Lecture Notes in Computer Science 3637

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

J. Manuel Moreno Jordi Madrenas
Jordi Cosp (Eds.)

Evolvable Systems: From Biology to Hardware

6th International Conference, ICES 2005
Sitges, Spain, September 12-14, 2005
Proceedings

Springer

Volume Editors

J. Manuel Moreno
Jordi Madrenas
Jordi Cosp
Technical University of Catalunya
Department of Electronic Engineering
Campus Nord, Building C4, c/Jordi Girona 1-3
08034 Barcelona, Spain
E-mail: {moreno,madrenas,jcosp}@eel.upc.edu

Library of Congress Control Number: 2005931797

CR Subject Classification (1998): B.6, B.7, F.1, I.6, I.2, J.2, J.3

ISSN 0302-9743
ISBN-10 3-540-28736-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-28736-0 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11549703 06/3142 5 4 3 2 1 0

Preface

The flying machines proposed by Leonardo da Vinci in the fifteenth century, the self-reproducing automata theory proposed by John von Neumann in the middle of the twentieth century and the current possibility of designing electronic and mechanical systems using evolutionary principles are all examples of the efforts made by humans to explore the mechanisms present in biological systems that permit them to tackle complex tasks. These initiatives have recently given rise to the emergent field of bio-inspired systems and evolvable hardware. The inaugural workshop, Towards Evolvable Hardware, took place in Lausanne in October 1995, followed by the successive events of the International Conference on Evolvable Systems: From Biology to Hardware, held in Tsukuba (Japan) in October 1996, in Lausanne (Switzerland) in September 1998, in Edinburgh (UK) in April 2000, in Tokyo (Japan) in October 2001, and in Trondheim (Norway) in March 2003.

Following the success of these past events the sixth international conference was aimed at presenting the latest developments in the field, bringing together researchers who use biologically inspired concepts to implement real systems in artificial intelligence, artificial life, robotics, VLSI design, and related domains. The sixth conference consolidated this biennial event as a reference meeting for the community involved in bio-inspired systems research.

All the papers received were reviewed by at least three independent reviewers, thus guaranteeing a high-quality bundle for ICES 2005. The conference included three keynote talks entitled: “Perspectives in Complex Systems Research”, “Neural Coding of Auditory Information” and “Evolutionary Approaches to Articulated Robot Locomotion”. The conference program consisted of 21 technical presentations and a panel debate. Additionally, a varied social program was set up to foster the exchange of ideas in an enjoyable environment.

We would like to thank the reviewers for their time and effort in reviewing all of the submitted papers. We would also like to thank the other members of the Organizing Committee. We wish to thank the following for their direct support of this conference: the Technical University of Catalunya (UPC), the Department of Electronics of the Technical University of Catalunya, the Spanish Ministry of Education, Culture and Sports, the Funding Agency for Universities and Research of the Generalitat de Catalunya (AGAUR), and Xilinx, Inc. Last, but not least, we would like to thank all the authors who invested so much time and effort in their research work and decided to join us in making ICES 2005 a successful event.

And what is to come next? It is not so easy to make forecasts in a research field that is moving as fast as ours about findings and understanding relating to the basic mechanisms that underlie the living forms we can observe. Of course, technology will play a major role in allowing for an actual realization of these principles, and this is where nanotechnology and new FPGA architectures will provide the necessary

substrate. However, in our opinion it will be the close cooperation between bioscientists, mathematicians and engineers that will result in a framework able to permit the construction of artifacts with emergent properties similar to those we can see even in the simplest living being. For sure in the next ICES conference we will see most of the topics that we have covered in the past, including evolving hardware design (both digital and analogue); evolutionary hardware design methodologies; self-repairing hardware; self-replicating hardware; embryonic hardware and self-developing systems; morphogenesis; neural hardware and adaptive hardware platforms; autonomous robots; evolutionary robotics; and molecular computation. As for the new topics that will emerge in this research field, it is our feeling that the breakthroughs coming in the life sciences in the coming years will provide avenues for facing challenges that, like consciousness, still constitute what Schopenhauer termed the world's knot.

We hope you enjoy reading these proceedings as much as we enjoyed putting them together.

September 2005

J. Manuel Moreno
Jordi Madrenas
Jordi Cosp

Organization

Organizing Committee

General Chair	Juan Manuel Moreno Arostegui, Technical University of Catalunya (UPC), Spain
Program Co-chair	Gianluca Tempesti, Swiss Federal Institute of Technology, Lausanne, Switzerland
Program Co-chair	Joan Cabestany, Technical University of Catalunya (UPC), Spain
Local Chair	Jordi Madrenas, Technical University of Catalunya (UPC), Spain
Publicity Chair	Jordi Cosp, Technical University of Catalunya (UPC), Spain

International Steering Committee

Pauline C. Haddow, Norwegian University of Science and Technology, Norway
Tetsuya Higuchi, Electrotechnical Laboratory, Japan
Julian Miller, University of York, UK
Jim Torresen, University of Oslo, Norway
Andy Tyrrell, University of York, UK

Program Committee

Wolfgang Banzhaf, University of Newfoundland, Canada
Peter Bentley, University College London, UK
Stefano Cagnoni, Universitá di Parma, Italy
Prabhas Chongstitvatana, Chulalongkorn University, Thailand
Carlos A. Coello, CINVESTAV-IPN, Mexico
Marco Dorigo, Université Libre de Bruxelles, Belgium
Rolf Drechsler, University of Bremen, Germany
Marc Ebner, Universität Würzburg, Germany
Stuart J. Flockton, Royal Holloway University of London, UK
Dario Floreano, Swiss Federal Institute of Technology, Lausanne, Switzerland
Andrew Greensted, University of York, UK
Tim Gordon, University College London, UK
Darko Grundler, Univesity of Zagreb, Croatia
Pauline C. Haddow, Norwegian University of Science and Technology, Norway

David M. Halliday, University of York, UK
Alister Hamilton, Edinburgh University, UK
Arturo Hernandez Aguirre, Tulane University, USA
Francisco Herrera, University of Granada, Spain
Tetsuya Higuchi, Electrotechnical Laboratory, Japan
Masaya Iwata, National Institute of Advanced Industrial Science and Technology (AIST), Japan
Tatiana Kalganova, Brunel University, UK
Didier Keymeulen, Jet Propulsion Laboratory, USA
William B. Langdon, University College London, UK
Yong Liu, University of Aizu, Japan
Jason Lohn, NASA Ames Research Center, USA
Daniel Mange, Swiss Federal Institute of Technology, Lausanne, Switzerland
Karlheinz Meier, University of Heidelberg, Germany
Julian Miller, University of York, UK
David Montana, BBN Technologies, USA
Juan Manuel Moreno Arostegui, Technical University of Catalunya (UPC), Spain
Masahiro Murakawa, National Institute of Advanced Industrial Science and Technology (AIST), Japan
Eduardo Sanchez, Swiss Federal Institute of Technology, Lausanne, Switzerland
Lukas Sekanina, Brno University of Technology, Czech Republic
Moshe Sipper, Ben-Gurion University, Israel
Giovanni Squillero, Politecnico di Torino, Italy
André Stauffer, Swiss Federal Institute of Technology, Lausanne, Switzerland
Adrian Stoica, Jet Propulsion Lab, USA
Gianluca Tempesti, Swiss Federal Institute of Technology, Lausanne, Switzerland
Christof Teuscher, University of California, San Diego (UCSD), USA
Jon Timmis, University of Kent at Canterbury, UK
Adrian Thompson, University of Sussex, UK
Jim Torresen, University of Oslo, Norway
Gunnar Tufte, Norwegian University of Science and Technology, Norway
Andy Tyrrell, University of York, UK
Milan Vasilko, Bournemouth University, UK
Alessandro Villa, Université Joseph Fourier, Grenoble, France
Moritoshi Yasunaga, University of Tsukuba, Japan
Ricardo Zebulum, Jet Propulsion Lab, USA

Sponsoring Institutions

We wish to thank the following for their contribution to the success of this conference:

Ministry of Education, Culture and Sports of Spain
Funding Agency for Universities and Research of the Generalitat de Catalunya (AGAUR)
Technical University of Catalunya (UPC)
Department of Electronics of the Technical University of Catalunya
Xilinx, Inc.

Table of Contents

Fault Tolerance and Recovery

Platforms for Evolving Digital Systems

Evolution of Analog Circuits

Evolutionary Robotics

Evolutionary Hardware Design Methodologies

Bio-inspired Architectures

Applications

An Adaptive Self-tolerant Algorithm for Hardware Immune System

Wenjian Luo, Xin Wang, Ying Tan, Yiguo Zhang, and Xufa Wang

Department of Computer Science and Technology,
University of Science and Technology of China, Hefei 230027, China
{wjluo, ytan, xfwang}@ustc.edu.cn
{sinbarwx, ygzhang}@mail.ustc.edu.cn

Abstract. Hardware immune systems have been studied with some initial achievements in recent years. Hardware immune systems are inspired by biological immune systems and they are expected to have many interesting characteristics, such as self-adaptive, self-learning and fault tolerant abilities. However, as novel intelligent systems, hardware immune systems are faced with many problems. This paper focuses on autoimmunization that is an inevitable problem when designing a complex hardware immune system. After the co-stimulation mechanism of biological immune system is simply introduced as a metaphor, a novel self-adaptive and self-tolerant algorithm for hardware immune systems is proposed in this paper. Inspired by the co-stimulation mechanism, the algorithm endows hardware immune systems with the capability of self-tolerance by automatically updating detector set and making the self set more complete. It can increase the accuracy of detection and decrease the rate of false positive effectively. Results of simulation experiments demonstrate the validity of this algorithm.

1 Introduction

Many works have been devoted to computational methods that are inspired by biological immune system in recent years [1-2]. As novel computational methods of Computational Intelligence (CI), this kind of research is called as Artificial Immune Systems (AISs) or methods. Among the many works about AIS, the concept of hardware immune systems is a younger one, which is proposed as a novel approach to designing a kind of hardware system with the fault tolerant ability [3-4].

So far, some works about hardware immune system have already been done. The architecture of a hardware immune system is firstly discussed and studied by D. W. Bradley and A. M. Tyrrell [3]. Also, A. M. Tyrrell and his colleagues proposed the concept of Immunotronics, and tried to construct a new theory about the design of fault tolerant hardware [5-6]. Based on Embryonic Array, R. Canham and A. M. Tyrrell proposed a multi-layered hardware artificial immune system with learning ability, which used the fact that the immune system consists of acquired immune subsystem and innate immune subsystem for reference. The acquired layer of the immune system monitors the behaviors of system for unusual activities, and the non-learning innate layer is then employed to localize the fault if possible [7]. R. Canham and A. M. Tyrrell also developed a novel

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 1 – 11, 2005.

artificial immune system, in which a detector of an immune system can be defined as a column in a 2-D feature space, and the generation and learning of detectors are fully automatic. It has been applied to robotics as an error detection system [8]. A. Tarakanov and D. Dasgupta proposed a novel architecture for building immunochips. The immunochip, by which information can be processed in a parallel and distributed manner, was evaluated with the problem of detecting dangerous ballistic situations in near-Earth space [9].

Generally, when the negative selection algorithm [10] is used to perform error detection in a complex fault tolerant hardware system, a complete set of self strings can not be obtained. Therefore, a part of matured detectors could become a threat to the system under monitoring because these detectors may match some unknown self strings. This problem is similar to the autoimmunization in biological immune system. For complex hardware systems, this problem seems inevitable, but there is no effective solution up to now.

Inspired by the co-stimulation mechanism which is used to maintain self-tolerance in biological immune systems, an adaptive self-tolerant algorithm for hardware immune system is proposed in this paper, it adopts Concurrent Error Detection (CED) technology [11] to provide the co-stimulation signal for the error detection system. It is named as ASTA-CED (the Adaptive Self-tolerant Algorithm with Concurrent Error Detection). The co-stimulation signal drives the error detection system to update the detector set automatically, delete detectors which bring autoimmune behaviors and generate new valid detectors. Therefore, ASTA-CED can avoid the occurrences of autoimmunization. Simulation experiments are carried out to show that this proposed algorithm can increase the accuracy of detection and decrease the ratio of false positives effectively.

The co-stimulation mechanism of biological immune system is simply discussed in section 2. Section 3 gives an introduction of the ASTA-CED in detail. Section 4 demonstrates the design of simulation experiments and the experimental results. And discussions are also given in section 4. Finally, section 5 is devoted to conclusions and future studies.

2 Immune Metaphor

In the natural immune system, an inactive T-cell's activation needs not only the antigen recognition signal (the first signal), but also co-stimulation (the second signal) [12]. The source of the second signal can be various, mainly coming from the combination of B7 molecules on the surface of antigen presentation cells (APC) and CD28 molecules on the surface of T-cells. Although the second signal does not have specificity, without the second signal, a T-cell that has already obtained the first signal will become an anergy cell (which can not take its own responsibility), and even die. The activation process of a T-cell is presented in Fig. 1 [12].

Among many kinds of cells interacting with T-cells, only the professional APCs (playing a professional role of presenting a peptide of an antigen to T-cells and providing other corresponding signals) can provide the first and second signals to activate T-cells at the same time [12]. If a T-cell recognizes an antigen's peptide from

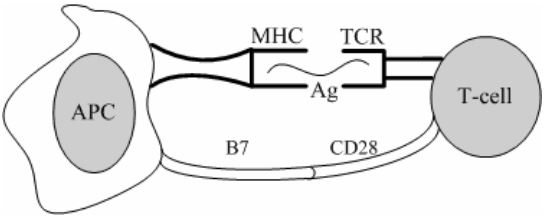


Fig. 1. This briefly shows that the activation of a T-cell should get the first and second signals. Here both first and second signals come from APC. However, the second signal, as a signal of co-stimulation, can come from various immune cells or molecules.

cells other than professional APCs, generally it will become an anergy cell because of the lack of the second signal. In fact, this is not likely to be a bad thing, because the antigen recognized is generally a self antigen at this time. In the biological immune system, new birth lymphocytes will undergo a maturation process, in which lymphocytes that bind with self proteins are destroyed. Hence, when released within the body, binding to a protein indicates it is non-self and may be a harmful pathogen. But the fact indicates that not all self proteins are presented to the maturing lymphocytes. This means some of the matured lymphocytes are still dangerous to the body. Thus, making the lymphocytes threatening the body become anergy cells or dies. This process is very helpful for the maintaining of self-tolerance [12].

Artificial hardware immune systems are presented with a similar problem, the current learning requires a period of fault free operation during which all the self states are presented. Although there are applications where this is possible, this can become a non-trivial task in some complex systems [7]. So, a mechanism to provide the second signals for artificial hardware immune systems is required.

3 ASTA-CED Algorithm

The negative selection algorithm is used for performing the detection of invalid state transitions. The negative selection algorithm, developed by Forrest and her colleagues [10, 13], is based on the generation process of T-Cells within the immune system. Forrest and her colleagues use a string to represent the self and non-self individuals. Partial matching between these self strings and non-self strings is used as a matching rule to distinguish between self and non-self. A set of detector strings are generated such that they do not match with all self strings, and they only match with non-self strings. Hence, the matching between a detector and the strings being protected gives an indication that some abnormal behaviors have occurred, and this indication is used as the first signal to the error detection system. In ASTA-CED algorithm, strings are used for representing the system's state transition but not just states because invalid state transitions can occur between valid states.

Concurrent Error Detection (CED) is widely used in highly dependable computing systems. It is a kind of on-line parity checking technology [11]. In the ASTA-CED algorithm, CED is used for performing parity checking on system's outputs and gen-

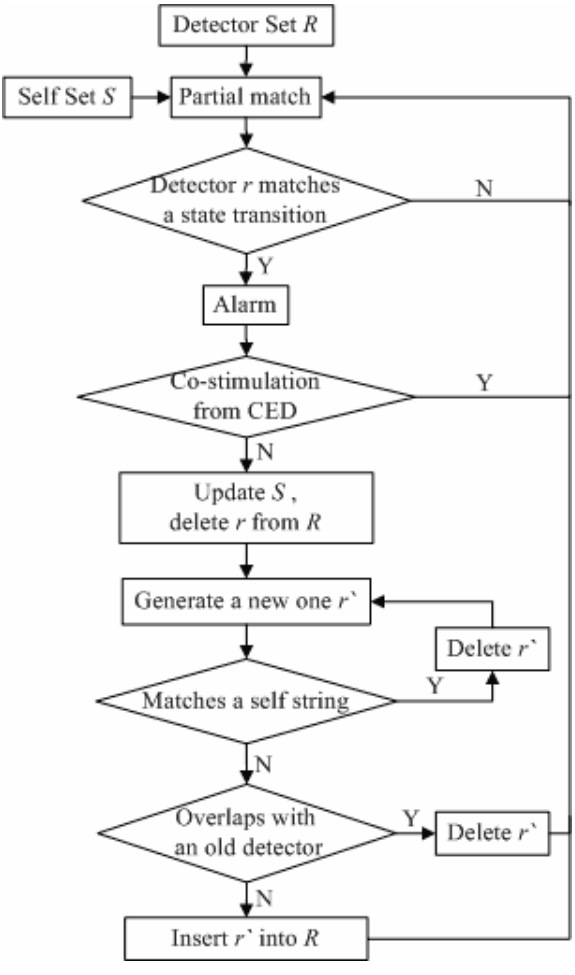


Fig. 2. This is the flow chart of ASTA-CED algorithm. The CED is used as a co-stimulation signal in ASTA-CED. Driven by this co-stimulation signal, the self set becomes more and more complete and the detector set evolves to be more and more efficient.

erating co-stimulation signals, because it is a simple and comparatively inexpensive technology to implement.

The ASTA-CED algorithm can be specified as shown in Fig. 2, in which S is the set of self (valid state transition) strings, and R is the set of detectors. The following is the description of the algorithm shown in Fig. 2. Here, it assumes that the initial set S is incomplete because a complete self set can not be obtained in general.

(1) Perform partial matching between state transitions and detectors in R one by one;
(2) If a detector r matches a state transition string, go to (3), or else back to (1);
(3) Report the error, if there is no co-stimulation from CED, go to (4), or else back to (1);

(4) Update the set *S* by inserting the new self string obtained by detector *r*, and delete *r* from *R*;
(5) Generate a new premature detector *r'* randomly;
(6) If *r'* doesn't match any string in *S*, go to (7), or delete it and back to (5);
(7) If *r'* is included in the current set *R*, delete it, or else insert it into *R*;
(8) Back to (1).

The aim of steps (5)-(8) is to generate a new detector and prevent the generating of new detectors from consuming too many resources of the system. The process of error detection can be regarded as the evolutionary process of the detector set *R*, by which the algorithm endows the hardware immune system with the capability of adaptive self-tolerance.

In the following experiments, every state transition of the whole string space appears once in a single cycle. When the detector set undergoes such a cycle, in fact it has evolved for one generation.

4 Experiments

4.1 Design of the Experiments

Based on the experiment of error detection of Finite State Machine (FSM) designed by A. M. Tyrrell [5], a co-stimulation generating module – CED and a controller are added to the simulation experiment, which is described as shown in Fig. 3. Partial matching of a state transition string from the FSM and a detector string from the detector set *R* will generate the first signal to the controller, and send an alarm to the results record module. At the same time, the self flag (which is set for every self string in *S* in advance) of the current state transition is sent to the results record module too. So the results recorded can be used for validating the performance of the ASTA-CED. In the CED module, the result of parity checking of the system's output will be sent to the controller as the second signal (co-stimulation), and then the controller decides whether *R* need to be updated according to the first and second signals. If necessary, *R* would be updated.

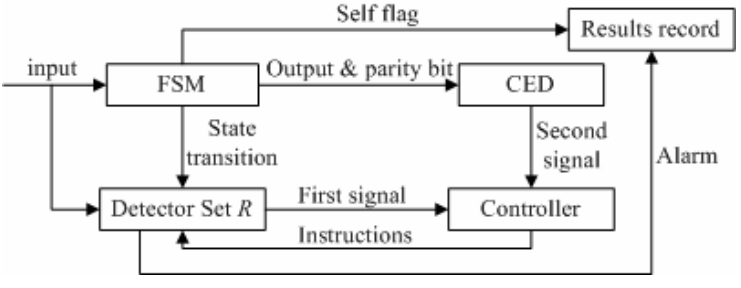


Fig. 3. This shows the design of the simulation experiment. In this experimental system, the CED is used to generate the second signal. The first and second signals are sent to the Controller Module that justify whether a matching result represented by the first signal is really an abnormal behavior or not. The final experimental results are stored in Results Record module.

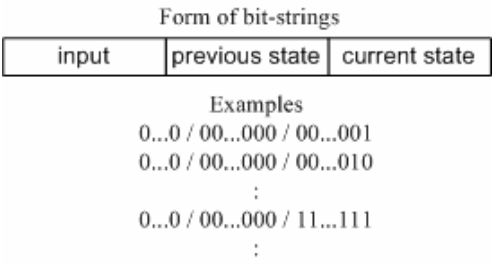


Fig. 4. This shows the bit-string representation of state transitions. Only previous state and current state are used because only the state transition is considered in this paper. The input state is ignored.

Here it supposes that valid state transitions generate valid outputs, and invalid state transitions generate invalid outputs, moreover, and every invalid output just fails at a single bit (here, the "invalid outputs" means outputs that fails to pass the parity checking). Bit-strings are used for representing state transitions and detectors in a form shown in Fig. 4 [4].

4.2 Results

The ASTA-CED algorithm is compared with the traditional Negative Selection Algorithm (traditional NSA) which does not have a co-stimulation mechanism in the same situation. The length of the bit-string is 10, and then the size of string space O is 1024. The number of total self strings N_s is fixed at 60. The partial match length c is 8.

The parameter a, which is the proportion of self strings already known in advance among the complete set of self strings, is set to {1.0, 0.9, 0.8, …, 0.1} for observing the change of the results against it.

Every state transition in space O appears for 40 times in an independent run of the algorithm. In other words, the detector set R in ASTA-CED algorithm will evolve for 40 generations in an independent run. And the results take the average values over 15 independent runs for every value of the parameter a.

The self set of every independent run is generated randomly. The size of initial immature detector set is fixed to 324. Both initial immature and mature detector sets of ASTA-CED are the same as that of traditional NSA. The size of initial mature detector set is N_{r1}, Table 1 lists the average values of N_{r1} against values of a.

It should be noted that the detector set of traditional NSA is fixed. However, the detector set of ASTA-CED will evolve step by step, and its evolution is driven by the co-stimulation signal (i.e. CED).

In an independent run, it is assumed that PS is the number of valid state transitions detected as normal behaviors by the system; FS is the number of valid state transitions detected as abnormal behaviors; PN is the number of invalid state transitions detected as normal behaviors; and FN is the number of invalid state transitions detected as abnormal behaviors. The following three statistical results are defined to make the comparisons between ASTA-CED and traditional NSA.

$$P_r = FN/(FS + FN), \quad P_w = FS/(PS + FS), \quad P_f = PN/(PN + FN)$$

Table 1. The average sizes of initial mature detector sets of the two algorithms against values of a. The mature detector set is the detector set after being filtered out the immature detectors that matches self individuals from the immature detector set. The size of initial immature detector set is always set as 324.

a	1.00	0.90	0.80	0.70	0.60
Average N_{r1}	195.5	206.5	220.0	225.0	240.5
a	0.50	0.40	0.30	0.20	0.10
Average N_{r1}	253.5	264.5	274.5	296.0	309.5

Table 2. P_r of traditional NSA and ASTA-CED

a	Traditional NSA	ASTA-CED	
		Average value from 1 to 20 generations	Average value from 21 to 40 generations
1.00	1.0000	1.0000	1.0
0.90	0.9934	0.9993	1.0
0.80	0.9881	0.9989	1.0
0.70	0.9816	0.9978	1.0
0.60	0.9741	0.9969	1.0
0.50	0.9698	0.9963	1.0
0.40	0.9634	0.9953	1.0
0.30	0.9586	0.9947	1.0
0.20	0.9505	0.9936	1.0
0.10	0.9466	0.9923	1.0

Table 3. P_w of traditional NSA and ASTA-CED algorithms

a	Traditional NSA	ASTA-CED	
		Average value from 1 to 20 generations	Average value from 21 to 40 generations
1.00	0.0000	0.0000	0.0
0.90	0.0814	0.0081	0.0
0.80	0.1546	0.0138	0.0
0.70	0.2441	0.0281	0.0
0.60	0.3580	0.0407	0.0
0.50	0.4231	0.0476	0.0
0.40	0.5289	0.0635	0.0
0.30	0.6103	0.0696	0.0
0.20	0.7567	0.0838	0.0
0.10	0.8381	0.1046	0.0

Table 4. P_f of traditional NSA and ASTA-CED algorithms

a	Traditional NSA	ASTA-CED	
		Average value from 1 to 20 generations	Average value from 21 to 40 generations
1.00	0.2431	0.2431	0.2431
0.90	0.2218	0.2413	0.2421
0.80	0.1813	0.1970	0.1974
0.70	0.1714	0.1971	0.1990
0.60	0.1403	0.1771	0.1797
0.50	0.1319	0.1807	0.1839
0.40	0.1112	0.1443	0.1475
0.30	0.0982	0.1635	0.1673
0.20	0.0722	0.1729	0.1766
0.10	0.0514	0.1393	0.1449

Table 2, Table 3 and Table 4 show the comparison between ASTA-CED and traditional NSA on average values of P_r, P_w and P_f. The comparisons in the first 20 generations, those are shown separately, indicate that on P_r and P_w, the results of ASTA-CED algorithm is much better than that of traditional NSA, this is due to the detectors matching self strings are deleted and new detectors are generated, and the P_r of the two algorithms increase with a, and both the P_w of them decrease as a increases; but only P_f of ASTA-CED algorithm is a little higher than that of traditional NSA, this will be discussed in the next section.

The comparisons in the last 20 generations are also shown separately in Table 2, 3 and 4. It is clear that in the last 20 generations, there is no detector matching a self string. In other words, the evolution of detector set R has been finished before these generations. For P_f, the results are similar to that of the first 20 generations.

Some other aspects of ASTA-CED algorithm can be observed in Fig. 5. Firstly, in Fig. 5(a), it indicates that the number of new detectors inserted into R is smaller than

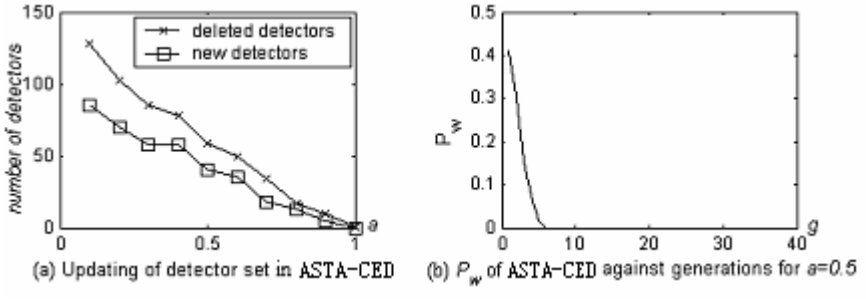


Fig. 5. This shows some characteristics of ASTA-CED. (a) The updating process of detector set R. (b) The changing curve of P_w in generations.

Table 5. The average sizes of final detector sets of ASTA-CED against values of a

a	1.00	0.90	0.80	0.70	0.60
Average N_{r2}	195.5	201.5	215.5	208.5	225.5
a	0.50	0.40	0.30	0.20	0.10
Average N_{r2}	235.0	244.0	246.5	263.0	266.5

the number of deleted detector from R, this means that the size of R has shrunk in ASTA-CED algorithm. The size of final detector set of ASTA-CED is N_{r2}. Table 5 lists the average values of N_{r2} against values of a over 15 independent runs. On the other hand, in Fig. 5(b), the probability of false positive P_w declines rapidly to 0 against generations.

4.3 Discussions

In Table 1, Table 2 and Table 3, the experimental results indicate that ASTA-CED algorithm is better than the traditional NSA both at P_r and P_w, but is poorer than traditional NSA at P_f. The following is a brief discussion about this phenomenon.

Since partial matching mechanism is being used here, even if a complete detector set can be obtained, the self strings and non-self strings matching over c contiguous bits will result in the presence of undetectable non-self strings, namely *holes* [13-14]. There are strings in the "hole" that are unable to be detected because any detectors matching them would also match some self strings. The relevant analyses of this phenomenon have been investigated by D'haeseleer in [14].

In Fig. 6, for a particular partial match length c, given S_0 as the incomplete set of self got in advance and S_1 as the more complete set of self obtained in the error detection procedure. Suppose h_0 and h_1 are the holes induced by S_0 and S_1 respectively, and R and R' are the detector sets generated against S_0 and S_1 respectively, and

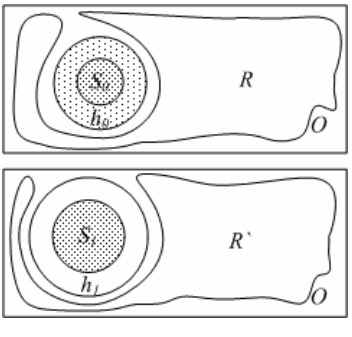


Fig. 6. Analyses on failure probability P_f. O is the whole string space. S_0 is the initial incomplete set of self, while S_1 is the more complete set of self obtained in the course of error detection. R and R' are the detector sets generated against S_0 and S_1 respectively, while h_0 and h_1 are the holes induced by S_0 and S_1 respectively.

O is the whole string space. It is noted that here R and R' are regarded as complete or almost complete detector sets. This can be guaranteed by randomly generating enough premature detectors or using the greedy detector generating algorithms proposed in [13].

Firstly, it should be noted that S_0 is a subset of S_1, because S_1 is more complete than S_0. Strings in h_0 can not be detected by R, and strings in h_1 are undetectable to R'. Secondly, there can be self strings in the h_0, but no self string in the h_1. Suppose that all of self strings in h_0 make up of a set of S_{h0}. Finally, according to the counting method of holes in [14], $(h_0 - S_{h0})$ is a subset of h_1. Therefore, on the one hand, non-self strings in h_1 can not be detected by R', while possible to be detected by R. On the other hand, non-self strings in $(h_0 - S_{h0})$ can not be detected by R' and R. Thus, if P_{f0} is the failure probability of R, and P_{f1} is the failure probability of R', it can be concluded that $P_{f0} < P_{f1}$. Fortunately, the problem with holes can be avoided by adopting a matching rule with a variable matching length c [14].

5 Conclusion

In the design of a complex hardware immune system, autoimmunization is an inevitable problem. In this paper, a novel self-adaptive algorithm, namely ASTA-CED, is proposed as a solution. Concurrent Error Detection (CED) technology is used for providing co-stimulation to the error detection system, and the detector set is updated automatically, then the occurrence of autoimmunization can be avoided. Results of simulation experiments prove that this new algorithm has increased the accuracy of detection and decreased the ratio of false positives effectively.

However, there are also some future works that should be studied to improve this algorithm. Firstly, CED is a technology based on parity checking, so it can only detect single bit fault. Secondly, since CED is used for checking the system's output in this paper, this might be unsuitable when considering some applications. We are looking for more suitable co-stimulation mechanisms, and we will explore better approaches to building up self-tolerance in hardware immune systems.

Acknowledgement. This work is supported by NSFC Foundation (No. 60404004), Nature Science Major Foundation from Anhui Education Bureau (No. 2004kj360zd) and Chinese Post-doc Science Foundation (No. 2003034433).

References

[1] Castro, de L.N., Timmis, J.. Artificial Immune Systems: a New Computational Intelligence Approach. Springer-Verlag, London, (2002)

[2] Dasgupta, D., Ji, Z, et al. Artificial Immune System(AIS) Research in the Last Five Years. Proceedings of the IEEE Congress on Evolutionary Computation (CEC'2003), Canberra, Australia, (2003) 123-130

[3] Bradley, D.W., Tyrrell, A.M.. The Architecture For A Hardware Immune System. Proceedings of the 3rd NASA/dod Workshop on Evolvable Hardware, Long Beach, Cailfornia, July 12 – 14 (2001) 193-200

[4] Bradley, D.W., Tyrrell A.M.. A Hardware Immune System for Benchmark State Machine Error Detection. Proceedings of the 2002 Congress on Evolutionary Computation, Honolulu, USA, (2002) 813-818

[5] Bradley, D.W., Tyrrell A.M.. Immunotronics - Novel Finite-State-Machine Architectures with Built-in Self-Test Using Self-Nonself Differentiation. IEEE Transactions on Evolutionary Computation, Vol.6, 3 (2002) 227-38

[6] Tyrrell, A.M.. Computer know Thy self!: A Biological Way to Look at fault Tolerance. Proceedings of 2nd EuroMicro/IEEE Workshop Dependable Computing Systems, September, (1999) 129–135

[7] Canham, R., Tyrrell, A.M.. A Learning, Multi-Layered, Hardware Artificial Immune System Implemented upon an Embryonic Array. Proceedings of 5th International Conference on Evolvable Systems, (2003) 174–185

[8] Canham, R., Jackson, A.H., Tyrrell, A.M.. Robot Error Detection Using an Artificial Immune System. Proceedings of NASA/DoD Conference on Evolvable Hardware, July (2003) 199 – 207

[9] Tarakanov, A., Dasgupta, D.. An Immunochip Architecture and its Emulation. Proceedings of NASA/DoD Conference on Evolvable Hardware, July 15-18 (2002) 261-265

[10] Forrest, S., Perelson, A.S., Allen, L., Cherukuri, R.. Self-nonself Discrimination in a Computer. Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy. Los Alamitos, CA: IEEE Computer Society Press, (1994) 202-212.

[11] Chaohuang, Z., Saxena, N., McCluskey, E.J.. Finite State Machine Synthesis with Concurrent Error Detection. Proceedings of International Test Conference, Sept. 28-30 (1999) 672 – 679.

[12] Guang-yan, Zhou. Principles of Immunology. Scientific and Technical Documents Publishing House, Shanghai (2000)

[13] D'haeseleer, P., Forrest, S., Helman, P.. An Immunological Approach to Change Detection: Algorithms, Analysis and Implications. Proceedings of the IEEE Symposium on Security and Privacy, IEEE Computer Society Press (1996)

[14] D'haeseleer, P.. Further Efficient Algorithms for Generating Antibody Strings. Department of Computer Science, University of New Mexico, Technical Report CS95-3 (1995)

Consensus-Based Evaluation for Fault Isolation and On-line Evolutionary Regeneration

Kening Zhang, Ronald F. DeMara, and Carthik A. Sharma

Department of Electrical and Computer Engineering,
University of Central Florida, Orlando, FL 32816-2450
demara@mail.ucf.edu

Abstract. While the fault repair capability of Evolvable Hardware (EH) approaches have been previously demonstrated, further improvements to fault handling capability can be achieved by exploiting population diversity during all phases of the fault handling process. A new paradigm for online EH regeneration using Genetic Algorithms (GAs) called Consensus Based Evaluation (CBE) is developed where the performance of individuals is assessed based on broad consensus of the population instead of a conventional fitness function. Adoption of CBE enables information contained in the population to not only enrich the evolutionary process, but also support fault detection and isolation. On-line regeneration of functionality is achieved without additional test vectors by using the results of competitions between individuals in the population. Relative fitness measures support adaptation of the fitness evaluation procedure to support graceful degredation even in the presence of unpredictable changes in the operational environment, inputs, or the FPGA application. Application of CBE to FPGA-based multipliers demonstrates 100% isolation of randomly injected stuck-at faults and evolution of a complete regeneration within 135 repair iterations while precluding the propagation of any discrepant output. The throughput of the system is maintained at 85.35% throughout the repair process.

1 Introduction

Evolutionary mechanisms can actively restore mission-critical functionality in SRAM-based reprogrammable devices such as *Field Programmable Gate Arrays* (FPGAs). They provide an alternative to device redundancy for dealing with permanent degradation due to radiation-induced stuck-at-faults, thermal fatigue, oxide breakdown, electromigration, and other local permanent damage without the increased weight and size normally associated with spares. Hence, recent research has focused on employing the capability for reconfiguration inherent in field programmable devices to increase reliability and autonomy [1], [2], [3], [4], [5]. In these experiments, fault-tolerance is evolved at design time, or achieved at repair-time using evolution after taking a detected failed unit offline. In both cases, GAs provided a population-based optimization algorithm with the objective of producing a single best-fit individual as the final product. They rely on a pre-determined static fitness function that does not consider an individual's utility relative to the rest of the

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 12 – 24, 2005.

population. The evaluation mechanisms used in previous approaches depend on the application of exhaustive test vectors to determine the individual with the best response to all possible inputs. However, given that partially complete repairs are often the best attainable [4], [2], other individuals may outperform the best-fit individual over the range of inputs of interest. In particular, there is no guarantee that the individual with the best absolute fitness measure for an exhaustive set of test inputs will correspond to the individual within the population that has the best performance among individuals under the subset of inputs actually applied. Thus, exhaustive evaluation of regenerated alternatives is computationally expensive, yet not necessarily indicative of the optimal performing individual among a population of partially correct repairs. Hence, two innovations are developed herein for self-adaptive EH regeneration:

1) Elimination of additional test vectors, and
2) Temporal Assessment based on aging and outlier identification

In *Consensus-based Evaluation (CBE),* an initial population of functionally identical (same input-output behavior), yet physically distinct (alternative design or place-and-route realization) FPGA configurations is produced at design time. During runtime, these individuals compete for selection based on discrepancy favoring fault-free behavior. Discrepant behaviour, where the outputs of two competing individuals do not agree on a bit-by-bit basis, is used as the basis for the performance evaluation process. Any operationally visible fault will decrease the fitness of just those configurations that use it. Over a period of time, as the result of successive comparisons, a consensus emerges from the population regarding the relative fitness of all individuals. This allows the classification of configurations into ranges of relative reliabilities based on their observed performance during online operation.

2 Related Work

Adaptive regeneration has been investigated as an alternative to using pre-determined spares. Most researchers [2], [3], [5], [6] focus on using traditional GAs to identify a single best-fit individual at the termination of the evolutionary computation. Keymeulen, Stoica, and Zebulum [1] use a design-time emphasis to improve fault tolerance. They develop evolutionary techniques so that a circuit is initially designed to remain functional even in presence of various faults. Their *population-based* fault tolerant design method evolves diverse circuits and then selects the most fault-insensitive individual. In this paper we propose a system that achieves improved fault tolerance by using a runtime adaptive algorithm that emphasizes the utilization of responses observed during the actual operation of the device. While their population-based fault tolerance approach provides passive run-time tolerance, CBE is dynamic and actively improves the fault tolerance of the system according to environmental demands.

Yao and Liu [7] emphasize that in evolutionary systems, the population contains more information than any one individual. They develop two examples to demonstrate the use of the information contained in the population in the domains of artificial neural networks and rule based systems respectively. The last population is

used efficiently and out-performs the single best-fit individual in these two examples. [8] presents four methods for combining the different individuals in the final population to generate the solution. They provide results for three data sets, namely the Australian credit card assessment problem, the heart disease problem and the diabetes problem, which show that solutions obtained by combining individuals outperform any single individual. While the authors devise a method to utilize the information contained in the population to improve the final solution, they fail to use the information in the population to improve the learning and optimization process itself. Also, the authors emphasize that learning systems are different from optimization problems, and that information contained in the population is only useful in learning systems. The proposed approach clearly indicates that even optimization and repair problems can benefit from population information. More recently, in [9] the authors describe using fitness sharing and negative correlation to create a diverse population of solutions. A combined solution is then obtained using a gating algorithm that ensures the best response to the observed stimuli. In EHW, it may not always be possible to combine solutions without additional physical resources that may be fault-prone. In our approach, all individuals in the population are recognized as possible solutions, with the best emerging candidate being selected based on their runtime response and performance track record. The authors also claim that applying the described techniques to EHW should be a straightforward matter, but do not describe any applications or examples. They state the absence of an optimal way of predicting the future performance of evolved circuits in unseen environments. We show that it is possible for an adaptive system to keep track of the relative performances of individuals and implicitly build a consensus.

Layzell and Thompson [10] identify Populational Fault Tolerance (PFT) as an inherent quality of EHW. They state that due to the incremental nature of evolutionary algorithms, the solution changes along the course of evolution to adapt to faults. The evolutionary history of the evolved circuit was used to arrive at the conclusion that PFT is an inherent quality in evolutionary design due to the incremental incorporation of additional components into a prototype depending on conditions. They speculate that PFT is less likely to occur for online evolution in varying environments. An evolutionary process that uses absolute fitness measures and exhaustive tests may not be able to provide adaptive fault tolerance.

Previous research has not focused on leveraging the robustness of a population to improve the detection and isolation phases, or to achieve an online evolution process. Problems related to fault tolerance in online evolution identified by the existing approaches are addressed by the new Consensus-based Evaluation scheme. Online evolution defines an essentially different problem from a traditional GA optimization problem. To address the problem effectively, a new fitness evaluation paradigm is required. With relative fitness measures based on competition, a running consensus is produced regarding the fitness of individuals in repsonse to the actual environmental stimuli. This can be used by the regeneration process to adapt to runtime requirements and improve the fault tolerance of the population. The CBE approach presents a new online adaptive repair mechanism that fully exploits the advantages of population-based evolutionary methods. It utilizes a *temporal voting* approach whereby the outputs of two competing instances are compared at any instant and alternative pairings are considered over time. The presence or absence of a discrepancy is used

to adjust the *discrepancy values* (DVs) of both individuals without rendering any judgment at that instant on which individual is actually faulty. The faulty, or later exonerated, configuration is determined over time through other pairings of competing configurations. The competitive process is applied repeatedly to form a strong consensus across the diverse pool of alternatives. The fitness of individuals is determined through this continuing runtime process by evaluating the real time performance of individuals in comparison to others in the population. Instead of using an absolute fitness function, with the concomitant exhaustive testing, relative discrepancy values are used as the threshold to identify faulty individuals. Also, the system actively selects individuals that perform the best, given the current environment. Healthy individuals are used to achieve the repair of individuals affected by faults. The proposed approach makes full use of the fact that repair complexity is far less than design complexity. CBE achieves improved fault tolerance by making extensive use of the information contained in the population – both as raw material for creating new individuals, and as information that enables faster and more accurate fault isolation. Any improvement in the fault isolation process speeds up the regeneration process by directing the GA search in the proper direction. The use of a relative fitness measure and temporal consensus improves the fault tolerance and adaptability of the population.

3 Autonomous Regeneration Using CBE

A GA performs a multi-directional search by maintaining a population of potential solutions and encouraging information formation and exchange along these directions. By encouraging direct competition between individuals in the population, a relative fitness measure based on consensus can be generated. The objective fitness function used in traditional GAs can be effectively replaced by the emergent consensus and relative fitness measure. The relative fitness measure is inherently dynamic, and by using an *Evaluation Window* for the individuals, an accurate reflection of the environmental conditions and changes can be achieved. Multiple potential directions for future exploration can be created and utilized depending on the conditions prevalent during the evolutionary process.

In the CBE approach, an initial population of *Pristine* individuals is created by manual design. These primordial configurations are functionally-identical (same input-output behavior), yet they utilize physically-distinct resources (alternative design or place-and-route implementations). For puposes of illustration, assume two competing *half-configurations* labeled Functional Logic Left ("*L*") and Functional Logic Right ("*R*") are loaded in tandem on the physical FPGA platform. The half-configurations occupy mutually exclusive physical resources to implement identical functionality. This realizes a conventional Concurrent Error Detection (CED) arrangement to identify at least any single resource fault with certainty [11]. As in traditional CED approaches, comparison of the outputs of the two resident half-configurations will produce either *discrepant* or *matching* outputs which will indicate the presence or absence of faulty resources in the FPGA hardware platform respectively.

Under CBE, whenever two half-configurations disagree, the *Discrepancy Value (DV)* of both half-configurations are incremented. By repeated pairing over a period of time, only those half-configurations which do not use faulty resources will eventually become preferred. This is because the DV of a faulty half-configuration is always increased regardless of its pairing, yet the DV of fault-free half-configurations which are paired together do not increase. This process occurs as part of the normal processing throughput of the FPGA without additional test vectors or other diagnostic routines. The determination of a configuration's health state is based on its cumulative DV relative to DV of the other individuals in the population evaluated over a period called the *Evaluation Window*, denoted by E^W.

3.1 CBE Procedure

The procedure begins with pre-designed individuals that are fault-free. These individuals are divided into two groups, *L* and *R,* where each group of individuals uses mutually exclusive physical resources. This is essential to ensure that one individual each form both groups can reside and compete in tandem on the FPGA. In addition, every individual can belong to one of four states – *Pristine, Suspect, Under-repair* or *Refurbished.* In the beginning, all individual are pristine. At any given point of time, one individual each from the *L* and *R* groups compete with each other. State transistions occur according to the result of pairwise output comparison. A comparison can lead to two results - "$L=R$" and "$L\neq R$" indicating whether the two resident half-configurations produce either *matching* or *discrepant* outputs, respectively. When $L=R$ occurs then both individuals retain their *Pristine* state. However when their outputs disagree then both the configurations are demoted to the *Suspect* pool and the DV of both individuals is increased. Whenever such a transition occurs, a *Fault Alert* indicator is issued because two functionally-identical circuits disagree. Hence at least one resource fault must have occurred.

More formally, the *i*-th half configuration remains in the *Suspect* pool until its DV f_i evaluated over the preceding E^W pairings rises above the *Repair Discrepancy Value* ($f_i < DV_R$) which is defined as average DV of entire population accumulated over E^W. The *i*-th half-configuration is then marked as *Under Repair* until its DV drops below the *Operational Discrepancy Value* ($f_i \geq DV_O$) which is defined as average DV of the healthy individuals among the population (*Pristine, Suspect and Refurbished*) accumulated over E^W. Under the fault-free circumstance, $DV_O = DV_R$ until the faulty individuals appear in the population as a result of emergent hardware faults. Thereafter, f_{OT} is modified such that $DV_O \leq DV_R$ which provides dithering immunity such that the configuration is indeed *Refurbished.*

Over a period of time the DV of an individual could increase further and *complete regeneration* becomes possible though not necessarily externally distinguishable from *partial regeneration*. Competing half-configurations remain *Refurbished* unless their DV rises above the Repair DV, at which time they again demoted to the *Under Repair* state.

The procedural flow of the CBE algorithm that calculates the health state transitions is depicted in Figure 1. After initialization, *Selection* of the *L* and *R* half-configurations occurs which are then loaded into the FPGA. The *Detection* process is

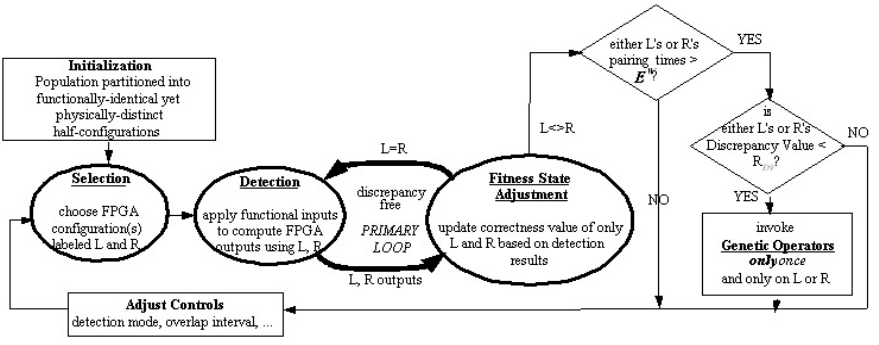


Fig. 1. Procedural Flow in the CBE Technique

conducted when the normal data processing inputs are applied to the FPGA. Based on agreement or disagreement among the outputs of the two competing *L* and *R* half-configurations, *Discrepancy Value Adjustment* for both individuals occurs. The central *PRIMARY LOOP* representing *discrepancy-free* behavior can repeat indefinitely without any reconfiguration of the FPGA. Only when outputs disagree do alternate configurations need to be loaded. For *Under Repair* individuals, if $f_i > DV_R$ then Genetic Operators are invoked only once on the resident configurations. The modified configuration is then immediately returned to the pool of competing configurations and the Selection step is resumed under normal FPGA throughput processing operations.

3.2 Selection and Detection Process

The *Selection* and *Detection* processes are shown in Figure 2. The usual flow is for *Pristine, Suspect*, and then *Refurbished* individuals to be preferred in that order for one half-configuration. On the other hand, the other half-configuration is selected based on a stochastic process determined by the *Re-introduction Rate* (λ_R). In particular, *Under Repair* individuals are selected as one of the competing half-configurations on average at a rate equal to λ_R. Henceforth, this now genetically-modified configuration will be *re-introduced* into the operational throughput flow as a new competitor to potentially exhibit fault-free behavior against the larger pool of configurations not currently undergoing repair.

An additional innovation is that λ_R is not only a continuous variable, but can be adapted under autonomous control. In particular, we strive for Mean-Time-To-Repair (*MTTR)* < Mean-Time-Between-Failures *(MTBF)* by monitoring the ratio of the number of computations elapsed between and adjusting λ_R accordingly.

The Detection process is presented in the lower right corner of Figure 2. If a discrepancy is observed as a result of output comparison, the FPGA is reconfigured with a different pair of competing configurations and the output of the device is temporarily held to be recalculated by the newly selected *L* and *R* half-configurations. These repeated computations and comparisons imply no additional cost since the device remains online and operational and the normal data throughput continues uninterrupted.

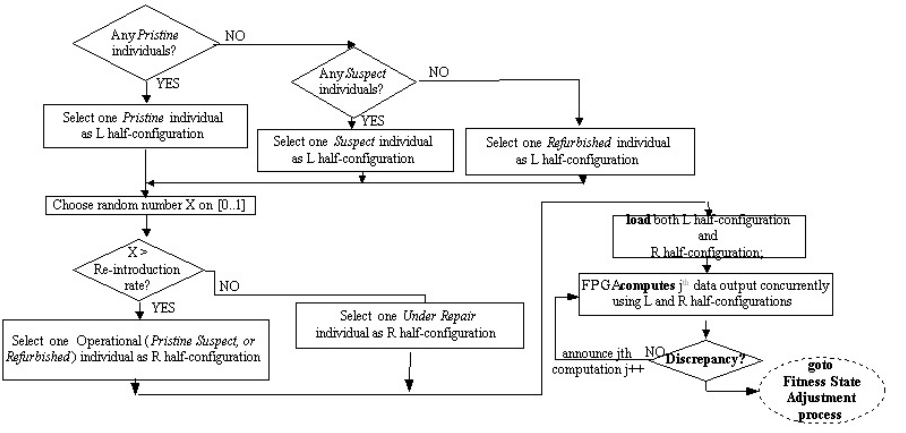


Fig. 2. Selection and Detection in the CBE Technique

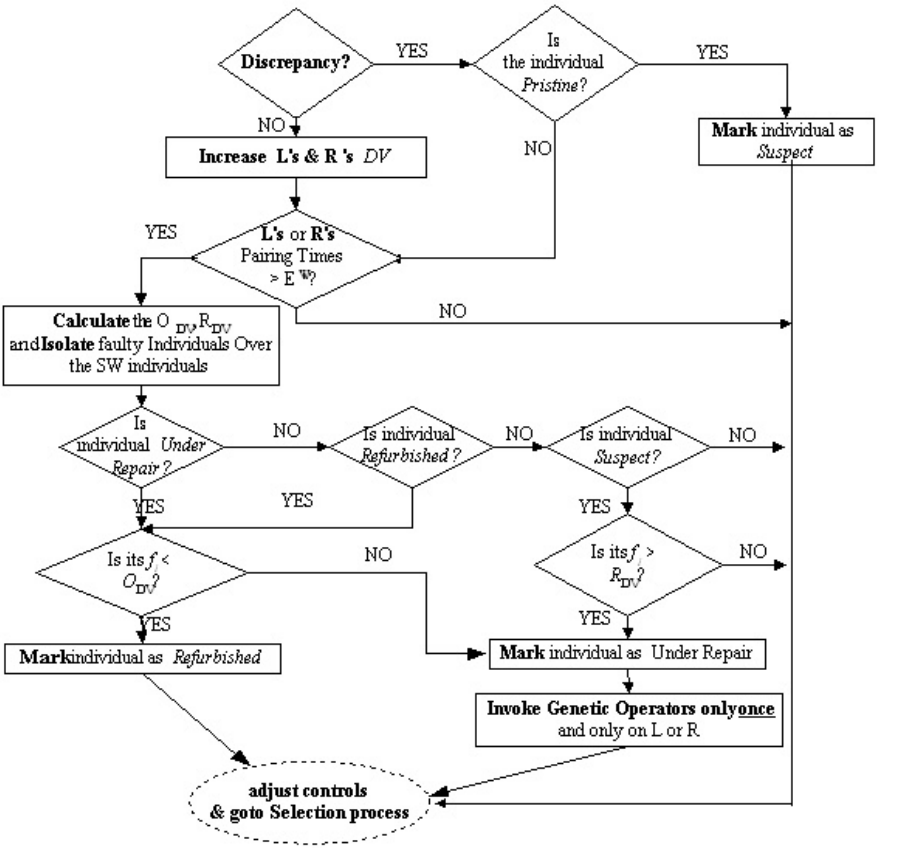


Fig. 3. Fitness State Adjustment Processes in the CBE Technique

3.3 Fitness State Adjustment Process

Figure 3 depicts the *Fitness State Adjustment Process* in CBE. Whenever a discrepancy is detected, the discrepancy values of the individuals involved are updated. The new discrepancy values are then compared to the *Repair Discrepancy Value* DV_R and *Operational Discrepancy Value* DV_O to determine whether the individuals move from one fitness state to another. Ideally, the repair and operational discrepancy values are computed over E^W comparisons for the population. As soon as the all the individuals in the population have completed at least E^W comparisons, new values of these thresholds are obtained. Since it may be impractical to wait for all individuals to complete the requisite iterations, an individual can undergo a state transition after it finishes E^W iterations. A *Sliding window* is defined, which reduces the latency involved in updating DV_R and DV_O by considering a subset of individuals instead of the whole population. With a sliding window, the values of these thresholds are updated upon the completion of the requisite number of iterations by the number of individuals defined by the sliding window. For *Under Repair* individuals, GA operators are invoked once every E^W iterations.

4 Evolutionary Fault Repair Circuit

The hypothetical FPGA structure used in the CBE approach is the same as that in Miller, Thomson[12]. The feed-forward combinational logic digital circuit uses a rectangular array of nodes with two inputs and one output. Each node represents a Look-up Table (LUT) in the FGPA device, and a Configurable Logic Block (CLB) is composed of four LUTs. In the array, each CLB will be a row of the array and two LUTs are represented as four columns of the array. There are five dyadic functions -- OR, AND, XOR, NOR, NAND -- and one unary-function NOT, each of which can be assigned to an LUT. The LUTs in the CLB array are indexed from 1 to *n.* Array routing is defined by the internal connectivity and the inputs/outputs of the array. Internal connectivity is specified by the connections between the array cells. The inputs of the cells can only be the outputs of cells with lower row numbers. Thus, the linear labelling and connection restrictions impose a feed-forward structure on the combinational circuit.

A 3×3 Multiplier is implemented using the above FPGA structure. XOR gates are purposely excluded from the initial designs which leads to designs with a higher number of the gates than conventional 3×3 Multiplier designs to increase the design space. The entire configuration needs 21 CLBs. The population of competing alternatives is then divided into two groups, *L* and *R*, where each group uses an exclusive set of physical resources. For crossover to occur such that offspring are guaranteed to utilize only mutually-exclusive physical resources with other resident half-configurations, a two-point crossover operation is carried out with another randomly selected *Pristine, Suspect* or *Refurbished* individual belonging to the same group. By enforcing speciation breeding occurs exclusively in *L* or *R*, and non-interfering resource use is maintained. The random crossover points are chosen along the boundary of CLBs so that intra-CLB crossover is not possible. The mutation

operator randomly changes the LUT's functionality or reconnects one input of the LUT to a new randomly selected output inside the CLB.

5 Experimental Results

An initial population of 20 fault-free configurations was partitioned into mutually exclusive sub-populations *L* and *R*, each containing 10 configurations. Varying stuck-at faults were injected into the architecture that represents permanent physical faults. Several fault isolation and regeneration experiments were carried out using a software simulator. The E^W used in the experiment is 600, which can statistically guarantee that all of 64 input combinations appear at the inputs at least once with probablity of 99.5%, when input combinations are selected at random. For the 3×3 multiplier, the total possible number of input combinations is 2^6=64. Thus $n = 64$ represents the total number of unique input combinations to the simulated FPGA. In the simulation, m $(0 \leq m \leq 64)$ is defined as the number of input combinations for which a fault is manifested at the output of the simulated circuit. The number of input combinations for which the output does not match the desired value measures the impact of a fault on an individual. Fault isolation characteristics are analyzed first without considering the regeneration process.

The second set of regenerative experiments investigates the regeneration of functionality using CBE. The GA uses a two-point crossover, with a crossover rate of 0.05 and the mutation rate is 0.8. The re-introduction rate is 10%. With the simulated FPGA remaining partially online, all of the regeneration experiments achieved full fault recovery within a few hundred repair operations with normal functional data input. During the regeneration period, data throughput is average 87.94. That is, only 13.16% of the total computations had to be recalculated in order to preclude propagation of discrepant outputs.

5.1 Fault Isolation Experiments

Pairs of individuals, one each from the *L* and the *R* groups are loaded on the FPGA in a repetitive random process. The outputs are compared to check for discrepancies. Judgment on the fault characteristics of an individual is reserved till it completes E^W pairings, and an *Observation Interval* is complete. A *Sliding Window* of evaluation is defined as five E^W, after which one observation interval is complete and individuals who have completed an E^W are evaluated to identify outliers. The DV of a faulty configuration will increase each time it is compared to another individual. A fault-free individual will see increases in its DV only when it is compared to a faulty individual. Individuals with a DV that exceeds the observed arithmetic mean by one standard deviation are identified as faulty. For example, if 1-out-of-64 outputs are affected in one *L* individual due to a fault, the expected DV of this individual after E^W pairings is DV_L= 1/64* E^W =9.375, assuming equal likelihoods for inputs. A faulty individual can be expected to be identified once every two observation intervals, since the width of each observation interval is defined by $5*E^W$. The average *DV* of the R individuals that this is paired with be DV_R =1/64* E^W /10=0.9375, assuming equal selection likelihoods.

Two metrics *Operational DV* (DV_O) and *Repair DV* (DV_R) are calculated and used in the CBA evaluation. DV_O is defined as arithmetic mean of the observed DV of all healthy individuals over a sliding winodw and the DV_R is defined as arithmetic mean of the DV of all individuals considered in the sliding window, including any that may be faulty. If no faulty individuals have been detected, DV_O will equal DV_R, otherwise the $DV_O < DV_R$ as the faulty individuals substantially increase the mean DV. DV_O *and* DV_R are subsequently used in the CBE fault repair mechanism to define the state transitions of individuals. If an individual has a DV < DV_O, it is probably fault-free and can be used for fault-free computation. If the DV of an individual exceeds DV_R, then the individual is placed in the *Under-Repair* group.

In the first experiment, only one individual is affected by a failure in the physical resource, which causes a 1-out-of-64 fault in the individual. Before the fault occurs, the system operates with a 100% throughput, and all individuals have a DV equal to zero. As shown in Figure 4, the fault occurs at time t = 0 and the faulty individual is repeatedly detected and identified at various observation intervals. $DV_O = DV_R$ whenever no faulty individual have been detected over a sliding window. The faulty individual is always detected, but since it has not completed E^W pairing, judgment is reserved, as shown in the plot. When a faulty individual is isolated, the DV_O will be less than DV_R and the faulty DV will be located outside of the DV_R+DV_σ, where DV_σ represents the standard deviation of the discrepancy values.

Figure 5 shows that the isolated individual's DV deviates by 1σ or more, typically 3σ. This shows error-free isolation and that faults are never incorrectly identified. Also, 100% of the faulty individuals are identified within statistically acceptable values for their discrepancies.

The average DV of individuals will increase proportionately with fault impact. This leads to increased isolation latency, as shown in Figure 6, for the second experiment, where the characteristics of isolating a single faulty individual with a *10-out-of-64* fault impact are shown. Since there are more faults, the faulty individual is expected to show a discrepancy (10/64)*600 = 93.75 times over its evaluation window. To complete these iterations, it will therefore require (93.75/5) = 18.75 observation intervals, as opposed to 1.88 previously, which leads to both increased discrepancy values for the isolated individuals and an increased time between

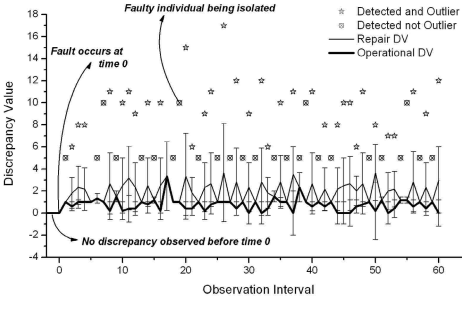


Fig. 4. Isolation of a single faulty *L* individual with a 1-out-of-64 fault impact

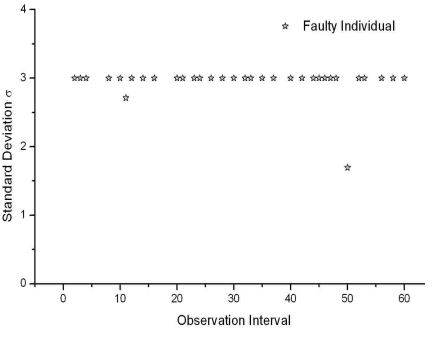


Fig. 5. Plot of Standard Deviations of DV with a 1-out-of-64 fault impact

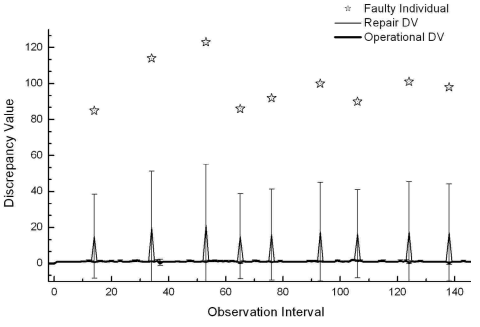


Fig. 6. Isolation of a single faulty *L* individual with a 10-out-of-64 fault impact

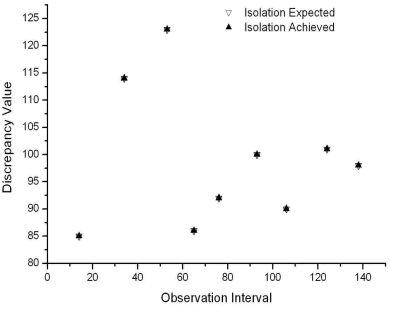


Fig. 7. Performance with a single faulty individual with 10-out-of-64 fault impact

successive isolations as compared to Figure 4. The detection latency remains unaffected. Figure 7 shows that for a single faulty L individual, with a *10-out-of-64* fault impact, isolation always succeeds when expected.

However, when more than one individual is affected by a resource fault, isolation is more time-consuming and difficult as shown in Figure 8, which depicts the isolation characteristics when 4 *L* and 4 *R* individuals are affected by *1-out-of-64* faults. Expected isolations do not occur approximately 40% of the time, as the average discrepancy value of the population is higher, making outlier isolation difficult. The faulty individuals are always detected, but the higher number of discrepancies prevents them from completing E^W iterations within an observation interval. However, a fault-free individual is never incorrectly identified as being faulty.

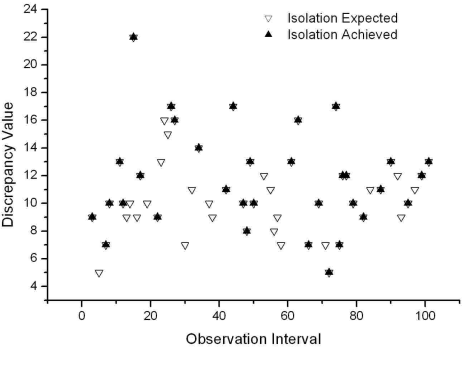


Fig. 8. Isolation of 8 faulty individuals, 4 *L* and 4 *R*, each with a *1-out-of-64* fault impact

5.2 Regeneration of Functionality

CBE-based regeneration experiments were performed on a simulated FPGA platform for the 3x3 multiplier application. Starting with an initial population of 20 viable configurations, random stuck-at faults were injected randomly into one of the 21

CLBs that were utilized to implement the multiplier. The fault reduced the number of correct outputs from 64-out-of-64 to 54-out-of-64. Regeneration was performed using a fitness-state adjustment process that utilized the results of the isolation process described in previous sections. A re-introduction rate of 10% was selected for selecting individuals under repair for performance evaluation. Higher re-introduction rates would lower the throughput whereas if the re-introduction rate is too low, the repair process will be unduly slowed down due to the decreased opportunities to evaluate the performance of the individuals under repair. A low crossover rate of 0.05 was used to ensure that the diversity in the population is preserved. The initial seeding population consists entirely of diverse hand-designed individuals. The mutation rate of 0.8 is required to ensure that the algorithm can explore alternatives by changing the logical functionality of LUTs and the interconnections between them.

While the simulated FPGA remained partially online, regeneration improved correctness to *64-out-of-64* possible outputs. Including iterations that produced functional outputs, the process concluded after a total of 218076 iterations. Complete repair was achieved after only 135 repair iterations when starting with a highly diverse initial population. The fault-affected individual was loaded on the FPGA for a total of 31636 iterations. During the regeneration period, data throughput was 85.54%. Hence, only 14.46% of the total computations needed to be redundant in order to preclude propagation of any discrepant outputs, even when candidate repairs were being re-introduced to refurbish the impacted FPGA configuration without additional test vectors. The throughput will be significantly higher when the system starts from a fault-free situation, since a large number of the initial iterations before the occurrence of the fault will contribute to improving the throughput. Fault isolation using consensus-based evaluation improved the performance of the repair process eliminating the use of an absolute fitness function. The diversity of the initial population provides for increased fault tolerance and also the raw material for realizing the repair.

6 Conclusion

Online EH regeneration essentially defines a problem that is different from offline EH design. CBE leverages the fact that a failed system's *Repair Complexity* can often be much more computationally tractable than either its original *Design Complexity* or its *Re-Design Complexity*, both of which operate in the absence of a diverse population of previously completely correct alternatives. In particular, "repair" implies working design(s) being available before the occurrence of a resource failure. A population of working designs can thus facilitate repair by providing diverse alternates. Conventional fitness evaluation associates a rigid *individual-centric* fitness measure defined at design-time. CBE uses instead, a self-adapting *population-centric* assessment method at run-time. Population-centric assessment methods such as CBE can provide an additional degree of adaptability and autonomy. Finally, an additional benefit of CBE is that fitness evaluation becomes independent of the application running on the FPGA enabling *model-free* assessment during evolutionary repair.

Acknowledgments

This research was supported in part by NASA Intelligent Systems NRA Contract NNA04CL07A.

References

1. D. Keymeulen, A. Stoica, and R. Zebulum, "Fault-Tolerant Evolvable Hardware using Field Programmable Transistor Arrays," *IEEE Transactions on Reliability*, Vol.49, No. 3, Sept. 2000
2. S. Vigander, "Evolutionary Fault Repair of Electronics in Space Applications", *Dissertation*, Norwegian University Sci. Tech., Trondheim, Norway, February 28, 2001
3. J. D. Lohn, G. Larchev, and R. F. DeMara, "A Genetic Representation for Evolutionary Fault Recovery in Virtex FPGAs," *Proceedings of the 5th International Conference on Evolvable Systems (ICES)*, Trondheim, Norway, March 17-20, 2003
4. J. D. Lohn, G. Larchev, and R. F. DeMara, "Evolutionary Fault Recovery in a Virtex FPGA Using a Representation That Incorporates Routing," *Proceedings of 17th International Parallel and Distributed Processing Symposium,* Nice, France, April 22-26, 2003
5. M. Garvie and A. Thompson, "Scrubbing away transients and Jiggling around the permanent: Long survival of FPGA systems through evolutionary self-repair," *Proceedings of the 10th IEEE Intl. On-Line Testing Symposium*, pp. 155-160, 2004
6. A. P. Shanthi and Ranjani Parthasarathi, "Exploring FPGA Structures for Evolving Fault Tolerant Hardware", *Proceedings of the 5th NASA / DoD Workshop on Evolvable Hardware*, pp. 184-191, 2003
7. Yao. X, Liu. Y and Darwen. P, "How to make best use of evolutionary learing," In R. Stocker, H. Jelinek, and B. Durnota, editors, Complex Systems: From Local Interactions to Global Phenomena, Amsterdam, pp. 229-242, 1996
8. Yao. X and Liu. Y,"Making use of population information in evolutionary artificial neural networks", IEEE Trans. On Systems, Man and Cybernetics, Part B: Cybernetics, 28(3), pp. 417-425, 1998
9. Yao. X and Liu. Y, "Getting most of evolutionary approaches," In A. Stoica, J. Lohn, R. Kata, D. Keymeulen & R. Zebulum(eds), Proceedings of 2002 NASA/DOD Conference on Evolvable Hardware, IEEE Computer Society, Alexandria, Virginia, pp. 8-14, 15-18 July 2002
10. Layezll. P and Thompson. A., "Understanding the inherent Qualities of Evolved Circuits: Evolutionary History as a Predictor of Fault Tolerance," *Proceedings of Third Conf on Evolvable Systems: From Biology to Hardware (ICES00), Vol. 1801 of LNCS*, pp. 133-142, Springer, April, 2000
11. Subhasish Mitra and Edward J. McCluskey, "Which Concurrent Error Detection Scheme to Choose?" *Proceedings of 2000 International Test Conference*, Atlantic City, NJ, pp. 985-994, Oct. 3-5, 2000
12. Julian F. Miller, Peter Thomson, "Cartesian Genetic Programming", *Proceedings of the Third European Conference on Genetic Programming (EuroGP2000).LNCS*, Vol. 1802, (2000), pp.121-132, Springer-Verlag, 2000

Hardware Fault-Tolerance Within the POEtic System

Will Barker and Andy M Tyrrell

Department of Electronics, Heslington, University of York, York YO10 5DD, UK
amt@ohm.york.ac.uk
http://www.elec.york.ac.uk/intsys/inspired/inspired.html

Abstract. The ideas presented in this paper are results of a new research project, "Reconfigurable POEtic Tissue". The goal of the project was the development of a hardware platform capable of implementing bio-inspired systems, in digital hardware. In particular, the final hardware device, while similar to other FPGAs, was designed with a number of novel features which facilitate fault-tolerance. These include dynamic reconfiguration and on-chip re-programming. This paper considers these features in the context of fault-tolerant system design and shows how an ensemble of different, but often complementary, techniques might be produced using these novel device features. Such characteristics are crucial for many control systems, particularly with safety implications.

1 Introduction

Ensuring the reliability of computing and electronic systems has always been a challenge. As the complexity of systems increases the inclusion of reliability measures becomes progressively more complex, but are often a necessity for VLSI circuits where a single error could potentially render an entire system useless.

Reducing the failure probability and increasing reliability has been a goal of electronic systems designers ever since the first components were developed. No matter how much care is taken designing and building an electronic system, sooner or later an individual component will fail. For systems operating in remote environments such as space, control and deep-sea applications, the effect of a single failure could results in a multi-million pound installation being rendered useless. With safety critical systems such as aircraft, mobile robotics the effects are even more severe. Reliability techniques need to be implemented in these applications and many more. The development of fault tolerant techniques was driven by the need for ultra-high availability, reduced maintenance costs, and long life applications to ensure systems can continue to function in spite of faults occurring. The implementation of a fault tolerant mechanism requires four stages: Detection of the error, *Confinement of the error*, to prevent propagation through the system, *Error recovery,* to remove the error from the system, *Fault treatment and continued system service,* to repair and return the system to normal operation.

We deal with the detection of errors and error recovery in this paper. But first we should comment on the new architectural aspects of the POEtic device that make it so amenable to fault-tolerant designs.

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 25 – 36, 2005.

The aim of this paper is to present some of the ideas developed in the framework of a new research project, called "Reconfigurable POEtic Tissue" (or "POEtic" for short) [1], recently completed under the aegis of the European Community. After a short introduction to the POE project for the design of bio-inspired hardware, the paper will present an outline of the POEtic device built during the project and of the main features designed into the device that aids fault-tolerant system design, experimental results are given to illustrate the efficacy of the device and these special features.

2 The POEtic Project

The goal of the POEtic project was:

> "*... development of a flexible computational substrate inspired by the evolutionary, developmental and learning phases in biological systems*."

The POEtic tissue is a multi-cellular, self-contained, flexible, and physical substrate designed to interact with the environment, to develop and dynamically adapt its functionality through a process of evolution, development, and learning to a dynamic and partially unpredictable environment, and to self-repair parts damaged by aging or environmental factors in order to remain viable and perform similar functionalities.

Following the three models of bio-inspiration, the POEtic tissue was designed logically as a three-layer structure (Figure 1 gives an abstract view of this relating to hardware):

- The phylogenetic model acts on the genetic material of a cell. Each cell can contain the entire genome of the tissue. Typically, in the architecture defined above, it could be used to find and select the genes of the cells for the *genotype layer*.
- The ontogenetic model concerns the development of the individual. It should act mostly on the *mapping or configuration layer* of the cell, implementing cellular differentiation and growth. In addition, ontogenesis will have an impact on the overall architecture of the cells where self-repair (healing) is concerned.
- The epigenetic model modifies the behavior of the organism during its operation, and is therefore best applied to the *phenotype layer*.

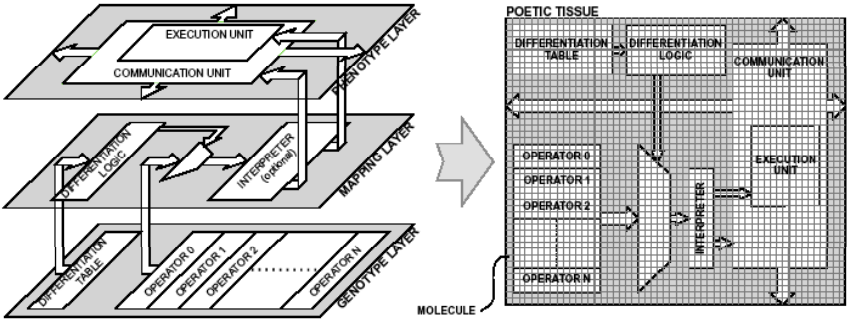


Fig. 1. The three organizational layers of the POEtic project

Defining separate layers for each model has a number of advantages, as it allows the user to decide whether to implement any or all models for a given problem, and lets the structure of each layer to be adapted to the model. This adaptability is achieved by implementing the cells on a *molecular substrate*, in practice a surface of programmable logic.

The final hardware design (VLSI device) has a number of specific novel features built into its fabric to assist with bio-inspired designs. It is shown here that these features can also be used effectively for the design of fault-tolerant systems.

2.1 Operational Mode

A molecule has eight different operational modes, to speed up some operations, and to use the routing plane. Here we briefly describe the different modes, and they will be completely described in [2].

In **4-LUT** mode, the 16-bit LUT supplies an output, depending on its four inputs.

In **3-LUT** mode, the LUT is split into two 8-bit LUTs, both supplying a result depending on three inputs. The first result can go through the flip-flop, and is the first output. The second one can be used as a second output, and is directly sent to the south neighbor (can serve as a carry in parallel operations).

In **Comm** mode, the LUT is split into one 8-bit LUT, and one 8-bit shift register. This mode could be used to compare a serial input data with a data stored in the 8-bit shift register.

In **Shift Memory** mode, the 16 bits are used as a shift register, in order to store data, for example a genome. One input controls the shift, and another one is the input of the shift memory.

In **Input** mode, the molecule is a cellular input, connected to the inter-cellular routing plane. One input is used to enable the communication. When inactive, the molecule can accept a new connection, but won't initiate a connection. When active, a routing process will be launched at least until this input connects to its source. A second input selects the routing mode of the entire POEtic tissue.

In **Output** mode, the molecule is a cellular output, connected to the intercellular routing plane. One input is used to enable the communication. When inactive, the molecule can accept a new connection, but won't initiate a connection. When active, a routing process will be launched at least until this output connects to one target. Another input supplies the value sent to the routing plane, as so to another cell.

In **Trigger** mode, the 16-bit shift register should contain "000...01" for a 16- bit address system. It is used by the routing plane to synchronize the address decoding during the routing process. One input is a circuit enable, that can disable every DFFs in the tissue, and another one can reset the routing, and so start a new routing.

In **Configure** mode, the molecule can partially configure its neighborhood. One input is the configuration control signal, and another one is the configuration shifting to the neighbors.

The mode of a molecule is stored in 3 bits of the configuration.

2.2 Partial Reconfiguration

The configuration system of the molecules can be seen as a shift register of 76 bits split into 5 blocks: the LUT, the selection of the LUT's input, the switch box, the mode of operation, and an extra block for all other configuration bits. Each block contains, as shown in Figure 2, together with its configuration, one bit indicating, in case of a reconfiguration coming from a neighbour, if the block has to be bypassed. This bit can only be loaded from the microprocessor, and remains stable during the entire lifetime of the organism.

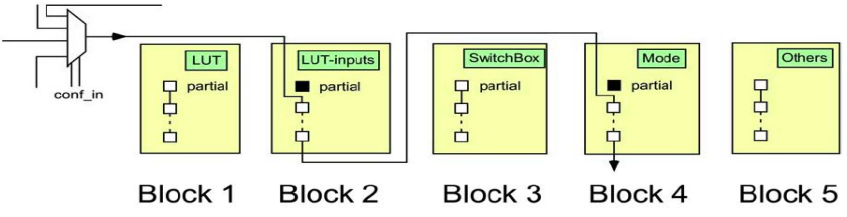


Fig. 2. Organisation of the configuration bits for partial reconfiguration

The special configure mode allows a molecule to partially reconfigure its neighbourhood. It sends bits coming from another molecule to the configuration of one of its neighbours. By chaining the configurations of neighbouring molecules, it is possible to modify multiple molecules at the same time.

3 Reconfiguration and Fault-Tolerance

The POEtic project provides a unique platform for investigating mechanisms at work in biological systems which exhibit fault-tolerant behaviours and it is the intent of the following work to demonstrate this through the development of a cellular ontogenetic fault-tolerant mechanism on the POEtic tissue based upon growth.

Self-repair, or healing, is a critical mechanism within an organism's response to damage involving the growth of new resources, in the form of cells, and their integration into the organism replacing damaged ones. An electronic system cannot grow new silicon resources in response to device faults in the same way and so growth in silicon is generally emulated by having redundant resources into which the system can grow. The POEtic architecture provides novel features which are particularly useful for implementing models of growth in digital hardware including the underlying molecular architecture, dynamic routing and self-configuration of the tissue.

3.1 Growth

The work reported here is inspired by two important features of growth: those of cell division and cellular differentiation. Cell division is a process of self-replication through which cells produce copies of themselves. Cellular differentiation is the proc-

ess through which cells organise themselves by taking on specific functional types depending upon their positions within an organism.

Prompted by these distinct modes of growth a novel cell design has been implemented on the POEtic tissue in the context of a test application. Implementation of an embryonic array emulating the processes of cellular differentiation is described in this paper.

3.2 Test Application

In order to investigate issues regarding the implementation of cellular fault-tolerant mechanisms on the POEtic tissue a test application has been defined based upon the audio application presented in [3]. The test application consists of a cell constructed from nine one-dimensional waveguide mesh elements (for those not familiar with waveguide meshes, each mesh element can be considered as a particular type of processing element, the Ps in Figure 3 represent values passing between neighbouring mesh (processing) elements) – this application requires real-time (audio) processing. Cells can be chained together to form a one-dimensional waveguide with length an arbitrary multiple of nine, Figures 3 & 4. While this is a specific application executed within the POEtic project, the work reported here is generic and appropriate for any application on one or more devices.

3.3 Fault Detection

Biological mechanisms for fault detection in themselves provide a rich field of research to which the POEtic platform is applicable [4]. However as the aim of the cell designs is to investigate growth mechanisms a standard hardware redundancy technique has been chosen to provide fault detection in the designs. It is based upon the assumptions that faults will occur discretely in time and that a fault is only of significance if it causes the cell function at its outputs to deviate from correct behaviour. Based upon these assumptions we can duplicate and compare two systems, a fault will cause the values at the outputs of cell function copies to differ. This discrepancy is detected by a XOR logic functions comparing the cell function outputs and combined into a fault flag by an OR logic function. The advantages of this method for fault detection are that it is simple, acts at the resolution of a single clock cycle, operates on-line and is applicable to any cell function.

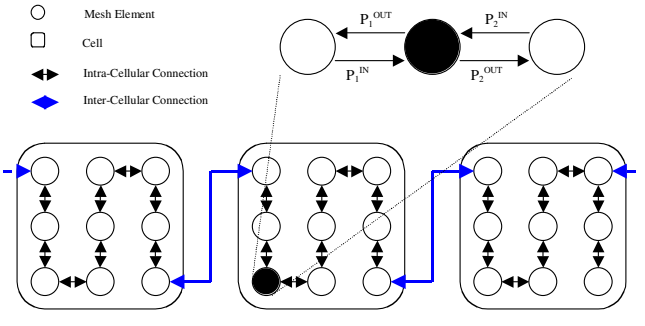


Fig. 3. Nine mesh-element one-dimensional waveguide cell with left and right input streams

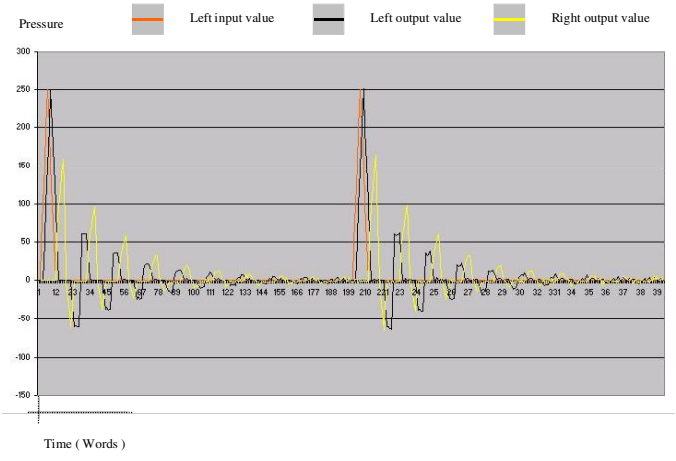


Fig. 4. Cell output in response to pulses applied to one input

4 Embryonic Implementation on the POEtic Tissue

An embryonic array consists of an array of cells implemented in reconfigurable logic each of which contains a set of configuration strings describing every cell function within the system which the cells are to form. This set of configuration strings is analogous to the biological genome contained within every living cell. Each configuration string is analogous to a gene and can be directly translated into the cell function it describes in the hardware of the cell. Development of the system is achieved through differentiation during which each cell identifies its configuration string with respect to its location within the array and uses it to configure its function [5], detailed in Figure 5.

4.1 Cell Function Areas

Cell function areas are the areas of the cell where the cell application functionality is performed. Duplication of the cell function enables fault detection by the method described above. The areas are initially blank and require configuration from a stored gene. They consist of molecules which have the partial configuration inputs from their neighbours chained together as illustrated in Figure 6 and all configuration registers enabled for configuration. This allows an arbitrary cell function to be configured within them. The stored gene therefore consists of the contents of the configuration registers for each molecule in the function listed in the order in which they appear in the chain from the head to the tail.

4.2 Genome Storage

The stored genome consists of individual gene blocks each of which can be selected by the differentiation system to be the source for configuration of the function areas within the cell. The genes consist of shift memory molecules which store the configu-

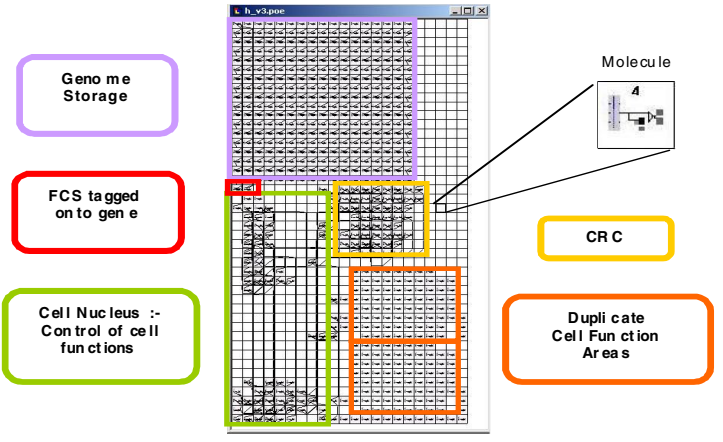


Fig. 5. Embryonic cell design on the POEtic tissue (Cell has a single gene in its genome for illustrative purposes)

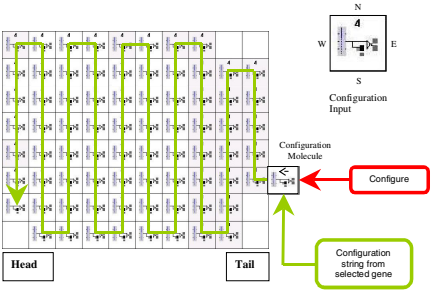


Fig. 6. Cell function area configuration chain

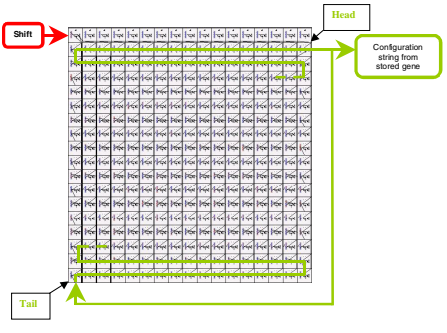


Fig. 7. Gene block

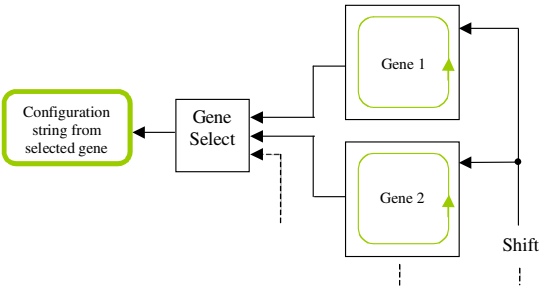


Fig. 8. Stored genome consisting of selectable gene blocks

ration string in their look up tables (LUTs). The inputs and outputs of the memory molecules are chained together in the same way as the molecules in the function configuration areas. During configuration of the function areas every gene in the stored genome shifts its contents out from its head with the string from the gene selected by the differentiation process being channelled into the cell function areas.

The head of each gene block is looped back into the tail by connecting the memory molecule output of the head molecule to the input of the tail molecule so that the contents of the gene block are retained, illustrated in Figures 7 & 8.

4.3 Fault Detection in the Stored Genome: Cyclic Redundancy Check

Approximately four times as many molecules are required to store each gene than are used in the function block that it describes and an embryonic cell will require as many genes as there are different cells in the system. As both function copies are configured from the same stored gene, a fault in the gene will go undetected by the redundancy fault-detection system as both copies will be producing the same erroneous outputs.

A second fault-detection system has therefore been implemented in the embryonic cell design in the form of a cyclic redundancy code (CRC) which can detect faults in the gene being used to configure the cell function by means of a frame check sequence (FCS) which is tagged onto the end of every gene [6].

On configuration of the cell function areas the configuration string for the selected gene including the FCS is passed through the CRC register as the cell function areas are configured. If the stored gene is incorrupt then the output of all of the CRC register elements will be zero at the end of this process. Otherwise at least one of the outputs of the register elements will be high indicating a fault in the gene.

4.4 Cell Nucleus

The cell nucleus is responsible for controlling the five main processes of the embryonic cell. These are cellular differentiation, cell function configuration, fault detection, apoptosis and routing.

Cellular Differentiation: Each of the embryonic cells in the array has a differentiation input and output molecule. These inputs and outputs are linked in a chain across the tissue. On receiving a zero at its input each cell asynchronously sets its output to zero. The first cell in the chain has its input connected to a source external to the tissue into which the differentiation signal is driven. This signal consists of a series of ones equal in length to the number of cells in the organism being developed. The first cell in the chain therefore receives a series of ones equal in number to this value before receiving a zero. At this point the output of cell one is asynchronously reset to zero causing a chain reaction through which all cell differentiation outputs down the chain asynchronously reset to zero. This terminates the differentiation process in the cells, each of which will have received one less one at its differentiation input than the previous cell in the chain. The cells then select their allocated genes depending upon the number of ones received at their differentiation inputs. Cells which receive no ones at their inputs blank their function areas and are left as unused spare cells. This process can be instigated at any point by simply driving the differentiation signal into the differentiation chain.

Cell Function Configuration: Completion of the differentiation process triggers a molecule configuration counter. The counter enables the shift input to each of the gene blocks in the stored genome and enables a configuration molecule which feeds the output of the selected gene into the configuration input of the tail of the function area configuration chain.

When the counter indicates that the number of molecules in a cell function area have been configured, the enable to the configuration molecule is disabled and the counter resets. At this point the function areas of the cell have been programmed with the selected gene and are ready for integration into the system. Before this can occur however the FCS must be shifted through the CRC register to check that the gene is correct. This is controlled by a second counter which is triggered by the overflow of the first and shifts the gene blocks by a further 32 bits driving the FCS out of the gene block through the CRC register.

Fault Detection: In the cell nucleus the values at the outputs of the two cell function copies are compared and a fault flagged in response to a discrepancy. The integrity of the configuration of these function copies is tested by the CRC register on configuration and if corrupt a fault is flagged. The cell nucleus combines these two fault flags into a single signal which triggers cell apoptosis and differentiation of the system.

Differentiation in response to a fault is triggered by the faulty cell setting the '*mol_enable_out*' signal on its trigger molecule low. This is detected by the external system controlling the differentiation signal input to the tissue which drives the signal into the differentiation chain in response.

Apoptosis: Apoptosis in a faulty embryonic cell is achieved by selecting the blank gene, simply a source of zeros, and bypassing the delay which the cell would otherwise introduce into the differentiation chain. This shifts the differentiation values received downstream of the faulty cell one cell down the chain and causes the cell to blank its function areas removing any molecules such as input and output molecules which may interfere with the operation of the system.

Routing: Having completed the processes of cellular differentiation and configuration the final step in producing the functioning embryonic system is to route together any input and output molecules which have been configured in the function areas of the cells.

The cell which receives the differentiation value equal to the number of cells in the system, i.e. the first healthy cell in the differentiation chain, is assigned the task of triggering the routing process. Every cell contains a trigger molecule capable of this. On completing the configuration of their function areas every cell sends a pulse on the '*start_routing_enable*' signal to its inputs entering them into the routing process. This pulse is also sent to the '*reset_routing*' input on the cell's trigger molecule via a gate. The output of this gate is enabled if it is the first working cell in the chain thereby triggering the routing process. This system is required as firing multiple trigger molecules on the tissue will result in multiple routing processes being instigated wasting clock-cycles [2].

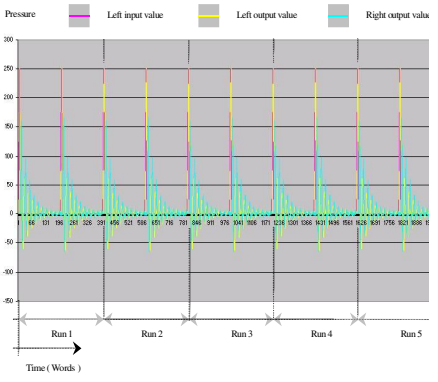


Fig. 9. Fault-free cell I/O

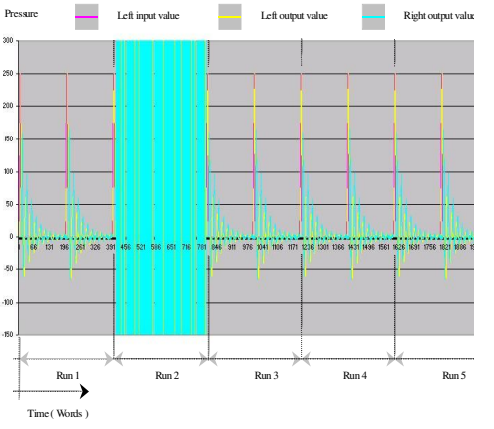


Fig. 10. I/O with faults. Fault-tolerance OFF.

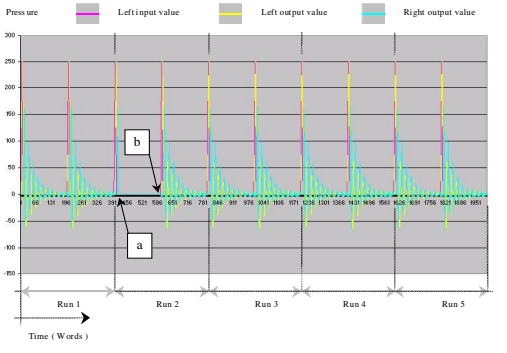


Fig. 11. I/O with faults. Fault-tolerance ON.

4.5 Simulations and Results

The cell design described above has been simulated in the presence of randomly generated faults using the POEtic design tool POEticmol [7]. As POEticmol simulates the behaviour of the POEtic device using the VHDL description from which the device is fabricated accurate simulations of the effects of faults on system behaviour can be made.

A number of signal elements are randomly chosen and are forced into a fault condition during the simulation. Each signal element is randomly allocated a clock-cycle number from the pre-specified duration of the simulation upon which to become faulty. The fault model used is the ‘stuck-at’ fault, or single hard error. The number of faults forced in the simulation is set at a value high enough to guarantee a satisfactory yield of terminal cell faults rather than at a value which is a realistic representation of fault rates for the real device with respect to the test application.

Some example results showing the behaviour of the embryonic cell design in response to randomly generated faults can be seen in Figures 9 to 11. The figures each show the input and output data for two cells, a working cell and a spare cell (both working and spare cells are exposed to faults during the simulations), simulated over five runs of a fixed number of output words. Each run has a different randomly generated set of faults which are to be forced into the tissue. The design is reset after the end of each run and any faults forced into the tissue are removed. Each cell design is simulated under three conditions. In the first simulation no faults are forced into the tissue. This generates the target output which the fault-tolerant system is aiming to achieve. In the second simulation faults are forced into the tissue but the fault-detection and growth mechanisms are disabled. In the third simulation the same faults are forced into the tissue with the fault-detection and growth mechanisms enabled.

In Figure 10 it can be seen that unprotected embryonic cell sustains a terminal fault in run 2. In run 2 of Figure 11 it can be seen that with the fault-tolerant systems enabled the system has detected the induced fault and instigated apoptosis of the faulty cell and re-growth of the system. Data loss in the embryonic system is illustrated by the zero output between points a) and b) produced by the newly grown system. At point a) the system is repaired and fully functional but its response to the input pulse previous to the fault being detected has been wiped by re-growth. By point b) the correct system response to this input has become negligible and a new input pulse stimulates the repaired embryonic system producing incorrupt output data.

5 Conclusion

An embryonic cellular fault-tolerant mechanism has been successfully implemented in simulation on the POEtic tissue. The transparency of the process of mapping an embryonic design onto the POEtic architecture has also been demonstrated. Unlike embryonic implementations on generic FPGA architectures which require complex stages of synthesis and careful tailoring of the embryonic architecture for the target device, compact embryonic designs can be built directly on the POEtic tissue at the molecular level. Results of preliminary simulations in the presence of randomly introduced faults show that the cell design is capable of successfully detecting, repairing and recovering from terminal faults in cell function.

Acknowledgements

The authors would like to acknowledge the contribution of all the other members of the POEtic project funded by the Future and Emerging Technologies programme (IST-FET) of the European Community, under grant IST-2000-28027 (POETIC). The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication. The Swiss participants to this project are supported under grant 00.0529-1 by the Swiss government.

References

1. Tyrrell, A.M., Sanchez, E., Floreano, D., Tempesti, G., Mange, D., Moreno, J.M., Rosenberg, J. and Villa, A.E.P. “POEtic Tissue: An Integrated Architecture for Bio-Inspired Hardware”, Proceedings of 5th International Conference on Evolvable Systems, Trondheim, pp 129-140, March 2003.
2. Thoma, Y., Tempesti, G., Sanchez, E. and Moreno J-M. “POEtic: An Electronic Tissue for Bio-Inspired Cellular Applications”, BioSystems, pp. 191-200, 74: 1-3, 2004
3. Cooper, C.H.V., Howard, D.M. and Tyrrell, A.M. ‘Using GAs to Create a Waveguide Model of the Oral Vocal Tract’, 6th European Workshop on Evolutionary Computation in Image Analysis and Signal Processing, Coimbra, Portugal, pp 2880-288 April 2004.
4. Canham, R. O., Tyrrell, A. M., “A Multilayered Immune System for Hardware Fault Tolerance within an Embryonic Array”, 1st International Conference on Artificial Immune Systems, Canterbury, September 2002.
5. D. Mange, M. Sipper, A. Stauffer, G. Tempesti. "Towards Robust Integrated Circuits: The Embryonics Approach", Proceedings of the IEEE, vol. 88, no. 4, April 2000, pp. 516-541.
6. Costello D.J. Jnr, Lin S. “Error Control Coding”, 2nd Ed. Prentice Hall 2004, pp 136-188
7. Thoma Y., Sanchez E., Hetherington C., Roggen D., Moreno J-M. “Prototyping with a bio-inspired reconfigurable chip”, Proc. 15th International Workshop on Rapid System Prototyping (RSP 2004), Geneva, Switzerland, June 2004.

Evolvable Hardware System at Extreme Low Temperatures

Ricardo S. Zebulum, Adrian Stoica, Didier Keymeulen, Lukas Sekanina[*], Rajeshuni Ramesham, and Xin Guo[**]

Jet Propulsion Laboratory/Caltech, NASA, 4800 Oak Grove Drive, Pasadena, CA 91109, USA
ricardo.zebulum@jpl.nasa.gov

Abstract. This paper describes circuit evolutionary experiments at extreme low temperatures, including the test of all system components at this extreme environment (EE). In addition to hardening-by-process and hardening-by-design, "hardening-by-reconfiguration", when applicable, could be used to mitigate drifts, degradation, or damage on electronic devices (chips) in EE, by using re-configurable devices and an adaptive self-reconfiguration of their circuit topology. Conventional circuit design exploits device characteristics within a certain temperature/radiation range; when that is exceeded, the circuit function degrades. On a reconfigurable device, although component parameters change in EE, a new circuit design, suitable for new parameter values, may be mapped into the reconfigurable structure to recover the initial circuit function. This paper demonstrates this technique for circuit evolution and recovery at liquid nitrogen temperatures (-196.6°C). In addition, preliminary tests are performed to assess the survivability of the evolutionary processor at extreme low temperatures.

1 Introduction

Future NASA missions to Moon, Mars and Beyond will face Extreme Environments (EE), including environments with large temperature swings, such as between -180°C and 120°C at the initial landing sites on the Moon, low temperatures of -220 °C to -233°C during the polar/crater Moon missions, and -180°C for Titan *in-situ* mission. High temperatures of 460°C and harsh sulfuric acid environment will be encountered for Venus Surface Exploration and Sample Return mission. High radiation levels will be faced for Jupiter's Icy Moons Orbiter (JIMO) missions: 5MRad Total Ionizing Dose (TID) for Europa Surface and Subsurface mission. These extreme environments of extreme low temperatures and high radiation induce drifts, degradation, or damage into electronic devices and reliability issues of package designs and associated materials.

[*] Brno University of Technology, Czech Republic (Visiting scientist at JPL/Caltech during fall of 2004).
[**] Chromatech, Alameda CA 94501, USA.

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 37 – 45, 2005.

The current approach for space electronics designs is to use commercial-of-the-shelf or military range electronics protected through passive (insulation) or active thermal control, and heavy metal (high weight) shielding for radiation reduction. This increases weight and volume, and is compounded by power loss, and leads to additional cost for the mission. More importantly, as missions will target operations with smaller instruments/rovers and operations in areas without solar exposure, these approaches sometimes become infeasible and it will be more expensive. In many cases the electronics must be co-located with the sensor or actuator in the extreme environment, without the option of being insulated or shielded properly, for example panoramic camera and its electronics in Mars Exploration Rover project. Therefore, developing EE-robust electronics would have several advantages including lower costs, less power, no thermal control, and offering in some cases, the only reasonable solution.

Conventional approaches to Extreme Environment Electronics include *hardening-by-process* (HBP), i.e. fabricating devices using materials and device designs with higher tolerance to EE, (e.g using special materials like Silicon Carbide for high temperatures, or Silicon-on Insulator for radiation, ceramic materials for packaging). Another promising approach is *hardening-by-design* (HBD), i.e. use of special design/compensation schemes. For example, circuit techniques, such as auto-zero correction, are used to alleviate the problem of the (temperature dependent) offset voltages in Operational Transconductance Amplifiers (OTA) operated at low temperatures [1]. Both these hardening approaches are limited, in particular for analog electronics, by the fact that current designs are fixed and, as components are affected by EE, these drifts alter functionality.

A recent approach pioneered by JPL is to mitigate drifts, degradation, or damage on electronic devices in EE by using re-configurable devices and an adaptive self-reconfiguration of circuit topology. This new approach, referred here as hardening-by-reconfiguration (HBR) mitigates drifts, degradation, or damage on electronic devices in EE by using reconfigurable devices and an adaptive self-reconfiguration of circuit topology. In HBR, although device parameters change in EE, while devices still operate (albeit on a different point of their characteristic) a new circuit design, suitable for new parameter values, is mapped into the reconfigurable system to recover the initial circuit functionality. Partly degraded resources are still used, while completely damaged resources are bypassed. The new designs, suitable for various environmental conditions, can be determined prior to operation or determined in-situ by reconfiguration algorithms running on a built-in digital controller.

The scope of this paper is on HBR for extreme low-temperatures, since other studies have been performed for high temperatures and radiation environments [2]. The application here described encompasses the separate testing of the whole Evolvable Hardware system (Evolutionary Processor + Re-configurable chip) at low temperatures, following the assumption that the entire system will be exposed to the space EE. In the experiments, we demonstrate the evolution and recovery of circuits at liquid nitrogen temperatures (-196.5°C) and verify the operational limitation of the evolutionary processor at low temperatures. This adds to our previous experiments where only the re-configurable chip was exposed to EE [2].

The Stand-Alone Board Level Evolvable (SABLE) system [3] designed by JPL is used in the experiments described in this paper. This system consists of a Digital Signal processor (DSP) working as an evolutionary processor and a reconfigurable mixed signal chip, the Field Programmable Transistor Array (FPTA). Section 2 of this paper overviews the SABLE system. Section 3 describes the experiments and section 4 concludes the research work performed.

2 Overview of SABLES

SABLES integrates an FPTA and a DSP implementing the Evolutionary Processor (EP) as shown in Figure 1. The system is stand-alone and is connected to the PC only for the purpose of receiving specifications and communicating back the results of evolution for analysis [3].

Fig. 1. Block diagram of a simple stand-alone evolvable system

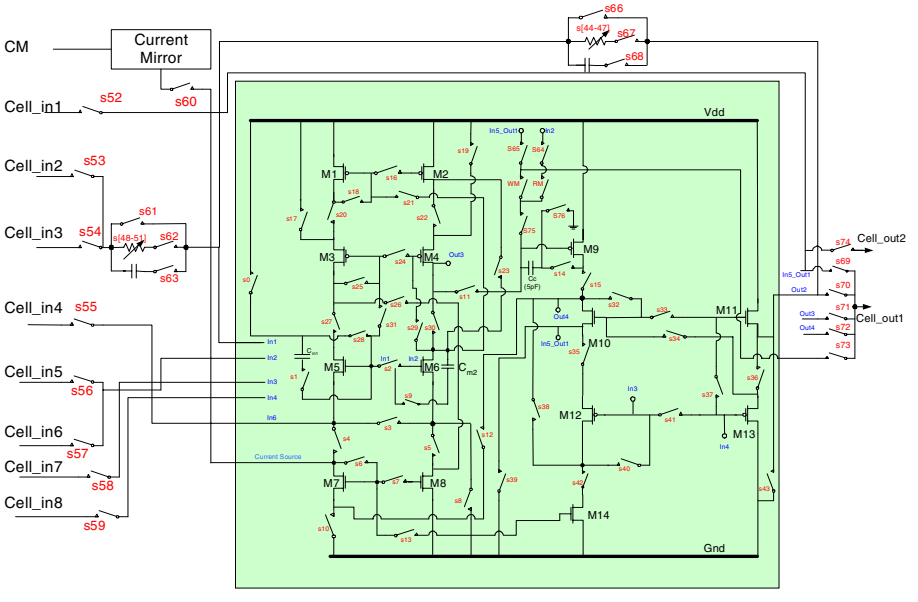


Fig. 2. Schematic of the FPTA-2 Cell

The FPTA has transistor level reconfigurability, consisting of an 8x8 array of reconfigurable cells. Each cell has a transistor array as well as a set of other programmable resources, including programmable resistors and static capacitors. Figure 2 provides a detailed view of the reconfigurable transistor array cell. The reconfigurable circuitry consists of 14 transistors connected through 44 switches and is able to implement different building blocks for analog processing, such as two- and three-stage OpAmps and Gaussian computational circuits. Details of the FPTA-2 can be found elsewhere [2,3].

The evolutionary algorithm is implemented in a DSP that directly controls the FPTA-2, together forming a board-level evolvable system with fast internal communication ensured by a 32-bit bus operating at 7.5MHz. Details of the evolutionary platform (EP) were presented in [4]. Over four orders of magnitude speed-up of evolution was obtained on the FPTA-2 chip compared to SPICE simulations on a Pentium processor (this performance figure was obtained for a circuit with approximately 100 transistors; the speed-up advantage increases with the size of the circuit).

3 Low Temperature Experiments

This paper particularly focuses on analog/digital electronics at low-temperatures [5]. The experiments cover separate tests of the whole Evolvable Hardware system: the Evolutionary Processor (the DSP in the SABLE system) and the FPTA tested at low temperatures. Table 1 summarizes the experiments setup.

Table 1. Summary of Experiments

Function	Device Tested	Temperature	Individuals/Generations
Maximization of chromosome value	DSP	Between -110°C and -120°C	100/464
Half-Wave Rectifier	FPTA	-196.5°C	100/300
NOR Gate	FPTA	-196.5°C	100/300
Controllable Oscillator	FPTA	-196.5°C	100/300

3.1 DSP Tests at Low-Temperatures

Previous experiments focused exclusively on the tests of the FPTA chips at extreme environments. However, no tests have been reported so far on the behavior of the Evolutionary Processors (EP) at extreme environments. This particular experiment focuses on low-temperature characterization of the DSP working as the EP.

A 320C6701 DSP was tested in a board fabricated by Innovative Integration (SBC62). The board communicates with a PC through a JTAG connection. During the test only the DSP board was placed on the low-temperature chamber: the PC and the JTAG were outside.

The FPTA chip was not used in this arrangement. The DSP was tested by running a simple Genetic Algorithm (GA) whose target was a simple optimization problem (the maximization of the number of '1's in the chromosomes). This problem is solved in less than 1 minute, after 464 generations. The GA results are deterministic, i.e., the same for each run.

The temperature of the chamber/test article has been driven to 0°C with a scan rate of 5°C/min from room temperature. The dwell time at 0°C temperature was for 8 minutes and electrical measurements were made during this time. Later, the temperature of the chamber has been driven to –30°C, –60°C, –90°C, –120°C at a scan rate of 5°C/min and electrical measurements were made respectively during the dwell (Figure 3).

A Failure was observed during the testing at –120°C step. Electrical measurements were made at –90°C again and the DSP regained its characteristics. This procedure was repeated again: the temperature was driven to –90°C, –100°C, –110°C and –120°C to narrow the temperature range. The dwell time at each temperature was for 5 minutes and electrical measurements were made during this time. The DSP was functioning at –90°C, –100°C, and –110°C. The failure was again observed during the testing in a temperature range of –110°C to –120°C. During the failure the DSP did not communicate with the PC. The PC-DSP communication link was the only means to read out the DSP outputs in this experiment.

Other Evolutionary Processors implementations, including FPGAs and other DSP models, will be tested. The final goal of the experiments is to have an implementation operational at -180°C or below.

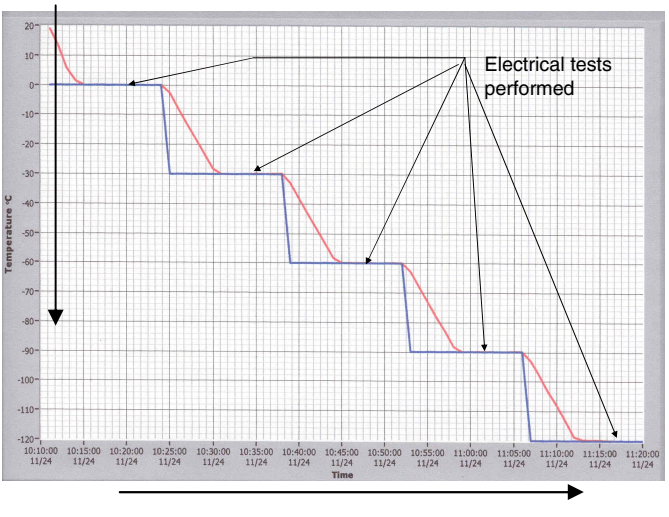


Fig. 3. Temperature Profile in the DSP Test. Time in the horizontal axis and temperature in the vertical axis.

3.2 Half-Wave Rectifier

The low temperature test bed for these experiments used liquid nitrogen, establishing a temperature of -196.5°C. In order to study the effect of low temperatures on the FPTA device only (the DSP was at room temperature), the chip was placed on a separate board that was immersed into liquid nitrogen. This setup did not allow a control for intermediate temperatures between room ambient and liquid nitrogen as described in the previous experiment. A standard ceramic package was used for the chip. A half-wave rectifier was then evolved at -196.6°C with the following setup.

The fitness function given below does a simple sum of errors between the target function and the output from the FPTA. The input was a 2 kHz excitation sine wave of 2V amplitude, while the target waveform was the rectified sine wave. The fitness function rewarded those individuals exhibiting behavior closer to target (by using a sum of differences between the response of a circuit and the target) and penalized those farther from it. The fitness function was:

$$F = \sum_{t_s=0}^{n-1} \begin{cases} R(t_s) - S(t_s) & \text{for } (t_s < n/2) \\ R(t_s) - V_{\max}/2 & \text{otherwise} \end{cases}$$

where $R(t_s)$ is the circuit output, $S(t_s)$ is the circuit stimulus, n is the number of sampled outputs, and V_{max} is 2V (the supply voltage). The output must follow the input during half-cycle, staying constant at a level of half way between the rails (1V) in the other half.

After the evaluation of 100 individuals, these were sorted according to fitness and a 9% (elite percentage) portion was set aside, while the remaining individuals underwent crossover (70% rate), either among themselves or with an individual from the elite, and then mutation (4% rate). The entire population was then reevaluated. The experiment used 2 cells and was run for 300 generations.

The oscilloscope caption is shown in Figure 4a. This was not a robust solution (and it was not even expected to be, since evolutionary algorithm was not asked, through

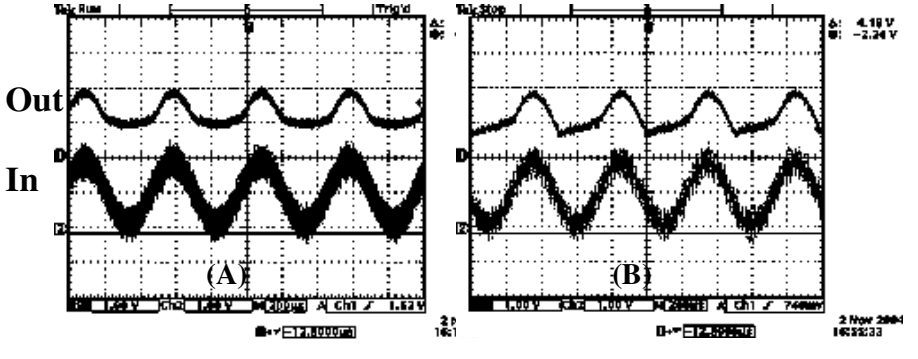


Fig. 4. Half-wave rectifier evolved at -196C (A); solution is not robust and degrades when returned to room temperature (B). An environmental noise signal is also present at the circuit input.

the fitness function, to respond to an entire temperature range) and when taken out to room temperature the response deteriorated as shown in Figure 4b.

3.3 NOR Gate

A NOR gate was evolved at -196.5°C using the same method described in section 3.2. Two FPTA cells were used and the experiment processed 100 individuals along 300 generations. Figure 5.a shows the oscilloscope picture of the evolved solution at -196.6°C. The same solution was tested at room temperature using another FPTA chip, producing an almost identical behavior (Figure 4b). This is in contrast to the rectifier behavior.

3.4 Recovery of Controllable Oscillator at Low Temperatures

Four cells of the FPTA were used to evolve a controllable oscillator. This circuit receives a digital input and it should oscillate when the input is at one digital level

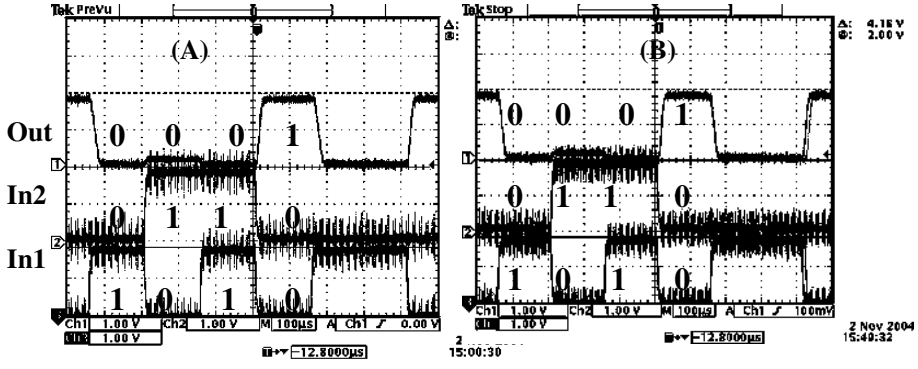


Fig. 5. NOR circuit evolved and tested at -196.5°C (A); the same circuit was tested successfully at room temperature (B). An environmental noise signal is also present at the circuit input.

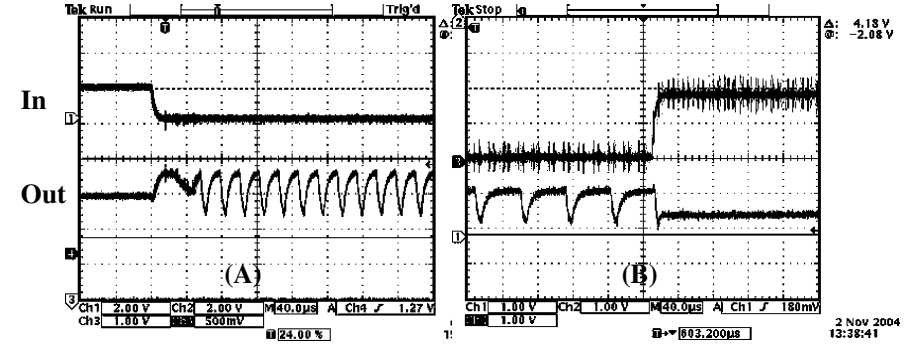


Fig. 6. Evolved controllable oscillator at room temperature and deteriorated response at –196.6°C

(either ‘0’ and ‘1’) and stay at ground for the other level. Initially, a controllable oscillator was evolved at room temperature, the circuit behavior being depicted in Figure 6a. The circuits output a 70kHz sine wave (with a small degree of harmonic components) when the input is ‘0’. When the same circuit is tested at -196.5°C, it can be observed a distortion (increase in harmonics) at the output (Figure 6b).

The controllable oscillator was evolved again at -196.5°C, the response being displayed in Figure 7. It can be observed that the output distortion largely has been removed. In addition, evolution found a circuit that oscillates for a high level input, in contrast with the room temperature solution.

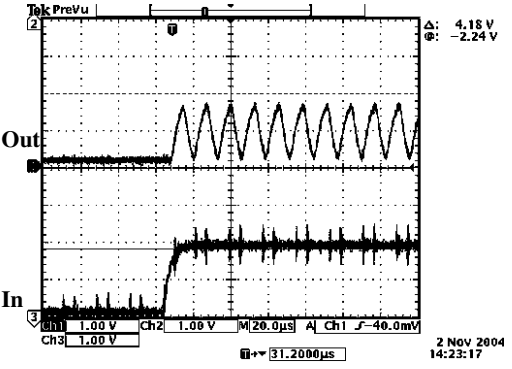


Fig. 7. Evolved controllable oscillator at low temperature

4 Conclusions and Future Work

The results summarized above prove the concept, yet have the following limitations: 1) the tests were of short duration, 2) did not implement temperature cycling, 3) did not use the combined EHW system (DSP and FPTA) at low temperature simultaneously, 4) were not demonstrated on complex analog or digital circuits performing in an application.

Particularly, the DSP Board worked down to -110°C, but failed for further lower temperatures. A short-term goal is to test other Evolutionary Processor implementations, such as FPGAs, for an extended operation at -180°C.

Longer term goals planned for this effort are: demonstrate the integrated reconfigurable array-reconfiguration logic in the same chip under temperatures cycles accurately replicating those in Moon and Mars and for longer duration and in combined radiation/temperature tests, performing a sensor processing function. More specifically, the overall objective of the new effort is to develop/demonstrate reconfigurable analog electronics performing characteristic analog functions (filtering, amplification, etc) for extended operations in extreme environment with temperatures cycling in the range of –180°C and 120°C and cumulative radiation of at least 300kRad total ionizing dose (TID). The objective is to develop and validate Self Reconfigurable Electronics for Extreme Environments (SRE-EE) technology by demonstrating a Self-Reconfigurable Analog Array (SRAA) IC in sustained (over 200

hours) operation at temperatures between -180°C and 120°C, and irradiated to 300kRad total ionizing dose (TID). The temperature range of -180°C and 120°C covers the temperature range for both Moon and Mars environments and 300kRad TID reflects accumulative dose during very long Mars missions (100kRad for near-term missions), or missions beyond the Moon and Mars, such as to Jupiter's Icy Moons. This would validate the technology for Moon and Mars temperature and Jupiter radiation environments and the even harsher radiation environments for missions beyond.

Acknowledgement

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology and was sponsored by the National Aeronautics and Space Administration.

References

1. S. C. Terry, B. J. Blalock, J. R. Jackson, S, Chen, C.S. Durisety, M.M. Mojarradi, E. Kolawa, " Development of Robust Analog and Mixed-Signal Electronics for Extreme Environments Applications", IEEE Aerospace Conference, Big Sky, MT, March 2004.
2. A. Stoica, D. Keymeulen, T. Arslan, V. Duong, R. Zebulum, I. Ferguson and X. Guo, "Circuit Self-Recovery Experiments in Extreme Environments", Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware, pp. 142-145, Seattle, USA, June 2004.
3. A Stoica, R.S. Zebulum, M.I. Ferguson, D. Keymeulen, V. Duong. "Evolving Circuits in Seconds: Experiments with a Stand-Alone Board-Level Evolvable System.", 2002 NASA/DoD Conf. on Evolvable Hardware, July 15-18, 2002, IEEE Computer Press, pp. 67-74.
4. M.I. Ferguson, A. Stoica, D. Keymeulen and R. Zebulum and V. Duong, " An Evolvable Hardware Platform based on DSP and FPTA ". In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2002), July 7-11, 2002, USA. Menlo Park, CA. Pages: 145-152: AAAI Press.
5. Hairapetian, A., Gitlin, D., Viswanathan, C.R., "Low-Temperature Mobility Measurements on CMOS Devices", IEEE Transactions on Electron Devices, V. 36, N. 8, August, 1989.

Intrinsic Evolution of Sorting Networks: A Novel Complete Hardware Implementation for FPGAs

Jan Kořenek and Lukáš Sekanina

Faculty of Information Technology, Brno University of Technology,
Božetěchova 2, 612 66 Brno, Czech Republic
{korenek, sekanina}@fit.vutbr.cz

Abstract. A specialized architecture was developed and evaluated to evolve relatively large sorting networks in an ordinary FPGA. Genetic unit and fitness function are also implemented on the same FPGA. We evolved sorting networks up to N=28. The evolution of the largest sorting networks requires 10 hours in FPGA running at 100 MHz. The experiments were performed using COMBO6 card.

1 Introduction

Sorting networks (SN) have recently been recognized as potentially suitable objects for the evolutionary design and optimization [2,4]. They are also interesting from a hardware viewpoint because of their regular and combinational nature suitable for pipeline processing. For instance, Koza et al. have used genetic programming to evolve small sorting networks directly in a field programmable gate array (FPGA) [6].

Similarly, effective hardware implementations of median circuits are crucial for high-performance signal processing. By the median circuit we mean a circuit calculating the median value from its inputs. That can be accomplished either by reading the middle value of the output sorted vector calculated by a corresponding sorting network or by designing of a specialized median circuit [10]. All the mentioned approaches share a common feature – the time of a candidate SN evaluation grows exponentially with growing number of inputs.

The objective of this paper is to evolve as large as possible sorting networks in a reasonable time. In order to perform these investigations, a novel virtual reconfigurable circuit architecture optimized for evolution of sorting networks has been proposed and implemented on the top of a conventional FPGA. The architecture is configured using the chromosomes generated by evolutionary algorithm which is implemented on the same FPGA. The chromosome encodes the functions performed by virtual programmable elements; however, the interconnection of these elements remains fixed. Since the FPGA implementation of the programmable element is inexpensive, it can operate as a wire and thus in fact the evolutionary algorithm also modifies the interconnection. As the fitness calculation is also carried out in the same FPGA, we can benefit from pipeline processing allowing reasonable time of a candidate circuit evaluation. The main

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 46–55, 2005.

feature of the proposed implementation is that everything is implemented in a cutting-edge reconfigurable hardware platform available today. For the experiments presented we utilized the COMBO6 card developed in the Liberouter project [7]. A personal computer is used only for a communication with the COMBO6 card, i.e. for reading the results. We evaluated various variants of the evolvable sorting network, including the size of the virtual reconfigurable circuit and the parameters of the evolutionary algorithm. The main objective is to find as large correct sorting network as possible in minimal time; neither area nor delay are optimised.

The paper is organized as follows. Section 2 briefly introduces sorting and median networks and evolutionary approaches to their design. In Section 3 the proposed complete hardware implementation is described. Results of synthesis for COMBO6 are reported in Section 4. Section 5 summarizes the obtained results. Section 6 deals with discussion of the obtained results and directions of future work. Conclusions are given in Section 7.

2 A Brief Survey of Relevant Research

2.1 Sorting and Median Networks

A *compare–swap* of two elements (a, b) compares and exchanges a and b so that we obtain $a \leq b$ after the operation. A sorting network is defined as a sequence of compare–swap operations that depends only on the number of elements to be sorted, not on the values of the elements [5]. The advantage of the sorting network is that the sequence of comparisons is fixed. Thus it is suitable for parallel processing and hardware implementation, especially if the number of sorted elements is small. Figure 1 shows an example of a sorting network.

The number of compare–swap components and the delay are two crucial parameters of any sorting network. Table 1 shows the number of compare–swap components and delay of the best currently known sorting networks (for $N \leq 16$). These values are derived from the Knuth's book [5] and from paper [1].

Having a sorting network for N inputs, the *median* is simply the output value at the middle position (odd Ns only). For example, efficient calculation of the

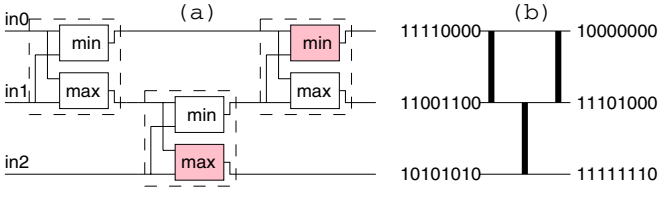


Fig. 1. (a) A 3-sorting network consists of 3 components, i.e. of 6 subcomponents (elements of maximum or minimum). A 3-median network consists of 4 subcomponents. (b) Alternative symbol.

Table 1. Parameters of the best-known sorting networks

N	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Delay	0	1	3	3	5	5	6	6	7	8	8	9	10	10	10	10
Comparators	0	1	3	5	9	12	16	19	25	29	35	39	45	51	56	60

median value is important in image processing where median filters are widely used with N = 3x3 or 5x5 [9]. Note that the popular implementation of the 9-median circuit in an FPGA proposed by Smith is also area-optimal (in terms of the number of components) [14].

The *zero–one* principle helps with evaluating sorting networks (and median circuits as well). It states that if a sorting network with N inputs sorts all 2^N input sequences of 0's and 1's into nondecreasing order, it will sort any arbitrary sequence of N numbers into nondecreasing order [5]. This principle will be utilized in the fitness function.

2.2 Evolutionary Approaches

Some of sorting and median networks were (re)discovered using evolutionary techniques [2,4,6,10]. Evolutionary techniques were also utilized to discover fault-tolerant sorting networks [12]. Since fitness function is typically based on the use of the *zero–one* principle, the evolution of larger sorting networks is not scalable (because the size of the test set doubles by increasing the number of inputs by 1). It is usually impossible to obtain the perfect solution (that sorts all 2^N input vectors) if only a subset of input vectors is utilized during the evolutionary design [3].

2.3 Intrinsic Evolution in FPGAs

In order to speed up candidate networks evaluation, Koza et al. have evaluated candidate sorting networks in Xilinx XC6216 FPGA. Genetic programming utilized for designing sorting networks was running in PC. For example, using population size 60k, minimal 8-SN was evolved on generation 58, and using a population size 100k, minimal 9-SN was evolved on generation 105. The evolution of minimal 7-SN required 69 minutes on the FPGA (31 generations, population size 1000). The evaluation of a candidate sorting network in XC6216 FPGA was 46 times faster than in Pentium 90MHz [6].

Some other FPGA-based implementations of complete evolvable systems have been proposed for various problems in the recent years. There are some examples: In Tufte and Haddow's approach only register values representing coefficients of a digital filters were evolved [15]. Sloarch and Sharman [13] have proposed intrinsic evolution of small combinational circuits in FPGA. An automatic feature identification algorithm that utilizes functional level operators was developed for multi-spectral images in [8]. The concept of virtual reconfigurable circuit (i.e. the second level of reconfiguration implemented in a conventional

FPGA) was utilized in papers [11,16]. We use a hardware implementation of evolutionary algorithm because it overcomes the bottleneck introduced by slow communication between the FPGA and a personal computer (in which the evolution is usually performed). The proposed architecture for evolution of large sorting networks is based on Koza's seminal work [6] and Sekanina and Friedl's complete hardware implementation of an evolvable combination circuit [11]. Unlike in Koza's approach evolutionary algorithm will be implemented in hardware.

3 The Proposed Architecture

The proposed architecture for sorting network evolution consists of four basic components—Control Unit, Fitness Unit, Genetic Unit and Virtual Reconfigurable Circuit Unit (VRC Unit). All the units are implemented on a single FPGA. The block structure of the architecture is shown on the Figure 2.

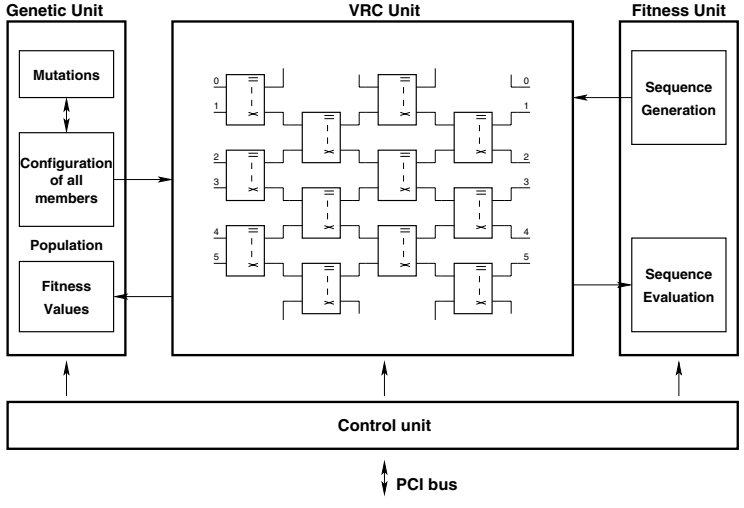


Fig. 2. Block structure of the proposed architecture

All operations are controlled by the control unit which is connected to PCI bus and executes the commands entered by user. For example, evolution can be started or stopped, the number of iterations can be specified, etc. The Genetic Unit executes genetic algorithm and contains all population members. VRC unit is a reconfigurable circuit in which the evolution is performed. That is implemented as a second level of reconfiguration on the FPGA. VRC is configured using chromosomes generated by Genetic Unit.

3.1 Genetic Unit

Genetic algorithm is based only on the mutation operator (bit inversion); crossover is not taken into account in this paper. We are going to investigate its

usefulness in next research. Population size is configurable. The new population is always generated from the best member of the previous one. Genetic algorithm operates in following steps: (1) Initialization Unit generates the first population at random (Linear Feedback Shift Register seeded from software is utilized). (2) Mutation Unit changes a given number of genes (bits) of a population member (this number is configurable) and the modified member is loaded into the VRC—it represents an image operators. (3) Genetic Unit is waiting for the evaluation performed by Fitness Unit and if the fitness value obtained is better than the parent's fitness then the chromosome replaces its parent. (4) This is repeated until the appropriate number of generations is produced.

3.2 VRC Unit

The unit consists of VRC elements that can perform different operations according to the selected configuration. Figure 3 shows its inteface.

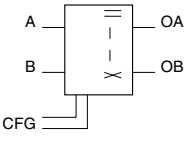


Fig. 3. VRC element architecture

Each element consists of two input and two output ports (everything is 1 bit). The functionality is determined by two configuration bits, which are used to select one of four different operations. All operations can be performed in one clock cycle. The result is stored into the register (local in each element) which offers to use VRC elements in a pipelined structure.

Sorting networks are usually composed of Compare&Swap components with various interconnection. For this reason, the proposed VRC element is designed to perform Compare&Swap operation. Alternatively, it can operate as a wire or cross-wire. The relation between configuration bits and functionality is shown in the following list:

- 00 – direct connection from inputs to outputs
- 01 – Compare&Swap operation – maximum on the upper output
- 10 – Compare&Swap operation – minimum on the upper output
- 11 – cross connection inputs to outputs

The VRC unit is composed of VRC elements in a fixed structure which is shown in Figure 3. Although the interconnection is invariable it can be changed if CFG = 00 or 11 is selected. The architecture is different for even Ns (left side in Fig. 4) and odd Ns (right side in Fig. 4).

A VRC element is connected to the four nearest neighbors. Even columns have inputs and outputs shifted by one item. This one-item-shift is necessary to

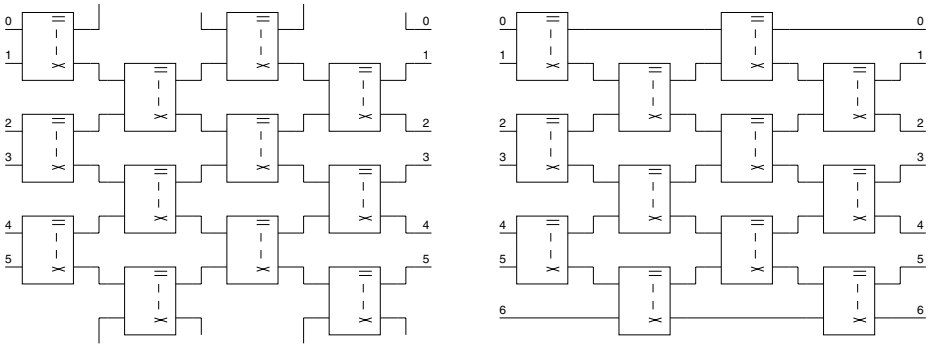


Fig. 4. Odd and even VRC array architecture

establish interconnection between arbitrary two Compare&Swap components or to compare different items. The architecture for odd Ns differs only in the first and last row where the remaining item is always connected to the next column via the synchronization register.

The proposed VRC array supports fully pipelined processing because every element contains synchronization register for the output values. Therefore, VRC array can produce one result in a clock cycle. Unlike VRCs in [11,16] this VRC element architecture is also optimized for hardware resources. Only two LUTs have to be utilized to create one VRC element. This optimization enables to fit a large VRC array within the single chip and thus to find sorting networks with many inputs.

3.3 Fitness Unit

The Fitness unit is used to generate unsorted sequences and evaluate results calculated by VRC unit. N-bit counter generates the unsorted input vectors, i.e. all possible combinations over N bits. In the evaluation process all vectors unsorted by VRC have to be identified.

The fitness value is defined as the number of unsorted vectors coming from the VRC unit. The unsorted vector is detected if for any item a_i in the vector $a_i < a_{i+1}$ does not hold. This is performed in parallel by a set of comparators in only one clock cycle. The number of unsorted vectors is stored in a counter which is incremented when an unsorted vector is identified. The content of the counter is presented as a fitness value which is valid after all input vectors are evaluated. In this paper, we are interested only in functionality of sorting networks; the number of components is not optimized.

4 Results of Synthesis

The proposed architecture is designed for evolution of sorting networks having different number of inputs. From this point of view, the size of the VRC array

Table 2. XC2V-3000 FPGA utilization for various VRC and N

N length	VRC Elements	Slices	Chip Utilization
10	5x30	1731	12 %
12	6x36	2262	15 %
14	7x42	2735	19 %
16	8x48	3207	22 %
18	9x54	3795	26 %
20	10x60	4431	30 %
22	11x66	5675	39 %
24	12x72	6412	44 %
26	13x78	7314	51 %
28	14x84	8173	57 %
30	15x90	9232	64 %
32	16x96	10223	71 %
36	18x108	12468	86 %

has to be scalable and support as many rows and columns as possible. On the other hand, the design with VRC array has to fit within the single chip. For this reason, the proposed VRC element was optimized for the hardware resources utilization.

The implementation of this architecture and evaluation of results have been performed on available COMBO6 hardware platform. COMBO6 is a PCI card equipped with a Field Programmable Gate Array XC2V-3000, TCAM memory, static and dynamic Random Access Memories and some other components.

The synthesis results in Table 2 show hardware resources utilization of the XC2V-3000 FPGA for different size of the VRC array and N. It can be seen that the FPGA utilization for the largest VRC array 18×108 is only 86 % without performance lost.

5 Experimental Results

Various VRC architectures were synthesized up to $N = 20$. For each VRC size, 80 independent experiments are performed and analyzed. We used four-member population and produced 50000 generations. Only mutation operator is used; 4 bits are inverted in chromosome in average.

Table 3 shows that it is possible to find correct sorting networks (for relatively large Ns) in a reasonable time. The first column shows the vector length (i.e. N). The number of correct sorting networks discovered out of 80 runs is reported in the second column. In 4th column there is the average number of generations needed to find the perfect solution (and its standard deviation in 5th column). The last column contains the time needed to generate and evaluate one individual (i.e. 2^N test cases). The evaluation of a candidate network requires 1.3 ms for $N = 16$ and 40s for $N = 32$ (at 50MHz).

Table 3. Sorting networks evolved in FPGA

N length	VRC size (elements)	#Perfect solutions	Average num. of generations	Standard deviation	Evaluation time of one candidate
4	2x8	80	94	2.55556	512 ns
6	3x16	80	458	31.44444	1.28 us
8	4x16	80	2217	24.66667	5.12 us
10	5x32	80	6378	65.66667	20.48 us
12	6x32	76	8673	666.88889	81.92 us
14	7x32	75	11322	718.55556	327.67 us
16	8x32	66	19467	477.44444	1.31 ms
18	9x64	20	25306	3732.77778	5.24 ms
20	10x64	17	31344	150.00000	20.97 ms

Table 4. Large sorting networks evolved in FPGA

N length	VRC size (elements)	Number of generation	Evaluation time of one candidate	Total time of evolution
22	11x64	4044	83.89 ms	5.6 min
24	12x64	4804	335.54 ms	26.9 min
26	13x64	10027	1.342 s	3.7 h
28	14x64	13483	5.368 s	20.1 h

In order to evolve larger sorting networks we applied an adaptive mutation. With respect to N we mutated $4 - 12$ bits per chromosome. If no improvement in fitness value is observed in last 1000 generations, the number of mutated bits is increased by 2. If an improvement is observed, the mutation ratio is changed back to the previous value. Table 4 presents some of the evolved sorting networks up to $N = 28$. The evolution of a 28-input sorting network requires more than 20 hours (at 50 MHz). The design can easily work at 100 MHz as well.

6 Discussion

In fitness functions, all possible input combinations are evaluated, i.e. 2^N test vectors are evaluated for N-input sorting network. In [10] median networks (whose evaluation is of the same complexity as for sorting networks) were evolved up to $N = 25$ in software. However, component-optimal solutions were not obtained for larger N. It was reported that the fitness calculation (performed in software) is very time consuming for $N \geq 23$ and evolution requires days to find a solution. Here we demonstrated that a special architecture implemented in hardware could make the evolutionary design significantly faster. Alternatively, we could reduce the training set; however, we have never obtained a perfect solution with the reduced training set.

We have evolved relatively large combinational circuits (28 inputs, 28 outputs) from scratch in a relatively short time (about 20 hours) and in (relatively low-cost) commercial off-the-shelf hardware. On the other hand, we have used a lot of domain knowledge for solving this problem (the usage of compare&swap components, invariable interconnection of components etc. is typical only for this problem). We demonstrated what complex circuits can be evolved on commercially available FPGAs. The evaluation of a single candidate sorting network for $N = 28$ was compared against highly optimised SW implementation running in Xeon 3 GHz. Our FPGA evaluation running at 100 MHz is 40× faster then the software approach.

A strongly generic approach was utilized during VHDL design. All the implemented units are parameterized using various constants (such as the size of chromosome, the number of mutations etc.). Therefore, it is easy to modify the design and to obtain a totally different evolvable system in a very short time. The FPGA communicates with PC via special software allowing designer to prepare scripts describing experiments that have to be performed. Typically, designer specifies the VRC, EA and fitness function, perform synthesis, upload the evolvable system into FPGA and execute all experiments described in scripts.

7 Conclusions

A specialized architecture was developed and evaluated to evolve relatively large sorting networks in an ordinary FPGA. We evolved sorting networks up to $N = 28$. The evolution of the largest sorting networks requires 10 hours in FPGA running at 100 MHz. In next research, the number of components utilized in the evolved networks will be optimized. As target future application of this approach we consider adaptive routing in computer networks.

Acknowledgments

The research was performed with the financial support of FRVS 3042/2005/G1 project *Evolutionary design of sorting and median networks in FPGAs*. Lukas Sekanina was supported from the research project of the Grant Agency of the Czech Republic under No. 102/03/P004 *Evolvable hardware based applications design methods*.

References

1. Devillard, N.: Fast Median Search: An ANSI C Implementation. 1998 http://ndevilla.free.fr/median/median/index.html
2. Hillis, W. D.: Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D 42 (1990) 228–234
3. Imamura, K., Foster, J. A., Krings, A. W.: The Test Vector Problem and Limitations to Evolving Digital Circuits. In: Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware, IEEE CS Press, 2000, p. 75–79

4. Juillé, H.: Evolution of Non-Deterministic Incremental Algorithms as a New Approach for Search in State Spaces. In Proc. of 6th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1995, p. 351–358
5. Knuth, D. E.: The Art of Computer Programming: Sorting and Searching (2nd ed.), Addison Wesley, 1998
6. Koza, J. R., Bennett III., F. H., Andre, D., Keane, M. A.: Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann, 1999
7. Liberouter project. www.liberouter.org
8. Porter, R.: Evolution on FPGAs for Feature Extraction. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2001, p. 229
9. Sekanina, L.: Evolvable components: From Theory to Hardware Implementations, Springer-Verlag, Natural Computing Series, 2003
10. Sekanina, L.: Evolutionary Design Space Exploration for Median Circuits. In: Applications of Evolutionary Computing, Coimbra, Portugalsko, LNCS 3005, Springer Verlag, 2004, p. 240-249
11. Sekanina, L., Friedl, S.: On Routine Implementation of Virtual Evolvable Devices Using COMBO6. In: Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, USA, IEEE Computer Society Press, 2004, p. 63-70
12. Shepherd, R., Foster, J.: Inherent Fault Tolerance in Evolved Sorting Networks. In Proc. of GECCO 2003, LNCS 2723, Springer Verlag, 2003, p. 456–457
13. Sloarch, C., Sharman, K.: The Design and Implementation of Custom Architectures for Evolvable Hardware Using Off-the-Shelf Progarmmable Devices. In: Proc. of the 3rd International Conference on Evolvable Systems: From Biology to Hardware ICES'00, LNCS 1801, Springer-Verlag, Berlin, 2000, p. 197–207
14. Smith, J. I.: Implementing Median Filters in XC4000E FPGAs. Xcell 23, Xilinx, 1996 http://www.xilinx.com/xcell/xl23/xl23_16.pdf
15. Tufte, G., Haddow, P.: Evolving an Adaptive Digital Filter. In: Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware. IEEE Computer Society, 2000, p. 143–150
16. Zhang, Y., Smith, S., Tyrrell, A.: Intrinsic Evolvable Hardware in Digital Filter Design. In: Applications of Evolutionary Computing, Berlin, DE, Springer, LNCS 3005, 2004, p. 389-398

Evolving Hardware by Dynamically Reconfiguring Xilinx FPGAs

Andres Upegui and Eduardo Sanchez

Ecole Polytechnique Fédérale de Lausanne – EPFL,
Logic Systems Laboratory – LSL, 1015 Lausanne, Switzerland
(andres.upegui, eduardo.sanchez)@epfl.ch

Abstract. Evolvable Hardware arises as a promising solution for automatic digital synthesis of digital and analog circuits. During the last decade, a special interest has been focused on evolving digital systems by directly mapping a chromosome on the FPGA configuration bitstream. This approach allowed a great degree of flexibility for evolving circuits. Nowadays, FPGAs routing scheme does not allow doing it in such flexible and safe way, so additional constraints must be introduced. In this paper we summarize three techniques for performing hardware evolution by exploiting the capacities of Virtex families. Among our proposals there are high and low level approaches, and coarse and fine grained components. A modular based evolution, with pre- placed and routed components, provides a coarse grain approach. Two techniques for directly modifying LUT contents on hard macros provide a fine grained evolution. Finally, integrating both approaches, coarse and fine grain, provides a more general and powerful framework.

1 Introduction

Designing analog and digital electrical circuits is, by tradition, a hard engineering task, vulnerable to human errors, and no one can guarantee the optimality of a solution. Design automation has become a challenge for tool designers, and given the increasing complexity of circuits, higher description levels are needed. Evolvable Hardware (EHW) arises as a promising solution to this problem: from a given behavior specification of a circuit, an evolutionary algorithm (EA) can find a circuit able to implement this function. EAs take inspiration from the principles of biological evolution decoding a phenotype from a genotype. The genotype is a number string, where the genetic operations, cross-over and mutation, are applied. Reproduction is performed by cross-over of genomes and mutation is performed on a probabilistic way. From this genome a phenotype is decoded for obtaining a circuit with a given set of components and connectivity (in the case of EHW). A fitness note is assigned to this individual given the performance exhibited according to a fitness function. EHW have shown to perform well finding solutions [1] from simple Boolean functions to complex analog circuits, sometimes performing better than hand-made solutions.

For evolving hardware there is a first main issue to address: the hardware substrate supporting the evolved circuit. Different custom chips have been proposed for this

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 56 – 65, 2005.

purpose with very interesting results: the main interest on proposing an architecture is that commercial FPGAs are designed for general purpose applications, so they would not necessarily fit the requirements for evolvable architectures. Custom evolvable chips use to provide dynamic and partial reconfiguration, dispose of multi-context memories and can be configured with random configuration bitstreams. The commercial options main advantage is the absence of non-recurrent engineering, as any general purpose architecture, under the cost of reduced flexibility and performance.

Different chips and platforms have been developed providing the flexibility necessary for evolving analog, digital and mixed circuits; some of them have been designed specifically targeting evolvable hardware, while others have just found in evolvable hardware another application field. Among them one can find different levels of granularity, different types of reconfiguration, including dynamic and static reconfigurations, possibility of loading partial configuration bitstreams, and the utilization of context memories.

One of the more recent chips is the POEtic tissue [2], a platform for bio-inspired hardware composed of three layers: phenotype, mapping, and genotype, each one of them supporting each of the three axes of life: phylogenesis (evolution), ontogenesis (development) and epigenesis (learning). Previous work on evolvable architectures have been done by Moreno et al. with FIPSOC [3, 4], a chip integrating digital and analog programmable circuits, with a dynamic multi-context reconfiguration for the digital section, focusing on evolution of parallel cellular machines. Higuchi's group has developed an evolvable LSI chip [5], which includes a genetic algorithm unit, and the ability to process two chromosomes in parallel. Layzell developed the Evolvable Motherboard (EM) [6], a diagonal matrix of analogue switches, connected to a set of daughter-boards, which contain the basic components for performing the evolution.

Other platforms, such as MorphoSys [7], DREAM [8], and Palmo [9], were not initially designed for bio-inspired systems; however, their flexible and performing architecture fits well with EHW requirements.

Among commercial options, the FPGA XC6200 from Xilinx (already obsolete for commercial reasons) constituted the perfect platform for intrinsic evolvable hardware; it was possible to download any arbitrary bitstream without risking contentions, given its multiplexer-based connection architecture. Additionally, this FPGA family allowed dynamic reconfiguration, making it more flexible for adaptive algorithms in a general sense. Maybe the most known work using these devices is that of Adrian Thompson [10, 11] who refers to intrinsic as "belonging to the point at issue" and it reflects very well his work. He evolved analog circuits, by exploiting the dynamics inherent to the physical properties of the FPGA internal components. On the same way cooperative robot controllers have been also evolved with these FPGAs with impressive results [12]. Such success of these families has motivated researchers to implement the same architecture on other still available FPGAs: [13] presents an "emulated" 6200-like cell on commercial available architectures (XC4010 and Altera EPF6010A), and they evolve a configuration bitstream which does not configure the FPGA itself but the "emulated" 6200-like cells.

More recent work on evolvable circuits on commercial FPGAs has focused on Virtex and Virtex-II architectures from Xilinx (and will extend to Virtex IV). The special interest on these devices is their partial dynamic reconfigurability, with the limitation, compared with the XC6200, that no arbitrary configuration bitstreams can

be loaded. This limitation is given by the multidirectional nature of the connectionism: it is possible, with an incorrect bitstream, to interconnect two logic gate outputs damaging the device.

For evolving circuits on Virtex architectures one must take care of not generating invalid bitstreams – i.e. bitstreams causing internal contentions – and different approaches have been proposed for dealing with that problem. At the University of York they have used Jbits, a Java API for describing circuits and manipulating configuration bitstreams, for evolving circuits. From a genome they map LUTs contents for evolving simple combinatorial functions [14], or robot controllers for obstacle avoidance [15]. Also using Jbits, Levi and Guccione from Xilinx have developed a tool called GeneticFPGA [16], which from a chromosome translates a configuration bitstream, making easy to generate legal bitstreams. Even if Jbits provides interesting features for EHW, it has several limitations, such as the impossibility to run on an embedded platform (for on-chip evolution), dependability on supported FPGA families and supported boards, incompatibility with other hardware description languages (HDLs), and a limited support from Xilinx, mainly reflected in an insufficient documentation.

In this paper we present 3 techniques to evolve circuits on Virtex families without depending on Jbits. Two of these techniques are mainly based on the two flows for partial reconfiguration proposed by Xilinx in [17], while the third one consists on directly manipulating the bitstream without depending on any Xilinx tool. In section 2 we describe the two design flows proposed by Xilinx for performing dynamic partial reconfiguration in a safe way. In section 3 we describe how these techniques can be used for evolving hardware systems. And, finally, section 4 concludes.

2 Dynamic Partial Reconfiguration on Xilinx Families

FPGAs are programmable logic devices that permit the implementation of digital systems. They provide an array of logic cells that can be configured to perform a given function by means of a configuration bitstream. Some FPGAs allow performing partial reconfiguration, where a reduced bitstream reconfigures only a given subset of internal components. Dynamic Partial Reconfiguration (DPR) is done while the device is active: certain areas of the device can be reconfigured while other areas remain operational and unaffected by the reprogramming. For the Xilinx's FPGA families Virtex, Virtex-E, Virtex-II, Virtex-II Pro (applicable also for Spartan-II and Spartan-IIE) there are two documented flows to perform DPR: Module Based and Difference Based [17].

With the Difference Based flow the designer must manually edit low-level changes. Using the FPGA Editor, a low level edition tool, the designer can change the configuration of several kinds of components such as: look-up-table equations, internal RAM contents, I/O standards, multiplexers, flip-flop initialization and reset values. After editing the changes, a partial bitstream is generated, containing only the differences between the *before* and the *after* designs. For complex designs, the Difference Based flow results inaccurate due to the low-level edition in the bitstream generation.

The Module Based flow allows the designer to split the whole system into modules. For each module, the designer generates a configuration bitstream starting from an HDL description and going through the synthesis, mapping, placement, and routing procedures, independently of other modules. Some of these modules may be reconfigurable and others fixed (see figure 1). A complete initial bitstream must be generated, and then, partial bitstreams are generated for each reconfigurable module. Hardwired Bus Macros must be included. These macros guarantee that each time partial reconfiguration is performed routing channels between modules remain unchanged, avoiding contentions inside the FPGA and keeping correct inter-module connections.

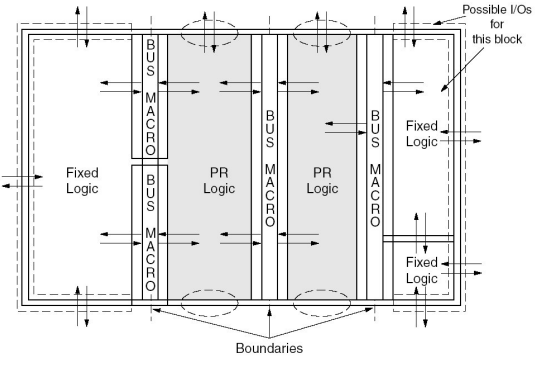


Fig. 1. Design Layout with Two Reconfigurable Modules (From [17])

3 Evolving Techniques

In this section we present 3 techniques for EHW on Virtex families. The first one is a coarse grained high level solution, well suited for architecture exploration. The second and the third one, very related among them, constitute a fine grained low level solution, well suited for fine tuning.

3.1 Module Based

The main consequence of the aforementioned features of DPR is a modular structure, where each module communicates solely with his neighbor modules through a bus macro (Figure 1). This structure matches well with modular architectures, such as layered neural networks, fuzzy systems, multi-stage filtering, etc. All systems with high needs of adaptability, and which can largely benefit from architecture exploration. Anyway, some design constraints must be respected: inputs and outputs of the full network must be previously fixed, as well as the number of layers and the connectivity among them (number and direction of connections). While each layer can have whatever kind of internal connectivity, connections among them are fixed through bus macros and restricted to neighbor layers.

Evolving artificial neural networks topologies by using this method have been reported in [18, 19]. For each module, there exists a pool of different possible configurations. Each configuration may contain a layer topology - i.e. a certain number of neurons with a given connectivity. Each module can be configured with different layer topologies, provided that they offer the same external view (i.e. the same inputs and outputs). Several generic layer configurations are generated to obtain a library of layers, which may be used for different applications.

A GA is responsible for determining which configuration bitstream is downloaded to the FPGA. The GA considers a full network as an individual. For each application the GA may find the combination of layers that best solves the problem. Input and output fixed modules contain the required logic to code and decode external signals and to evaluate the fitness of the individual depending on the application.

Two ways of generating bitstreams can be identified by using this technique: (1) by letting the EA to modify HDL or the netlist descriptions of the system or (2) by preplacing and routing all the possible modules to be used. The first option, letting the EA to modify HDL or netlist specifications, would definitely result in prohibitive execution times: a full placement and routing process should be executed for each individual, which is typically a very heavy computing task. The second option, preplacing and routing modules, results more accurate for EHW. Under this approach one can see each module as a coarse grain configurable block that can be configured with a set of predefined components. The EA would select the best combination of components to solve the problem. This technique results accurate for a global coarse search; however, for fine tuning it must be used another adaptation technique. For instance, in [18] a spiking neural network topology is evolved with this technique, but hebbian learning adjusts synaptic weights for each individual.

3.2 Hard-Macros Difference Based

Lower level partial bitstreams can be generated by using the Difference-Based flow. Using this technique to modify circuits requires a previous knowledge of the physical placement of the logical components implementing the target function – i.e. the logical function to be evolved – in the FPGA. By using hard macros one can define placement constraints; one can place each hard macro and, knowing LUT positions, one can modify them by using Difference-Based reconfiguration [17]. Hard macros must be designed by low level specification of a system: using the FPGA_editor one can define a system in terms of the FPGA basic components. Every CLB, LUT and flip-flop must be manually placed, and a semi-automatic routing must be performed.

Cooperative coevolution of fuzzy systems using this technique is described in [20]. They define two hard macros: a parameter macro and a fuzzy rule macro. The functionality of a parameter macro is just storing a constant parameter. After specifying placement constraints for this macro one can access and modify its contents automatically by using the FPGA editor. On the same way, the fuzzy rule macro can be automatically configured to implement a fuzzy-OR or a fuzzy-AND function (different from their Boolean counterparts).

For using this technique, the first step is to define an initial HDL description of the system. This description must include the hard macros to be evolved as black boxes. The hard macros must be designed before the placement and routing process. Place-

ment constraints must be specified for the hard macros, taking care of not overlapping them. After placing and routing the design, one must check that hard macros have been placed as desired. Now the system is ready to be evolved: a genetic algorithm running on your favourite programming language will generate LUT contents from a chromosome and will run a script for modifying the LUT contents on the FPGA_editor. Then a partial bitstream, just containing the LUT modifications, will be generated and downloaded to the FPGA.

This technique provides the possibility of fine tuning systems, under the cost of not allowing topological or connectionism modifications. It is well suited for evolving systems with cellular structures, such as neural networks, fuzzy system rules, or cellular automata, among others, with the main drawback of a dependence of Xilinx tools for modifying LUT contents and generating the bitstream. Even if the placement and routing process must not be executed for every individual, it is still not suited for on-system evolution.

3.3 Bitstream Manipulation

Up to now, all described evolving techniques are highly dependant on Xilinx tools, making them restrictive for on-chip evolution. An attempt for providing a similar functionality from Jbits running on-chip has been proposed by Xilinx engineers [21]: XPART (Xilinx Partial Reconfiguration Toolkit) is an application program interface (API), for Microblaze or PowerPC microprocessors, that provides methods to read and modify select FPGA resources by using the ICAP (Internal Configuration Access Port). Anyway, XPART was never released.

Directly evolving the configuration bitstream has been a very common technique. It has been widely used with the XC6200 family and on other custom platforms summarized in section 1. However, in every case one must maintain a fixed section – i.e. not evolved – in the bitstream. For instance, Thompson in [10], uses an XC6216 with an array of 64x64 logic cells, but the evolved circuit uses just an array of 10x10 logic cells, while keeping fixed input and output. In this case the evolved section of the bitstream is just that containing the 10x10 array while the sections for IO blocks and the remaining cells are kept constant during the evolution.

Exactly the same principle can be applied for Virtex families, including Virtex II, Virtex II-Pro and eventually Virtex 4: LUT contents can be evolved, while keeping a fixed routing. By using hard macros, as described in 3.2, one can describe a computing cell. This computing cell can implement a neuron, a fuzzy rule, a simple LUT, or any function, including one or several LUTs; it can include also flip-flops for making the design synchronous, or it can just implement combinatorial circuits. LUTs configuration can be modified in an arbitrary way; however, routing must remain fixed. Connectivity among components of a computing cell is manually set when designing the hard macro; connectivity among computing cells is defined by an HDL description of the full system. Although routing must remain fixed during evolution, LUTs can be evolved as multiplexers, where the selection is done by the configuration bitstream. An implementation using this principle is described in [22], where they present a cellular automata evolution running on a Virtex-E.

For the Virtex family, the XAPP151 [23] describes in a detailed way the configuration bitstream, specifying the position of LUT contents on the bitstream. However, for

the Virtex II family this documentation is not available and just a limited bitstream description can be found in [24]. A Virtex-II bitstream is divided by columns and each column is composed by frames. There are different column types, each type with a given number of frames, as described in figure 2 (for more details refer to [24]). However, there is no documentation about frame composition, consequently, no information about LUT contents.

We present in the following paragraphs how to address LUT contents on a bitstream. This problem can be solved by exploring the bitstream content. In the Virtex-II architecture each CLB has 4 slices arranged 2x2. This arrangement makes that each CLB column has 2 slices columns, which are numerated in the format XiYj, with i from 0 to 2n-1 beginning from the left (n is the number of CLB columns) and j from 0 to 2m-1 beginning from the bottom (m is the number of CLB rows). For instance, for an XC2V40 (with array 8x8) the slice placed at the top left of the component is called the slice X0Y15. Each one of these slices has 2 LUTs called G-LUT and F-LUT.

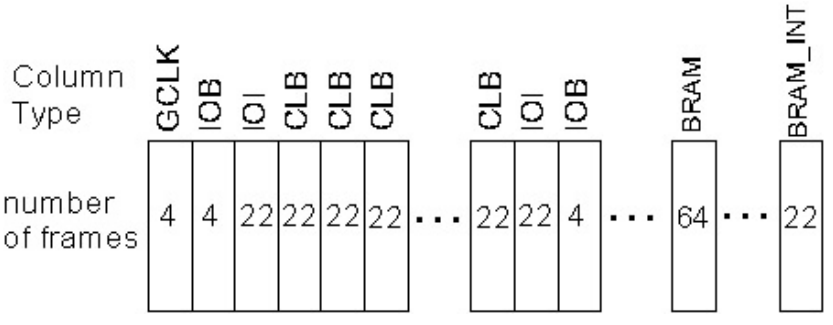


Fig. 2. Virtex-II configuration bitstream composition

Table 1. Frame description. The first 12 bytes configure the IOB, the next 2 bytes configure the G-LUT contents for the top slice, the next byte has an unknown functionality, the next 2 configure the G-LUT, … This sequence is repeated for every slice, and finishes by the bottom IOB configuration. * Supposing it is the second frame of the first CLB column for an XC2V40.

Description	Size (bytes)
Top IOB	12
Top slice G-LUT (slice X0Y15)*	2
--	1
Top slice F-LUT (slice X0Y15)*	2
2nd slice G-LUT (slice X0Y14)*	2
--	1
2nd slice F-LUT (slice X0Y14)*	2
…	
…	
Bottom slice F-LUT (slice X0Y0)*	2
Bottom IOB	12

Even if it is not documented, LUT contents can be localized in the configuration bitstream. As shown in figure 2 a CLB column contains 22 frames; the contents for the first slices column LUTs – i.e. with an even X – can be found in the second frame, while for the second slices column – i.e. with an odd X – are in the third frame. Frame contents are described in Table 1. It must be noticed also that, as in Virtex family, LUT configurations are stored inverted – i.e. for an 4-input AND function, LUT contents must be 1000 0000 0000 0000, but actually it stored like 0111 1111 1111 1111 or 7F FF in hex format –. Additionally, the bit order is swapped in F-LUTs respective to G-LUTs – i.e. the same AND function in a G-LUT is stored 7F FF in the configuration bitstream, while for a F-LUT function it is stored FF FE –.

Based on this description one can determine any LUT content position on the bitstream by applying the following equation:

```
Position=    Size of the header
           + #GCLK_col_frames X #bytes/frame
           + #IOB_col_frames  X  #bytes/frame
           + #IOI_col_frames   X  #bytes/frame
           + #Xcoord_of_CLB_col  X  #CLB_col_frames  X  #bytes/frame
           + 1 frame X  #bytes/frame            if slice X coord is even
           + 2 frames X  #bytes/frame           if slice X coord is odd
           + 12 bytes                           -- IOB config.
           + 5 bytes X slice_Ycoord(from top)
           + 0 bytes                            if G-LUT
           + 3 bytes                            if F-LUT
```

Almost all these values are constant for every Virtex-II family devices; just the #bytes/frame depends on the number of CLB rows of the device. The header size is variable, and depends on the configuration options enabled for the bitstream (Details on the header can be found in [24]).

Accessing LUT contents on a partial bitstream is even easier, since one can directly address the target frame by setting the frame address in the bitstream header as described in [24]. Then, the table 1 description may be used for directly localizing LUT contents on the frame.

Accessing the configuration bitstream, as described in this section, allows evolving circuits in a very flexible way. By defining an initial interconnection schema one can let an on-chip – or off-chip – processor to simply modify partial bitstreams just containing the LUT-frames. This powerful approach can be largely improved when combined with the technique proposed in section 3.1, allowing additionally different connectionism schemas (unfortunately predefined and fixed).

4 Conclusions

In this paper we presented three techniques for evolving hardware on Virtex families. The first technique, module based, provides a high level abstraction of the system and fits well for coarse topology exploration; anyway, it remains inaccurate for fine tuning, given its coarse grain nature. The second technique, hard macros difference

based, is a good complement for the first technique: by manually placing hard macros one can modify CLB contents. It allows fine tuning, but the overall routing must remain fixed. The third method, bitstream manipulation, provides almost the same features that the second one with the great advantage that, given its independence from Xilinx tools, it can run on-system, under the cost of a slight flexibility loses.

By integrating the coarse grain with a fine grain approach one can largely enhance the search space and increase the flexibility of the evolvable platform. By evolving a system at topology and LUT contents level, one can play with the trade-off exploration vs. exploitation: fine tuning parameters while major topology changes are performed. Different basic computing units may be used, specified as hard macros, such as artificial neurons, fuzzy rules, or cellular automata.

Even if Virtex families are not specifically conceived for EHW, methodologies can be proposed for exploiting their performance and flexibility features. These methodologies may rely on the techniques presented in this paper and should deal with the kind of basic element - modules or hard macros – to evolve, as well as the genome coding or the type of EA best suited for a given structure.

References

[1] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, M. Salami, N. Kajihara, and N. Otsu, "Real-world applications of analog and digital evolvable hardware", *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 220-235, 1999.

[2] Y. Thoma, G. Tempesti, E. Sanchez, and J. M. M. Arostegui, "POEtic: an electronic tissue for bio-inspired cellular applications", *Biosystems*, vol. 76, pp. 191-200, 2004.

[3] J. M. Moreno, J. Cabestany, J. Madrenas, E. Canto, J. Faura, and J. M. Insenser, "Approaching Evolvable Hardware to Reality: The Role of Dynamic Reconfiguration and Virtual Meso-Structures", presented at Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems, Granada, Spain, 1999.

[4] J. M. Moreno, J. Madrenas, J. Faura, E. Canto, J. Cabestany, and J. M. Insenser, "Feasible evolutionary and self-repairing hardware by means of the dynamic reconfiguration capabilities of the FIPSOC devices", *Evolvable Systems: From Biology to Hardware*, vol. 1478, pp. 345-355, 1998.

[5] M. Iwata, I. Kajitani, Y. Liu, N. Kajihara, and T. Higuchi, "Implementation of a Gate-Level Evolvable Hardware Chip", *Evolvable Systems: From Biology to Hardware*, vol. 2210, pp. 38, 2001.

[6] P. Layzell, "'Evolvable Motherboard': A test platform for the research of intrinsic hardware evolution", *Tech rep. University of Sussex*, 1998.

[7] G. Lu, E. M. C. Filho, V. Castro Alves, H. Singh, M.-h. Lee, N. Bagherzadeh, and F. J. Kurdahi, "The MorphoSys Dynamically Reconfigurable System-on-Chip", *Proceedings of The First NASA/DOD Workshop on Evolvable Hardware*, pp. 152, 1999.

[8] J. Becker, T. Pionteck, and M. Glesner, "DReAM: A Dynamically Reconfigurable Architecture for Future Mobile Communication Applications", *Field-Programmable Logic and Applications. The Roadmap to Reconfigurable Computing: 10th International Conference, FPL 2000, Villach, Austria, August 27-30, 2000. Proceedings*, vol. 1896, pp. 312, 2000.

[9] A. Hamilton, K. Papathanasiou, M. R. Tamplin, and T. Brandtner, "Palmo: Field programmable analogue and mixed-signal VLSI for evolvable hardware", *Evolvable Systems: From Biology to Hardware*, vol. 1478, pp. 335-344, 1998.

[10] A. Thompson, "An evolved circuit, intrinsic in silicon, entwined with physics", *Evolvable Systems: From Biology to Hardware*, vol. 1259, pp. 390-405, 1997.

[11] A. Thompson and P. Layzell, "Evolution of robustness in an electronics design", *Proc. of Evolvable Systems: From Biology to Hardware,*, vol. 1801, pp. 218-228, 2000.

[12] D.-W. Lee, C.-B. Ban, K.-B. Sim, H.-S. Seok, Lee Kwang-Ju, and B.-T. Zhang, "Behavior evolution of autonomous mobile robot using genetic programming based on evolvable hardware", *Proceeding of 2000 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, pp. 3835-3840, 2000.

[13] C. Slorach and K. Sharman, "The design and implementation of custom architectures for evolvable hardware using off-the-shelf programmable devices", *Evolvable Systems: From Biology to Hardware, Proceedings*, vol. 1801, pp. 197-207, 2000.

[14] G. Hollingworth, S. Smith, and A. Tyrrell, "The intrinsic evolution of Virtex devices through Internet reconfigurable logic", *Evolvable Systems: From Biology to Hardware, Proceedings*, vol. 1801, pp. 72-79, 2000.

[15] A. M. Tyrrell, R. A. Krohling, and Y. Zhou, "Evolutionary algorithm for the promotion of evolvable hardware", *IEE Proceedings-Computers and Digital Techniques*, vol. 151, pp. 267-275, 2004.

[16] D. Levi and S. A. Guccione, "GeneticFPGA: Evolving Stable Circuits on Mainstream FPGA Devices", *Proceedings of The First NASA/DOD Workshop on Evolvable Hardware*, pp. 12, 1999.

[17] Xilinx_Corp., "XAPP 290: Two Flows for Partial Reconfiguration: Module Based or Difference Based": www.xilinx.com, Sept, 2004.

[18] A. Upegui, C. A. Peña-Reyes, and E. Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks", *Microprocessors and Microsystems. In press*, 2005.

[19] A. Upegui, C. A. Peña-Reyes, and E. Sanchez, "A methodology for evolving spiking neural-network topologies on line using partial dynamic reconfiguration", presented at ICCI - International Conference on Computational Intelligence, Medellin, Colombia, 2003.

[20] G. Mermoud, A. Upegui, C. A. Peña-Reyes, and E. Sanchez, "A Dynamically-Reconfigurable FPGA Platform for Evolving Fuzzy Systems", in *The 8th International Work-Conference on Artificial Neural Networks (IWANN'2005)*, (To appear) 2005.

[21] B. Blodget, P. James-Roxby, E. Keller, S. McMillan, and P. Sundararajan, "A self-reconfiguring platform", *Proceedings of Field-Programmable Logic and Applications*, vol. 2778, pp. 565-574, 2003.

[22] G. Tufte and P. C. Haddow, "Biologically-inspired: A rule-based self-reconfiguration of a virtex chip", *Computational Science - Iccs 2004, Pt 3, Proceedings*, vol. 3038, pp. 1249-1256, 2004.

[23] Xilinx_Corp., "XAPP 151:Virtex Series Configuration Architecture User Guide": www.xilinx.com, Oct, 2004.

[24] Xilinx_Corp., "Virtex-II Platform FPGA User Guide": www.xilinx.com, March 2005.

A Flexible On-chip Evolution System Implemented on a Xilinx Virtex-II Pro Device

Kyrre Glette and Jim Torresen

Department of Informatics, University of Oslo,
P.O. Box 1080 Blindern, N-0316 Oslo, Norway
{kyrrehg, jimtoer}@ifi.uio.no

Abstract. There have been introduced a number of systems with evolvable hardware on a single chip. To overcome the lack of flexibility in these systems, we propose a single-chip evolutionary system with the evolutionary algorithm implemented in software on a built-in processor. This architecture is implemented in a Xilinx Virtex-II Pro FPGA with an embedded PowerPC processor. This allows for a rapid processing of the time consuming parts in hardware and leaving other parts to more easily modifiable software. This platform will be beneficial for future work regarding both cost and compactness. Experiments have been performed on the physical device with software running in parallel with fitness computation in digital logic. The results show that the system uses only twice as much time when compared to a PC running at 10 times faster clock speed.

1 Introduction

Evolution time is critical for online evolvable systems. Further, often the compactness and cost of the system would be important. Thus, integrating as much as possible of a system on a single chip would be important.

There have been undertaken some implementations earlier. Kajitani et al have introduced several LSI (Large-Scale Integrated Circuits) devices with evolution undertaken in hardware [2,3]. 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. On-chip evolution using a prototype of the VLSI (Very Large-Scale Integration) POEtic chip has also been reported [7]. A robot controller and logic functions (3-input multiplexer and full adder) were evolved. The architecture contains an on-chip custom 32-bit processor, and a bio-inspired array of building blocks. This chip is specialized for the implementation of bio-inspired mechanisms.

In this paper, we demonstrate how a commercial FPGA (Field Programmable Gate Array) can provide a platform for System-On-Chip evolution. This is by integrating the evolution running as software on a processor with the target evolvable hardware implemented in reconfigurable logic. This allows for fast fitness computation – normally the most time consuming part of evolution, by measuring fitness in hardware communicating with a processor within the same hardware chip.

Implementing complete evolution in an FPGA has been proposed by Tufte and Haddow in [12]. The evolving design is implemented in the same device as the evolutionary

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 66–75, 2005.

algorithm. A similar approach is proposed by Perkins et al in [6]. 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 [10].

Running complete evolution within a Virtex XC2V3000 FPGA has been reported by Sekanina [9]. In this work the evolution (mutation only) is implemented in reconfigurable logic. Correctly working 3×3 and 3×4 bit multipliers were evolved.

In the work presented in this paper, a XC2VP7 Virtex-II Pro FPGA has been applied. It consists of reconfigurable logic, a PowerPC 405 hard-core processor block, on-chip RAM and high speed serial links for external interfaces.

There has been developed one other system for the Virtex-II Pro device [8]. This work is based on designing a co-processor for an analog neural network ASIC. This is contrasted to our work, where we focus on evolution of digital circuits and an evolutionary system in a single device. Further, in our system, all parts of the evolution (except the fitness evaluation, 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.

Another motivation, for having on-chip evolution and fitness computation in a single unit, is that it allows for scalable systems. By connecting a number of such units into a grid, they can perform concurrent evolution. Thus, this will be a very scalable architecture as well as flexible. In addition to the flexibility provided by software, the hardware would also be relatively easy to modify.

To demonstrate the achievable performance, experiments will be based on evolving small multiplier circuits. This will be to document the speed of evolution rather than evolution of very large and complex circuits which time has not allowed us to do so far. A number of papers have earlier contained work on evolving multipliers off-chip and extrinsically [4,5,13,11].

The next section introduces the applied FPGA followed by our architecture in Section 3. Results from the implementation are given in Section 4. Finally, Section 5 concludes the paper.

2 The Virtex-II Pro FPGA

The design is synthesized for a Xilinx Virtex-II Pro (XC2VP7-FG456-7) – see Fig. 1. This device contains 11,088 logic cells, 792 Kbit dual-port SRAM - named Block SelectRAM (BRAM), and one PowerPC 405 (PPC) embedded processor. The maximum processor speed is 300MHz.

The FPGA is situated on a Memec Design Virtex-II Pro development board. The board also contains two Xilinx XC18V04 configuration EEPROMS, 32MB SDRAM, Rocket I/O ports, an RS-232 port, an LCD panel and other useful connectors.

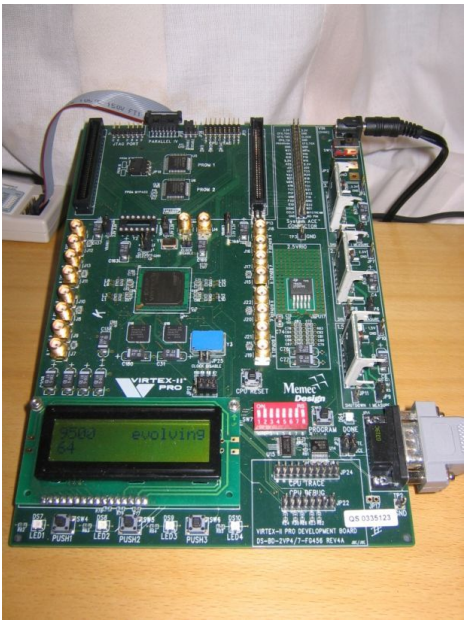


Fig. 1. The prototype board with the Virtex-II Pro FPGA

3 Implementing Evolutionary Algorithm on FPGA

In this section, the implementation of the evolvable hardware system will be detailed.

3.1 System Overview

The on-chip system is built using the Xilinx Embedded Development Kit (EDK) [15]. 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.

The architecture consists of a set of modules interconnected with buses. The bus system is a part of IBM's CoreConnect architecture [16]. Two main buses are used to connect the on-chip peripherals – the Processor Local Bus (PLB) and the On-chip Peripheral Bus (OPB), as seen in Fig. 2. The PLB is a high-performance 64-bit datapath bus, while the OPB is a 32-bit wide bus designed for peripherals with lower requirements. These buses can be run in different clock domains, and they are interconnected with the PLB to OPB bridge.

The communication intensive modules are connected to the PLB: The PPC CPU and 64KB BRAM for program instructions. The PPC processor is connected as a bus master to the PLB. In addition to the PLB interface, there are two On-Chip Memory (OCM) interfaces in the PPC [14]. These are used as dedicated interfaces between the

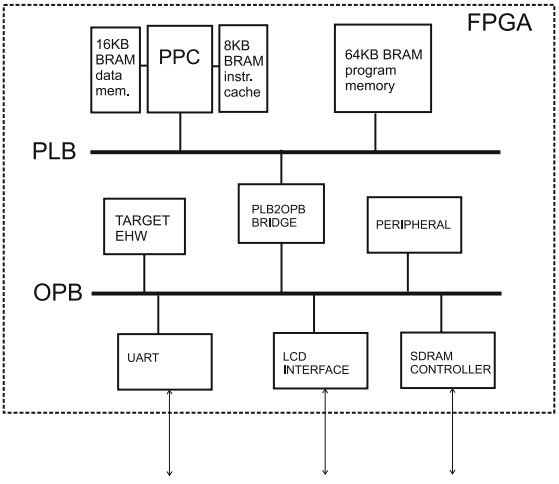


Fig. 2. Hardware architecture including our target EHW

FPGA BRAM and the PPC core. One OCM is for the instruction-side memory space and the other is for the data-side memory space. These interfaces are usually used for accessing instruction and data caches, built from BRAM (there is no cache inside the PPC block). The advantage of these OCM interfaces over the PLB interface is that no bus arbitration is necessary for memory access, and the instruction and data accesses do not have to share the same interfaces. In our design, 8KB of BRAM is connected to the instruction side OCM interface, used as an instruction cache. 16KB is connected to the data side OCM interface. This memory is used as storage for all program data, ie. no access to the PLB is needed at all. This increases program execution speed. In our case, a $2\times$ program execution speed increase was obtained by introducing instruction caching, and another $3\times$ increase was obtained by accessing all program data through the data side interface.

The target evolvable hardware is at the moment connected to the OPB. This will be detailed in section 3.3. Various on-chip peripherals are also connected to the OPB, including a UART for RS-232 serial communications, an LCD interface and a LED interface. An SDRAM controller can also be connected to the OPB should more memory be needed.

3.2 Implementing a Genetic Algorithm on the PPC

A Genetic Algorithm (GA) was implemented to run on the PPC. The program was written in C and compiled and linked using the PPC405 version of the GNU GCC compiler tools. Some system limitations had to be taken into consideration when implementing the GA on the embedded PPC system.

Firstly, program memory is limited. A maximum of 64KB of BRAM was allowed for the executable size. Although there exists 32MB of SDRAM on the development board, it was decided to only use BRAM internal to the FPGA. The BRAM is faster,

and it allows for the program to be loaded directly from the bitstream that configures the FPGA. A program which is too large to fit into the BRAM would have to be loaded into SDRAM using a boot loader from external nonvolatile memory at startup (although during development it is possible to initialize SDRAM through the JTAG interface). SDRAM could still be used for data storage, but this would be a rather slow solution, at least if no data caching is used.

Secondly, there is no floating point support on the PPC405. Floating point operations used in C programs have to be emulated, unless a floating point co-processor is available. Emulating floating point is not speed efficient, and it increases the executable size.

The limitations on the executable size led us to use of C instead of C++, and to minimize the use of standard library functions. Speed considerations made us avoid using floating point in time-critical program parts. Ideally, only fixed point or integer solutions should be employed in order to reduce the executable size.

The combination of C-only programming, restricted use of library functions and floating point operations, makes the implementation slightly more challenging and time consuming than it would have been on a PC. However, the degree of program flexibility and the speed of algorithm implementation is still very high compared to assembly programming or custom hardware solutions. Having the above-mentioned limitations in mind, the program was developed mostly using Microsoft's Visual Studio, and it can be run on both the PC and the FPGA platform. Only a few code paths had to be written specifically for the PPC. The PC version of the program is equally fast as if it would have been developed for PC only.

The GA implemented for this experiment follows the Simple GA style, given by Goldberg [1]. 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.

3.3 Target Evolvable Hardware

The target EHW is implemented as an OPB slave peripheral module – see Fig. 3. 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. IPIF cores exist for both OPB and PLB buses, so an adaptation of the target EHW to the PLB should be a feasible task.

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 each functional unit. Registers are also provided for feeding the EHW with inputs and for storing the outputs.

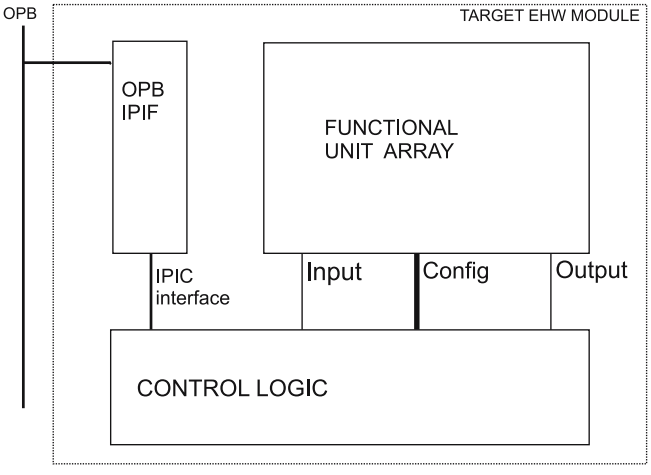


Fig. 3. The architecture of the target EHW system

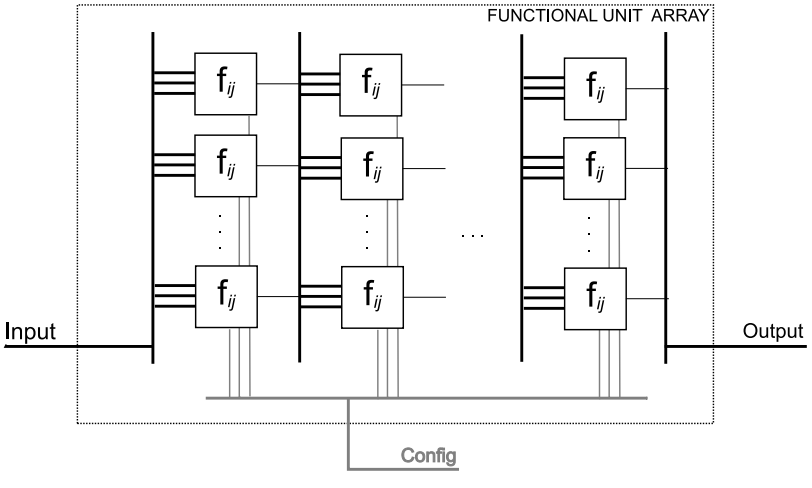


Fig. 4. The architecture of the functional unit array subsystem

As an example application, a configurable functional unit array has been implemented – see Fig. 4. Each subsystem evolved consists of a fixed-size array of functional units. The array consists of n unit layers from input to output.

Each unit's three inputs in layer l are connected to the outputs of three units in layer $l-1$. Each of the input signals can be inverted. Each unit can have one of four functions: *BUF*, *MUX*, *AND*, or *XOR*. The function of each unit and its three inputs are configurable and determined by evolution. The encoding of each functional unit in the genome string is as follows, in the case of 4 different functions for each unit and 8 units in each layer:

Function (2 bit)	Input 1 (4 bit)	Input 2 (4 bit)	Input 3 (4 bit)

Of the 4 bits for each input, one bit is toggling an inversion of the input signal, while the 3 others code for which output from the previous layer to use. For our array consisting of 6 layers with 8 units, the genome string length becomes $(2+(3\times 4))\times 6\times 8 = 672$ bit long. 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 training vector is evaluated at a time.

3.4 GA Parameters and Fitness Function

For the evolution, a population size of 20 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 0.5, thus the cloning rate is 0.5. 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
$p(n)$	$\frac{1}{10}$	$\frac{6}{10}$	$\frac{2}{10}$	$\frac{1}{10}$

The fitness function is computed in the following way:

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

For each output the computed output y is compared to the target d. If these are equal then 1 is added to the fitness function. The function sum these values for the assigned outputs (outp) for the assigned truth table vectors (vec).

4 Results

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

4.1 Device Utilization and Clock Speed

Table 1 shows the amount of logic used for our target EHW module containing an 8×6 functional unit array. A maximum of 20% of the FPGA's total resources are used. The total resource usage for the system, including bus structure and peripherals, is 43%. This

Table 1. Device utilization for the EHW module

Resource	Used	Available	Percent
Slices	1025	4928	20
Slice Flip Flops	896	9856	9
4 input LUTs	1231	9856	12

indicates that there is more space for either more complex EHW modules, or a larger number of them. The FPGA used for these experiments is also relatively small compared to the larger Virtex-II Pro and Virtex-4 FX devices - devices with up to 142,000 logic cells are available at the time of writing.

The maximum clock frequencies currently attained are 200MHz for the PPC core, and 50MHz for the rest of the system, including both PLB and OPB modules. The maximum possible speed for the PPC is stated to be 300MHz, and 100MHz for the rest of the system. Since no analysis has currently been done to localize bottlenecks, it should be possible to increase these frequencies later.

4.2 Evolution Speed

Evolution runs were conducted on our on-chip evolution system and a Pentium 4 (P4) workstation for speed comparisons. The P4 workstation has a clock frequency of 2GHz. For the speed test, 10,000 generations of 20 individuals were evolved. The fitness evaluation was for a 2×2 bit multiplier, thus 16 input/output vectors were used.

Raw GA Execution Speed. The execution time of only the GA operation without fitness evaluation was measured. The results are shown in the first row of table 2. As can be seen, the P4 system outperforms the on-chip system. This was expected, as the P4 processor is running at a much higher clock frequency and operates with a more efficient memory interface and caches. However, although the clock frequency of the P4 is 10 times greater than on the PPC, the evolution speed is only around 6.4 times greater. This may be explained by processor architecture factors, such as the high number of pipeline stages on the P4, in effect giving a lower instructions per clock cyle count.

Table 2. Evolution speeds on PPC and P4 systems

Configuration	PPC	P4
GA without fitness evaluation	8.3s	1.3s
GA with fitness evaluation	20.0s	8.6s
Fitness time in % of total	59	85

GA with Fitness Evaluation Speed. The execution time of the GA operation including fitness evaluation was then measured. The phenotype is evaluated in hardware on the on-chip system and simulated in software on the P4 for comparison.

The results are shown in the second row of table 2. Here, the P4 system is still faster than the on-chip system, but this time only by a factor of 2.3. The hardware evaluation is advantageous to the on-chip system, but it is still limited by time-consuming fitness evaluation administration in software. An estimate of the time for a complete hardware fitness evaluation is around 9.7s, but this number would increase if the number of training vectors becomes higher.

The on-chip system will be little affected by increased complexity in the phenotype structure, whereas the simulated fitness evaluation will be more time-consuming.

4.3 Circuits Evolved

Correct 2×2 bit multiplier circuits were evolved after an average of 5702 generations over 10 evolution runs. The same experiment was conducted as a verification on the PC platform, where the average was 5649. The different numbers can be explained by the different programs using different random number generators. Still, the results are rather similar, which indicates that the FPGA implementation works correctly.

4.4 Discussion

The system architecture should be analyzed more thoroughly in order to obtain higher clock frequencies. A higher bus speed would be beneficial for the configuration phase of the target EHW, especially as the genomes get large. This could be combined with moving the target EHW module to the PLB bus, in order to get a wider datapath. A way of bursting data could also be explored. A direct connection to the other side of the PPC's data BRAM would also be possible, since the BRAM is dual-ported.

To speed up fitness evaluation, certain software operations can be moved into hardware. For digital circuits with defined training vectors, like the multiplier circuits, it would be advantageous to make a system that feeds the EHW circuit with one training vector per clock cycle. As the functional unit array is pipelined, the number of cycles needed for one complete fitness evaluation of an individual would be roughly equal to the number of training vectors. Another option would be to increase the number of EHW units on the same chip. However, an increased degree of hardware specialization may come at the prize of reduced flexibility and, naturally, a higher implementation effort.

We have presented the result of our first initial experiments on the evolutionary platform. Our motivation for implementing evolution on the PPC is mainly that we would like to apply the platform for evolving systems for real-world applications in the future. A large part of the total computation time is used for the fitness computation even for evolving the small (2 bit) multiplier in a relatively small circuit above. Thus, we expect that for more complex problems with much data to measure fitness on, the amount of time used for evolution compared to fitness computation will be small. On the other hand, to experiment with several evolutionary and bio-inspired methods, flexibility would be important. This is obtainable with our platform with the processor containing the parts not critical on computation time. This would also be important with our goal of designing a scalable system with incremental evolution.

5 Conclusions

The paper has presented an approach for evolving digital circuits in a new technology – System-On-Chip by FPGA. The work is focused on demonstrating the potential of running evolution on an embedded processor in an FPGA. The first experiments – by measuring performance of the real device, are promising. As this technology progresses, it will probably be an interesting platform for cost effective evolution in embedded systems.

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. D. Goldberg. *Genetic Algorithms in search, optimization, and machine learning*. Addison–Wesley, 1989.
2. I. Kajitani et al. A myoelectric controlled prosthetic hand with an evolvable hardware lsi chip. *Technology and Disability*, 15(2):129–143, 2003.
3. 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.
4. T. Kalganova. Bidirectional incremental evolution in extrinsic evolvable hardware. In J. Lohn et al., editor, *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 65–74. IEEE Computer Society, Silicon Valley, USA, July 2000.
5. J.F. Miller, D. Job, and V.K. Vassilev. Principles in the evolutionary design of digital circuits – Part I. *Journal of Genetic Programming and Evolvable Machines*, 1(1):8–35, 2000.
6. 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.
7. 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.
8. T. Schmitz et al. Speeding up hardware evolution: A coprocessor for evolutionary algorithms. In P. Hadddow 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 274–285. Springer-Verlag, 2003.
9. L. Sekanina and S. Friedl. On routine implementation of virtual evolvable devices using combo6. In *Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware*, pages 63–70. IEEE, 2004.
10. 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.
11. J. Torresen. Evolving multiplier circuits by training set and training vector partitioning. In P. Hadddow 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. Gunnar Tufte and Pauline C. Haddow. Prototyping a ga pipeline for complete hardware evolution. In *1st NASA / DoD Workshop on Evolvable Hardware (EH '99)*, pages 18–25, 1999.
13. D. Job V. Vassilev and J. Miller. Towards the automatic design of more efficient digital circuits. In J. Lohn et al., editor, *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 151–160. IEEE Computer Society, Silicon Valley, USA, July 2000.
14. Xilinx Inc. *PowerPC 405 Processor Block Reference Guide*, August 2004.
15. Xilinx Inc. *Embedded System Tools Reference Manual*, February 2005.
16. Xilinx Inc. *Processor IP Reference Guide*, February 2005.

An Evolvable Image Filter: Experimental Evaluation of a Complete Hardware Implementation in FPGA

Tomáš Martínek and Lukáš Sekanina

Faculty of Information Technology, Brno University of Technology,
Božetěchova 2, 612 66 Brno, Czech Republic
{martinto, sekanina}@fit.vutbr.cz

Abstract. This paper describes an evolvable image filter which is completely implemented in a field programmable gate array. The proposed system is able to evolve an image filter in a few seconds if corrupted and original images are supplied by user. The architecture is generic and can easily be modified to realize other evolvable systems. COMBO6 card with Xilinx Virtex xc2v3000 FPGA is used as a target platform.

1 Introduction

As image processing deals with large data sets, a hardware implementation of image processing algorithms becomes unavoidable in many applications to ensure reasonable processing time. Furthermore, efficient image processing algorithms require a certain level of intelligence to correctly interpret and present the input data. An adaptation is required in many cases. Hence image processing in general, and image filtering in particular, belong to most popular applications of evolvable hardware [4]. Evolvable hardware can be utilized either to find the required solution at the design time (i.e. to assist the designer during the design process) or to ensure (perhaps real-time) adaptation of hardware at the runtime. The both approaches have been reported in literature (see [1, 9, 13, 17]).

The objective of this paper is to explore the performance of an evolvable image filter that is *completely* implemented in a Field Programmable Gate Array (FPGA). In order to perform these investigations, a smoothing filter (whose function can be evolved) was implemented in FPGA. The main feature of the proposed implementation is that everything is implemented in a cutting-edge reconfigurable hardware platform available today. For the presented experiments we utilized the PCI COMBO6 card developed in the Liberouter project [7]. Therefore, our results should indicate what is possible to do with such the FPGA-based evolvable systems nowadays. Evolutionary algorithm, implemented in hardware, is used to find the filter which minimizes the difference between the corrupted image and training image. These images are stored in RAM memories available on COMBO6. A personal computer is used only for communication with the COMBO6 card, i.e. for writing/reading the images to/from RAMs etc. The performed experiments should allow us to exactly determine the adaptation time

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 76–85, 2005.

and the quality of the evolved filters and, consequently, to specify the class of applications in which the filters could be evolved in real-time. We evaluated various variants of the evolvable filter, including the size of the virtual reconfigurable circuit and the parameters of the evolutionary algorithm. A crucial feature of the proposed architecture is that all EA operations as well as reconfiguration are completely overlapped by evaluation of candidate circuits, i.e. they are for free.

A lot of work has been done in the area of image filter evolution. Section 2 briefly introduces the field and emphasizes the differences between our approach and the existing approaches. In Section 3 the proposed complete hardware implementation is described. Section 4 summarizes the obtained results. Advantages and disadvantages of the evolvable system and potential applications are discussed in Section 5. Conclusions are given in Section 6.

2 Evolution of Image Filters in FPGA: A Survey

Various approaches have been proposed to the evolutionary design of image operators (filters). The authors of paper [5] evolved circuits for edge detection using elementary binary operations supported in FPGAs. Other edge detectors (also evolved in an FPGA) were represented as 2D arrays of integers that defined the convolution kernel [2]. Ebner evolved an edge detector using genetic programming which approximates the Canny edge detector [3]. An automatic feature identification algorithm that utilizes functional level operators like mean, standard deviation, convolution and linear scale was developed for multi-spectral images [9]. The evolutionary approach usually works in the time domain and produces non-linear filters. In many cases the evolved filters have exhibited better properties (the cost, quality of filtering) than conventional filters (such as median and mean filters, Sobel operators etc.) in tasks such as Gaussian or salt-and-pepper noise removal or edge detection [13, 11].

This work extends the approach initially developed by Sekanina [10, 11] who has evolved novel image filters (for 3×3 neighbourhood) using Cartesian genetic programming (CGP) applied at the functional level. Furthermore, the hardware implementation of CGP was proposed for FPGAs [13, 12]. However, in his approach image filters were evolved only by using a virtual reconfigurable circuit simulated in software that could eventually be implemented of the top of a conventional FPGA (no evolution in hardware was performed). Recently, Zhang et al. have implemented a very similar virtual reconfigurable circuit in FPGA and evolved some image operators in FPGA [17, 16]. Smith et al. used the same approach [15]. However, they have not evaluated the complete hardware implementation (including the evolutionary algorithm) in FPGA. Their papers do not indicate the time of evolution and speeding up against the software approach. It seems that some parts of evolutionary algorithm (such as circuit evaluation) have been carried out in software. As their system was proposed as asynchronous utilizing local handshaking protocols [16] they were not able to make the system completely pipelined. For similar purposes Sekanina and Friedl have proposed a completely pipelined implementation of an evolvable combinational unit for

FPGAs that can be considered as an evolvable IP core [14]. They obtained a significant speedup against the software approach. In this paper we will derive a complete hardware implementation of the evolvable filter according to the approach [14].

3 The Proposed Architecture

Architecture of the evolvable image filter is based on the component approach to evolvable hardware [13, 12, 11]. As Fig. 1 shows, it is composed of three main components—Fitness Unit, Genetic Unit and Virtual Reconfigurable Circuit.

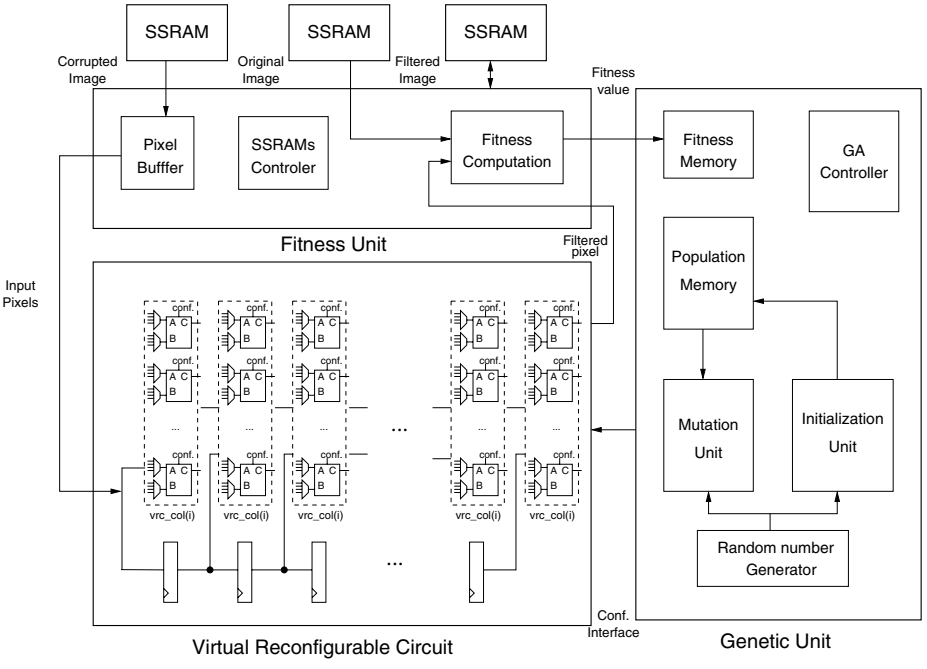


Fig. 1. Architecture of the evolvable image filter in FPGA

3.1 Virtual Reconfigurable Circuit

Every image operator will be considered as a digital circuit of nine 8bit inputs and a single 8bit output, which processes gray-scaled (8bits/pixel) images. As Fig. 2 shows every pixel value of the filtered image is calculated using a corresponding pixel and its eight neighbours in the processed image.

We approached the problem using Cartesian Genetic Programming (CGP) operating at the functional level. In contrast to the conventional CGP [8]—where gates and 1 bit connection wires are utilized—Configurable Functional Blocks

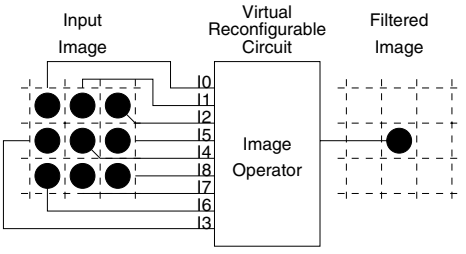


Fig. 2. Image Operator

(CFBs) and 8bit datapaths are employed [6]. Our model of the reconfigurable circuit consists of 2-input CFBs placed in a grid of n_c columns and n_r rows.

Any input of each CFB may be connected to the primary circuit inputs. Any input of each CFB may be connected to the output of a CFB, which is placed anywhere in the preceding column. The interconnection is implemented using multiplexers. Any CFB can be programmed to realize one of functions given in Table 1. These functions were recognized as useful for this task in [11]. Configuration bits of VRC are directly connected to the multiplexers that control the selection of CFB inputs and CFB functions. The reconfiguration is performed column by column. The computation is pipelined; a column represents a stage of the pipeline. Registers are inserted between columns in order to synchronize the input pixels with CFB outputs.

Table 1. Functions in CFBs

Number	Function	Description
0	$x \vee y$	binary or
1	$x \wedge y$	binary and
2	$x \oplus y$	binary xor
3	$x + y$	addition
4	$x + y^s$	addition with saturation
5	$(x + y) >> 1$	average
6	$Max(x, y)$	maximum
7	$Min(x, y)$	minimum

3.2 Fitness Unit

The design objective is to minimize the difference between the filtered image and the original image. Let u denote a corrupted image and let v denote a filtered image. The original (uncorrupted) version of u will be denoted as w. The image size is $K \times K$ (K=256) pixels but only the area of 254×254 pixels is considered because the pixel values at the borders are ignored and thus remain unfiltered. The fitness value of a candidate filter is obtained as follows: (1) the

VRC is configured using a candidate chromosome, (2) the circuit created is used to produce pixel values in the image v, and (3) the fitness value is calculated as

$$fitness = 255.(K-2)^2 - \sum_{i=1}^{K-2}\sum_{j=1}^{K-2} |v(i,j) - w(i,j)|.$$

Fitness computation is realized in Fitness Unit. The pixels of corrupted image u are loaded from external SSRAM memory and forwarded to inputs of VRC. Pixels of filtered image v are sent back to the Fitness Unit, where they are compared with the pixels of original image w. Filtered image is simultaneously stored into the additional SSRAM memory. Note that all image data are stored in external SSRAM memories due to the limited resources available in the FPGA chip.

3.3 Genetic Unit

Genetic algorithm is based only on the mutation operator (bit inversion); similarly to experiments reported in [13] a crossover is not taken into account. Population size is configurable. The new population is always generated from the best member of the previous one. Genetic algorithm operates in following steps: (1) Initialization Unit generates the first population at random (LFSR seeded from software is utilized). (2) Mutation Unit changes a given number of genes (bits) of a population member (this number is configurable) and the modified member is loaded into the VRC; it represents an image operators. (3) Genetic Unit is waiting for the evaluation performed by Fitness Unit and if the fitness value is better that the parent's fitness then the chromosome replaces its parent. (4) This is repeated until an appropriate number of generations are produced.

4 Experimental Results

4.1 Target Platform

COMBO6 developed in the Liberouter project is a PCI card primarily dedicated for a dual-stack (IPv4 and IPv6) router hardware accelerator. This board offers a very high computational power (FPGA Virtex XC2V3000 by Xilinx, Inc. with more than 3 mil. equivalent gates, up to 2GB DDR SDRAM, up to 9Mbit context addressable memory, and the three 2MB SSRAM memories) and so it is well suited for development and the use in various application domains, including evolvable hardware. We decided to use this card for our experiments because it offers us a sufficient performance and capacity of FPGA. Furthermore, the design software is available for free.

4.2 Synthesis for COMBO6

In order to compare different implementations we have decided to synthesize the whole system with VRC of size 8×4 CFBs and 8×7 CFBs. The evolutionary algorithm operates in the same way for both implementations; however, the size of

chromosome depends on the number of CFBs. The results of synthesis obtained using Leonardo Spectrum and Xilinx ISE tools are shown in the following table.

Table 2. Results of synthesis for Virtex II xc2v3000 FPGA

Resource	Avail	Used 8x4	Utilization	Used 8x7	Utilization
Function Generators	28672	10638	37.1%	18432	64.2%
Slices	14336	6175	43.0%	10042	70.0%
Dffs or Latches	30724	3172	10.3%	3668	11.9%
IOBs 256	684	236	34.0%	236	34.0%
Block RAMs	96	2	2.0%	3	3.0%

4.3 Time of Evolution

The evaluation of candidate filters consists of three basic activities: (1) preparation of a new candidate chromosome (filter), (2) reconfiguration of VRC circuit according to the prepared chromosome, and (3) evaluation of the filter. As most time is spent in filter evaluation, the architecture of evolvable image filter is designed in order to overlap the evaluation by other activities (1, 2). Therefore, because there is no overhead for reconfiguration of VRC (VRC is reconfigured at the beginning of filter evaluation) and the preparation of a new candidate circuit configuration is performed during the evaluation, it is possible to express the time of evaluation of a single filter as:

$$t_{eval} = (K-2)^2.\frac{1}{f} = (256-2)^2.\frac{1}{50.10^6} = 1.29ms$$

if the size of images is 256 x 256 pixels and the system operates at 50 MHz. Time of evolution can be expressed as follows:

$$t_e = t_{init} + g.n.t_{eval},$$

where g is the number of generations, n is population size and t_{init} is time needed to generate the initial population (t_{init} is negligible).

4.4 Functional Evaluation

The proposed evolvable image filter has been used to remove two types of noise– Gaussian noise ($\sigma = 16$) and Salt-and-Pepper noise (5% pixels with white or black shots)—that are popular in evolvable hardware literature [13, 11, 16, 15]. Original as well as filtered versions of Lena image were utilized in the fitness function. As the image is relatively large (256 x 256 pixels) we can assume that the evolved filter is general, i.e. the filter is able to remove the *same* type of noise

also from other images. Examples of filtered images and evolved filters are given in Table 5 and Figure 3.

We performed 100 runs for each problem and measured *mdpp* (mean difference per pixel) and checked the final generation. The average number of generations was calculated for the 100 runs. The evolution was stopped when no improvement in the best fitness value was detected over the last 5000 generations. Table 3 and 4 summarize the experiments. We performed the experiments with population size of four individuals and with the ratio of mutations 3 bits per chromosome; then we repeated all the experiments with mutation of 6 bits in chromosome. All the experiments were also performed for two different sizes of VRC: 8×4 and 8×7 CFBs.

Fig. 3. Evolved Salt and Pepper filter

Table 3. Results for Salt-and-Pepper noise

VRC size	Number of mutations	The best mdpp	In generation	Average mdpp	Average num. of gen.
8x4	3	0.64	4548	3.75	10475.00
8x4	6	0.51	36765	3.06	12189.00
8x7	3	0.77	8130	4.11	8617.00
8x7	6	0.48	22346	3.69	10340.00

Table 4. Results for Gaussian noise

VRC size	Number of mutations	The best MDPP	In generation	Average MDPP	Average num. of gen.
8x4	3	6.47	13767	7.98	10496.00
8x4	6	6.49	16058	7.77	10698.00
8x7	3	7.33	8435	10.10	7602.00
8x7	6	6.43	26647	8.27	8553.00

Table 5. The filter was evolved using Lena image and tested on other images

5 Discussion

We can compare the best-obtained results with the results reported in literature. In reference [11], the three best Salt-and-Pepper noise filters have *mdpp* 0.379, 0.507 and 0.656. The three best values of *mdpp* for Gaussian noise are 6.243, 6.312 and 6.326. Tables 3 and 4 show that the filters evolved here and in [11] exhibit a very similar quality. A small improvement visible in [11] is probably due the fact that some other properties (such as testability) were required for the filters in [11]; these properties were not considered in our hardware implementation. The influence of the mutation ratio and size of VRC is unclear from these experiments. Some other experiments will be arranged to clarify it.

From Tables 3 and 4 it can be derived that 9871 generations (i.e. 51 seconds at 50MHz) are needed in average to finish the evolution. The obtained time is

very reasonable if the proposed system should operate "instead" of a designer in the image filter design task. For some application, our solution could also operate as real-time evolving filter. However, if we consider that 100MHz operation frequency is easily reachable at COMBO6 and the training image could consist of 128 x 128 pixels only then the time of evolution is 6.3 second. Note that the speedup we obtained against the software approach (Pentium III/800MHz) is 50 if the FPGA operates at 100 MHz.

Our VHDL design benefits from a generic approach. All the implemented units are parameterized using various constants (such as the size of chromosome, the number of mutations, the size and topology of VRC, the size of input images etc.). Therefore, a novel FPGA-based implementation for some other evolvable systems can be obtained in a very short time. The FPGA communicates with PC via a special software allowing the designer to prepare scripts describing experiments that have to be performed. Typically, designer specifies the VRC, EA and fitness function, performs synthesis, uploads the evolvable system into FPGA and executes all experiments described in scripts. This approach can be considered as user-friendly interface to evolvable hardware.

6 Conclusions

A complete FPGA-based implementation of an evolvable image filter was realized and experimentally evaluated in an FPGA. The proposed system is able to evolve image filters in a few seconds. The architecture is generic and can easily be modified to realize other evolvable systems. Future research will be devoted to integrating the proposed solution to a real-world industrial application.

Acknowledgements

The research was performed with the financial support of the FRVS 2987/2005/-G1 project *The utilization of evolutionary algorithms for implementations of image filters in FPGAs*. Lukas Sekanina was supported from the research project of the Grant Agency of the Czech Republic under No. 102/03/P004 *Evolvable hardware based applications design methods*.

References

[1] Burian, A., Takala, J.: Evolved Gate Arrays for Image Restoration. Proc. of 2004 Congress on Evolutionary Computing, IEEE Publ. Press, 2004, p. 1185-1192

[2] Dumoulin, J. et al.: Special Purpose Image Convolution with Evolvable Hardware. In: Real-World Applications of Evolutionary Computing – Proc. of the 2nd Workshop on Evolutionary Computation in Image Analysis and Signal Processing. LNCS 1803, Springer, 2000, p. 1–11

[3] Ebner, M.: On the Evolution of Edge Detectors for Robot Vision Using Genetic Programming. In: SOAVE 97: Selbstorganisation von Adaptivem Verhalten, VDI Verlag 1997, p. 127–134

[4] Higuchi, T. et al.: Real-World Applications of Analog and Digital Evolvable Hardware. IEEE Trans. on Evolutionary Computation, Vol. 3, No. 3, 1999, p. 220–235
[5] Hollingworth, G., Tyrrell, A., Smith, S.: Simulation of Evolvable Hardware to Solve Low Level Image Processing Tasks. In: Proc. of the Evolutionary Image Analysis, Signal Processing and Telecommunications Workshop. LNCS 1596, Springer, 1999, p. 46–58
[6] Murakawa, M. et al.: Evolvable Hardware at Function Level. In: Proc. of the Parallel Problem Solving from Nature PPSN IV, LNCS 1141, Springer-Verlag Berlin, 1996, p. 62–72
[7] Liberouter project. www.liberouter.org
[8] Miller, J., Job, D., Vassilev, V.: Principles in the Evolutionary Design of Digital Circuits – Part I. Genetic Programming and Evolvable Machines, Vol. 1, No. 1, 2000, p. 8–35
[9] Porter, R.: Evolution on FPGAs for Feature Extraction. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2001, p. 229
[10] Sekanina, L.: Image Filter Design with Evolvable Hardware. In: Applications of Evolutionary Computing – Proc. of the 4th Workshop on Evolutionary Computation in Image Analysis and Signal Processing EvoIASP'02, LNCS 2279, Springer-Verlag, Berlin, 2002, p. 255–266
[11] Sekanina, L., Ruzicka, R.: Easily Testable Image Operators: The Class of Circuits Where Evolution Beats Engineers. In: Proc. of the 2003 NASA/DoD Conference on Evolvable Hardware, USA, IEEE Computer Society Press, 2003, p. 135-144
[12] Sekanina, L.: Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware. In: Proc. of the 5th International Conference Evolvable Systems: From Biology to Hardware, ICES 2003, Trondheim, Norway, LNCS 2606, Springer-Verlag, 2003, p. 186-197
[13] Sekanina, L.: Evolvable components: From Theory to Hardware Implementations, Springer-Verlag, Natural Computing Series, 2003
[14] Sekanina, L., Friedl, S.: On Routine Implementation of Virtual Evolvable Devices Using COMBO6. In: Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, USA, IEEE Computer Society Press, 2004, p. 63-70
[15] Smith, S., Leggett, S., Tyrrell, A.: An Implicit Context Representation for Evolving Image Processing Filters. In: Applications of Evolutionary Computing, LNCS 3449, Berlin, Springer Verlag, 2005, p. 407-416
[16] Zhang, Y., Smith, S., Tyrrell, A.: Intrinsic Evolvable Hardware in Digital Filter Design. In: Applications of Evolutionary Computing, Berlin, DE, Springer, LNCS 3005, 2004, p. 389-398
[17] Zhang, Y., Smith, S., Tyrrell, A.: Digital Circuit Design Using Intrinsic Evolvable Hardware. In Proc of the 2004 NASA/DoD Conference on Evolvable Hardware. Seattle, USA, IEEE CS Press, 2004, p. 55-62

Operational Amplifiers: An Example for Multi-objective Optimization on an Analog Evolvable Hardware Platform

Martin Trefzer, Jörg Langeheine, Karlheinz Meier, and Johannes Schemmel

Ruprecht-Karls-University of Heidelberg, Kirchhoff Institute for Physics,
Im Neuenheimer Feld 227, 69120 Heidelberg, Germany
+49 (0)6221 54-9838
martin.trefzer@kip.uni-heidelberg.de
http://www.kip.uni-heidelberg.de/vision/projects/eh/

Abstract. This work tackles the problem of synthesizing transferable and reusable operational amplifiers on a field programmable transistor array: the Heidelberg FPTA. A multi-objective evolutionary algorithm is developed, in order to be able to include various specifications of an operational amplifier into the process of circuit synthesis. Additionally, the presented algorithm is designed to preserve the diversity within the population troughout evolution and is therefore able to efficiently explore the design space. Furthermore, the evolved circuits are proven to work on the chip as well as in simulation outside the FPTA. Schematics of good solutions are presented and their characteristics are compared to those of basic manually created reference designs.

1 Introduction

Analog circuit development is a discipline of electronic design that demands a lot of knowledge and experience as well as a considerable amount of creativity in solving diverse problems from the designer. The design of task specific operational amplifiers (OP) is an example for an exercise that has to be done by experienced designers and exactly such OPs are essential building blocks of many electronic circuits. Contrary to digital circuit design there is still a lack of supporting tools for automatic synthesis and sizing of transistor circuits.

To date, to the authors knowledge, only a few analytic solutions for analog design automation are available. Examples in which previously known topologies are tested while the sizing of the components is done by an optimization algorithm are given in [1, 2]. In a great number of approaches, the topology is also to be found automatically, therefore, developmental strategies [3,4,5,6] or heuristic interconnection of building blocks [7] are applied, in order to deal with the high complexity of amplifiers. An alternative possibility is to choose a multi-objective evolutionary algorithm [8, 9], in order to face the fact that, for the solution of almost every complex problem, numerous variables have to be taken into account for optimization. Operational amplifiers as well as other transistor circuits found to this point by means of hardware evolution in conjunction with multi-objective optimization (MO) are reported in [10, 11, 12]. Furthermore, a multi-objective approach provides the designer with a variety of choices

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 86–97, 2005.

instead of only one more or less good solution. This is a great advantage, especially in cases in which trade-offs have to be made, e.g. between gain and speed of an amplifier.

In this paper a multi-objective evolutionary algorithm, based on previous work [13] and referred to as the *MO-Turtle GA*, is presented and successfully used for the synthesis of differential amplifiers on the Heidelberg FPTA [14]. Other current results, obtained by using this FPTA, can be found in [15]. As proposed in an earlier publication [13], one of the aims is to synthesize circuits that contain only relevant components, thus, are easier to understand according to engineering criteria. The evolved circuits are proven to work on the chip as well as in simulation outside the FPTA. Schematics of good solutions are presented in this work and their characteristics are compared to those of manually created OPs. Two series of experiments are carried out using in one case a pair of PMOS transistors and in the other case a pair of NMOS transistors as input.

2 Evolvable Hardware System

The evolution system consists of three main parts: The FPTA that hosts the configurable CMOS transistor array, a controller with a PCI interface that connects the FPTA to a standard PC and the software that runs the multi-objective evolutionary algorithm and communicates with the FPTA via the controller. Thus, the experimental setup and the candidate configurations for the transistor array are generated on the PC and then transferred to the controller. Subsequently, the controller configures the FPTA and measures the output of the circuits under test. The software on the PC reads back the results and carries out the evolutionary steps. These components provide a real time test environment for the evolved circuits.

The transistor array consists of 16x16 configurable CMOS transistor cells (Fig. 1). Each cell contains a transistor that can be configured by selecting values for its width W and length L within $W = 1, 2, ..., 15\,\mu\text{m}$ and $L = 0.6, 1, 2, 4, 8\,\mu\text{m}$. The terminals (source, drain and gate) can be connected to one of the cells outside connections (N,S,W,E), vdd or gnd. Additionally, it is possible to directly connect the nodes (N,S,W,E) to each other, which provides routing capabilities. Half of the cells are designed as programmable PMOS and NMOS transistors respectively and are arranged in a checkerboard pattern. Owing to the four nodes available for routing and terminal connections, one cell mostly serves either as transistor cell or routing cell. However, both capabilities are not separated. The array is enclosed by IO cells that can apply voltages to the border cells or measure the output voltages of the evolved circuit. A detailed description of the FPTA is given in [14].

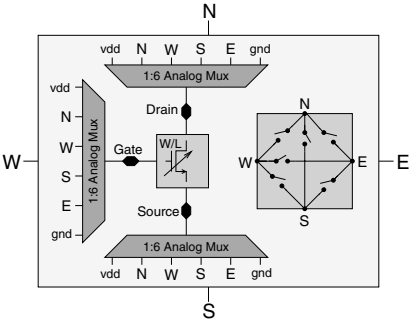


Fig. 1. The block diagram of an FPTA MOS transistor cell

3 The Multi-objective Evolutionary Algorithm

Since the evolution of operational amplifiers is a challenging task, a multi-objective strategy, first proposed in [16], is used for the experiments. This allows for a separate evaluation and optimization of different properties of the circuits, which would not be possible with a single objective algorithm. The *MO-Turtle GA* consists of a non-dominated sorting algorithm and a crowding distance measure, which are described in Sec. 3.2 and are based on those from the non-dominated sorting genetic algorithm, presented in [8, 17]. Using an MO approach offers two important advantages: First, numerous results can be harvested from the non-dominated front (NDF) instead of only one, providing trade-off solutions for the different objectives. Second, the population is of great diversity during the whole evolution, for the reason that individuals with a bad over-all performance survive as long as they are superior in at least one objective. Thus, crossover gains importance by combining differently specialized individuals.

3.1 Variation Operators of the MO-Turtle GA

The variation operators of the *Turtle GA*, reported in [13], are employed, namely the *Random Wires* mutation and the *Implanting Block of Cells* crossover. The implementation of both operators is adapted to the FPTA's architecture and described in the following. A complete description is reported in [13].

Random Wires (Mutation). The mutation operator consists of the create mode and the erase mode. The create mode connects random nodes within the FPTA's transistor array and thereby randomly inserts components into the active circuit. Contrary to that, the erase mode randomly disconnects nodes and removes transistors. The mutation operator is carried out recursively until the resulting circuit contains no dangling nodes and no floating transistor terminals. The width and length of all active transistors is mutated due to a configurable probability.

Implanting a Foreign Block of Cells (Crossover). The *implanting* crossover operator is carried out in two stages. The first stage exchanges randomly sized and positioned rectangular blocks of transistor cells between two randomly selected individuals. While the size of both blocks has to be the same for each individual, the positions of the blocks may differ. Since this operation in general breaks the layout of both previously intact circuits, the second stage fixes the occurring floating nodes by executing the random wires mutation operator for each of them. Thus, again, the resulting circuits contain no floating nodes.

3.2 Non-dominated Sorting and Crowding Distance

Non-dominated Sorting. All individuals are classified by calculating their level of non-domination, as shown in Fig. 2, due to their objective values p_i. An individual p is said to dominate q, denoted by $p \preceq q$, if and only if p is partially less than q (Eq. 1).

$$\forall i \in (1,\ldots,n),\, p_\mathrm{i} \leq q_\mathrm{i} \quad \wedge \quad \exists i \in (1,\ldots,n) : p_\mathrm{i} < q_\mathrm{i} \tag{1}$$

$$\mathrm{NDF} := \{p \in P \mid \nexists p' \in P : p' \preceq p\} \tag{2}$$

All p satisfying Eq. 1, 2 provide the first non-dom. front NDF_1. The succeeding NDFs are found by removing the individuals of NDF_k from the population $P' = P \setminus \mathrm{NDF}_k$ and by recalculating Eq. 1, 2 for the new population P' until NDF_{k+1} is empty.

Crowding Distance. The crowding distance c_{dist} is a measure for the density of solutions within the vicinity of a particular individual p within the fitness landscape (Fig. 2). All objective values are considered for calculating the quantity c_{dist} which represents an average distance to the nearest neighbors of p and is assigned to each individual of the population. Therefore c_{dist} is used to steer the evolution towards a uniform distribution of the individuals over the NDF.

3.3 Evolutionary Step

Three populations are used for evolution: A repository population *RP* and a new population *NP* of size N and an intermediate population *IP* of size $2N$. The algorithm is initialized by randomly generating individuals for *IP* and measuring their objective values. Subsequently, the evolutionary loop is started by performing non-dominated sorting and calculating crowding distances for $IP = RP \cup NP$. The next step is to refill *RP* with the best individuals of *IP* by using tournament selection with the first selection method (SM_1), described in the next subsection, on the obtained NDFs. Hereby, NDF_k is allowed to occupy at most $\frac{1}{2^k}$ of the available space in *RP*. In case the size of NDF_k is less than or equal to the available space, the whole NDF_k is copied to *RP*. Finally, a new population *NP* of size N is created from *IP* by using tournament selection with SM_2 and applying mutation and crossover.

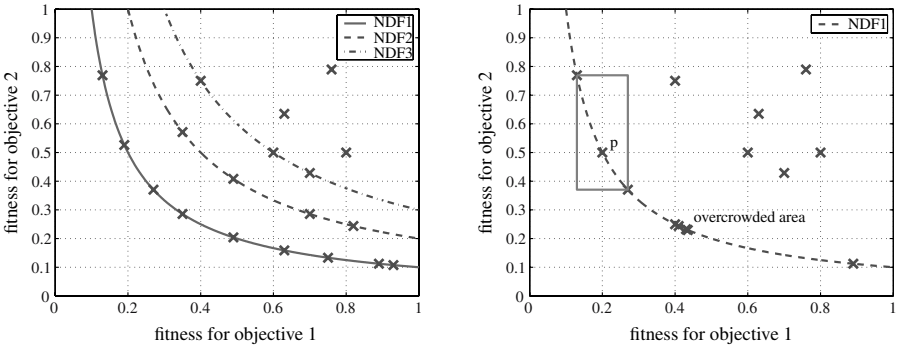


Fig. 2. *Left:* An example set of individuals—which are to be optimized for two objectives—is depicted. The first three NDFs, obtained by evaluating Eq. 1, 2, are drawn in. It is expected that the NDFs propagate towards better fitness values throughout evolution. Additionally, the rank of the NDF is equal to the level of non-domination for each individual of the respective NDF. *Right:* In this example, the individuals are not distributed uniformly over the NDF. Therefore, in order to be able to drive evolution towards such a uniform distribution, a partial order of the individuals within an NDF is defined by the crowding-distance c_{dist}. The value of c_{dist} for an example individual p is derived from the distance to the next neighbors of p.

Table 1. An overview of all test-modes (TM) and their corresponding objectives. The aim is to minimize the fitness; thus, in the cases where the objective value is to be maximized, the reciprocal or absolute value is used as fitness. *Pull to rails* is chosen as main objective, for the reason that it judges a fundamental behavior of an amplifier and the fitness-value improves smoothly.

TM objective	fitness	description
TM_1 pull to rails	min.	$(V_{tar} - V_{out})^2$ (main objective)
TM_1 DC offset	min.	sum of DC offsets of the set of curves
TM_1 dev. of DC offset	min.	standard deviation of the DC offsets
TM_2 slew-rate	max. (use recip.)	sum of slew-rates of all steps
TM_2 settling-time	min.	time when V_{out} settles within $\pm 10\%$ of V_{tar}
TM_2 deviation from V_{tar}	min.	$(V_{tar} - V_{out})^2$
TM_3 magnitude	max. (use abs.)	damping of the fundamental frequency at unity gain
TM_3 harmonic distortion	min.	sum of ampl. of harmonics if above $-60dB$
TM_4 phase-shift	min.	phase-shift of sin between V_{out} and V_{I+}
TM_4 sin-curve deviation	min.	$(V_{tar} - V_{out})^2$
— resource consumption	min.	sum of used transistors

3.4 Tournament Selection Schemes

Tournament selection with a tournament size of 2 is used as selection scheme. The selection mechanism (SM) for creating the repository is slightly different from that for creating the new population. In the first case (SM_1), the decision which competitor wins is simply based on the comparison between the individuals' level of non-domination and crowding distance c_{dist} (Cond. 3 is true), whereas in the second case (SM_2) it is additionally based on a randomly selected objective and on the main objective (Tab. 1) (more than one of the Cond. 3-5 are true).

These two kinds of tournament selection provide on the one hand high diversity within the repository population by making pure pareto-decisions (SM_1) and, on the other hand, drive evolution to improve single objectives and the main objective (SM_2).

$$p, q \in P : p \preceq q \ \vee \ (p = q \ \wedge \ c_{\text{dist}}(p) > c_{\text{dist}}(q)) \tag{3}$$

$$\text{Fitness}(p_{\text{main-objective}}) < \text{Fitness}(q_{\text{main-objective}}) \tag{4}$$

$$\text{Fitness}(p_{\text{random-objective}}) < \text{Fitness}(q_{\text{random-objective}}) \tag{5}$$

4 Experimental Setup

The experiments are run at a generation size of 200 for *IP* and a number of 4000 generations per evolution run. Individuals are mutated with a probability of 0.6 and crossover is carried out with a probability of 0.4 and a maximum block-size of 4×4 transistor cells. An area of 9×9 transistor cells is provided to the evolving circuit. Both, the non-inverting (I_+) and the inverting (I_-) input of the circuit are statically connected to the gate of a transistor of the same flavor, in order to avoid meaningless amplifiers. Two series of experiments, each of 20 evolution runs, are carried out using PMOS input in the first case and NMOS input in the second case. Free resources of the transistor array are

used to attach a randomly (by mutation) variable capacitive load to the circuits output and to implement two test benches for the circuit under test: One for open loop testing and another one with full feedback to the inverting input. Thus, a gain of 1 is assumed for the latter. Since the feedback is realized using only the configuration capabilities of the transistor array—where no constant resistors, capacities or current sources are available—it is not feasible to measure properties like gain or common-mode rejection ratio (CMRR) directly on the chip. Nevertheless it is possible to measure and evaluate important properties of an amplifier, namely open-loop behavior, slew-rate, settling-time, DC offset, harmonic distortion and phase-shift, directly on the FPTA.

4.1 Test Modes for the Measurements on the FPTA

Three kinds of test-modes (TM_i) have been developed to perform these measurements delivering a total of 11 objective values listed in Tab. 1.

TM_1: Open-Loop Behavior, Offset. The task is to pull V_{out} to $V_{tar} = 5V$ if $V_{I+} > V_{I-}$ and to $V_{tar} = 0V$ if $V_{I+} < V_{I-}$ and to keep the offset voltage V_{os} low or at least constant. A set of nine curves at $V_{I+} = 1.5, 1.75, \ldots, 3.5V$, each consisting of 100 randomly applied sample voltages for $V_{I-} = 0 \ldots 5V$, is used as test pattern. TM_1 delivers fitness values for three objectives, namely *pull to rails*, *DC offset* and *deviation of DC offset*.

TM_2: Slew-Rate, Settling-Time. The challenge for the output is to follow two voltage-steps from $V_{I+} = 1.5V$ to $2.5V$ and from $V_{I+} = 2.5V$ to $3.5V$ in $t_{step} = 0.25\,\mu$s. Fitness values for the *slew-rate* and the *settling-time* are calculated from the period of time between the step and the point of time when V_{out} has settled at the new target voltage $V_{tar} \equiv V_{I+}$. An additional objective is given by the *deviation of V_{tar} from V_{out}*.

TM_3 & TM_4: Magnitude, Phase-Shift, Harmonic Distortion. A further demand on an OP is to distort and damp the input signal as less as possible and to keep the phase-shift constant below $180\,^\circ$ in order to cause the amplifier to remain stable. These properties are measured in TM_3 by applying three different sinusoidal signals with $f = 5, 50$ and $500\,$kHz to the input and comparing them to the circuits output $V_{tar} \equiv V_{I+}$. A discrete fourier transform is used to calculate the power spectrum of the output signal for each frequency. Subsequently, fitness values for *magnitude* and *total harmonic distortion (THD)* are calculated from the power spectrum. Additionally, the output of a sinusoidal input signal of $f = 20\,$kHz is used in TM_4 to obtain values for the *phase-shift* and the *deviation of V_{I+} from V_{out}*.

4.2 Simulation Setup

The simulations are carried out with the SPICE3 simulator described in [18]. BSIM3v2 transistor models are used for simulation. SPICE netlists are extracted from the circuits that have been evolved on the transistor array by using the *MO-Turtle GA*. The input voltage patterns correspond to those used for the on-chip measurements. A load-capacity of $10\,$pF is attached to the circuits' output in simulation. Fitness values, calculated from the simulation results, are obtained by using the same fitness functions as throughout evolution.

5 Results

All evolution runs ended up in similar regions of fitness, although the overall performance of the circuits is slightly better for those with NMOS input than for those with PMOS input, as can be seen from Tab. 2. For all evolved circuits the simulation results are worse than those obtained from the chip and about half of them did not work at all outside the FPTA. Nevertheless, each evolutionary run features a significant amount of individuals performing at least similar in simulation and on the transistor array. Example NDFs for the resulting circuits are depicted in Fig. 3 and 4.

5.1 Performance of the Multi-objective Approach

An example of how the NDF develops throughout evolution is depicted in Fig. 3. For some objectives (e.g. *magnitude*, *offset*) the NDF converges towards better fitness values

Table 2. The no. of runs that contain at least one individual that achieved a better (or not more then 10% worse) fitness value than the manually made circuits for a given no. of objectives. In all cases the manually made OPs obtained better fitness values for *distortion* and *resource consumption* than the evolved circuits. The reason for this is the placement and routing of the evolved solutions which often contain longer wires and therefore produce more noise.

		in no. of objectives										
		1	2	3	4	5	6	7	8	9	10	11
no. of NMOS runs	better than ref.	20	20	18	5	1	0	0	0	0	0	0
	max. 10% worse than ref.	20	20	18	5	5	2	1	1	0	0	0
no. of PMOS runs	better than ref.	20	20	18	4	2	1	0	0	0	0	0
	max. 10% worse than ref.	20	20	19	6	5	3	1	1	1	0	0

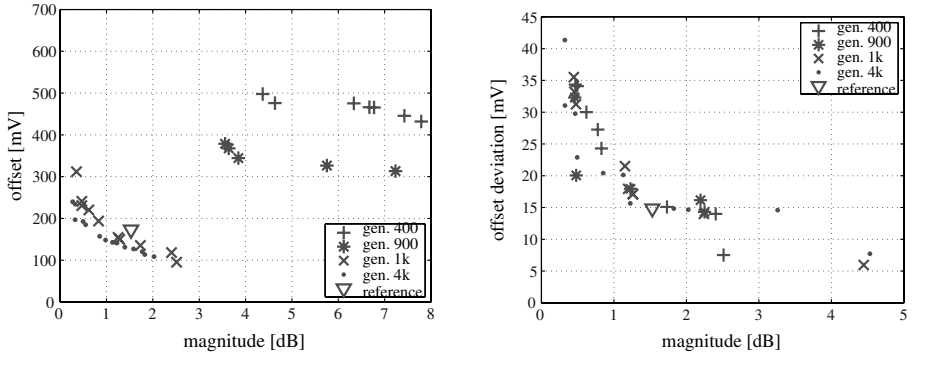


Fig. 3. An example run (NMOS input) with good performance is chosen and the depicted NDFs are recalculated by considering only the two objectives shown in the respective plot for illustration. The position of a manually made OP (reference), described in Sect. 5.2, is marked by a triangle. *Left:* The NDF for *offset* over *magnitude* converges towards better fitness over time. *Right:* In contrast to this, the NDF for *dev. of offset* over *magnitude* is spread over wide ranges of fitness.

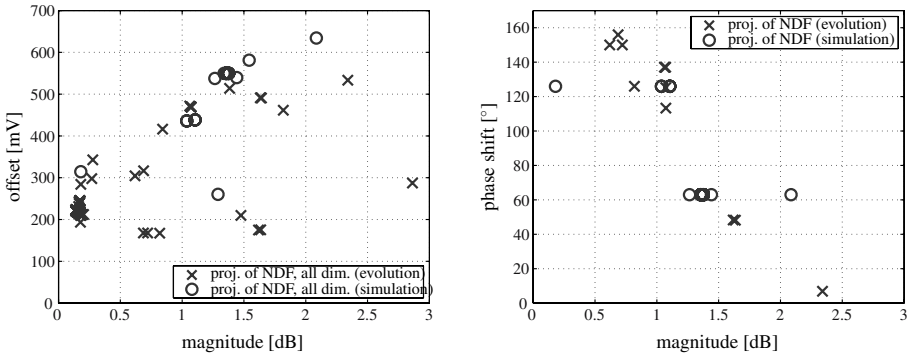


Fig. 4. The NDFs of a run (NMOS input) with good performance, obtained from the measuring on the FPTA and from the simulation, are depicted above. The graphs show a multi-dimensional projection of the NDF into the plane spanned by the respective objectives. *Left: Offset* over *magnitude*. *Right: phase-shift* over *magnitude*.

over time, as can be nicely seen from Fig. 3 (left). Other objectives (e.g. *magnitude*, *offset-deviation*) show a different behavior where the front as a whole does not further converge, but is spread over wide ranges of fitness. An example for the latter is shown in Fig. 3 (right). Additionally, the position of a manually created design, described in Sec. 5, within the objective space is marked by a triangle.

Projections of the whole NDF into the plane spanned by the respective objectives—taking all objectives into account for computation—are graphed in Fig. 4. This illustrates nicely the complexity of the NDF troughout the optimization process. After all, the main benefit of using an MO approach for the evolution of operational amplifiers on the Heidelberg FPTA is the possibility to efficiently explore the search space taking care of both, the diversity of the population and the various demands on the target circuit.

5.2 Solutions for the Operational Amplifier

The FPTA is configured with manually created circuits, one with PMOS and one with NMOS input respectively, in order to be able to assess the quality of the synthesized circuits compared to human-made solutions. Each of the references consists of a differential input stage and a simple inverter-output stage. The fitness values are measured for both reference designs, using exactly the same setup as throughout evolution, and are compared to those of the evolved circuits. As can be seen from Tab. 2, almost each run contains at least one individual that outperforms the corresponding reference OP in up to 3 objectives and about 5 runs feature similar performance in up to 5 objectives. In all cases the manually made OPs obtained better fitness values for *distortion* (noise) and *resource consumption* than the evolved circuits. The reason for this is the placement and routing of the evolved solutions which often contain longer wires and therefore produce more noise.

Opposite to the competition with the reference circuits on the FPTA, the evolved circuits come off worse if typical characteristics of OPs are compared in simulation. As can be seen from Tab. 3 especially those properties that cannot be measured directly on

Table 3. Comparison between characteristics of evolved circuits with a good performance and the reference circuits (NMOS and PMOS input). The values are obtained from SPICE simulations.

parameter	NMOS (evo)	NMOS (ref)	PMOS (evo)	PMOS (ref)
open-loop gain	33 dB	57 dB	29 dB	65 dB
0dB bandwidth	13 MHz	77 MHz	6 MHz	33 MHz
offset	−80 mV	28 mV	230 mV	20 mV
slew-rate (+)	40 $\frac{\mathrm{V}}{\mu\mathrm{s}}$	100 $\frac{\mathrm{V}}{\mu\mathrm{s}}$	15 $\frac{\mathrm{V}}{\mu\mathrm{s}}$	25 $\frac{\mathrm{V}}{\mu\mathrm{s}}$
slew-rate (-)	15 $\frac{\mathrm{V}}{\mu\mathrm{s}}$	30 $\frac{\mathrm{V}}{\mu\mathrm{s}}$	35 $\frac{\mathrm{V}}{\mu\mathrm{s}}$	45 $\frac{\mathrm{V}}{\mu\mathrm{s}}$
settling-time	0.4 μs	0.2 μs	0.3 μs	0.2 μs
phase-margin	91 °	50 °	92 °	50 °
common mode rejection	30 dB	> 40 dB	20 dB	> 40 dB
out voltage swing	2.2 V	4.8 V	2.8 V	4.8 V
input common mode range	2.5 V	4.2 V	3.5 V	4.3 V

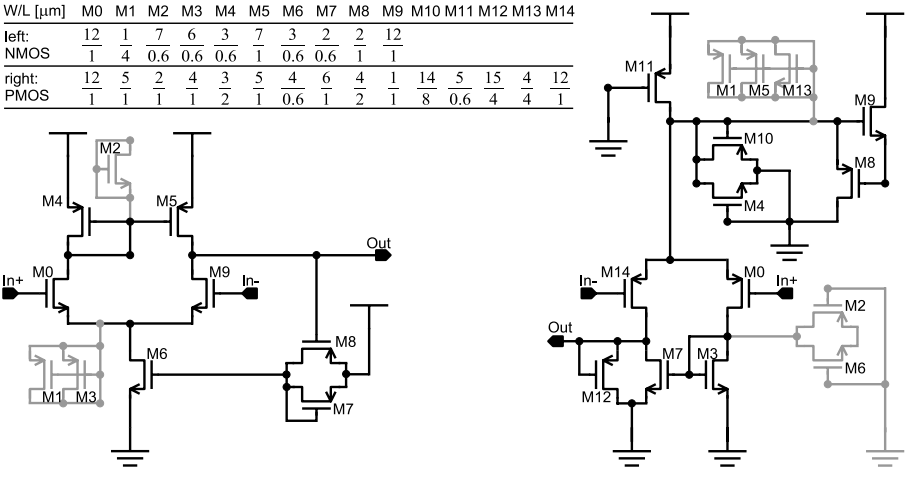


Fig. 5. Schematics of the evolved circuits; shorted transistors are grayed. *Left*: NMOS input transistors. *Right*: PMOS input transistors. In both cases the *MO-Turtle GA* achieved to synthesize differential input stages and some kind of biasing circuitry. The evolved solutions thus far lack of an output gain-stage.

the transistor array during evolution—thus, cannot be evaluated by a fitness function (e.g. open-loop gain)—return rather poor results. Contrary to that, the characteristics that are represented by an objective perform similar, e.g. *offset*, *slew-rate* and *settling-time*. Since the output voltage swing and the 0dB bandwidth are correlated to a good open-loop gain, those values are also not as good as those of the manually made OPs.

In both cases the phase-margin of the evolved solution is higher than those of the reference OPs. This is interesting insofar, that it is on the one hand a very good result, since the aim of the corresponding objective is to minimize the phase-shift. On the other hand, forcing the phase-shift towards zero could possibly thwart the evolution of output gain-stages. If this is the case, it would be better to allow for a certain phase-

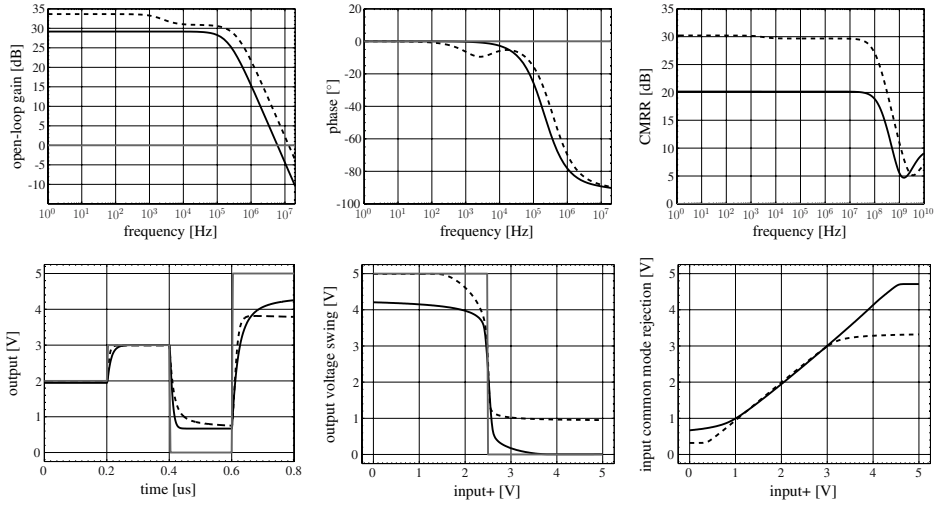


Fig. 6. The graphs above show characteristics of evolved operational amplifiers obtained from spice simulation. For illustration, evolved circuits with PMOS (—) and NMOS (- - -) input respectively and featuring good performance are chosen.

margin in the objective function. Hence, this could be the reason why in both examples depicted in Fig. 5—which represent evolved circuits with good performance—the algorithm was able to synthesize clearly recognizable differential input stages as well as biasing circuitry, but failed in appending a simple inverter, which would provide significantly better performance. Finally, some important characteristics of the evolved circuits are shown in Fig. 6.

6 Conclusion and Outlook

The main achievement of the presented method is that reusable and substrate-independent circuits are evolved successfully and human-understandable schematics of good solutions can be drawn. Hence, it is possible to analyze the resulting circuits and to investigate how the algorithm is solving problems on the hardware substrate. As an example, it has been shown that the presented algorithm is able to synthesize operational amplifiers on the Heidelberg FPTA. The fact that the evolution of OPs is a difficult task suggests that the *MO-Turtle GA* can be applied to a variety of problems.

The resulting circuits are extracted into netlists and simulated outside the substrate on which they were evolved. About 50% of the outcome is performing equally well on the chip and in simulation and can therefore be transferred to other technologies. The presented multi-objective approach allows for considering various objectives during evolution. Thus, it is possible to efficiently explore the design space and converge to regions of fitness comparable to those which are obtained by basic human reference designs measured on the chip. Unfortunately, the algorithm failed in synthesizing additional gain-stages. The reason for this is probably the lack of a suitable gain test bench

due to the fact that even well approved human designs do not achieve significantly better fitness. In this case it is more likely that the abilities of the FPTA limit the search for good solutions than the algorithm itself. This indeed, will only be solved by a second generation FPTA.

Future work will be done to understand to what extent the architecture of the transistor array influences the performance of the algorithm and what can be done to improve it. Furthermore, the *MO-Turtle GA* will be enhanced to allow the creation and deletion of structures like differential pairs or inverters in one step. Hereby, all transistors of those structures could be marked for a simultaneous W/L mutation.

Acknowledgment

This work is supported by the Ministerium für Wissenschaft, Forschung und Kunst, Baden-Württemberg, Stuttgart, Germany.

References

1. Hershenson, M., Boyd, S., Lee, T.H.: Optimal design of a cmos op-amp via geometric programming. In: IEEE Transactions on Computer-Aided Design. (2001) 1–21
2. Arpad Buermen, Janez Puhan, T.T.: Robust design and optimization of operating amplifiers. (2003) 745–750
3. Vieira, P.F., Botelho, L.B., Mesquita, A.: Evolutionary synthesis of analog circuits using only mos transistors. In Zebulum, Ricardo S., Gwaltney, David, Hornby, Gregory, Keymeulen, Didier Lohn, Jason, and Stoica, Adrian, ed.: Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware, Los Alamitos, IEEE Computer Society Press (2004) 38–45
4. Sripramong, T., Toumazou, C.: The invention of cmos amplifiers using genetic programming and current-flow analysis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **21** (2002) 1237–1252
5. Koza, J.R., Bennett III, F.H., Andre, D., Keane, M.A.: Design of a high-gain operational amplifier and other circuits by means of genetic programming. In In Angeline, Peter J., Reynolds, Robert G., McDonnell, John R., and Eberhart, Russ, ed.: Evolutionary Programming VI. 6th International Conference, EP97, Proceedings. Volume 1213 of Lecture Notes in Computer Science., Indianapolis, Indiana, USA, Springer-Verlag, Berlin (1997) 125–136
6. Koza, J.R., Bennett III, F.H., Andre, D., Keane, M.A.: Evolution using genetic programming of a low-distortion 96 decibel operational amplifier. In: Proceedings of the 1997 ACM Symposium on Applied Computing, San Jose, California, USA, New York: Association for Computing Machinery (1997) 207–216
7. Kruiskamp, W., Leenaerts, D.: Darwin: Cmos opamp synthesis by means of a genetic algorithm. In: DAC '95: Proceedings of the 32nd ACM/IEEE Conference on Design Automation, New York, NY, USA, ACM Press (1995) 433–438
8. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: Nsga-ii. In: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature. (2000) 849–858
9. Coello Coello, C.A., Van Veldhuizen, D.A., Lamont, G.B.: Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers, New York (2002)

10. Hernandez Aguirre, A., Zebulum, R.S., Coello Coello, C.A.: Evolutionary multiobjective design targeting a field programmable transistor array. In Zebulum, Ricardo S., Gwaltney, David, Hornby, Gregory, Keymeulen, Didier Lohn, Jason, and Stoica, Adrian, ed.: Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware, Los Alamitos, IEEE Computer Society Press (2004) 199–205
11. Zebulum, R., Pacheco, M., Vellasco, M.: A novel multi-objective optimization methodology applied to the synthesis of cmos operational amplifiers. Journal of Solid-State Devices and Circuits (2000) 10–15
12. Zebulum, R.S., Pacheco, M.A., Vellasco, M.: A multi-objective optimisation methodology applied to the synthesis of low-power operational amplifiers. In Cheuri, I.J., dos Reis Filho, C.A., eds.: Proceedings of the XIII International Conference in Microelectronics and Packaging. Volume 1., Curitiba, Brazil (1998) 264–271
13. Trefzer, M., Langeheine, J., Schemmel, J., Meier, K.: New genetic operators to facilitate understanding of evolved transistor circuits. In Zebulum, R.S., Gwaltney, D., Hornby, G., Keymeulen, D., Lohn, J., Stoica, A., eds.: Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware, Los Alamitos, IEEE Computer Society Press (2004) 217–224
14. Langeheine, J., Becker, J., Fölling, S., Meier, K., Schemmel, J.: A CMOS FPTA chip for intrinsic hardware evolution of analog electronic circuits. In: Proc. of the Third NASA/DOD Workshop on Evolvable Hardware, Long Beach, CA, USA, IEEE Computer Society Press (2001) 172–175
15. Garvie, M.: Reliable Electronics through Artificial Evolution. PhD thesis, University of Sussex (2004)
16. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley (1989)
17. Deb, K., Goel, T.: Controlled Elitist Non-dominated Sorting Genetic Algorithms for Better Convergence. In Zitzler, E., Deb, K., Thiele, L., Coello, C.A.C., Corne, D., eds.: First International Conference on Evolutionary Multi-Criterion Optimization, Springer-Verlag. Lecture Notes in Computer Science No. 1993 (2001) 67–81
18. Quarles, T., Newton, A., Pederson, D., Sangiovanni-Vincentelli, A.: SPICE3 Version 3f3 User s Manual. Department of Electrical Engineering and Computer Sciences, University of California Berkeley, Ca., 94720. (1993)

Intrinsic Evolution of Controllable Oscillators in FPTA-2

Lukáš Sekanina and Ricardo S. Zebulum

[1] Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66 Brno, Czech Republic
[2] Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA
sekanina@fit.vutbr.cz, rzebulum@mail2.jpl.nasa.gov

Abstract. Simple one- and two-bit controllable oscillators were intrinsically evolved using only four cells of Field Programmable Transistor Array (FPTA-2). These oscillators can produce different oscillations for different setting of control signals. Therefore, they could be used, in principle, to compose complex networks of oscillators that could exhibit rich dynamical behavior in order to perform a computation or to model a desired system.

1 Introduction

The conventional design of analog as well as digital oscillators is difficult since it requires a lot of experience. Designers must guarantee that their oscillators meet the specifications in terms of the frequency of oscillations, amplitude, phase, shape of signal, sufficient power and some other properties. Oscillators are also usually very sensitive to the environment (temperature, electromagnetic field, etc.) in which they operate. In the recent years various EA-based approaches have been proposed to design the oscillators automatically [1,3,8]. Oscillators were evolved at the opamp, transistor and gate levels. In general, the results show that evolution of oscillators with required properties is difficult.

Oscillators do play an important role not only in the area of electronic circuits. Oscillatory networks have been studied as information processors by many researchers because they can be constructed from realistic nonlinear dynamical systems and are biologically plausible (furthermore, for example, cellular neural networks or spiking neural networks have practical applications).

Networks of oscillators can be identified in neural systems or genetic regulatory networks. Recently, a synthetic network capable of producing sustained oscillations in protein concentrations was presented [2]. The "repressilator" consisted of three genes (for simplicity, called a, b, c), expressing three proteins (respectively, A, B, C). The network formed a ring: Protein A repressed transcription of gene b; B repressed c; and C repressed a. For certain biochemical parameters, this cyclic repression produced self-sustained roughly sinusoidal oscillations over the entire growth phase of the host *Escherichia coli* cells. In another work, a model for controlling a synthetic gene network of coupled oscillators was presented [11]. Unlike the repressilator, the oscillator consisted of

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 98–107, 2005.

only two genes (x and y) and was of the relaxation type. Both proteins were under the control of a promoter that was activated by the protein X, and protein Y was a protease of X. Oscillations arose because Y degrades X and thus reduces its own expression level (because X activates transcription of y). Neural oscillators inspired by *olfactory cortex* models were investigated in [13]. They can be utilized as a dynamical context addressable memory [7] or to perform logic computation in which synchronized oscillations are considered as logic 1 and desynchronized oscillations as logic 0. Logic gates AND, NOR and NXOR were implemented by means of these networks [13].

Networks of oscillators can be composed of a controllable oscillator as a building block, i.e. of an oscillator whose output can be controlled using the input signals enabling or disabling oscillations. These signals are taken from the outputs of other oscillators in the network. The first step to build networks of oscillators is creating the controllable oscillators. Therefore, the objective of this paper is to explore whether controllable oscillators can be evolved intrinsically in a physical platform reconfigurable at the transistor level. We decided to utilize the transistor level because we assume that more various and richer dynamic behavior can be obtained than at the gate level. In next step of research the evolved controllable oscillators will be connected in oscillator networks. As we are not primarily interested in the frequency of oscillations, we propose a simple fitness function operating in the time domain. In this work the controllable oscillators are evolved directly in the Field Programmable Transistor Array (FPTA-2). The oscillators have one or two digital control inputs and produce various oscillations for different input stimuli.

In practice, the networks of oscillators could perform useful parallel asynchronous computation in the way similar to cellular automata, for example, in signal processing tasks. Having inspiration in the mentioned genetic regulatory networks, the evolved networks of oscillators could implement non-trivial genotype-phenotype mappings useful for embryonic electronics [5,9]. Furthermore, in addition to traditional models of genetic control networks developed by Kauffman and others [4], the system could be used to model and study natural gene regulations (see the evolution of limit cycle dynamics in electronic models in [10]).

The paper is organized as follows. Section 2 briefly introduces the area of evolutionary design of oscillators. In Section 3 FPTA chip and SABLES system are described. The proposed evolutionary design method is formulated in Section 4. While Section 5 summarizes the obtained results, Section 6 discusses them. Conclusions are given in Section 7.

2 Evolutionary Design of Electronic Oscillators

Oscillators are difficult to design manually. Hence the evolutionary approach was utilized to perform this task. Oscillators are usually evolved in the way similar to other analog circuits evolution [15]. However, the evolutionary approach does not work as well as in case of other analog circuits (e.g. filters). That is also

demonstrated in Koza's list of human-competitive results that does not contain any oscillator circuits; on the other hand it contains about 20 analog circuits [6]. The construction of fitness function is very important especially in case of evolution of oscillators. The analysis of circuit behavior performed in the fitness function can be based on various principles: time domain analysis, frequency domain analysis or transfer function analysis. Corresponding fitness landscapes are usually extremely rugged; oscillations appear only in a very specific parts of the search space.

Huelsbergen et al. evolved oscillators (astable multivibrators) from primitive logic components in Xilinx XC6216 FPGA [3]. They reported results of in Silico oscillator evolution for ten target frequencies in three cell-array sizes (6x8, 8x8, and 16x16). Considering all three cell-array sizes, the system discovered relatively accurate oscillators – over 97% of their pulses correct – for five of the ten frequencies and required only a small number of GA runs. In fitness function, the output signal was compared against a binary string containing the required combinations of 0s and 1s; thus the number of missed pulses could be calculated. It was not at all understood how the evolved circuits function. For example, relative to the speed of the FPGA's gates (nanosecond transition times), the evolved oscillators are of rather low frequency.

Aggarwal has used genetic algorithm to evolve opamp-based sinusoidal oscillators [1]. His algorithm looks for a suitable passive network (consisting of a given number of resistors and capacitors) connected to a single opamp. In fitness function a symbolic analysis was used to find out the transfer function which contains specific expressions indicating oscillations. It was found that the GA rediscovered all the twelve canonic single opamp-based topologies. Some new interesting opamp-based topologies of oscillators were also discovered.

Field programmable analog array MPAA020 of Motorola was utilized to evolve opamp-based oscillators [14]. The fitness function tried to maximize the voltage difference between samples of the outputs at specified time points. The evolved circuit generated a close-to-perfect square wave of 3 Volts amplitude and frequency of 200 kHz.

Layzell and Thompson evolved oscillators in Evolvable Motherboard at the transistor level [8]. The circuit population was rich on oscillator circuits and GA was used to optimize the frequency – measured directly in the fitness calculation process.

Except Aggarwal's results (who has worked at symbolic level), the aim of the mentioned approaches was to demonstrate that oscillators can be evolved in the given target platform. The evolved oscillators were not used in any application. No other types of evolved oscillators, such as controllable oscillators or voltage-controlled oscillators have been reported in literature.

3 Evolvable Platform: FPTA-2 and SABLES

A complete stand-alone board-level evolvable system (SABLES) is built by integrating the FPTA and a DSP implementing the Evolutionary design algorithm

[12]. The system is connected to the PC only for the purpose of receiving specifications and communicating back the result of evolution for analysis. The system fits in a box 8“ x 8“ x 3“. Communication between DSP and FPTA is very fast with a 32-bit bus operating at 7.5MHz. The evaluation time depends on the tests performed on the circuit. Many of the tests attempted here require less than two milliseconds per individual, and runs of populations of 100 individuals from 100 to 200 generations require only 20 seconds.

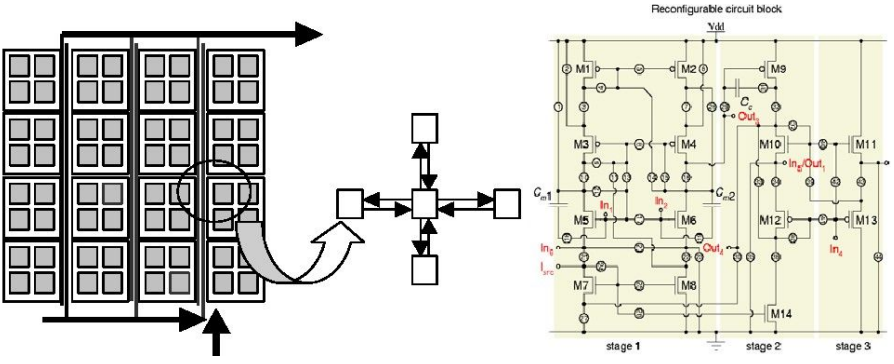


Fig. 1. FPTA-2 architecture (left) and schematic of cell transistor array (right). The cell contains additional capacitors and programmable resistors (not shown).

The FPTA is an evolution-oriented reconfigurable architecture (EORA). It has a configurable granularity at the transistor level. It can map analog, digital and mixed signal circuits. The architecture of the FPTA consists of an 8x8 array of re-configurable cells. Each cell has a transistor array as well as a set of programmable resources, including programmable resistors and static capacitors. Figure 1 provides a broad view of the chip architecture together with a detailed view of the reconfigurable transistor array cell. The reconfigurable circuitry consists of 14 transistors connected through 44 switches. A total of 5000 bits is used to program the whole chip. The pattern of interconnection between cells is similar to the one used in commercial FPGAs: each cell interconnects with its north, south, east and west neighbors. The reader can refer to [12] for more information on the FPTA-2.

4 Design Method

The controllable oscillators will be designed using a standard genetic algorithm operating directly with configurations of FPTA-2 as chromosomes. Only a few cells of the FPTA will be utilized for the experiments. Figure 2 shows the cells and the connection of input and output signals. No external components (such as RC circuits) were considered for these experiments. The frequency of oscillations

depends only on the configuration and internal characteristics (such as delay of transistors) of FPTA-2.

The genetic algorithm running in a DSP uses the roulette-wheel selection, crossover and mutation. Candidate solutions are evaluated directly in FPTA-2. In this process, all possible combinations of logic values over the input control signals (a and b) are applied at the circuit inputs and oscillations are detected at the output y. The genetic algorithm must promote the chromosomes that cause oscillations if they are required and keep the output invariable otherwise. In particular 240 values are sampled, digitized and utilized during the evaluation of a candidate circuit. Because of simplicity we decided to evaluate candidate circuits in the time domain. Oscillators controlled using a single input signal $a[i]$ are designed using the fitness function whose basic structure is given in the following pseudo-code:

Algorithm 1:

```
i = 0; fitness = 0;
while (i < samples)
{
  // oscillations
  ones = 0; zeroes = 0; penalty = 0;
  while (i < samples and a[i] is High)
  {
      if (y[i] < LL) zeroes = zeroes + 1;
      else if (y[i] > HL) ones = ones + 1;
      else penalty = penalty + 1;
  }
  fitness = fitness + k1 * abs(ones – zeroes) + kp * penalty;

  // no oscillations
  ones = 0; zeroes = 0; penalty = 0;
  while (i < samples and a[i] is Low)
  {
      if (y[i] < LL) zeroes = zeroes + 1;
      else if (y[i] > HL) ones = ones + 1;
      else penalty = penalty + 1;
  }
  fitness = fitness + k2 * (zeroes + ones – abs(zeroes + ones)) + kp * penalty;
}
```

If $a[i]$ is at log. 1 (High), the circuit should oscillate; otherwise, the circuit should not. Here, $i = 1 \ldots 240$ samples are evaluated at the circuit output $y[i]$. The *zeroes* counter indicates the number of output values that are considered as lower than a given threshold value LL ($LL = 0.45MV$ where MV determines the maximum output voltage 1.8V). The *ones* counter indicates the number of output values that are considered as higher than a threshold value HL ($HL = 0.55MV$). Note that, here, the fitness should be minimized. The situation in

which the circuit should oscillate (i.e. the number of *zeroes* and *ones* is similar but non-zero) is evaluated in the first nested while loop. The second nested loop deals with the situation in which the output should not oscillate. *Penalty* counter is used to avoid staying in the middle of MV range. The values of constants k_1, k_2 and k_p are determined experimentally, and $k_p \gg k_1 = k_2$. A very similar fitness function has been utilized to design oscillators controlled using two bits.

5 Experimental Results

If a single cell of FPTA is configured as an inverter and its output is connected to its input then oscillations are always observable. We utilized this property in our approach. Figure 2 shows the experimental setup used to evolve controllable oscillators using four and five FPTA-2 cells. The solid lines in Fig. 2 denote external physical connections (wires) used to connect the cells. These connections were utilized to promote a specific design pattern which is typical for the conventional oscillators composed of three inverters. In addition to these connections, the evolution could interconnect the cells using the internal switches of the FPTA-2. Behavior of a cell is defined using 77 configuration bits. However, three words (48 bits) are not evolved for the cells that belong to the cells that are connected in a ring; indeed, they are taken from the configuration bitstream of a conventional inverter and used during all experiments. This strategy is applied in order to obtain some oscillations in a shorter time. We know that conventional oscillators can be designed in this way. In fact we were not able to evolve any oscillators without this setup. Parameters of GA are as follows: the population size = 100, the crossover probability = 70%, and the mutation probability = 10%. Depending on experiment 300-1000 generations were produced.

5.1 One-Bit Controllable Oscillators

Various one-bit controllable oscillators were evolved using the setup from Fig 2A. Figure 3 shows typical oscillations we obtained (the frequency of oscillations is 90.9kHz). Similar other oscillators we evolved that operate at the following frequencies: 41.6kHz, 22.7kHz, 83.3kHz, and 38.5 kHz. The shape of the output signal is usually very close to the sine wave; however, with some distortions. We also attempted to change the frequency of oscillations by means of increasing

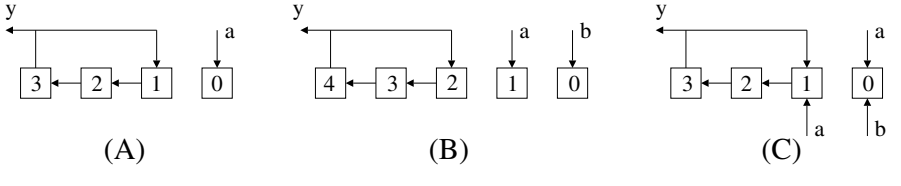


Fig. 2. Cells used and their connection. a and b are control signals; y is the output signal.

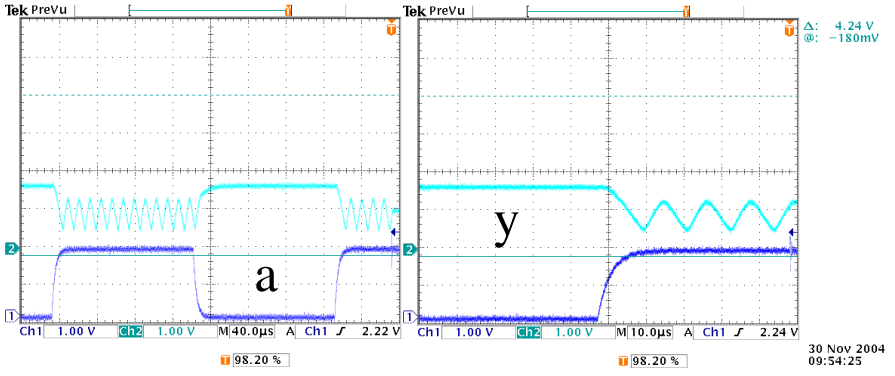


Fig. 3. Evolved 1-bit controllable oscillator (f = 90.9kHz)

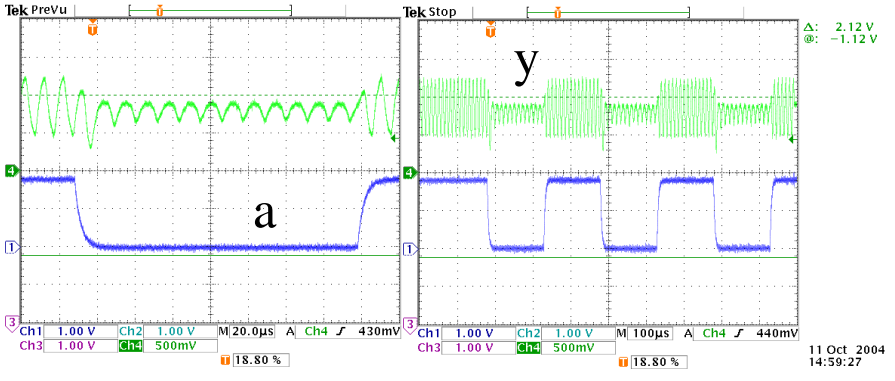


Fig. 4. Evolved 1-bit controllable oscillator (f = 83.3 kHz for both waves)

voltage at the control input. However, we were not able to evolve such a kind of controllable oscillators. In another setup, a circuit producing two types of oscillations was evolved (Figure 4). In order to obtain this result, we required in the fitness functions that $ones = 2 * zeroes$ when the control signal is at logic 0.

5.2 Two-Bit Controllable Oscillators

The two-bit controllable oscillators utilize two input signals, a and b, to control the oscillations. As shown in Fig. 2B, they consist of five cells. The oscillations, controlled through cells 0 and 1, should emerge in cells 2, 3 and 4. The proposed fitness function has been modified in order to consider all four combinations over the inputs a and b. For instance, we required to have oscillations only when $a = b = 1$. Figure 5 shows a typical behavior we obtained. Let us define the following logic interpretation of that behavior. Let oscillations mean logic 1 and let no oscillations mean logic 0. Then the evolved circuit whose behavior

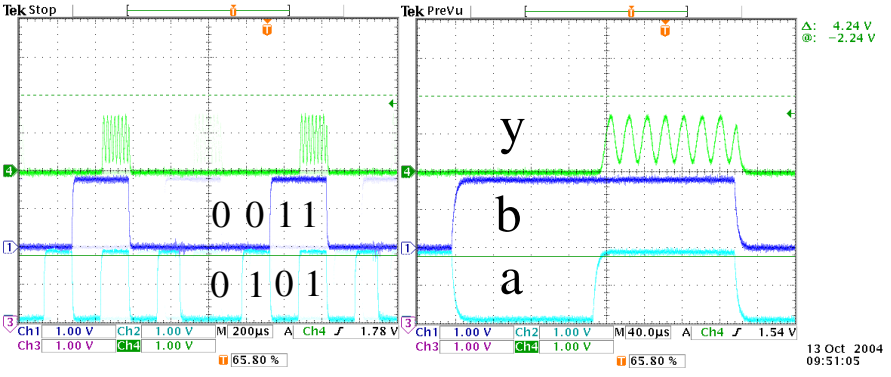


Fig. 5. Evolved 2-bit controllable oscillator operating as AND (f = 50 kHz)

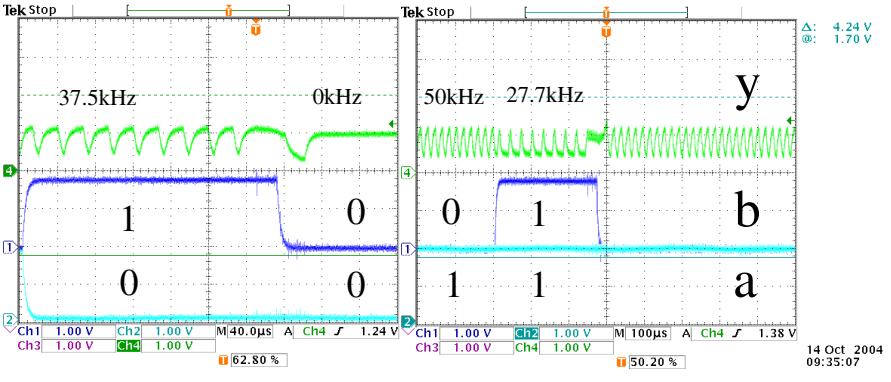


Fig. 6. Evolved 2-bit controllable oscillator generating four different behaviors

is depicted in Fig. 5 can be understood as logic function AND. Considering this interpretation we were able to evolved various other logic functions, and surprisingly, we also evolved exclusive-or (XOR) function.

In another experiment we evolved a circuit that exhibits four different behaviors for four different combinations of the control inputs. It generates a signal of frequency 27.7kHz for $a = 1$ and $b = 1$, 50kHz for $a = 1$ and $b = 0$, 35.7kHz for $a = 0$ and $b = 1$ and no oscillations for $a = 0$ and $b = 0$ (see Fig. 6).

6 Discussion

The presented work has addressed the question whether the evolutionary approach is able to discover controllable oscillators at the transistor level. The answer is positive, i.e. the transistors available for the evolutionary design can be composed together by means of an automated evolutionary process in order to establish one- and two-bit controllable oscillators. The search was not

performed completely from scratch. We promoted some "ring"-based structures and partially preconfigured the cells in the ring. No oscillations have appeared in case of a complete evolution from scratch. On the other hand no information showing a way how to stop/enable oscillations was provided for the evolution. Therefore, the evolutionary approach really discovered how to create controllable oscillators. It is interesting that we were able to repeat almost all experiments reported in Section 5.2 also using only four cells of FPTA-2. The setup is shown in Fig. 2C.

The success of evolution also depends on values of coefficient k_1, k_2 and k_p. If the penalty for oscillations is too high, no oscillating candidate circuits are visible. If the penalty for no oscillations is too high, the population contains many oscillators; however, it is impossible to control the oscillations via the input control signals. Looking for suitable values of these coefficients is a very time consuming experimental work requiring tens of runs of the GA. Once the values of coefficients are fixed, a 1-bit controllable oscillator is usually found in approximately 30% of runs and 2-bit controllable oscillator in 10% of runs.

The main disadvantage of the proposed fitness function is that it is difficult to specify the frequency of oscillations and shape of the wave. The time domain analysis allowed us to specify only the required number of values higher or lower than a given threshold value. More sophisticated search for a given frequency of oscillations (e.g. a multiobjective method) would probably require the analysis in the frequency domain which, however, requires more computational effort. On the other hand the oscillators in network have not to work at a predefined frequency. They can operate at different frequencies that are suitable for a given platform.

7 Conclusions

Simple one- and two-bit controllable oscillators were intrinsically evolved using only four cells at the transistor level directly in FPTA-2. We can control the oscillations using logic signals which in principle allows us to build networks of oscillators. The question for future research is whether the output oscillations are able to control other oscillators in order to connect them into a complex network.

Acknowledgments

The research described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology and was sponsored by the National Aeronautics and Space Administration (NASA). Lukas Sekanina was supported by the Fulbright scholarship and from the research project of the Grant Agency of the Czech Republic under No. 102/04/0737 *Modern methods of digital system synthesis.*

References

1. Aggarwal, V.: Evolving Sinusoidal Oscillators Using Genetic Algorithms. In Proc. of the 2003 NASA/DoD Conference on Evolvable Hardware, Chicago, USA, IEEE Computer Society, 2003, p. 67–76
2. Elowitz, M. B., Leibler, S.: A Synthetic Oscillatory Network of Transcriptional Regulators. Nature (London) Vol. 403, 2000, p. 335–338
3. Huelsbergen, L., Rietman, E., Slous, R.: Evolving Oscillators in Silico. IEEE Trans. on Evolutionary Computation. Vol. 1, No. 3, 1999, p. 197–204
4. Kauffman, S. A.: The Origins of Order: Self Organization and Selection in Evolution. Oxford University Press, 1993
5. Koopman, A.: Hardware-Friendly Genetic Regulatory Networks in POEtic Tissues. MSc Thesis. Utrecht University, 2004, 75 p.
6. Koza, J.: 36 Human-Competitive Results Produced by Genetic Programming. http://www.genetic-programming.com/humancompetitive.html
7. Kozma, R., Freeman, W.: Chaotic Resonance-Methods and Applications for Robust Classification of Noisy and Variable Patterns. Neural Networks. Vol. 4, No. 1., 2001, p. 103–121
8. Layzell, P., Thompson, A.: Understanding Inherent Qualities of Evolved Circuits: Evolutionary History as a Predictor of Fault Tolerance. In Proc. of the 3rd Evolvable Systems: From Biology to Hardware Conference, LNCS 1801, Springer-Verlag, 2000, p. 133–142
9. Mange, D. et al.: Towards Robust Integrated Circuits: The Embryonics Approach. Proc. of IEEE. Vol. 88, No. 4, 2000, p. 516–541
10. Mason, J., Linsay, P. S., Glass, L.: Evolving Complex Dynamics in Electronic Models of Genetic Networks. Chaos, Vol. 14, No. 3, 2004, p. 707–715
11. McMillen, D. et al.: Synchronizing Genetic Relaxation Oscillators by Intercell Signaling. Proc. of the National Academy of Sciences of the USA, Vol. 99, No. 2, 2002, p. 679–684
12. Stoica, A. et al.: Evolving Circuits in Seconds: Experiments with a Stand-Alone Board Level Evolvable System. In Proc. of the 2002 NASA/DoD Conference on Evolvable Hardware, Alexandria Virginia, USA, IEEE Computer Society, 2002, p. 67–74
13. Xu, D., Principe, J. C., Harris, J. G.: Logic Computation Using Coupled Neural Oscillators. In Proc. of IEEE Intl. Symposium on Circuits and Systems, 2004, p. 788–791
14. Zebulum, R. S., Pacheco, M., Vellasco, M.: Analog Circuits Evolution in Extrinsic and Intrinsic Mode. In Proc. of the 2nd Evolvable Systems: From Biology to Hardware Conference, LNCS 1478, Springer-Verlag, 1998, p. 154–165
15. Zebulum, R., Pacheco, M., Vellasco, M.: Evolutionary Electronics – Automatic Design of Electronic Circuits and Systems by Genetic Algorithms. CRC Press, Boca Raton 2002

The Role of Non-linearity for Evolved Multifunctional Robot Behavior

Martin Hülse, Steffen Wischmann, and Frank Pasemann

Fraunhofer Institute, Autonomous Intelligent Systems,
53754 Sankt Augustin, Germany
{martin.huelse, steffen.wischmann, frank.pasemann}@ais.fraunhofer.de
http://www.ais.fraunhofer.de/~INDY

Abstract. In this paper the role of non-linear control structures for the development of multifunctional robot behavior in a self-organized way is discussed. This discussion is based on experiments where combinations of two behavioral tasks are incrementally evolved. The evolutionary experiments develop recurrent neural networks of general type in a systematically way. The resulting networks are investigated according to the underlying structure-function relations. These investigations point to necessary properties providing multifunctionality, scalability, and open-ended evolutionary strategies in Evolutionary Robotics.

1 Introduction

Evolutionary robotics (ER) as the study and development of behavioral control for autonomous robots through self-organizing processes based on artificial evolution is a widely accepted approach [10,14]. With respect to natural evolution and simplest forms of natural life there are many researches criticizing the dissatisfying outcomes of current work in ER [4,5]. In [5] it is claimed that open-ended evolutionary processes are necessary to overcome crucial limitations of current ER models and to generate more complex and interesting results.

However ER models providing open-ended evolutionary processes are implemented, the agents must be incrementally evolved. With respect to behavioral control this means control structures must facilitate incremental evolution. Such an approach should also cope with the scalability problem of ER models in general [1,3,4].

The crux of incremental control structure evolution is the integration of new behavioral functionality without loosing existing capabilities. In this paper we propose an approach to make this problem more tractable. We present incrementally evolved control structures which are systematically investigate to study the underlying dynamical properties and control principles providing (1) coordination of different behavioral tasks, and (2) the development of multifunctionality. In [2] it is claimed that a serious and systematical analysis of concrete examples of evolved agents are the prerequisite for dynamical explanation and "abstracting" general principles" of situated autonomous agents. Therefore we present robotic tasks which at first might seem rather simple, but this simplicity allows

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 108–118, 2005.

minimal systems which are exactly what we need to give a detailed description of the effects of the purposed incremental evolutionary approach on the dynamical properties of already existing control structures with innate basic functionalities. Based on these results we discuss the role of non-linearity for (1) open-ended evolutionary processes and (2) the development of multifunctionality in a self-organized way.

2 Setup

We present experiments which systematically apply two methods of incremental evolution of recurrent neural networks (RNN), also referred to as neuro-modules: (1) expansion method and (2) fusion method [6]. Each method is realized by a restrictive and semi-restrictive technique. Restrictive means that neither already existing structural elements (hidden neurons and synapses) nor parameters (bias and weight terms) of the initial basic building modules can be changed. hereas, semi-restrictive means that parameters can be changed, while the structure remain fixed [6], too.

Expansion and fusion methods are realized with an evolutionary algorithm, the ENS^3-algorithm (described in [6,13]). Using a standard additive neuron model with sigmoidal transfer function $\sigma(x)$ and time discrete dynamics the ENS^3-algorithm evolves neural structures and optimizes the corresponding parameters at the same time. Besides from a task specific input-output structure, the neuron type, and the constraint that input neurons have only outgoing weights, nothing else is determined. Therefore, any kind of recurrent neural connections, like self-connections and loops can emerge during the evolutionary process.

As an incrementally evolved robot task, we chose a reactive light seeking behavior. Light seeking includes the coordination of a positive and negative tropism - phototaxis and obstacle avoidance. In the following light seeking behavior means that a robot has to follow a light source and has to stop in front of it while it is avoiding collision with any objects in its environment. For these studies the Khepera robot [9] and a 2-dimensional simulation software [8] is used. Note, that all evolution experiments and analysis are done in simulation but all resulting controllers were tested on the physical hardware as well to approve that the observed behavior in real world is qualitatively the same as in simulation.

3 Experiments

The Khepera robot is driven by two DC-motors (control signals m_l, m_r), which are able to move the left and right wheel forward (positive signals) and backward (negative signals). The sensor data of the Khepera are delivered by its eight infrared sensors. They can be executed in two modes, measuring light intensity (sensor values $l_0, l_1, \ldots l_7$) and distances to obstacles (sensor values $d_0, d_1, \ldots, d_7$). The sensor values l_n and d_n are mapped into the closed interval $[0.0; 1.0]$. For the light sensors, values $l_n = 0.0$ refers to darkness and $l_n = 1.0$ to the maximal

measurable light intensity. The proximity values d_n are zero if no obstacle is detected and value 1.0 represents a collision. In all presented experiments the sensor values d_n and l_n are summarized as follows:

$$i_1 := \frac{1}{3}(d_0 + d_1 + d_2), \qquad i_2 := \frac{1}{3}(d_2 + d_3 + d_4),$$

$$i_3 := \frac{1}{2}(l_0 + l_1), \quad i_4 := \frac{1}{2}(l_2 + l_3), \quad i_5 := \frac{1}{2}(l_4 + l_5), \quad i_6 := \frac{1}{2}(l_6 + l_7).$$

Where i_1 and i_2 represents the distance to obstacles at the robot's left and right side. The values i_3 and i_5 indicates the light intensity at the left and the right, i_4 the intensity at the front, and i_6 at the rear. According to this setup the input-output structure of the neuro-modules that were evolved for the light seeking task is represented by six input neurons $(I_1, I_2, \ldots I_6)$ and two output neurons (O_1 and O_2). The values $i_1, \ldots, i_6$ are the inputs of the corresponding input neurons $I_1, \ldots, I_6$ of the neuro-module. Since input neurons are only used as buffers the values i_n can also be seen as the outputs of the corresponding input neurons I_n. As transfer function we applied $\sigma(x) := \frac{1}{1+e^{-x}}$, the standard sigmoid. According to the problem of handicapped navigation possibilities with only positive control signals a special post-processing is implemented. We functionally decompose the two output neurons. The left output neuron O_1 controls the speed and O_2 the turning angle of the robot's movement. This is formalized as follows:

$$m_l := \lfloor (5.0 \cdot (o_1 - (2 \cdot o_2 - 1.0))) + 0.5 \rfloor,$$

$$m_r := \lfloor (5.0 \cdot (o_1 + (2 \cdot o_2 - 1.0))) + 0.5 \rfloor.$$

In such a way we get positive and negative integer values, used as motor control signals for the Khepera robot, simulated as well as real.

3.1 Basic Building Modules

For the following experiments we used two basic building modules. The neural structure of module $\mathcal{G}_{\mathcal{O}}$, solving an obstacle avoidance task, and its resulting behavior in a simulated environment is show in Fig. 1 (a). This module has an even 2-ring between O_1 and O_2. Its weight configuration is critical and therefore hysteresis effects can be expected. A robot controlled by this module generates a straight forward movement, if no obstacle is detected. The robot is able to escape from dead-ends and sharp corners.

Fig. 1 (b) shows the second basic building module $\mathcal{G}_{\mathcal{L}}$, performing a positive phototropism. Module $\mathcal{G}_{\mathcal{L}}$ is basically feedforward organized. Hence, this module can only provide fixpoint attractors. The resulting robot behavior shows a strong drive to the right, if no light is detected. The drive to the right is forced by the bias term 0.2 of O_2. The bias term causes an output value larger than 0.5 that generates a turning angle unequal zero. If the robot detects light it moves straight to it and stops in front of it.

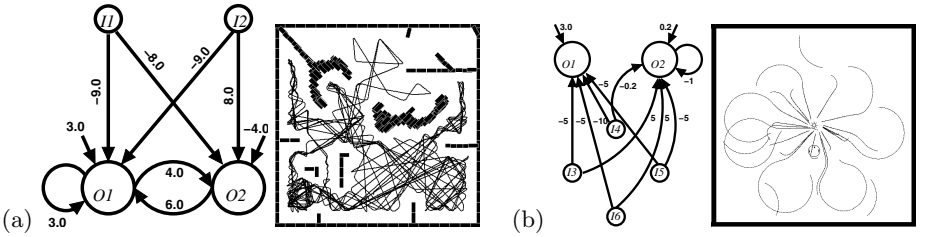


Fig. 1. (a) Neuro-module $\mathcal{G}_{\mathcal{O}}$ solving an obstacle avoidance task, (b) neuro-module $\mathcal{G}_{\mathcal{L}}$ performing a positive phototropism, and the resulting robot behavior in simulation

3.2 Expansion of a Basic Module

Two neuro-modules resulting from the expansion experiments are shown in Fig. 2. Neuro-module $\mathcal{G}_{\mathcal{O}\Rightarrow\mathcal{L}}$ is one outcome of the restrictive and $\mathcal{G}_{\mathcal{O}\rightarrow\mathcal{L}}$ one of the semi-restrictive expansion experiments.

Aside from the undercritical self-connection of O_2 the new structural elements of $\mathcal{G}_{\mathcal{O}\Rightarrow\mathcal{L}}$ are only feedforward organized (Fig 2(a)). These new connections are coming from the input neurons delivering light sensor data. With respect to the feedforward organization of the new connections one can not expect additional non-trivial dynamical effects. Robots controlled by this module show a drive to the right, if no obstacle and light is detected. If a light source is detected, the robot orients to it and comes to a halt in front of it. It has not lost its capability to escape from dead-ends (Fig 2(a)).

Considering the semi-restricted evolved neuro-module $\mathcal{G}_{\mathcal{O}\rightarrow\mathcal{L}}$ one can find only three new connections (Fig 2(b)). Like in module $\mathcal{G}_{\mathcal{O}\Rightarrow\mathcal{L}}$ all the new connections come from the input neurons delivering the light sensor data and are feedforward organized. However, the weights of the initial structure have strik-

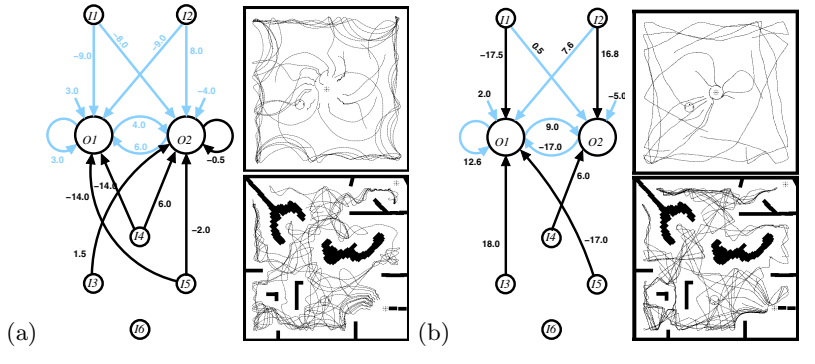


Fig. 2. Neuro-module $\mathcal{G}_{\mathcal{O}\Rightarrow\mathcal{L}}$ resulting from (a) the restrictive module expansion, (b) neuro-module $\mathcal{G}_{\mathcal{O}\rightarrow\mathcal{L}}$ resulting from the semi-restrictive module expansion, and the resulting light seeking behavior. The grey color indicates the unchanged elements.

ingly changed. The self-connection of O_1 has become critical and it is now able to generate a hysteresis effect. Furthermore the 2-ring between O_1 and O_2 has become odd. This 2-ring can generate periodic and chaotic attractors [11]. If the robot detects no obstacle and no light, its resulting behavior is characterized by irregular and slight drives to the left as well as to the right (Fig. 2(b)). The semi-restrictive evolved module $\mathcal{G}_{\mathcal{O}\rightarrow\mathcal{L}}$ can also escape from dead-ends as well as it comes to a halt in front of a light source.

3.3 Fusion of the Two Basic Modules

The initial structures of the following fusion experiments include the two modules $\mathcal{G}_{\mathcal{O}}$ and $\mathcal{G}_{\mathcal{L}}$. The output neurons of both modules become hidden neurons ($H_{1,\ldots 4}$) of the initial structure. During the evolutionary process the insertion of new connections coming from the input neurons was not allowed. This guarantees that no structural elements emerge, which could exclude the basic modules. Figure 3 shows two examples of resulting neuro-modules, restrictive ($\mathcal{G}_{\mathcal{O}\Leftrightarrow\mathcal{L}}$) and

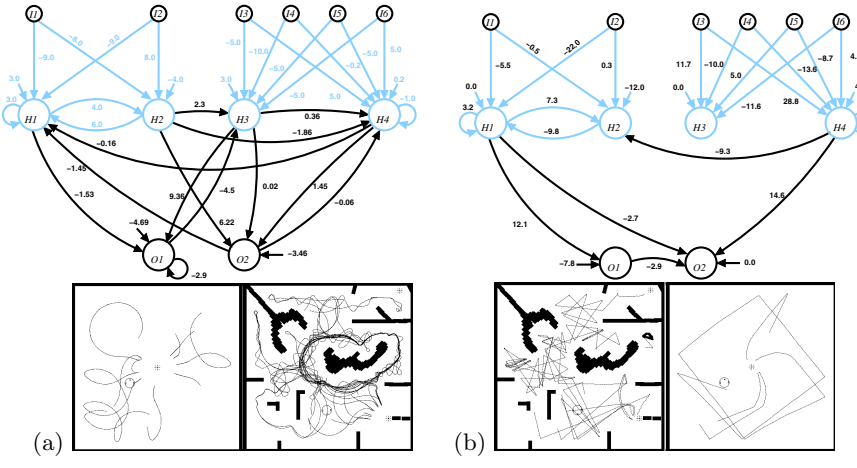


Fig. 3. (a) Neuro-module $\mathcal{G}_{\mathcal{O}\Leftrightarrow\mathcal{L}}$ resulting from the restrictive module fusion and (b) neuro-module $\mathcal{G}_{\mathcal{O}\leftrightarrow\mathcal{L}}$ resulting from the semi-restrictive module fusion and their resulting light seeking behavior. The grey color indicates the unchanged elements.

semi-restrictive ($\mathcal{G}_{\mathcal{O}\leftrightarrow\mathcal{L}}$) evolved by our fusion method. Considering the structure of module $\mathcal{G}_{\mathcal{O}\Leftrightarrow\mathcal{L}}$ (Fig. 3 (a)) the evolved coupling between the two basic modules $\mathcal{G}_{\mathcal{O}}$ and $\mathcal{G}_{\mathcal{L}}$ does not include any new hidden neurons but a lot of new synaptic connections. These connections show many recurrences, like self-connections and rings. Nevertheless, only one 2-ring (H_3 and O_1) has a critical weight parameter configuration, that provides non-trivial dynamical properties. The 2-ring between H_3 and O_1 is odd and can generate periodic as well as chaotic attractors. With respect to the resulting robot behavior (see Fig. 3 (a)) one can observe

a strong drive to the left, if no obstacle and light is detected. In the case of obstacle detection the module produces large turning angles to avoid a collision. Again, module $\mathcal{G}_{\mathcal{O}\Leftrightarrow\mathcal{L}}$ successfully produces a light seeking behavior including the escapes from dead-ends as well as a halt in front of a light source.

The semi-restrictive evolved coupling of module $\mathcal{G}_{\mathcal{O}\leftrightarrow\mathcal{L}}$ (Fig. 3 (b)) consists of only a few new connections. There are even no new recurrences. New dynamical properties originally generated by these new connections can not be expected. They can at most provide non-trivial dynamical features which are generated by the structures of the basic modules $\mathcal{G}_{\mathcal{O}}$ and $\mathcal{G}_{\mathcal{L}}$. And again, these basic modules have strikingly changed. The self-connection of hidden neuron H_4 has become critical. Hence, H_4 can generate period-2 oscillations. Similar to the semi-restrictive extended module $\mathcal{G}_{\mathcal{O}\rightarrow\mathcal{L}}$ the former even 2-ring of basic module $\mathcal{G}_{\mathcal{O}}$ has become an odd 2-ring. Therefore, this 2-ring between H_1 and H_2 can also generate periodic as well as chaotic oscillations. A robot controlled by module $\mathcal{G}_{\mathcal{O}\leftrightarrow\mathcal{L}}$ moves straight forward if no obstacle and light is detected. It also produces a halt in front of a light source. But the turning angles generated by this module during obstacle avoidance are very large. Note, that the generation of these large turning angles reduces the exploration capabilities, insofar as we understand and define well exploration by the robot's visited area.

3.4 Free and Starting from Scratch

The following two light seeking modules (Fig. 4) are evolved in such a way that either the underlying initial structures can be removed or no initial structure was given.

Neuro-module $\mathcal{G}_{\mathcal{O}-\mathcal{L}}$ (Fig. 4 (a)) is an example of a free expansion. That means, although the evolutionary process was initialized with the basic module $\mathcal{G}_{\mathcal{O}}$, the elements of this initial structure were not locked during evolution. Hence, all elements of the initial structure could be modified by the variation operator during the evolutionary process, including the deletion of initialized connections. As it can be seen in Fig. 4(a) the resulting structure is purely feedforward organized. All recurrences of the initial structure $\mathcal{G}_{\mathcal{O}}$ were removed during the evolutionary process. According to this feedforward structure the resulting robot behavior is determined only by fixpoint attractors. Nevertheless, also this simply feedforward structure enables the robot to escape from dead-ends, to stop in front of a light source and to robustly move straight forward, if no obstacle and light is detected.

The last light seeking module $\mathcal{G}_{\mathcal{O}\mathcal{L}}$ (Fig. 4 (b)) is evolved without any predefined control, because the evolutionary process was initialized with the empty initial structure. With respect to the number of synapses and hidden neurons this is the smallest control structure and also purely feedforward organized. Although its resulting behavior shows a strong drive to the left it successfully solves the light seeking task.

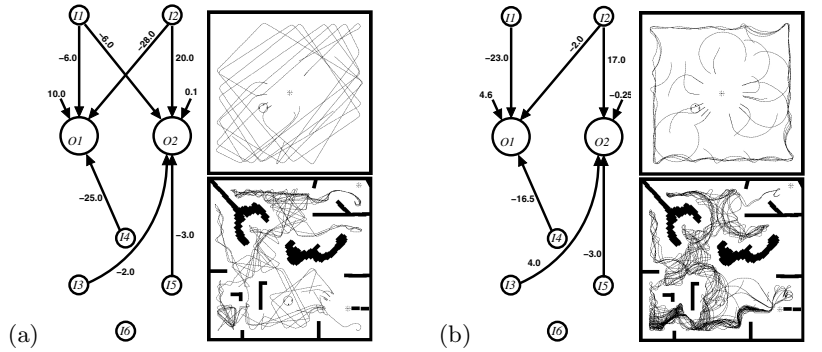


Fig. 4. (a) Neuro-module $\mathcal{G}_{\mathcal{O}-\mathcal{L}}$ resulting from the free module expansion and (b) neuro-module $\mathcal{G}_{\mathcal{OL}}$ resulting from evolution starting with an empty initial structure

4 Discussion of the Structure-Function Relations

Clarifying the relationship between evolved structure, its inherent dynamics, and the resulting robot behavior we identify five different behavioral patterns: (1) moving forward, (2) avoiding obstacles, (3) orientation to the light, (4) halt in front of an light source, and (5) solving the conflict between obstacle avoidance and approaching the light. In the following we will only focus on two behavioral patterns: orientation to the light and halt in front of a light source.

These patterns correspond to specific sensor value configurations. The halt in front of a light source is basically characterized by a high activation of all front light sensor values, while distance sensors and the light sensor at the rear have low activations ($i_{3,4,5} \uparrow i_{1,2,6} \downarrow$). The orientation to a light source can be characterized as the transition from behavioral pattern moving forward ($i_{1,2,\ldots 6} \downarrow$) to the halt in front of a light. We symbolize this transition as follows: $i_4 \uparrow i_{3,5} \updownarrow i_{1,2,6} \downarrow$.

To identify relevant attractors for specific parameter configurations the four neuro-modules were simulated as dynamical systems, de-coupled from constraints of the body and environmental interactions. Based on this simulations we have an indication which attractor generates the observed behavior patterns. The results are summarized in Table 1.

Due to its feedforward organization, the behavior relevant dynamics of neuro-modules $\mathcal{G}_{\mathcal{OL}}$ and $\mathcal{G}_{\mathcal{O}-\mathcal{L}}$ are purely based on fixpoint attractors. With respect to these attractors the two modules can be seen as a simple superposition of the two basic modules $\mathcal{G}_{\mathcal{O}}$ and $\mathcal{G}_{\mathcal{L}}$.

Contrary, all modules resulting from the expansion and fusion experiments show an increase of complexity according to the behavior relevant dynamical properties. This becomes most obvious, if the dynamical properties providing the orientation to the light are investigated. The bifurcation diagrams in Fig. 5 indicate that the dynamical features generating an orientation to the light are beyond simple fixpoint dynamics.

Table 1. Attractors of the neuro-modules under specific parameter configurations

modules	moving forward $i_n \downarrow$	obstacle avoidance $i_{1,2} \uparrow \quad i_{3,\ldots 6} \downarrow$	halt in front of a light source $i_{3,4,5} \uparrow$ $i_{1,2,6} \downarrow$	orienting to the light $i_4 \uparrow i_{3,5} \updownarrow$ $i_{1,2,6} \downarrow$
$\mathcal{G}_{\mathcal{O}}$	fixpoint	fixpoint	-	-
$\mathcal{G}_{\mathcal{L}}$	fixpoint	-	fixpoint	fixpoint
$\mathcal{G}_{\mathcal{O}\Rightarrow\mathcal{L}}$	fixpoint	fixpoint	fixpoint	hysteresis
$\mathcal{G}_{\mathcal{O}\rightarrow\mathcal{L}}$	chaotic	fixpoint	fixpoint	chaotic
$\mathcal{G}_{\mathcal{O}\Leftrightarrow\mathcal{L}}$	chaotic	chaotic	fixpoint	chaotic
$\mathcal{G}_{\mathcal{O}\leftrightarrow\mathcal{L}}$	perio-2	hysteresis and period-2	fixpoint	period-2
$\mathcal{G}_{\mathcal{O}-\mathcal{L}}$	fixpoint	fixpoint	fixpoint	fixpoint
$\mathcal{G}_{\mathcal{O}\mathcal{L}}$	fixpoint	fixpoint	fixpoint	fixpoint

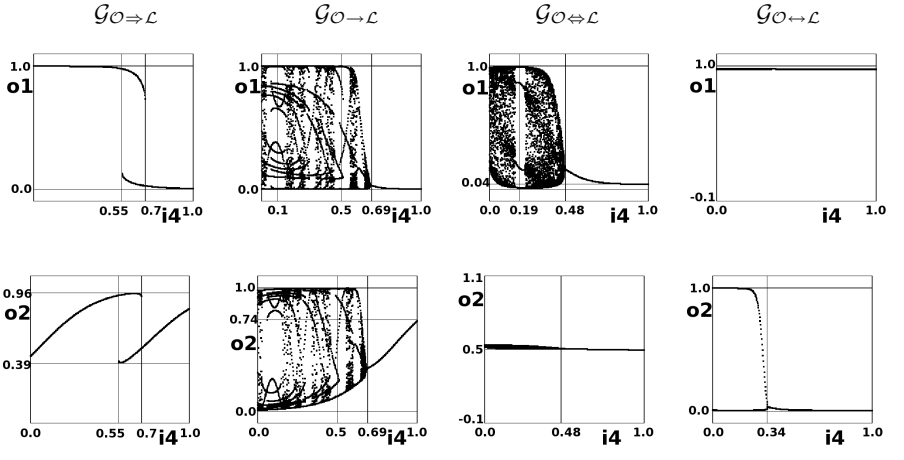


Fig. 5. Behavior relevant attractors causing a halt in front of a light source indicated by bifurcation diagrams of the four light seeking modules. **Upper:** output value o_1 over the input value i_4, other input values $i_n = 0$. **Lower:** output value o_2 over input value i_4, other input values $i_n = 0$.

For instance while a robot, controlled by neuro-module $\mathcal{G}_{\mathcal{O}\rightarrow\mathcal{L}}$ or $\mathcal{G}_{\mathcal{O}\Leftrightarrow\mathcal{L}}$, is approaching a light source the speed control is realized by chaotic attractors. The turning angle in module $\mathcal{G}_{\mathcal{O}\rightarrow\mathcal{L}}$ is also modulated by a chaotic attractor (Fig. 5).

Considering module $\mathcal{G}_{\mathcal{O}\leftrightarrow\mathcal{L}}$ we observe that orientation to a light source is provided by a period-2 attractor (Fig. 5). The period-2 oscillation creates a permanent alteration of o_2 between 0 and 1. Over time this generates an effective motor signal of 0.5, which is the mean of this stream of output values. The value 0.5 represents a turning value of zero and the robot moves straight due to $o_1 \approx 1.0$.

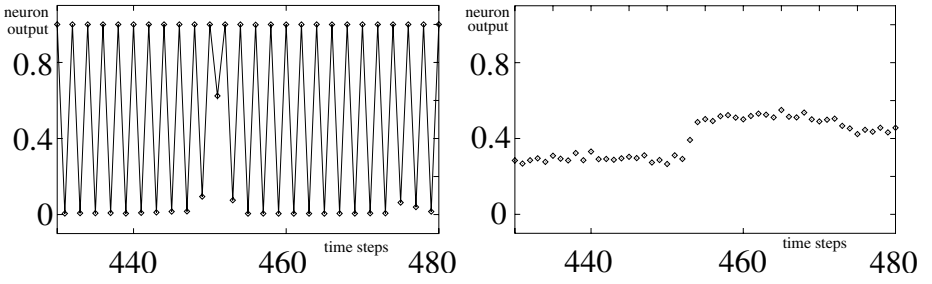


Fig. 6. The neuron outputs o_2 (left) and i_4 (right) over time of neuro-module $\mathcal{G}_{\mathcal{O}\leftrightarrow\mathcal{L}}$. The data are recorded while the robot is approaching a light source. The orientation to the light is realized by a modulation of the amplitude of a period-2 oscillation.

If the amplitude is changing, the mean of this output stream is changing, too (between time step 440 and 460 in the left diagram of Fig. 6). Therefore, the effective motor signals become unequal zero, which creates a turn towards the light, indicated by the increased activity of i_4 (right diagram Fig. 6).

Regarding neuro-module $\mathcal{G}_{\mathcal{O}\Rightarrow\mathcal{L}}$ there is a hysteresis effect active while the robot is approaching the light. This hysteresis effect creates a discrete switch between the turning angles represented by the values 0.39 and 0.96 of o_2 (Fig. 5). Such a hard switch produces the zigzag close to a light source (compare to path plot of Fig. 2 (a)).

5 Conclusion

In this paper we presented incrementally evolved neuro-modules solving a light seeking task for a Khepera robot. We have systematically applied two methods: expansion and fusion. These two methods based on a structure evolution of various recurrent neural networks. The expansion method extends the structure of RNNs, while fusion couples two RNNs to combine different behavioral functionalities. The scope of these experiment was (1) the study of the coordination of different behaviors within one neural control structure and (2) the control principles which allow the integration of new behavioral capabilities without losing the old functionality.

The extended and coupled neuro-modules show an increase of complexity with respect to the behavior relevant dynamical properties. Additionally, we have shown that if relevant dynamics of a behavioral function are modified, the control principles of this robot behavior also fundamentally change. For instance, neuro-module $\mathcal{G}_{\mathcal{O}\leftrightarrow\mathcal{L}}$ utilizes a "frequency and amplitude" coding to generate the required motor signals. Such a coding was not grounded in the basic modules, neither in $\mathcal{G}_{\mathcal{O}}$ nor in $\mathcal{G}_{\mathcal{L}}$. As simple as the presented evolution experiments are they provide a minimal setup which allowed us a detailed study of the effects of the presented incremental evolution on the dynamical properties of the resulting

control structures which leads us to the conclusion that in ER the role of non-linear control principles can hardly be overemphasized.

Indeed, the two examples $\mathcal{G}_{\mathcal{O}-\mathcal{L}}$ and $\mathcal{G}_{\mathcal{O}\mathcal{L}}$ demonstrate how the development of multifunctionality can be organized by a simple superposition of behavior relevant dynamical properties. Hence, multifunctionality can also be generated without new non-trivial dynamical features and non-linear couplings. But, these two examples also show: The development of a simple superposition either goes hand in hand with a remove of initial elements and functionality or has to start with an empty initial structure. These observations suggest that multifunctionality organized by linear control structures must start from scratch each time a new function has to be integrated. This will become unwieldy for open-ended evolutionary processes at a certain level of desired behavioral complexity. If those effects can already be observed in a simple combination of a positive and negative tropism, a stick to linear control structures in ER models must be carefully evaluated.

Relating our results to some state of the art research, we stress three major points: (1) Multifunctionality and task related switchings are natural properties of non-linear coupled systems [7]. (2) The development of multifunctionality is a indispensable prerequisite for open-ended artificial evolutionary processes [5]. (3) Open-ended artificial evolution must develop multifunctionality incrementally to cope the scalability problem [1,3]. Considering these aspects and our results, we claim, that control structures must be based on non-linear principles, if they should provide open-ended artificial evolutionary processes in Evolutionary Robotics.

References

1. Beer, R. D.: An dynamical systems perspective on agent-environment interaction. Artificial Intelligence **72** (1995) 173 – 215
2. Beer, R. D.: The dynamics of active categorical perception in an evolved model agent. Adaptive Behavior **11** (2003) 209 – 243
3. Brooks, R. A.: Artificial life and real robots. In: Proceedings of the First European Conference on Artificial Life, MIT Press (1992) 3–10
4. Clark, A.: *Being There: Putting Brain, Body and World Together Again.* MIT Press, 1997.
5. Bianco, R., Nolfi, S.: Toward open-ended evolutionary robotics: Evolving elemantary robotic units able to self-assemble and selfreproduce. Connection Science **16** (2004) 227–248
6. Hülse, M., Wischmann, S., and Pasemann, F.: Structure and function of evolved neuro-controllers for autonomous robots. Connection Science **16** (2004) 249–266
7. Kelso, S.: *Dynamic Patterns*, MIT Press, 1995.
8. Michel, O.: Khepera Simulator, Package version 2.0. Freeware mobile robot simulator written at the University of Nice Sophia-Antipolis by Oliver Michel. Downloadable from the World Wide Web at `http://wwwi3s.unice.fr/~om/khep-sim.html`.
9. Mondada, F., Franzi, E., Ienne, P.: Mobile robots miniturization: A tool for investigation in control algorithms. In: Proc. of ISER' 93, Kyoto, 1993.

10. Nolfi, S., and Floreano, D.: *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge, 2000.
11. Pasemann, F.: Characteristics of periodic attractors in neural ring networks. Neural Networks **8** (1995) 421 – 429.
12. Pasemann, F.: Structure and dynamics of recurrent neuro-modules. Theory in Biosciences **117** (1998) 1 – 17.
13. Pasemann, F., Steinmetz, U., Hülse, M., and Lara, B.: Robot control and the evolution of modular neurodynamics. Theory in Biosciences **120** (2001) 311–326
14. Walker, J., Garrett, S., Wilson, M.: Evolving controllers for real robots: A survey of the literature. Adaptive Behavior **11** (2003) 179–203.

An On-the-fly Evolutionary Algorithm for Robot Motion Planning*

Teddy Alfaro and María-Cristina Riff

Department of Computer Science, Universidad Técnica Federico Santa María, Valparaíso, Chile
{Teddy.Alfaro, María-Cristina.Riff}@inf.utfsm.cl

Abstract. Computation of a collision-free path for a movable object among obstacles is an important problem in the fields of robotics. The simplest version of motion planning consists of generating a collision-free path for a movable object among known and static obstacles. In this paper, we introduce a two stage evolutionary algorithm. The first stage is designed to compute a collision-free path in a known environment. The second stage is designed to make on-the-fly updates of the robot current path according to the dynamic environmental modifications. Evolutionary techniques have proven to be useful to both quickly compute a new path and to take advantage of the initial path from the first stage. The tests have been made using simulations and a Lego Mindstorms Robot.

1 Introduction

An important problem in the fields of robotics is to compute a collision-free path for a movable robot among obstacles from an initial position to a goal position through a known environment [1], [5], [7]. Many approaches have been proposed to tackle various versions of the same problem with different characteristics. From the control point of view, the goal is to have stability and controllability of the movable object using techniques coming from both linear and non-linear system theories [1]. Other kinds of techniques are complete searches like the automata theory which uses geometric characteristics of the environment [1], [3]. A number of recent publications have proposed methods based on heuristics, which have been successfully applied to solve complex combinatorial instances of the motion planning problem, [2], [4], [5], [6], [7], [8]. The most well known approaches use the map knowledge dividing it into a set of free zones and zones with obstacles. They are thus able to compute a static path usually named *offline planning*. However, the real-world problem is intrinsically dynamic, i.e., the robot is dropped in an environment which can continuously changes, [9], [10]. Some researchers have applied encouraging genetic algorithms for path planning in dynamic environments [5], [6], [8]. Usually, the path is constructed with a set of lines and the algorithms work on unions without taking into account some aspects as the controllability of the robot when it is walking on the lines.

* Supported by the Fondecyt Project 1040364.

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 119–130, 2005.

Furthermore, the dynamic planner works using a complete knowledge of the obstacle that the mobil robot will find.

However, in real applications the degree of this knowledge strongly depends on the capacity of the perception of the robot. In this paper, we propose a two stage evolutionary algorithm which uses a grid with variable resolution (from fine grain to coarse grain). The first stage is designed to compute a near-optimal collision-free path in a known environment. The second stage is designed to make on-the-fly updates of the robot current path according to the dynamic environmental modifications. The idea of this stage is to find a new path, as fast as possible, avoiding collisions. In the best case, the solution given by the second stage can also be a near-optimal one. The two stages are not completely independent. The second stage is called when the environment in the current robot path changes. The local planner uses the remaining part of the current path as a member of its starting population (coming from the first stage or from the previous local planner execution). Because we are working with Lego Mindstorms Robots we have both processing and available memory limitations. Moreover, these robots have also limitations in their sensorial system. Thus, in our approach, the local planner does not require a complete knowledge of the dynamic map modifications. Therefore, our aim here is to propose an efficient algorithm which requires a reasonable processing time and which also does not require neither a complex sensorial system nor a large amount of memory.

The article is organized as follows: in the next section we introduce the Evolutionary planner, in section 3 we present in detail the structure of the Global Planner. Section 4 introduces the Local Planner. In Section 5 we present the Tests. Finally, in section 6 we present the conclusions and future work.

2 The Evolutionary Planner

In our approach we consider the environment divided into cells (grid). The cell size is equal to the robot size. Thus, we only allow robot moves to one of the four cardinal points. We define a map by a grid and the obstacles. The map represents the priori information available for the robot containing the principal characteristics of the environment where it will walk. The goal of the evolutionary planner in the first stage is to find the shortest path from the initial map, beginning from (0,0) cell to the (n,m) cell. We called this stage global planner. In the second stage, the algorithm adapts the initial path to new conditions of the environment. We called this stage local planner. Thus, the solution is the sequence of visited cells representing the shortest path from the starting cell to the ending cell. In Figure 1, we present the general skeleton of the algorithm.

Both planners use the same map description based on cells, the same genetic representation and the same handling constraint strategies.

2.1 Genetic Representation

Our algorithm uses a string of cells with variable length as representation. Each cell is identified by its coordinates x and y on the nxm grid, and by a boolean

```
Evolutionary Planner( )
Begin
init_cell=(x_init,y_init);
goal_cell=(x_goal,y_goal); i=0;
path.cell[i++]=init_cell;
path=Global_planner(init_cell, goal_cell, map);
while (!goal_reached)
  if ( perturbation(path.cell[i]) )
  then
     map_update( perturbation(path.cell[i]) );
     path=Local_planner(i,map);
     i=0
  else
     execute_movement(i);
  endif
endwhile
End
```

Fig. 1. Evolutionary Planner Algorithm

value *obj* which indicates if the cell belongs to a free or a collision zone. The environment map is the whole set of cells.

$$[x_0, y_0, obj_0] \rightarrow [x_1, y_1, obj_1] \rightarrow [x_2, y_2, obj_2] \rightarrow \ldots [x_{p-1}, y_{p-1}, obj_{p-1}]$$

2.2 Constraints

The motion planning problem has many constraints. In our algorithm these constraints are divided in two sets. One set contains the constraints which we imposed to be satisfied by all the chromosomes in all the generations. They are:

- $0 \leq x_i < n$, $\forall i : 0..k$
- $0 \leq y_i < m$, $\forall i : 0..k$
- $(x_{i+1}, y_{i+1}) \in \{(x_i + 1, y_i), (x_i, y_i + 1), (x_i - 1, y_i), (x_i, y_i - 1)\}$

It means that any sequence of cells must be inside the map, and we also impose the path continuity. Each population satisfies this constraint set, as well as the initial population. The second set of constraints is:

- $obj_i \neq 1$, $\forall i$
- $(x_i, y_i) \neq (x_j, y_j)$ si $i \neq j$

The first constraint represents the collision-free condition. The second one indicates that the robot must move to another cell in the next step. This set of constraints is managed by the evaluation function with a penalty factor.

3 Evolutionary Planner: Global Stage

The goal of this stage is to find the shortest path on the map from the initial cell to a goal cell considering known and static obstacles. The generation of the

```
Initial_Population(path)
Begin
Set_data(init_cell,goal_cell,map_dimension);
while(last_cell_path != goal_cell)
  if(not in a corner)
  then
    prob=random_probability();
    if(prob<=Prob_back && backward<Max_backward)
    then
      N=random_number();
      add_backward(N,random_backward_successor);
      backward++;
    else
      add_to_path(random_backward_successor);
    endif
  sino
    add_to_path(random_forward_successor);
  endif
endwhile
eliminate_redundance(path);
End
```

Fig. 2. Initial Population algorithm

initial population is shown in Figure 2. This population is randomly generated but it satisfies the first set of constraints detailed in the above section.The move to go ahead or to come back are inserted randomly on the path.

3.1 Evaluation Function

As we mentioned before the evaluation function searches for a minimal path but also includes a penalization factor in relation to the violation of the collision-free condition. It is shown by the following equation:

$$F(path) = \left(\frac{path\_size()}{n+m-1}\right) + PENALTY \cdot \left(\sum_{i=0}^{path\_size()-1} obj_i\right) \qquad (1)$$

This equation takes into account that the theoretical optimal path size, without either obstacles nor backward moves, is equal to $n + m - 1$.

3.2 Genetic Operators

We have designed four genetic operators. Three of them are asexual operators. The key idea of the recombination operator named Intersection-Bridge crossover is to create two offsprings which inherit a sub-path from two parents. Each asexual operator has a specific task. Arc-operator is created to repair chromosomes which represent paths with collisions. Smooth-operator is designed to generate

a smoother path than a given path by discarding some unnecessary visited cells. Finally, the mutation operator is charged to include more exploration to the algorithm by applying a random path modification. They are described in the following sections.

Intersection-Bridge Operator. This operator creates two children from two parents. It has two modes which can be applied. If the two parents have some common cells in their paths, that is, there are intersections in their pathes, the cross-point is randomly selected from the intersection points. It is shown in Figure 3.

When there is no intersection point, the operator acts in bridge mode. The operator randomly selects a cross-point generating two children. Then, each child will be repaired by including a bridge in order to satisfy the continuity constraint. It is shown in Figure 4.

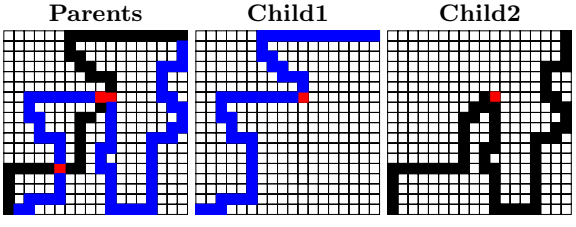


Fig. 3. Intersection Mode

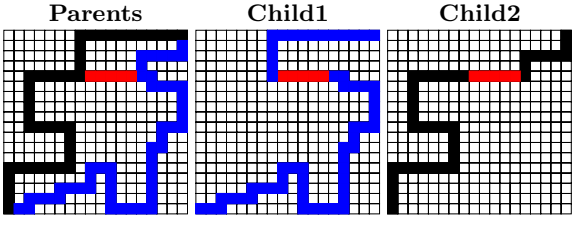


Fig. 4. Bridge Mode

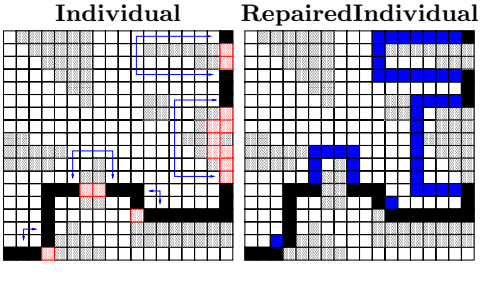


Fig. 5. Arc-Operator Example

```
Arc-Operator(path)
Begin
foreach cell in path
   set=collision_set(path);
  for i=0 to total_set
     size=1; accepted_change=false;
    while(size<Max_arc_size && !accepted_change)
       segment=projects_arc(size,set[i]);
      if(collisions(segment)==0)
      then
         replace(segment, set[i]);
         accepted_change=true;
      else
         size++;
      endif
    endwhile
    if(collisions(segment)!=0 && collision(segment)<collision(set[i]) )
    then
       replace(segment,set[i]);
    endif
  endfor
endforeach
End
```

Fig. 6. Arc Operator Algorithm

Arc-Operator. The goal of this operator is to repair collisions. It extends the path to the borders of the obstacle cells, and goes around the collision zone as is illustrated in Figure 5. The Arc-operator algorithm is shown in Figure 6. This operator repairs but does not worry about the path length.

Smooth Operator. The key idea of this operator is to discard some visited cells which are not needed. Applying this change could help to decrease the path length. The entry of this operator is a collision-free path, therefore it will be only applied to a chromosome where all obj_i are equal to zero.

Mutation Operator. This operator selects a k-cells-length sub-path changing it randomly either to the upper cells or to the lower cells, given a random generated width value, from the current path. The k value is also randomly selected. It takes into account only the first set of constraints. It allows for low cost exploration.

Structure of the Global Planner. The algorithm uses a Roulette Wheel algorithm to select individuals from the population. The complete procedure is shown in Figure 7.

```
Global_Planner(map, init_cell, goal_cell )
Begin
Initial_population()
while (gen < Max_Gen && !goal_reached )
   Evaluate(chromosomes)
   Select(chromosomes)
   Intersection-Bridge-Crossover(paths)
   Mutation(paths)
   Arc_Operator(paths)
   Smooth_Operator(paths)
   Update_population();
   gen++
  if ( collisions()==0 && is_optimal_size() )
  then
     goal_reached=true
  endif
endwhile
return_best_path()
End
```

Fig. 7. Global Planner Algorithm

4 Evolutionary Planner: Local Stage

The Local Stage of the algorithm allows the robot to find a new path given some new obstacles in its current path. In the beginning, its current path is the path given by the Global Stage. When the robot is in front of a new obstacle, the Local Planner starts building a new path which becomes its current path. The process continues until the robot reaches the goal cell. This algorithm is an online planning path, it only uses asexual operators because a new path must be found as fast as possible. The local planner starts its evaluation from the cell where an unknown object has been found. It does not include already visited cells from the current path.

5 Tests

We have divided the tests in simulation and real-world experiments.

5.1 Simulations

We use nine maps with various characteristics to evaluate the Evolutionary Planner. They are regular and irregular polygonals. The test cases are shown in Figure 9, they also have different dimensions. A_1 is 50x50, B_1 is 60x40, C_1 is 60x60, D_1 is 80x60, A_2 is 100x100, B_2 is 800x600, C_2 is 800x600, D_2 is 1000x1000, A_3 is 1500x1500, B_3 is 3000x2000 and the maps C_3 and D_3 using various resolution. Their size corresponds to the image resolution. The cell decomposition

```
Local_Planner(map, current_cell, goal_cell )
Begin
Initial_population(); gen=0; count=0
insert_old_path(paths)
while (gen < Max_Gen && !acceptable_solution )
   Evaluate(paths)
   Mutation(paths)
   Arc_Operator(paths)
   Smooth_Operator(paths)
   Update_population()
   gen++
  if(collisions()=0 && invariable_size_path())
  then
     count++
  endif
  if(collisions()==0 && (is_optimal_size() || count>Max_Count))
  then
     goal_reached=true
  endif
endwhile
return_best_path();
End
```

Fig. 8. Local Planner algorithm

is in pixel, where a white pixel indicates that it is a free collision cell. The benchmarks A_1, C_3, A_2, B_2 have been proposed in [5]. The other ones have been specially generated to include more complex shapes in the map as irregularities, labyrinth and with objects which have a different order of magnitude in size.

Hardware. The hardware platform for the experiments was a PC Pentium IV, 2Ghz with 256 MB RAM under the Fedora Core 2 Kernel 2.6 operating system. The algorithm has been implemented in C.

Gobal Planner Tests. The first test was carried out to test the Global Planner with known static environments.

The optimal path is the shortest free-collision one. Every tested case has an optimal solution. In all of the following results, the algorithm has been limited to 20 iterations. It uses a crossover probability equal to 0.9, mutation and smooth operator probabilities equal to 0.3 and arc-operator probability equal to 0.5. The population size was 10 and the PENALTY factor is equal to 1000. All of these parameter values were defined by tuning.

The Initial Best Chromosome Length is the number of cells of the best path from the initial population (from the evaluation point of view), and the number of its collisions is shown in the Initial Best Chromosome Collisions column. In the same way the Final Best Chromosome Length and the Final Best Chromosome Collisions indicates the characteristics of the best path found by the algorithm.

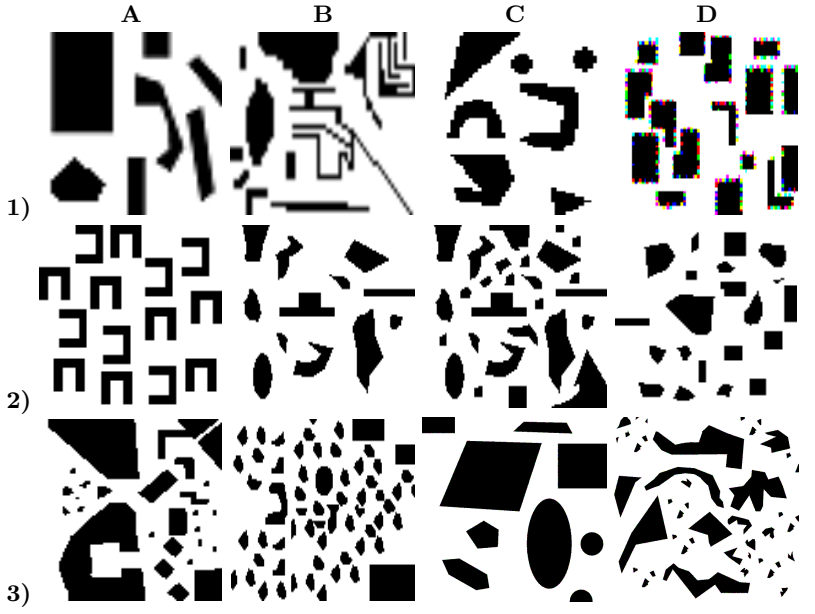


Fig. 9. Benchmarks Maps

Initial Conditions			Final Conditions				
ID map	Best Chromosome Length	Collisions	Best Chromosome Length	Collisions	Number Generations to find Optimal	to satisfy Constraints	CPU Time [m:s]
A1	99	28	99	0	3	2	0:1
B1	107	18	99	3	-	-	0:2
C1	127	32	119	0	1	1	0:1
D1	145	33	139	0	5	4	0:1
A2	205	28	199	0	10	10	0:1
B2	1405	256	1399	0	4	2	0:17
C2	1405	414	1399	0	6	5	0:24
D2	2007	474	1999	0	5	5	1:9
A3	2999	1429	2999	0	5	4	1:36
B3	5023	1116	4999	0	3	2	1:36

Fig. 10. Results of Global Planner

The Number Generations to find Optimal and the Number Generations to satisfy Constraints are the number of generations required by the algorithm to find the best path and the first chromosome which respectively satisfied all the constraints. We can observe that the algorithm is able to satisfy the constraints very quickly. The labyrinth and the non polygonal problems are the hardest ones for the Global Planner. The algorithm cannot find the optimal solution for B_1. It

Resolution	Number Generations		Final Conditions		
	to satisfy Constraints	to find Optimal	Best Chromosome Path Length	Collisions	Time[m:s]
C3 Map					
100x100	3	2	199	0	0:1
500x500	3	3	999	0	0:12
1000x1000	3	2	1999	0	0:50
2500x2500	4	4	4999	0	5:40
D3 Map					
100x100	25	3	199	0	0:3
500x500	6	5	999	0	0:17
1000x1000	27	13	1999	0	3:37
2500x2500	12	12	4999	0	14:58

Fig. 11. Tests with Various Maps Resolution

means that we must design a specific operator for this kind of map. However, for other maps, the algorithm has been very efficient and has found the optimal known values in a reasonable CPU time as shown in the Figure 5.1.

Another kind of test has been done to evaluate how the resolution can affect the performance of the algorithm. We use maps C_3 and D_3 because each one is representative of a polygonal and a non-polygonal map. Their initial resolution is 1000x1000 pixels. We have increased and decreased their resolution. The results are shown in Figure 11. The size of the search space increases when the resolution is higher. In these cases, this kind of algorithm will have a better performance than the complete techniques which in the worst case, visit all the cells.

Local Planner Tests. In order to test the dynamic adaptation of the algorithm some objects have been created in the initial path computed by the Global Planner, to generate unexpected collisions. The test reported here are using maps C_3 and D_3. The obstacles inserted have a 2x3 cells dimensions. We have considered small objets because of the limitations of the sensorial system of our robots.

Map	**Initial Path**	**Obstacles Coordinates**				**New Path Length**			
		a	**b**	**c**	**d**	**a**	**b**	**c**	**d**
C3									
100x100	199	(4,1)	(5,3)	(7,3)	(86,76)	198	195	189	37
500x500	999	(98,67)	(193,182)	(336,244)	(458,254)	838	624	423	287
1000x1000	1999	(405,18)	(620,36)	(973,84)	(978,265)	1580	1343	942	760
D3									
100x100	199	(1,4)	(17,24)	(26,48)	(35,48)	194	158	129	120
500x500	999	(157,8)	(219,175)	(330,333)	(393,428)	834	605	336	198
1000x1000	1999	(188,235)	(446,362)	(561,470)	(872,856)	1576	1191	968	271

Fig. 12. Test for Local Planner

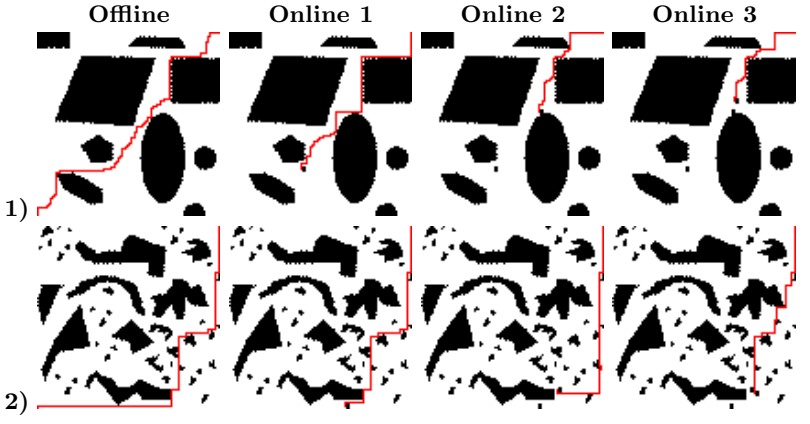


Fig. 13. Local Planner Tests

Figure 12 shows the length of the path obtained after four obstacles {**a,b, c,d**}. The Local Planner computes the New Path Length from the robot current position, where it found an obstacle, to the ending cell. Depending on both the current path and the obstacle cells, the robot could go backward to find a new collision-free path, as it shows Figure 13.

We have observed that the Local Planner is able to quickly find a new path without collision and the given solution usually shares sub-paths with the initial shortest path. This shows that the chromosome coming from the Global Planner's shortest path, helps the algorithm to converge faster.

5.2 Real-World Test: Lego Mindstorms Robot

The RCX is the programmable LEGO brick that controls the robot actions. It has a limited RAM of $32KB$. Using infrared communication RCX communicates with our computer, described in the above section, sending messages back and forth. The Lego Mindstorms robot uses marks on the floor as a navigation system. Its sensorial system is composed by:

- Three light sensors for positioning and navigation, to follow lines on the floor
- Three tactile sensors to identify collision with another object

The cell size was equal to the robot size. The algorithm allows the robot to go from an initial point to the ending point and also to adapt its path, according to the new obstacles introduced in its current path. Because of it is a real-world robot, the algorithm strongly uses the sensorial system of the robot to both, quickly act in case of collision, and if it is required, to modify its current path on-line.

6 Conclusions

The Evolutionary Planner Algorithm that we proposed in this paper can significantly contribute to do efficient motion planning for robots with difficult limitations like memory capacity and sensorial systems. The most important contribution is the Local Planner, which uses the specialized asexual operators to be able to quickly find a new path that avoids collisions. However, it still has several limitations. First of all, the four cardinal points movement must be extended to allow diagonal moves which are more realistic. A second limitation comes from environments with labyrinths, where the algorithm requires a specialized operator to handle this situation.

References

1. Latombe J.C., *Robot Motion Planning*, Kluwer Academic Pub., Boston, 1991.
2. Nilsson N.J., *A mobile automanton: an aplication of artificial intelligence techniques*, International Joint Conference on Artificial Intelligence, pp. 509-520, 1969.
3. Ahuja N., Ahuja Y.K., *Gross Motion Planning*, ACM Computing Surveys, 24, N.3, pp. 219-291, Sep. 1992.
4. Zalama E., *Arquitectura Neuronal no supervisada para el control de un robot móvil en entornos no estacionarios*, Universidad de Valladolid, España 1995.
5. Xiao J., Michalewicz Z., *Adaptive Evolutionary Planner/Navigator for Mobile Robots*, IEEE Transactions on Evolutionary Computation, vol. 1(1), pp.18-28, 1997.
6. Smierzchalski R., Michalewicz Z., *Path Planning in Dynamic Enviroments*, chapter in "Innovations in Machine Intelligence and Robot Perception", Springer-Verlag, 2004.
7. Elshamli A., Hussein A., Areibi S., *Genetic Algorithm for Dynamic Path Planning*, School of Enginnering, University of Guelph, Canada, May 2004.
8. Elshamli A., Hussein A., Areibi S., *Mobile Robots Path planning Optimization*, School of Enginnering, University of Guelph, Canada, 2004.
9. Borenstein J., Feng L., *Measurement and Correction of Systematic Odometry Error in Mobile Robot*, IEEE Journal of Robotics and Automation, vol.12, N.6, pp. 869-880, 1996.
10. Tonouchi Y., Tsubouchi T., Arimoto S., *Fusion of Dead-Recogning Position wiht a workspace model for a mobile robot by Bayesian Inference*, Int. Conf. on Intelligent Robots and Systems, Munich, Germany, pp. 1347-1354, Sep. 1994.

Improving the Evolvability of Digital Multipliers Using Embedded Cartesian Genetic Programming and Product Reduction

James Alfred Walker and Julian Francis Miller

Department of Electronics, University of York, Heslington, York, YO10 5DD, UK
{jaw500, jfm}@ohm.york.ac.uk

Abstract. Embedded Cartesian Genetic Programming (ECGP) is a form of Genetic Programming based on an acyclic directed graph representation. In this paper we investigate the use of ECGP together with a technique called Product Reduction (PR) to reduce the time required to evolve a digital multiplier. The results are compared with Cartesian Genetic Programming (CGP) with and without PR and show that ECGP improves evolvability and also that PR improves the performance of both techniques by up to eight times on the digital multiplier problems tested.

1 Introduction

The evolution of digital multipliers has proved to be very difficult for evolutionary techniques (particularly when the number of bits in the multiplicands is greater than three) [4][7][11][12][13]. Cartesian Genetic Programming (CGP) [5][6] is one technique that has been used to attack such problems. Even though CGP does not have the equivalent of Automatically Defined Functions (ADFs) it was empirically demonstrated to be more computationally efficient than Genetic Programming (GP) [3] with Automatically Defined Functions (ADF's) on the even parity and 2-bit multiplier problems [5]. Embedded Cartesian Genetic Programming (ECGP) is a development of CGP that allows the construction and evolution of modules that can be called from the main CGP code and has been shown to perform better than standard CGP on a series of parity problems [14]. In this paper we apply ECGP to the multiplier problem. We also introduce a new approach called Product Reduction (PR), which is designed to make evolving digital multipliers easier.

The plan for the paper is as follows: Section 2 is an overview of related work. In section 3 we describe ECGP and compare it with CGP before describing PR in section 4. The details of our experiments are shown in section 5 followed by the results and comparisons for all three experiments in section 6. Section 7 gives conclusions and some suggestions for future work.

2 Module Acquisition and Automatically Defined Functions

Module acquisition (MA) [1] adds two operators to the evolutionary process, *compress* that selects a section of the genotype to make it immune to manipulation from operators

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 131 – 142, 2005.

(the module) and *expand* which decompresses a module in the genotype therefore allowing this section of the genotype to be manipulated once more. The fitness of a genotype is unaffected by these operators. MA allows the possibility of having modules within modules. These techniques have been shown to decrease the time taken to find a solution. Rosca's method of Adaptive Representation through Learning (ARL) [8] also extracted program segments that were encapsulated and used to augment the GP function set. However, recently Dessi et al [2] showed that random selection of program sub-code for re-use is more effective than other Rosca's method across a range of problems. Once the contents of modules are themselves allowed to evolve (as in ECGP) they become a form of ADF, however in contrast to Koza's form of ADFs [3] and Spector's Automatically Defined Macros [9], there is no explicit specification of the number or internal structure of such modules. This freedom does exist in Spector's PushGP [10].

3 Embedded Cartesian Genetic Programming (ECGP)

3.1 Representation

ECGP and CGP share the same structure and represent a program as a directed graph (that for feed-forward functions is acyclic). The genotype is a list of integers that encode the connections and functions of each node of the directed graph. CGP used a program topology defined by a rectangular grid of nodes with a user defined number of rows and columns. However, later work in CGP always chose the number of rows to be one, thus giving a one-dimensional topology. This is always used in ECGP. In CGP, the genotype is a fixed length representation (in terms of genes) in which the number of nodes in the program (phenotype) can vary but is bounded. In ECGP the genotype is a variable length representation (in terms of genes and nodes) in which the number of nodes and genes in the graph is bounded. The variable number of nodes in the ECGP genotype is the result of the compression and expansion of modules and the variable number of genes (which allows each node to have a variable number of inputs) is a result of the re-use of modules and some of the module mutation operators, which can change the number of inputs of a node. In Fig. 1 an example of the differences between a CGP and an ECGP genotype are shown. Despite these differences, both CGP and ECGP are initialized with a CGP style genotype. This means that all of the initial genotypes in the population have the same number of nodes and genes and every node represents a primitive function (i.e. no modules are present). Each of the nodes consists of two parts: a node header and a node body. The node header encodes the primitive function or module (by their unique identifier) that the node represents and the type of the node (type I or type II) if the node represents a module (the concept of module type is explained in section 3.4). The node body encodes the inputs of the node. Each input is encoded by two integers: one represents the index of the node or program input (terminal) in the genotype and the other represents the output of the node (note nodes can have multiple outputs) – see Fig. 2. The number of inputs and outputs that a node has is dictated by the arity of its function.

The nodes take their inputs in a feed forward manner from either the output of a previous node or from a program inputs (terminals). The program inputs are

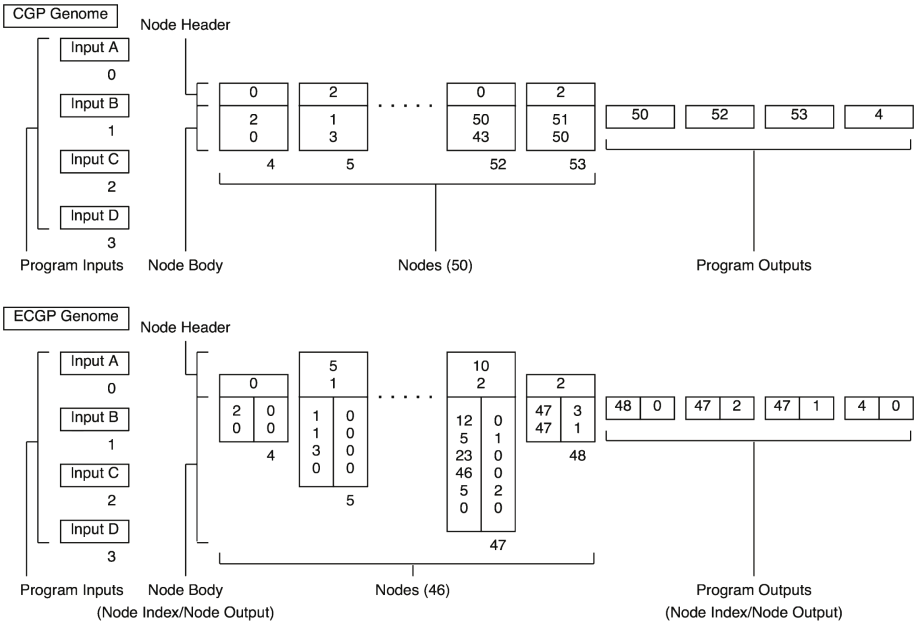


Fig. 1. Examples of evolved CGP and ECGP genotypes for a 2-bit digital multiplier (4 inputs, 4 outputs). Both genotypes were initialized with 50 nodes (150 genes). The top or only number in the node headers represents the function; the remaining number (where present) is the node type. The node body represents the node inputs, which in ECGP are split into two parts: the node index in the genotype and the points from which the node outputs are taken. The node index is underneath each node.

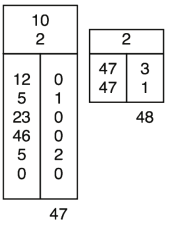


Fig. 2. A fragment of the ECGP genotype from Fig.1. Node 47 with function 10 is of type II (see later) and is a module with 6 inputs connected to node indexes 12(0), 4(1), 23(0), 46(0), 4(2), 0(0). The particular outputs of the nodes are in brackets. Node 48 with function 2 is a primitive function whose inputs are both taken from node 47. The first input comes from output 3 and the second comes from output 1.

numbered from 0 to n-1 where n is the number of program inputs. The nodes in the genotype are also numbered sequentially starting from n to $n+m$-1 where m is the user-determined upper bound of the number of nodes. If the problem requires k program outputs then k integers are added to the end of the genotype, each one representing a pointer

to the output of a node in the graph where the program output is taken from. These k integers are initially set as pointers to the outputs of the last k nodes in the genotype. Fig. 3 shows an ECGP genotype and how it is decoded (a 2-bit digital multiplier circuit).

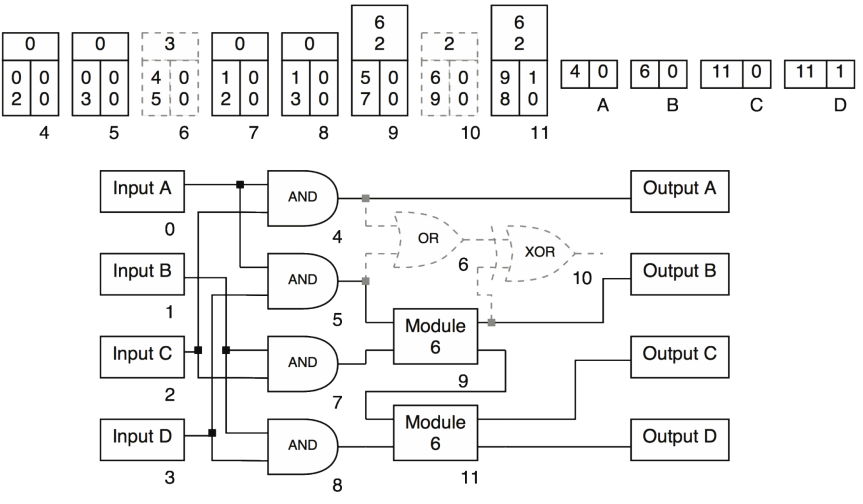


Fig. 3. An ECGP genotype and corresponding phenotype for a 2–bit digital multiplier circuit. Module 6 represents a possible structure for a half adder constructed from the function set. The inactive areas of the genotype and phenotype are shown in grey dashes.

Although each node must have a function and a set of inputs for that function, the outputs of a node do not have to be connected. This is shown in Fig. 3, where the output of node 10 is not used. This allows areas of the genotype to be inactive (nodes 6 and 10, shown in grey dashed lines), leading to a neutral effect on genotype fitness (neutrality). When point mutations are carried out on genes representing connections (the mutation is constrained to respect the directed and acyclic nature of the graphs) these inactive genes can be activated or active genes can be made inactive.

3.2 Evolutionary Strategy

We have used a 1+4 evolutionary strategy defined below:

1. Randomly generate an initial population of 5 genotypes and select the fittest.
2. Carry out point-wise mutation on the winning parent to generate 4 offspring.
3. Construct a new generation with the winner and its offspring.
4. Select a winner from the current population using the following rules:
   - If any offspring has a better fitness; the best becomes the winner.
   - Otherwise, an offspring with the same fitness as the best is randomly selected.
   - Otherwise, the parent remains as the winner.
5. Go to step 2 unless the maximum number of generations is reached or a solution is found.

3.3 Module Representation

A module is represented as a bounded variable length genotype that has the same characteristics of a standard CGP genotype. The module genotype consists of a list of integers and is split into two parts: the module header and the module body. The module header contains four integers and stores information about the module. Each of the four integers encodes the module identifier, the number of module inputs, the number of nodes contained in the module and the number of module outputs respectively. The module body encodes the connections and functions of the nodes contained in the module and the module outputs (similar to program outputs) in the same way as any standard CGP genotype. An example of a module genotype showing the separate components is shown in Fig. 4.

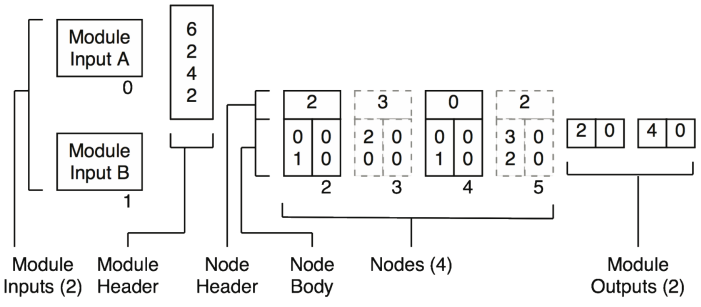


Fig. 4. An example of a module genotype. The four numbers in the module header represent the module identifier, the number of inputs, the number of nodes and the number of outputs of the module respectively. The nodes are represented the same as in ECGP. The module outputs represent which nodes the module takes its outputs from.

The size of a module genotype is determined by the number of nodes and module outputs that it encodes. The number of nodes encoded in the module genotype is bounded between a minimum limit of two (any fewer and it would either be an empty module or a primitive function) and a maximum limit that is set by the user. Likewise the number of module outputs encoded in the module genotype is also bounded between a minimum limit of one (otherwise there would be no way to connect to the module and access its result to the given inputs) and a maximum of n module outputs, where n is equal to the number of nodes contained in the module (one module output per node). The number of module inputs that a module is allowed to have is also restricted between a minimum of two and a maximum of 2n module inputs, where n is equal to the number of nodes contained in the module. However, the number of module inputs allowed does not affect the size of the module genotype, as they are not encoded in the module genotype. In its current form, ECGP only allows modules to contain nodes representing primitive functions rather than nodes representing other modules. An example is given in Fig. 5.

Once a module is created, the module genotype is stored in the module list, which is an extension of the primitive function list. This allows any node in the genotype of

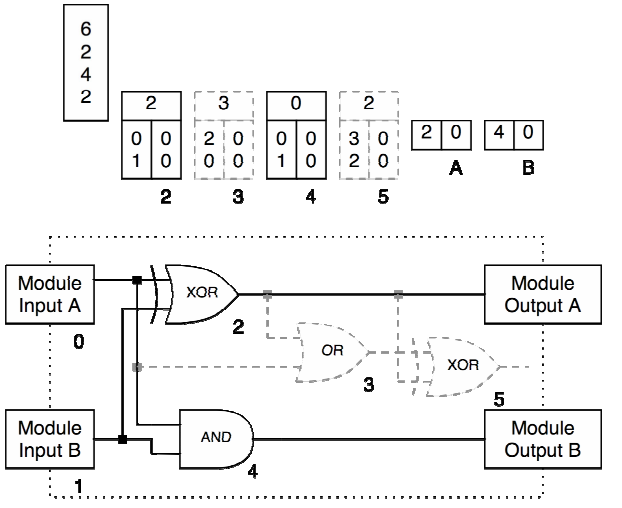


Fig. 5. The genotype and corresponding phenotype of a module representing a half adder. The inactive areas of the genotype and phenotype are shown in grey dashes. The dotted box represents the edges of the module.

an individual to be mutated into any module or primitive function present in either of these lists for that generation. The module list is dynamic and has no restrictions on its maximum size and is updated every generation when the fittest individual (chosen in accordance with the evolutionary strategy used in Section 3.2) in the generation is promoted to the next generation (i.e. the next generation inherits the module list of the fittest individual in the previous generation). This creates a regulatory control of the module list so that bloat never occurs.

The nodes contained inside the module are not necessarily connected and are immune from the main genotype point mutation operator. However, the module itself is allowed to be mutated by the module mutation operators (including a module point operator see section 3.4).

3.4 Operators

ECGP extends CGP by allowing the use of dynamic acquisition, evolution and the reuse of modules. This is achieved through extra mutation operators, which are used in conjunction with the genotype point mutation of CGP.

The compress operator constructs modules by selecting two random points in the genotype (in accordance with the rules for the module size restrictions) and encapsulates all the nodes between these two points into a new module, which is encoded into a module genotype as described earlier. Note, that if there any modules between the two selected points the compress operator does not take place (this is because at present we do not allow modules within modules). The number of module inputs that a module is initialized with is determined by the number of

connections between the inputs of the nodes that are going to be encapsulated into a module and the outputs of any previous nodes or program inputs (terminals) in the genotype when the module is created. Likewise, the number of module outputs possessed by a module is determined by the number of connections between the inputs of the latter nodes in the genotype and the outputs of the nodes that are going to be encapsulated in the module, when it is created. Any module created by the compress operator is represented in the genotype of an individual as a type I node. The node header (i.e. the primitive function or module that the node represents) in any type I node is immune from the genotype point mutation operator therefore allowing the type I node to remain in the genotype of an individual until it is removed by the expand operator (see Table 1).

Table 1. Nodes types and their properties

Node Type	Action of Compress	Action of Expand	Action of Genotype Point Mutation
I	Creation	Destruction	Change node inputs
II	Immune	Immune	Creation or destruction or change node inputs

The expand operator destroys a type I node by replacing it in the genotype of an individual with the nodes contained in the module that the type I node represented. The inputs of all of the latter nodes in the genotype of the individual are updated in the final stage of both the compress and expand operators so that all the connections remain intact. The reasons for this is that, the compress and expand operators only make a structural change to the genotype of an individual and have no affect on genotype fitness, as the genotypes before and after the action of these operators represent the same directed graph. The expand operator has twice the probability of being applied to the genotype than the compress operator. We found that this introduces a pressure for good modules to replicate quickly in the genotype of an individual in order to survive. This can be seen as survival-of-the-fittest modules within the genotype itself.

Modules can replicate within the genotype of an individual through the action of the genotype point mutation operator. This is identical to that used in CGP with the exception that it can mutate the function of a node to any of the primitive functions or any available modules in the module list. If a node is mutated to represent a module it is classed as a type II node and is treated like a standard node. This means the genotype point mutation operator can also mutate the function of a type II node to any of the pre-defined functions or any available modules in the module list. It can also mutate any of the inputs of the type II node in the same way it would mutate the inputs of a standard node. If the function of a standard node or type II node is mutated, the new node keeps however many of the original nodes inputs it needs and randomly generates any extra inputs it may require. Type II nodes are also immune from the expand operator as this could cause excessive growth of the genotype that could possibly lead to bloat.

To summarize the properties of node types I and II are shown in Table 1. The main reasons for the two types of module is to try and reduce the excessive growth of the genotype and to also help induce a selection pressure on the modules so that they have to replicate in the genotype (i.e. make the transition from being represented by type I to type II nodes) and be associated with a high fitness genotype in order to survive. Once the module is represented by a type II node it is harder for the module to be removed from the module list, as it has a lower probability that it will be removed from the genotype (i.e. it cannot be expanded). This is both advantageous as it allows good modules to stay in the module list but is also disadvantageous as it could possibly allow the evolution of the genotype to progress a lot slower.

The module genotypes contained in the module list can also be evolved through the action of five different operators: *module point mutation, add-input, add-output, remove-input* and *remove-output*. The module point mutation operator is a restricted version of the CGP genotype point mutation operator, as it can still mutate the inputs and function of any node contained in the module genotype but it is not allowed to introduce any type II nodes into the module genotype. It can also mutate which node output each of the module outputs are connected to.

The add-input and add-output operators allow greater connectivity to and from the contents of a module by increasing the number of module inputs or module outputs by one respectively each time either operator is applied, making a more generalized module. When the add-input operator is applied to a module, the gene representing the number of module inputs in the module header part of the module genotype is incremented by one and an extra gene is inserted into all nodes (type I and type II) representing the module in the genotype of the individual, as a randomly chosen value for the new module input. Likewise, when the add-output operator is applied to a module, the gene representing the number of module outputs in the module header part of the module genotype is incremented by one and two extra genes are added to the module output section of the module genotype, as randomly chosen values for the node index and node output that the new module output is connected to.

Alternatively, the remove-input and remove-output operators reduce the connectivity to and from the contents of a module, by decreasing the number of module inputs or module outputs by one respectively each time either operator is applied, therefore making a more specialized module. When the remove-output operator is applied to a module, the gene representing the number of module inputs in the module header part of the module genotype is decremented by one and the gene corresponding to the module input randomly chosen is removed from all nodes (type I and type II) representing the module in the genotype of an individual. Likewise, when the remove-output operator is applied to a module, the gene representing the number of module outputs in the module header part of the module genotype is decremented by one and the two genes corresponding to the randomly chosen module output are removed from the module output section of the module genotype. All of the operators: add-input, add-output, remove-input, and remove-output must comply with the restrictions on

the number of module inputs and module outputs at all times. Further information about all of the module operators (including figures explaining their operation) is available in our previous work [14].

4 Product Reduction (PR)

In digital multipliers we require n^2 AND gates to compute the product bits. Product reduction (PR) assumes that these have already been provided. It uses the outputs of these gates as inputs to the remaining circuit (which is evolved). PR transforms the standard truth table of 2^{2n} rows to an input-output table having $2^{2n} - 2(2^n) - 2$ rows. The width of the PR table is increased from the $2n$ inputs found in the standard truth table to n^2 inputs because n^2 AND Boolean functions are required to produce the product of every combination of bits. The length of the PR table however is reduced because the PR table contains multiple row entries all containing zeros due to multiplication by 0, which can be reduced to a single row.

5 Experiment Details

The performance of CGP and ECGP both with and without PR was tested on the digital multiplier problem (2x2 and 3x3 bit). The fitness is defined as the number of phenotype output bits that differ from the perfect n-bit digital multiplier. A perfect solution has score zero.

The parameter settings used for CGP and ECGP in all of the experiments are shown in Table 2. The probability values chosen for the ECGP operators were found to be optimal by a trial and error process in previous ECGP experiments.

Table 2. Parameter settings used for CGP and ECGP in all of the experiments. The operator rate is expressed as a percentage of the genotype length. Both the operator rates and probabilities are per generation. 50 independent runs used.

Parameter	Value
Population size	5
Initial genotype size	200 nodes (600 genes)
Function set	{AND, AND with one input inverted, OR, XOR}
Genotype point mutation rate	3% (18 Genes)
Genotype point mutation probability	1
Compress/Expand probability	0.1/0.2
Module point mutation probability	0.04
Add/Remove input probability	0.01/0.02
Add/Remove output probability	0.01/0.02
Maximum module size (ECGP only)	5 or 10 nodes
Module list initial state (ECGP only)	Empty

6 Results

For all experiments, the Computational Effort (CE) was calculated using the formula found in Fig. 6 [3] with z=99% and are shown in Table 3. They are only relevant when comparing CGP and ECGP with the same number of nodes in their genotypes and the same rate for the genotype point mutation operator because CE figures for CGP and ECGP vary significantly depending on these values, therefore potentially causing an unfair comparison. We have only compared the CE figures of ECGP with CGP because no other researchers have provided CE figures for their GP techniques on these problems.

$$P(M,i) = \frac{N_s(i)}{N_{total}}, \quad R(z) = ceil \ \frac{\log(1-z)}{\log(1-P(M,i))} \ , \quad I(M,i,z) = MR(z)(i+1)$$

Fig. 6. The Computational Effort (CE) formula from [3] where i represents the generation number, $N_s(i)$ represents the number of successful runs by generation i, N_{total} represents the total number of runs and M represents the number of individuals in the population. $P(M,i)$ represents the cumulative probability of success, $R(z)$ represents the number of independent runs required to give a probability of success z by generation i and $I(M,i,z)$ represents the minimum number of individuals which must be processed to give a probability of success z by generation i.

Table 3. The CE figures for CGP and ECGP for the digital multiplier problems with and without product reduction. The maximum module size is shown in brackets.

	2-Bit Multiplier	**3-Bit Multiplier**
CGP	37,600	18,509,600
CGP with PR	5,600	2,498,800
ECGP (5)	46,000	8,400,400
ECGP with PR (5)	6,000	1,560,400
ECGP (10)	61,600	2,795,200
ECGP with PR (10)	7,600	688,800
CGP-PR Speedup	6.7	7.4
ECGP-PR (5) Speedup	7.7	5.4
ECGP-PR (10) Speedup	8.1	4.1

For both of the digital multipliers tested over all fifty runs, both CGP and ECGP with and without PR produced 100% successful solutions. The results from both multipliers clearly show that CGP with PR performs between 6.7 and 7.4 times faster than CGP without PR and that ECGP with PR performs between 7.7 and 5.4 (with a maximum module size of five) or 8.1 and 4.1 (with a maximum module size of ten) times faster than ECGP without PR depending on the chosen maximum module size. We note that, rather unexpectedly, the speedup with CGP increases with problem difficulty, while the opposite is true with ECGP where the speedup decreases. We think this is because most of the time taken by CGP without PR to find a solution is used

organizing the AND Boolean functions in the 1-bit multiplication section of the circuit. However, ECGP without PR finds the 2x1-bit Multiplier module and re-uses it to quickly find and organize the 1-bit multiplication section. Therefore by eliminating the 1-bit multiplication section from the search space by using PR, saves CGP more time than ECGP as the problem scales in difficulty.

Comparing the results of CGP and ECGP (both with and without PR) on the individual problems shows that CGP performs quicker than ECGP on the 2-bit multiplier problem. This could be because the exploration of code in the modules hinders the performance of ECGP on small problems, as the results show that by reducing the maximum module size makes the performance of ECGP closer to that of CGP. However, ECGP does perform substantially better than CGP on the harder 3-bit multiplier problem, suggesting that ECGP may perform better on even larger, more complex problems. This speedup could be because ECGP is building and re-using modules containing useful partial solutions out of the primitive functions such as the 1-bit half adder and the 1-bit full adder. The results also show that for harder problems, ECGP performs better with a larger maximum module size (doubling the maximum module size, halved the computational effort for the 3-bit multiplier). This could be because the more nodes a module has the easier it is to find partial solutions. This is an interesting concept and will be investigated further in future work.

All of the experiments were run on a single processor desktop PC with 512MB of memory. The time taken to complete 50 runs of each problem varied between a few minutes to a few hours depending on problem difficulty and whether PR was used. ECGP only took fractionally longer to complete one thousand generations on any problem than CGP showing that the computational time required for the overhead of module acquisition is quite small and the computational time taken for fitness evaluation (both CGP and ECGP) is by far the dominant factor.

7 Conclusion

We have presented for the first time the application of PR with CGP and ECGP on the difficult digital multiplier problem. PR is shown to significantly speedup the performance of CGP and ECGP when compared with CGP and ECGP without PR on both multipliers tested. However, CGP was shown to perform better than ECGP on the simpler 2-bit multiplier problem but ECGP performed better on the harder 3-bit multiplier problem indicating that ECGP may perform substantially better than CGP on even larger problems. This is a promising result for ECGP as the results presented in this paper follow a very similar trend to those found in our previous work [14].

It was also found that the maximum module size chosen for ECGP can drastically affect performance and will be investigated further in future investigations. Currently ECGP does not allow modules within modules. However, we do have a working version of ECGP that allows embedded sub-modules but we are currently investigating the problem of bloat within the embedded sub-modules found in the inactive areas of the module genotype. When a solution is found, we intend to allow embedded sub-modules in future work as this could lead to an even greater boost in performance.

References

[1] Angeline, P. J. Pollack, J. (1993) Evolutionary Module Acquisition, Proceedings of the 2nd Annual Conference on Evolutionary Programming, pp. 154-163, MIT Press, Cambridge.

[2] Dessi, A. Giani, A. Starita, A. (1999) An Analysis of Automatic Subroutine Discovery in Genetic Programming, GECCO 1999: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 996-1001, Morgan-Kaufmann, San Francisco.

[3] Koza, J. R. (1992, 1994) Genetic Programming I and II. MIT Press, London.

[4] Miller, J. F., Thomson, P., and Fogarty T. C. (1997) Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study, Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications. Editors: D. Quagliarella, J. Periaux, C. Poloni and G. Winter, Wiley.

[5] Miller, J. F. (1999) An Empirical Study of the Efficiency of Learning Boolean Functions using a Cartesian Genetic Programming Approach, GECCO 1999: Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, Florida, pp 1135-1142, Morgan Kaufmann, San Francisco.

[6] Miller, J. F. and Thomson, P. (2000) Cartesian Genetic Programming, Proceedings of the 3rd European Conference on Genetic Programming, Edinburgh, Lecture Notes in Computer Science, Vol. 1802, pp 121-132, Springer-Verlag, Berlin.

[7] Miller, J. F., Job D., and Vassilev, V. K (2000) Principles in the Evolutionary Design of Digital Circuits – Part I, Genetic Programming and Evolvable Machines, Vol. 1, pp. 8-35.

[8] Rosca, J. P. (1995) Genetic Programming Exploratory Power and the Discovery of Functions, Proceedings of the 4th Annual Conference of Evolutionary Programming, San Diego, pp 719-736, MIT Press, Cambridge.

[9] Spector, L. (1996) Simultaneous Evolution of Programs and their Control Structures, Advances in Genetic Programming II, pp. 137-154, MIT Press, Cambridge.

[10] Spector, L. (2001) Autoconstructive Evolution: Push, PushGP, and Pushpop, Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001, pp. 137-146. San Francisco, CA: Morgan Kaufmann Publishers

[11] Torresen, J. (2003) Evolving Multiplier Circuits by Training Set and Training Vector Partitioning, Proceedings of the 5th International Conference on Evolvable Hardware, ICES03, Lecture Notes in Computer Science, Vol. 2606, pp. 228-237, Springer-Verlag, Berlin.

[12] Torresen, J. (2004) Exploring Knowledge Schemes for Efficient Evolution of Hardware, Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware (EH-2004), pp. 209-216, IEEE Comp. Society Press.

[13] Vassilev, V. K. and Miller J. F. (2000) Scalability Problems of Digital Circuit Evolution, Proceedings of the 2nd NASA/DOD Workshop on Evolvable Hardware, pp. 55-64, IEEE Comp. Society Press.

[14] Walker, J. A. Miller, J. F. (2004) Evolution and Acquisition of Modules in Cartesian Genetic Programming, Proceedings of the 7th European Conference on Genetic Programming, Lecture Notes in Computer Science, Vol. 3003, pp 187-197, Springer-Verlag, Berlin.

Benefits of Employing an Implicit Context Representation on Hardware Geometry of CGP

Xinye Cai, Stephen L. Smith, and Andy M. Tyrrell

Department of Electronics, The University of York, Heslington, York YO10 5DD, UK
{sls5, amt}@ohm.york.ac.uk

Abstract. Cartesian Genetic Programming (CGP) has successfully been applied to the evolution of simple image processing filters and implemented in intrinsic evolvable hardware by the authors. However, conventional CGP exhibits the undesirable characteristic of positional dependence in which the specific location of genes within the chromosome has a direct or indirect influence on the phenotype. An implicit context representation of CGP (IRCGP) has been implemented by the authors which is positionally independent and outperforms conventional CGP in this application. This paper describes the additional benefits of IRCGP when considering alternative geometries for the hardware components. Results presented show that smaller hardware arrays under IRCGP are more robust and outperform equivalent arrays implemented in conventional CGP.

1 Introduction

A form of genetic programming (GP) [1] termed Cartesian Genetic Programming (CGP) [8,9] has been successfully adapted for the evolution of simple image processing filters [11,12] and subsequently implemented in hardware [14,15]. A criticism of CGP (and GP in general) is that the location of genes within the chromosome has a direct or indirect influence on the resulting phenotype [6]. In other words, the order in which specific information regarding the definition of the GP is stored has a direct or indirect effect on the operation, performance and characteristics of the resulting program. Such effects are considered undesirable as they may mask or modify the role of the specific genes in the generation of the phenotype (or resulting program). Consequently, GPs are often referred to as possessing a direct or indirect context representation.

An alternative representation for GPs in which genes do not express positional dependence has been proposed by Lones and Tyrrell [3-7]. Termed *implicit context representation,* the order in which genes are used to describe the phenotype (or resulting program) is determined after their self-organised binding, based on their own characteristics and not their specific location within the genotype. The result is an implicit context representation version of traditional parse-tree based GP termed *Enzyme Genetic Programming*. The authors have since implemented an implicit context representation of CGP, termed *Implicit Context Representation Cartesian Genetic Programming* (IRCGP), specifically for the evolution of image processing filters [13].

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 143 – 154, 2005.

This paper reports the additional benefits of IRCGP when considering alternative geometries for the constituent hardware components. Specifically, the performance of IRCGP and conventional CGP are compared over a range of hardware component configurations.

Section 2 of the paper gives a brief introduction to the use of conventional CGP for evolving image processing filters. Section 3 describes the implementation of implicit context representation of CGP (IRCGP). Section 4 presents results obtained from both conventional and implicit context representation CGP for a range of different hardware configurations. Conclusions are presented in Section 5.

2 Cartesian Genetic Programming for Evolving Image Processing Filters

Cartesian Genetic Programming (CGP) was first proposed by Miller [8,9] as an alternative representation for genetic programming which does not require the use of a parse-tree based programming language and does not exhibit uncontrolled expansion commonly termed bloat [2]. As opposed to the rigid tree structure representation of traditional GP, CGP permits the arrangement of functions in a far more flexible, typically rectangular format, referenced by conventional Cartesian co-ordinates.

An extension of CGP for evolving image processing filters was proposed by Sekanina [11-12] and subsequently implemented in hardware by Zang et al. [14,15] as shown in Figure 1.

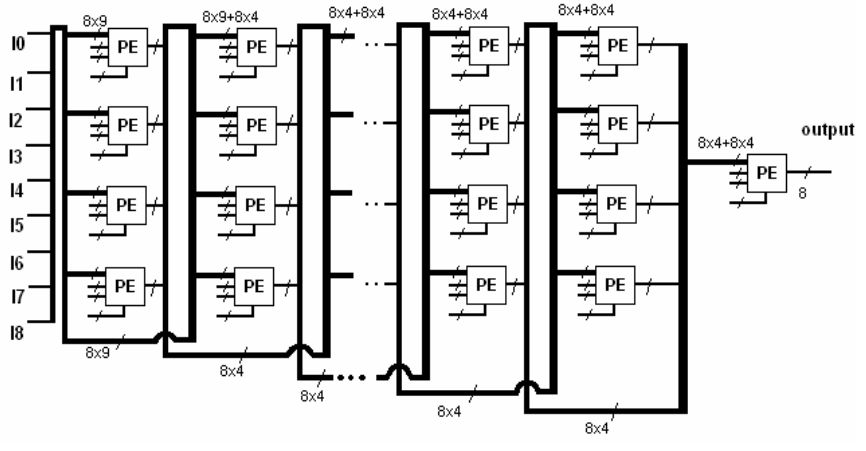


Fig. 1. Extended Cartesian Genetic Programming for evolution of image processing filters

A number of processing elements (PEs) are arranged in a rectangular format, each connected to a data bus. The inputs I0 to I8 are the pixel values obtained from a conventional 3 x 3 neighborhood image filter; these are manipulated by the PEs and the output replaces the pixel of interest in the processed image. The structure of the PE, shown in Figure 2, comprises two multiplexers and a functional block. The multi-

plexers can be configured, according to the values of cfg1 and cfg2 respectively, to select the output of another PE or image pixel input I0 to I8, as long as it is connected to the same data bus. In the specific hardware representation considered here, this requires that the PE or input be located in the two columns immediately preceding the PE containing the multiplexer in question. The outputs of the two multiplexers are then provided as inputs to the functional block; the function applied to them is determined by cfg3 and selected from the available functions listed in Table 1.

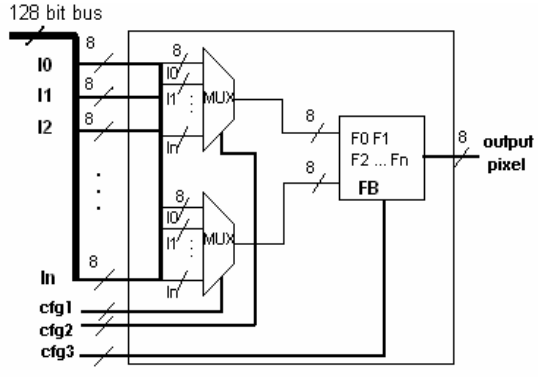


Fig. 2. Architecture of the processing element

Table 1. Functions available for configuration of the processing element's functional block

Code	Function	Code	Function
F0:	X >> 1	F8: (3)	(X+Y+1) >> 1
F1:	X >> 2	F9:	X & 0x0F
F2:	~ X	F10:	X & 0xF0
F3:	X & Y	F11:	X | 0x0F
F4:	X | Y	F12:	X | 0x F0
F5: (1)	X ^ Y	F13: (4)	(X&0x0F) | (Y&0xF0)
F6:	X + Y	F14:	(X&0x0F) ^ (Y&0xF0)
F7: (2)	(X+Y) >> 1	F15:	(X&0x0F) & (Y&0xF0)

&	bitwise AND function
|	bitwise OR function
^	bitwise XOR function
~	bitwise inverter or NOT function
>>	bitwise right shift
X, Y	inputs to functional block

7 4 3 4 0 3 6 1 3 8 5 3 2 11 3 2 1 3 0 4 3 12 3 3 10 9 1 12 16 3 9 13 1 14 11 3 17 18 3 19 20 3 20 20 3 15 20 3 17 19 4 24 19 2 19 24 4 21 22 3 22 26 4 27 22 3 28 21 1 25 23 3 28

Fig. 3. Example chromosome for configuration of the extended CGP

The PEs within the architecture are configure by means of a chromosome, an example of which is given in Figure 3.

The chromosome consists of a string of integer values, arranged logically in groups of three, providing values for cfg1 (multiplex 1 input), cfg2 (multiplex 2 input) and cfg3 (functional block function index), respectively, for each PE in the representation.

A number of these chromosomes form the individuals of a population which are initialised with random values. Each chromosome is then used to configure the hardware representation which, in turn, is used to process a test image. The image resulting from this operation is compared with an ideal (uncorrupted image), and a fitness score derived, which is then associated with the respective individual's chromosome. After all the individuals in the population have been evaluated in this manner, the fittest is retained and used as the parent for a subsequent generation of individuals. These new individuals are generated by simply mutating the parent in a non-deterministic manner.

Yang et al demonstrated that the image filters evolved in this way out performed conventional median and Gaussian image filters [14,15].

3 Implict Context Representation

3.1 Overview

As described in Section 1, CGP can be described as an *indirect context representation*; the position a particular gene occupies in the chromosome has an influence on the resulting phenotype, or in the case of extended CGP considered in Section 2, the configuration of the hardware representation.

The effect or meaning of a component in the evolved or resulting program is determined by its absolute or relative position in the program representation. The manner in which components are referenced in CGP is considered arbitrary as there is no correlation between a component's absolute coordinates and its behaviour. Therefore it can be argued that indirect context representation has no effect beyond describing the connectivity of a specific program. This is also the case when considering the behaviour of components in different programs. Components with the same functionality may have different coordinates and those with different functionality the same coordinates. Hence, any form of recombination, such as crossover is unlikely to be constructive in the evolutionary process and could explain why this has not been found to be useful in CGP [4].

A lack of positional independence also has an important effect on the relationship between genes that in combination effect good performance. Recombination will not preserve the relationship of these genes, commonly referred to as building blocks, when conventional forms of crossover are employed. Various attempts have been adopted to minimise the destructive effect that such positional dependence in the representation by preserving building blocks that describe the beneficial relationship between particular genes. In GAs this is termed linkage learning, and has been implemented by limiting the destructive effect of crossover operations limiting by using special crossover templates [10]. However, this does not overcome the underlying problem which is associated with the program representation. A further concern with an indirect context representation such as CGP is that when a component's input references are mutated, the resulting arrangement of components is in no way represents the degree of mutation applied and hence, cannot be varied in a gradual manner.

Therefore, ideally, the evolution of a system should be independent of the position of genes within the chromosome, but should still be a result of the values of those

genes. This is termed an *implicit context representation* by Lones and Tyrrell [3], who have developed a form the conventional parse-tree type GP that exploits this representation, called *Enzyme Genetic Programming* (EGP). The biological inspiration for Enzyme GP is the metabolic pathway, and the role of enzymes which express computational characteristics. This is not dissimilar to the logic network employed in this work to evolve the image filters described in Section 2 [6].

Implicit context representation employs an enzyme model comprising a shape, activity and specificities (or binding sites) [5], as shown in Figure 4. Along with inputs and outputs, the enzyme model can be considered a program component from which a genetic program may be constructed. The shape describes how the enzyme is seen by other program components. Similarly, the binding sites determine the shape (and hence type) of program component the enzyme wishes to bind to. Finally, the activity determines the logical function the enzyme is to perform. A typical EGP will comprise a set number of inputs and outputs and a number of enzyme models or components. Initial values for each component's binding sites and logical function are assigned non-deterministically; the component's shape, however, is derived from a combination of its binding sites' shapes and logical function which is considered in Section 3.2.

Once initialized, components are bound together to form a network, as shown in Figure 5. The order in which components are bound is determined by the closeness of match between a component's binding site shape and another component's shape. The best matching components are bound first and the process is repeated until a network has formed in which no further binding is possible.

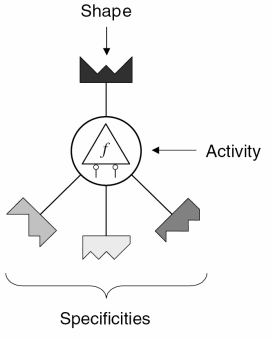


Fig. 4. Enzyme model illustrating shape, activity and specificities (binding sites) [5]

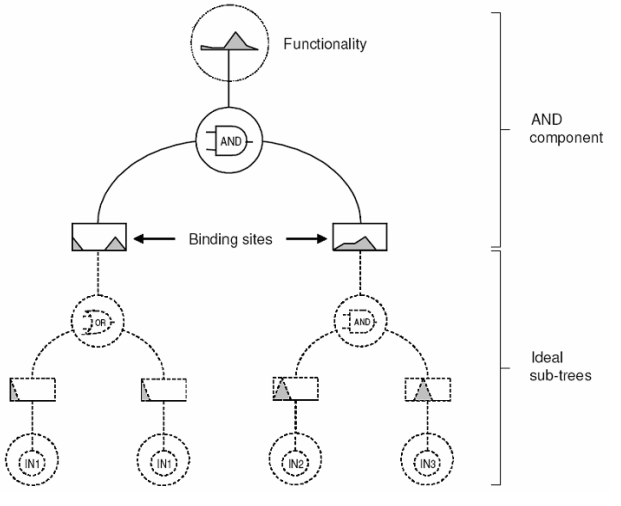


Fig. 5. Calculation of a component's shape from its binding site shapes and logical function [6]

Over time, components may evolve through mutation. Mutation is applied to the component's binding sites and logical function with a pre-determined probability. When this occurs, a new component shape is derived accordingly and may lead to different binding between components occurring. This in turn may result in a modified network.

3.2 Implicit Context Representation CGP

The purpose of Implicit Context Representation CGP (IRCGP) is to combine the benefits of an implicit context representation (described above in Section 3.1) with the extended Cartesian Genetic Program for evolving image filters described in Section 2.

The processing elements within the extended CGP are particularly suited to the implicit context representation implementation. However, instead of employing a parse-tree arrangement, the existing CGP Cartesian arrangement is maintained. The significant difference to conventional CGP is the manner in which components are selected and interconnected within the representation. To achieve this, each component is equipped with two binding sites and a shape (as shown in Figure 6), which relate directly to the inputs and outputs of a component in the existing extended CGP representation.

As previously described, the values for the elements in the binding site shapes are initially assigned non-deterministically, as is the component's function. However, the output shape is numerically derived from the binding site shape and component function as illustrated in Figure 7.

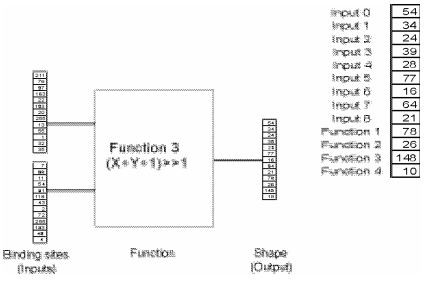


Fig. 6. Component binding sites and shape

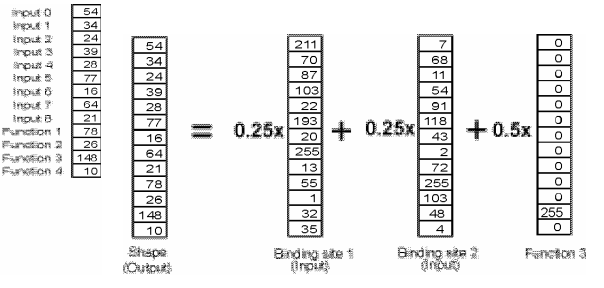


Fig. 7. Calculation of component's shape from binding sites' shape and component function

Formation of the CGP network begins with the assignment of an output component (Figure 8a); this will ultimately provide a new value for the pixel under consideration in the filtered image. The binding sites of the output component are then made active and will bind to components according the closeness of match between their respective shapes (Figures 8b-8d). (Closeness of match is based on the sum of differences between the elements of the two shapes.) Once bound, component's binding sites will also become active and will bind to other components in the same way. Binding between components is always undertaken on a "best-fit first" basis until no further binding is possible.

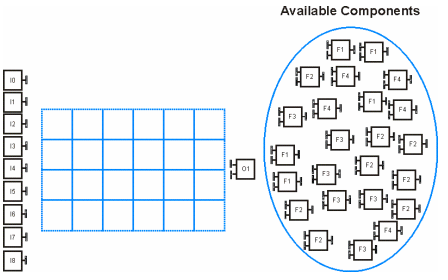


Fig. 8a. Network composition begins with non-deterministic allocation of the output component, O1

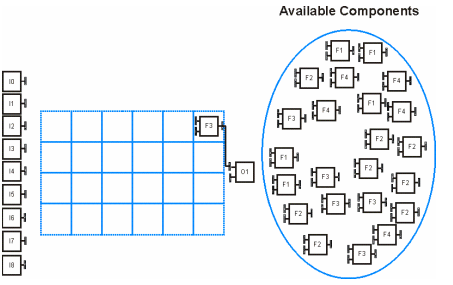


Fig. 8b. The component whose shape matches either of the output component's binding sites most closely, is chosen for binding

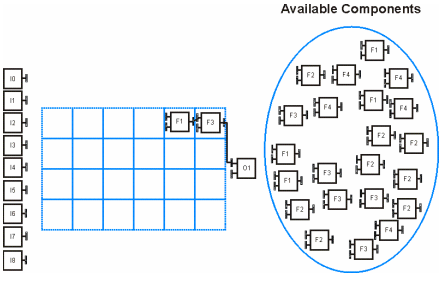


Fig. 8c. Binding continues on a "best match-first" basis

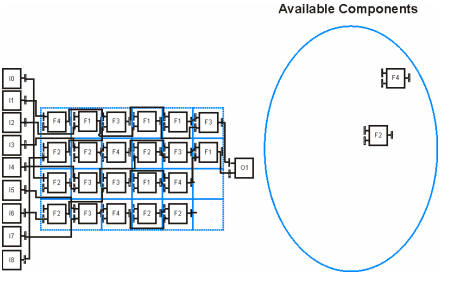


Fig. 8d. Binding is completed when no further components can be bound

The physical hardware places constraints in the manner with which the formation of the network takes place. Successful binding of a new component may only take place if there is sufficient space for that component in the hardware representation. Typically, this means that any newly bound component must be placed in one of the two columns to the left of the existing component. Similarly, input components I0 to I8 (holding the image pixel values) may only be bound to components one or two columns to their right in the representation.

Once all possible binding has completed, the resulting network is applied to a test image and, the resulting filtered image, compared with the original, uncorrupted image. A fitness score for the individual is described by equation (1), which is identical to that used in previous image filtering evolution by Sekanina [11,12] and Yang [14,15].

$$fitness = 255.(H-2).(W-2) - \sum_{i=1}^{H-2}\sum_{j=1}^{W-2} ideal(i,j) - filt(i,j) \quad (1)$$

Where:

i,j	are the image co-ordinates
filt(i,j)	is the image resulting from the resulting filter operation
ideal(i,j)	is the ideal (original uncorrupted) image
H, W	is the height and width of the image respectively

For the purpose of clarity, this fitness score is presented as a percentage of the image score possible, in this case, the original image.

3.2.1 Redundancy and Reuse

It can be seen from the way in which components are selected for inclusion in the network that there is provision for both redundancy and reuse of components. The pool of available components need not be restricted to the number required to populate the network; additional components can be made available to facilitate redundancy. Further, reuse of components is permitted, by allowing an output of one component to satisfy the input of more than one other component.

3.2.2 Mutation

Two separate mutation operations are performed according to predefined probabilities: (i) to the binding sites of the components and, (ii) to the index that selects the component's function from those available (as defined in Table 1). Once these mutations have been performed, new shapes for each component are derived as described in Section 3 and shown in Figure 7.

3.2.3 Selection Scheme

A conventional, q-tournament selection scheme is adopted; one of the advantages being that it does not require a global fitness comparison of all individuals in the population. From the population, a group of q individuals is randomly chosen (where q is the *tournament size*). The fittest individual from the tournament group will be selected and placed in a pool for recombination. The process is repeated until the required number of individuals has been attained. Experimentation suggests that for this application, a 9-tournament scheme is most likely to provide best performance for both maximum and average fitness over 20 runs.

3.2.4 Crossover Operator

An important benefit of an implicit context representation is that recombination supports meaningful variation filtering, i.e. the effects of inappropriate variation events are suppressed, whilst promoting meaningful change, leading to fitter solutions. For the implicit context representation of CGP described here, a conventional 2-point crossover was used to exchange components available to the two individuals. This is simply implemented as each component available to each individual is held in a sequential list.

4 Results

Results are presented for 20 runs of the implicit context representation CGP (IRCGP) compared with the conventional extension of CGP (ECGP), for each of the hardware component configurations shown in Table 2. The task was for the algorithms to evolve an image processing filter that reduces noise on a version the 'Lena' image corrupted with Gaussian noise ($\sigma = 16$), shown in Figure 10b. A list of algorithm parameters for IRCGP is given in Table 3.

Results for the fitness score of IRCGP compared with ECGP are presented graphically in Figure 9 and listed in Table 2. These values are represented as a percentage

Table 2. Fitness scores for evolved image filters

	Hardware Configuration (Rows x Columns)				
	1 (3x3)	2 (4x3)	3 (4x4)	4 (4x5)	5 (4x6)
IRCGP Average Best Fitness (%)	97.460	97.498	97.500	97.513	97.522
ECGP Average Best Fitness (%)	97.377	97.390	97.450	97.486	97.507
IRCGP Overall Best Fitness (%)	97.517	97.519	97.533	97.537	97.544
ECGP Overall Best Fitness (%)	97.480	97.480	97.499	97.520	97.528
Two-tailed P value	0.00163	1.16e-06	0.000271	0.002563	0.026045

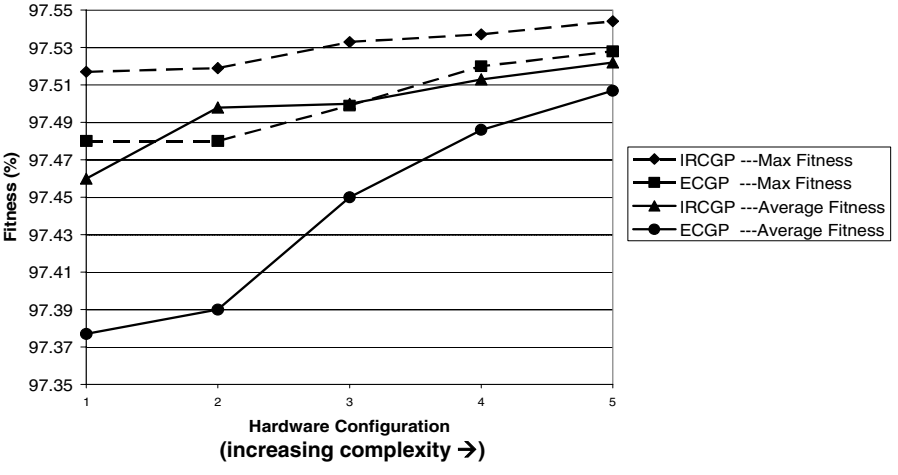


Fig. 9. Results for Implicit Context Representation CGP (IRCGP) and Conventional Extended CGP (ECGP)

Table 3. Program parameters for IRCGP

Parameter	Value
Population size	150
Number of generations	150
Number of runs	20
Function mutation rate	1.0%
Binding site mutation rate	0.6%
Number of available functions	4
Available functions	X^Y (X+Y+1)>>1 (X+Y)>>1 (X&0x0F)|(Y&0xF0)

Fig. 10a. Original 'Lena' image

Fig. 10b. 'Lena' image corrupted with added Gaussian noise (σ =16)

Fig. 10c. Image after applying ECGP evolved filter

Fig. 10d. Image after applying IRCGP evolved filter

of the best possible fitness score, in this case, the original image (Figure 10a). The results show that IRCGP outperforms ECGP for all hardware configurations, both in overall and average best fitness values. Examples of the resulting images for ECGP and IRCGP are shown in Figures 10c and 10d respectively. The two-tailed P value gives support for statistical significance. It is of particular interest that the fitness of IRCGP for small hardware component configurations (particularly four rows and four columns) remains close to the performance of larger configurations (such as the conventional four rows and six columns).

Once evolved, the filter was also applied to another image, 'Baboon' (Figure 11a), again corrupted with guassian noise in the same way as for 'Lena' (as shown in Figure 11b). Results presented in Table 4 and Figures 11c and 11d, show that the IRCGP again outperforms ECGP.

Table 4. Fitness scores for 'Baboon' image corrupted with Gaussian noise ($\sigma = 16$)

	ECGP(%)	IRCGP (%)
Overall best fitness	93.965	94.007
Average best fitness	93.937	93.967

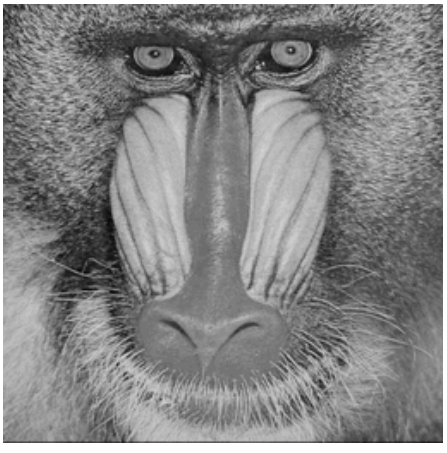


Fig. 11a. Original 'Baboon' image

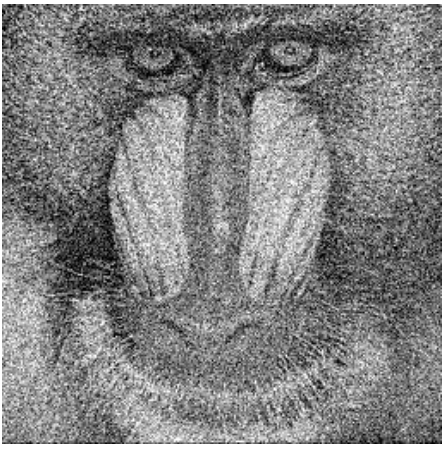


Fig. 11b. 'Baboon' image corrupted with Gaussian noise

Fig. 11c. Image after applying ECGP evolved filter

Fig. 11d. Image after applying IRCGP evolved filter

5 Conclusion

This paper considers the benefits of an implicit context representation of CGP (IRCGP) over a number of different hardware component configurations. The results presented show that IRCGP outperforms conventional extended CGP (ECGP) in all hardware configurations tested and also demonstrates a more consistent performance and stability.

References

1. J. Koza: Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press (1992)
2. W. Langdon: Quadratic bloat in genetic programming. In D. Whitley, D. Goldberg, and E. Cantu-Paz, editors, Proceedings of the 2000 Genetic and Evolutionary Computation Conference (2000) 451-458
3. M. A. Lones and A. M. Tyrrell: Enzyme genetic programming. Proc. 2001 Congress on Evolutionary Computation, J.-H. Kim, B.-T. Zhang, G. Fogel, and I. Kuscu (eds.), IEEE Press. Vol. 2 (2001) 1183–1190
4. M. A. Lones and A. M. Tyrrell: Crossover and Bloat in the Functionality Model of Enzyme Genetic Programming. Proc. Congress on Evolutionary Computation 2002 (2002) 986-992
5. M. A. Lones and A. M. Tyrrell: Biomimetic Representation with Enzyme Genetic Programming. Journal of Genetic Programming and Evolvable Machines. Vol.3 No.2 (2002) 193-217
6. M. A. Lones: Enzyme Genetic Programming. PhD Thesis, University of York, UK, (2003)
7. M. A. Lones and A. M. Tyrrell: Modelling biological evolvability: implicit context and variation filtering in enzyme generic programming. BioSystems (2004)
8. J. Miller and P. Thomson: Cartesian genetic programming. in Third European Conf. Genetic Programming, R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty (eds.). Vol. 1802 Lecture Notes in Computer Science, Springer (2000)
9. J. F. Miller, D. Job, and V. K. Vasilev: Principles in the evolutionary design of digital circuits—Part I. Genetic Programming and Evolvable Machines, Vol. 1 (2000) 7–36
10. J. Schaffer and A. Morishima: An adaptive crossover distribution mechanism for genetic algorithms". Proceedings of the Second International Conference on Genetic Algorithms and their Applications, 1987.
11. L. Sekanina, V. Drabek: Automatic Design of Image Operators Using Evolvable Hardware. Fifth IEEE Design and Diagnostic of Electronic Circuits and Systems (2002) 132-139
12. L. Sekanina. Image Filter Design with Evolvable Hardware. Applications of Evolutionary Computing – Proceedings of the 4th Workshop on Evolutionary Computation in Image Analysis and Signal Processing (EvoIASP'02), Lecture Notes in Computer Science. Vol. 2279 Springer-Verlag, Berlin (2002) 255-266
13. S.L. Smith, S. Leggett and A.M. Tyrrell: An Implicit Context Representation for Evolving Image Processing Filters– Proceedings of the 7th Workshop on Evolutionary Computation in Image Analysis and Signal Processing (EvoIASP'05), Lecture Notes in Computer Science. Vol. 3449. Springer-Verlag, Berlin (2005) 407-416
14. Z. Yang, S.L. Smith and A.M. Tyrrell: Intrinsic Evolvable Hardware in Digital Filter Design. Lecture Notes in Computer Science. Vol.3005 Springer-Verlag, Berlin (2004) 389-398
15. Z. Yang, S.L. Smith and A.M. Tyrrell: Digital Circuit Design using Intrinsic Evolvable Hardware. Proceedings of 2004 NASA/DoD Conference on Evolvable Hardware, Seattle (2004)

Evolution in Materio: Investigating the Stability of Robot Controllers Evolved in Liquid Crystal

Simon Harding and Julian F. Miller

Department of Electronics, University of York, York, UK
{slh, jfm}@evolutioninmaterio.com
http://www.evolutioninmaterio.com

Abstract. In our previous work, we have demonstrated that evolution can be used to program liquid crystal to act as a signal processing device. In this work we discuss the stability and reconfigurability of a real time robot controller evolved in liquid crystal. We envisage these issues will be important when programming or evolving in other physical systems.

1 Introduction

Allowing computer controlled evolution (CCE) to manipulate novel physical media can allow much greater scope for the discovery of unconventional solutions. Last year the authors demonstrated, for the first time, that CCE could manipulate liquid crystal to perform signal processing tasks (i.e frequency discrimination, robot control). In [4] Harding and Miller showed that liquid crystal could be used as a medium for evolution. They were able to rapidly evolve simple transistor like behaviour and in [3] they demonstrated that it was relatively easy to evolve a liquid crystal to discriminate between pairs of dissimilar frequencies. The task was first considered by Adrian Thompson (using an FPGA) [10]. Recently we have investigated other tasks including robot control.

Here we examine some practical issues relating to evolving devices in liquid crystal. In particular we look at these issues in the case of evolving a real time robot controller for obstacle avoidance. We found that solutions were relatively unstable, and were greatly influenced by previous configurations. In this paper we investigate this phenomenon in detail.

1.1 The Field Programmable Matter Array

In [8] a device that the authors referred to as a Field Programmable Matter Array(FPMA) was described. A FPMA is a device that can be used to manipulate a material under computer control by applying voltages that induce physical changes within a substance, and that these changes may interact in unexpected ways that may be exploitable under evolution.

Different candidate materials were cited for possible use as the evolvable substrate in the FPMA. They all share several characteristics : the material should be configurable by an applied voltage/current, the material should affect an incident signal (e.g. optical and electronic) and should be able to be reset back to

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 155–164, 2005.

its original state. Examples of these include electroactive polymers, voltage controlled colloids, bacterial consortia, liquid crystal, and nanoparticle suspensions. In previous work we have demonstrated that liquid crystal is indeed a suitable material to form the basis of the FPMA.

Liquid crystal (LC) is commonly defined as a substance that can exist in a mesomorphic state [1,7]. Mesomorphic states have a degree of molecular order that lies between that of a solid crystal (long-range positional and orientational) and a liquid, gas or amorphous solid (no long-range order). In LC there is long-range orientational order but no long-range positional order.

2 The Liquid Crystal Evolvable Motherboard

The Liquid Crystal Evolvable Motherboard (LCEM) is circuit that uses four cross-switch matrix devices to dynamically configure circuits that connect to the liquid crystal. The switches are used to wire the 64 connections on the LCD to one of 8 external connections. The external connections are: input voltages, grounding, signals and connections to measurement devices. Each of the external connectors can be wired to any of the connections to the LCD.

The external connections of the LCEM are connected a PC's analogue inputs and outputs. Connections can be assigned for the input signals, measurement, and for fixed voltages (plus a ground connection). The value of the fixed voltages is determined by a genetic algorithm[6], but is constant throughout each evaluation.

In the experiments presented here, the liquid crystal glass sandwich was removed from the display controller it was originally mounted on, and placed on the LCEM. The display is a passive monochromatic matrix LCD with a resolution for 180 by 120 pixels. Unfortunately neither the internal structure nor the electrical characteristics of the LCD are known. The display has a large number of connections (in excess of 200), and is roughly positioned over the pads on the PCB, with many of the PCB pads touching more than 1 of the connectors

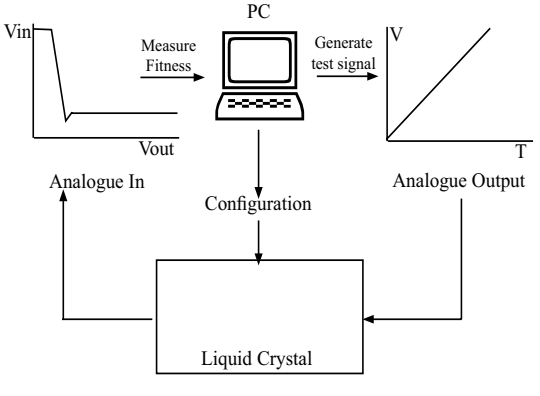


Fig. 1. Equipment configuration

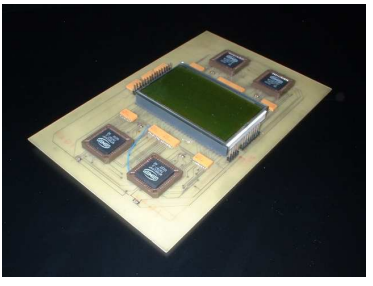


Fig. 2. The LCEM

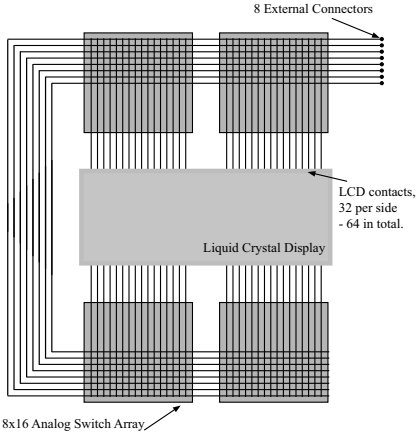


Fig. 3. Schematic of LCEM

on the LCD. This means that we are applying configuration voltages to several areas of LC at the same time.

It is important to note that other than the control circuitry for the switch arrays there are no other active components on the motherboard - only analogue switches, smoothing capacitors, resistors and the LCD are present.

3 A Liquid Crystal Robot Controller

In these experiments we used a simulated robot that has two sensors (mounted with 30 degrees of separation) and two wheels for mobility. The simulated sensor readings are converted into signals fed to the evolvable motherboard. Signals read from the evolvable motherboard are then used to control the behaviour of the simulated robot. The intention being that the signal processing, and majority of the robot control should be performed in the liquid crystal. Two sonar distances sensors and two motors can be considered to be "directly" connected to the evolvable motherboard, and then routed to the liquid crystal.

We defined each distance sensor to output a square wave with a frequency proportional to the distance in a straight line from the sensor to an obstacle. For near objects the output was 1Hz, for far objects the output frequency is 5000Hz. No artificial noise is added to the distance measured, however the mechanism by which the waves are generated by the computer will add noise and timing problems. There is also an expected 50ms delay between a distance reading and a change in frequency.

Two connections from the LCEM are used as inputs to the two motor controllers. The two motors are mounted either side of the simulated robot, and allow for the robot to be steered. If the voltage is high (i.e. above 0.3V) a motor is switched fully on, when low the motor is set to a slow speed. If both inputs are high the robot drives forward, with both inputs off the robot is stationery. If

only one motor is enabled, the robot turns. The threshold voltage for enabling a motor was chosen arbitrarily. The robot has a small turning circle, and does not pivot on the switched off wheel.

3.1 The Genetic Algorithm

The genetic representation for each individual is made of two parts. The first part specifies the connectivity; the second part determines the configuration voltages applied to the the LCD. Each of the 64 connectors on the LCD can be connected to one of the eight external connectors or left to float (see Figure 3). Each of the connectors is represented by a number from 0 to 7 and no connection is represented by 8. Hence the genotype for connectivity is a string of 64 integers in the range 0 to 8. The remainder of the genotype specifies the voltages applied to the pins on the external connector that are not used for signal injection / monitoring. One of the external connectors is always connected to ground. Two are reserved for the incident signals (distance readings) and two connections for motor control. The remaining three connectors have static voltages applied to them that are determined by evolution. All these connectors can be routed to various places in the liquid crystal display according to the connection scheme decided by evolution. Each voltage is represented as a 16-bit integer, the 65536 possible values map to the voltage levels output from -10V to +10V. The second section of the genotype is therefore represented as a string of three 16bit integers.

In all the following experiments, a population of 40 individuals was used. The mutation rate was set to 5 mutations per individual. Elitism was used, with 5 individuals selected from the population going through to the next generation. Selection was performed using tournament selection based on a sample of 5 individuals. Evolutionary runs were limited to 200 generations. With each individual taking approximately 60 seconds to evaluate. The fitness function rewarded controllers that were able to travel around the environment without colliding with obstacles and for exploring as much of the environment as possible.

Further details of the fitness function, genotype and related operators can be found in [5].

3.2 Results

The fitness function rewarded perfect solutions with a score of 10000, with 0 as the lowest possible score. Solutions that have a fitness of over 6700 represent robots that have navigated to leave the top section of the map. Solutions below this score fail to fully explore the map - however they may cover large areas of the top half of the map but never escape through the gap. In our evolutionary runs we found 36% of runs obtained a near perfect score. The average number of generations to find a good solution is 62, with the fastest solution found within 22 generations.

Figures 4 to 9 shows sections of the "fossil record" of the evolution of one controller. We can see that after learning not to drive in circles, the robot learns to move forward, and then learns to turn when it approaches a wall. After it

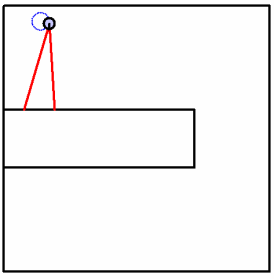


Fig. 4. Fitness=515

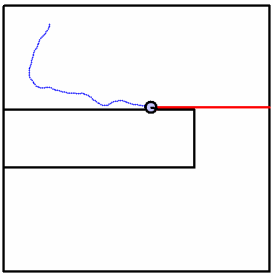


Fig. 5. fitness=3819

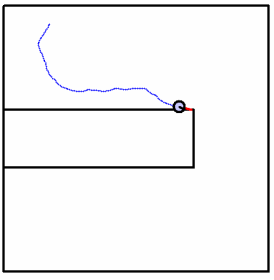


Fig. 6. fitness=4607

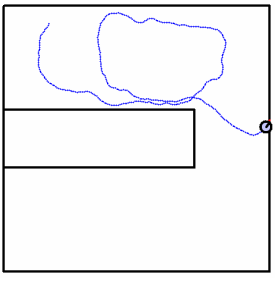


Fig. 7. fitness=6772

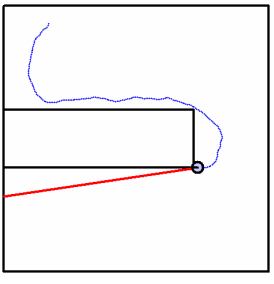


Fig. 8. fitness=7229

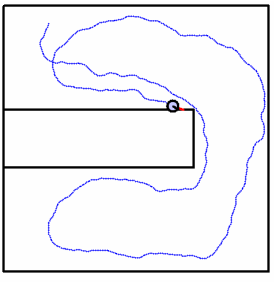


Fig. 9. fitness=9796

learns to start following the wall it quickly searches the entire map, and gets the highest fitness.

4 Investigating Solution Stability

When incident signals are applied to the liquid crystal display, we can see their effect - some of the pixels go dark - indicating that the molecular direction has been changed. This means that the configuration of the liquid crystal is changing as we are applying signals. To draw an analogy with circuit design, the incident signals would be changing component values or changing the circuit topology, which would have an effect on the behaviour of the system. This is likely to be detrimental to the measured performance of the circuit, and also we expect to a liquid crystal solution. When a solution is evolved the fitness function automatically measures it stability over the period of the evaluation. Changes made by the incident signals can be considered part of the genotype to phenotype mapping. Solutions that cannot cope with their initial configurations being altered will achieve a low score. However, the fitness function cannot measure the behaviour beyond the end of the evaluation time - however a stable solution is still desirable.

Another issue of stability is that of the genotype to phenotype mapping. When a configuration is applied to the liquid crystal, do the molecules go back to

exactly how they were when this configuration was tried previously? We cannot be sure - since we cannot directly measure the properties of every molecule, and in a highly complex system such as LC it would be unlikely to reorder in precisely the same way. Assuming, that there is a strong correlation between genotype and phenotype, then it is likely that evolution will cope with this extra noise. The fact that it is possible to evolve in liquid crystal, shows that we should expect good genotype/phenotype correlation, however as the results in this paper indicate, this is not the case.

In [9] it is noted that the behaviour of circuits evolved intrinsically can be influenced by previous configurations - therefore their behaviour (and hence fitness) is dependent not only on the currently evaluated individuals configuration but on those that came before. For example, in a circuit capacitors may still hold charge from a previously tested circuit. This charge would then effect the circuits operation, however if the circuit was tested again with no stored charge a different behaviour would be expected and a different fitness score would be obtained - and the fitness function would essentially be noisy. Not only does this effect the ability to evolve circuits, but would mean that some circuits are not valid. Without the influence of the previously evaluated circuits the current solution may not function as expected. The behaviour does not have to be worse when dependent on previous configurations - there is no reason why the previous configuration cannot have a positive influence. It is expected that such problems will have analogies in evolution in materio. The configurations are likely to be highly sensitive to initial conditions (i.e. conditions introduced by previous configurations), as the ability to configure a system is reliant on the emergent properties of the material. The behaviour of emergent systems, such as Conway's "Life" [2] or Wolfram's cellular automata [11] are highly dependent on the starting configuration. Small perturbations in the initial starting arrangement can prevent solutions from becoming stable.

In these experiments, we investigate the stability of the solution, by looking at the performance of the solution for extended periods of time and the effect of previous configurations on the behaviour of the system.

4.1 Observing Continued Behaviour

The wall avoiding robot task often produced solutions that would take approximately 30 seconds to evaluate before the robot collided with an obstacle. This raised the issue of for how long are solutions stable? This experiment investigates the performance of solutions over a period of time. If the liquid crystal was being affected by the incident signals, and being reprogrammed, then it would be expected that the behaviour would change. A change in behaviour would change the fitness score measured by the fitness function. The fitness function used for this task returns an absolute fitness value - any change in behaviour should result in a different fitness value.

In these experiments robot controllers were evolved (as described in section 3) and when an individual received a high fitness score we repeatedly ran the evaluation function and recorded the fitness for the subsequent evaluations. The

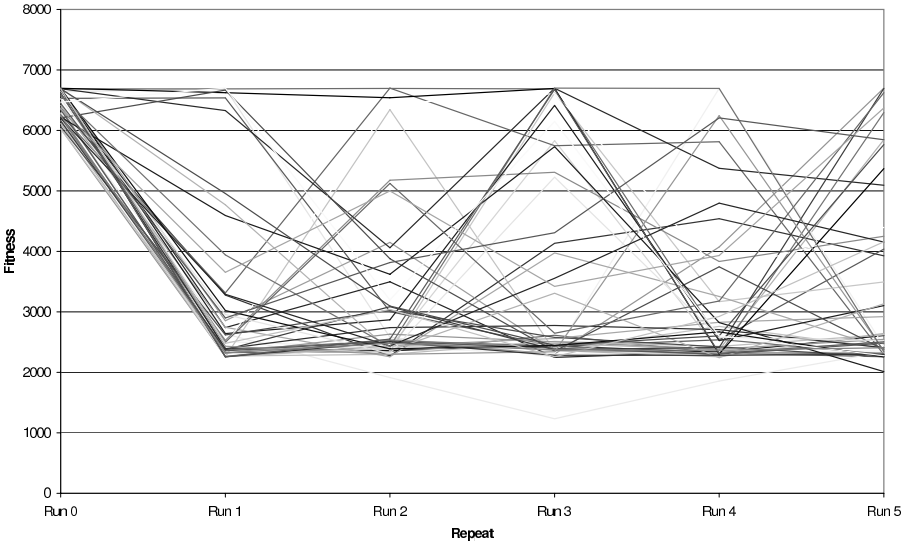


Fig. 10. Graph showing degradation of fitness when individuals are reloaded and tested, as described in section 4.1

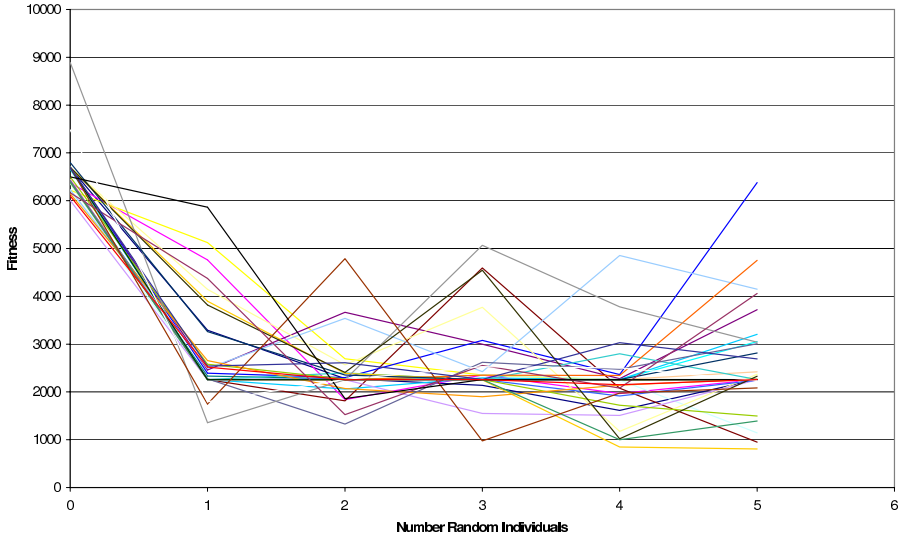


Fig. 11. Graph showing degradation of fitness when individuals are reloaded and tested, with intermediate individuals loaded in between, as described in section 4.2. The number of intermediate random individuals increases on each reload.

individuals were re-evaluated 10 times - giving a total running time of approximately 5 minutes.

4.2 Drift and the Reloading of Configuration

This experiment demonstrates the affect that previous configurations have on the behaviour of the current evaluation. The experiment is similar to that in section 4.1, however the configuration of the liquid crystal is modified in between evaluations. This was done by applying a number of random configurations, and then reapplying the configuration specified in the current individual. The intent is to disrupt the liquid crystal as much as possible, and to try and randomize the molecular configuration. This evaluation process can be summarised as:

```
1. Apply individual, and test fitness.
2. If the solution is good (i.e. with fitness>6700) then continue
   otherwise, goto step 1 and evaluate next individual.
3. N = 0
4. Apply N random Configurations to LC
5. Apply individual
6. Test and record fitness
7. N = N + 1
8. If N<6 repeat from step 4.
```

If configurations are dependent on previous configurations then the fitness values would be expected to be affected, and as there are few solutions the fitness would be expected to decrease. If the configurations are independent of each other, then the fitness results should be relatively consistent throughout the evaluation procedure.

4.3 Results

From Figure 10, it can be seen from the reduced fitness scores that most solutions fail to operate over long periods of time. However, some solutions do continue to function correctly for several evaluations. The results also show that the behaviour can deteriorate, and then recover. From Figure 11 it is apparent that previous configurations have a large effect on the behaviour of the system. Applying other configurations causes the evolved behaviour to worsen in every case. The fitness does vary through each iteration, however it never returns to the level initially achieved.

5 Conclusions

The effects observed in these experiments may preclude this set-up from having any real practical application. The issue of reliable genotype to phenotype mapping and reliable behaviour from the phenotype are serious problems. However, these are preliminary results from our first attempt at direct evolution in

material. With further investigation, we believe that there will be methods to overcome these problems.

The most obvious solution is to find a more stable material. Liquid crystal was chosen as a candidate material because of the ease at which we could manipulate it. However there may be more suitable materials, that can be made less sensitive to disruption caused by incident signals.

The next potential solution is to evolve stable solutions. In the experiments in this section, solutions were stable only for a few seconds - however there was no pressure imposed for evolution to find solutions that had any robustness. With the robot controller, solutions had to remain stable for a much longer period - typically 30 seconds. It is apparent when forced to produce time-robust solutions evolution in liquid crystal is capable of doing so.

Another approach, for certain tasks, may be to find a different way to interact with the material. At the moment the method of communication with the liquid crystal is through different frequencies of electrical signals. This will have the effect of continually reconfiguring the system. It is not clear if this is the most appropriate way to present information to the system.

Liquid crystals are normally associated with displays as they can modify light passing through them. We can assume that light does not effect the liquid crystal and is only affected by it. If we used a camera to observe changes in light coming through the display we could see any differences in the properties of the light and use this as an output. Liquid crystal is also able to change the properties of sound waves passing through it. This technique is normally used to study the structure of the liquid. If the structure is modified using computer controlled evolution, it should be possible to alter the effect it has on sound waves. Incident signals could be applied using a speaker, and the output detected with a microphone.

Despite the issues of stability, the system is still evolvable. It would appear from the experiment in section 4.2 that there is little inheritance of behaviour from one generation to the next - as applying similar (and even the same) configurations may not result in the same phenotypic behaviour. However, the evolutionary algorithm still succeeds in finding solutions despite the highly noisy search space. In future we intend to investigate this further, and determine if there are any properties of this system that could be utilised by other evolvable systems.

References

1. D. Demus, J Goodby, G W Gray, H W Spiess, and V Vill, editors. *Handbook of Liquid Crystals*, volume 1,2A,2B,3. July 1998.
2. Martin Gardner. Mathematical games: The fantastic combinations of john conways new solitaire game life. In *Scientific American*, volume 223, pages 120–123, 1970.
3. Simon Harding and Julian F. Miller. Evolution in materio: A tone discriminator in liquid crystal. In *In Proceedings of the Congress on Evolutionary Computation 2004 (CEC'2004)*, volume 2, pages 1800–1807, 2004.
4. Simon Harding and Julian F. Miller. Evolution in materio: Initial experiments with liquid crystal. In *Proceedings of 2004 NASA/DoD Conference on Evolvable Hardware (EH'04)*, pages 298–305, 2004.

5. Simon Harding and Julian F. Miller. Evolution in materio : A real time robot controller in liquid crystal. In *To appear in the proceedings of 2005 NASA/DoD Conference On Evolvable Hardware*, 2005.
6. John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, MA, USA, 1992.
7. I. C. Khoo. *Liquid Crystals: physical properties and nonlinear optical phenomena.* Wiley, 1995.
8. J. F. Miller and K. Downing. Evolution in materio: Looking beyond the silicon box. In *Proceedings of NASA/DoD Evolvable Hardware Workshop*, pages 167–176, 2002.
9. Adrian Stoica, Ricardo Salem Zebulum, and Didier Keymeulen. Mixtrinsic evolution. In *ICES*, pages 208–217, 2000.
10. A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In Tetsuya Higuchi, Masaya Iwata, and L. Weixin, editors, *Proc. 1st Int. Conf. on Evolvable Systems (ICES'96)*, volume 1259 of *LNCS*, pages 390–405. Springer-Verlag, 1997.
11. Stephen Wolfram. *A new kind of science.* Wolfram Media Inc., Champaign, Ilinois, US, United States, 2002.

Hardware Implementation of 3D Self-replication

André Stauffer, Daniel Mange, and Fabien Vannel

Logic Systems Laboratory, Swiss Federal Institute of Technology,
IN-Ecublens, CH-1015 Lausanne, Switzerland
{name.surname}@epfl.ch, lslwww.epfl.ch

Abstract. After a reminder of the Tom Thumb algorithm, originally designed for the self-replication of two-dimensional (2D) loops, this paper presents its application to the hardware implementation of 3D self-replicating structures. This self-replication process is achieved by translation and transcription of a configuration information in a three-dimensional data and signals cellular automaton (DSCA). The corresponding hardware implementation takes place in the BioCube, a new 3D reconfigurable electronic medium with input, output and computation abilities.

1 Introduction

The main goal of this paper is to present the hardware implementation of three-dimensional (3D) self-replicating structures endowed with universal construction and universal computation properties. Basically designed for the self-replication of two-dimensional (2D) loops with universal construction and universal computation, the Tom Thumb algorithm [5] is revisited here in order to deal with the third dimension. According to this algorithm, a configuration information made of flag data and code data is used twice during the self-replication process. First by translation, where the information ends up trapped in the new replicated loop, defining both its structure and its functionality. Secondly by transcription, where the information remains mobile and moves along the loop in order to allow further replications. By addition of only a few supplementary flags, the Tom Thumb algorithm allows the self-replication of 3D structures provided with universal construction and computation capabilities.

In Section 2, the 3D Tom Thumb algorithm will be described by means of a minimal structure composed of eight cells which will grow and then self-replicate for triggering the growth of three identical structures. This example is sufficient for deriving the detailed architecture of a three-dimensional data and signals cellular automaton (DSCA) [9]. The specifications of this three-dimensional, seven-neighbor DSCA and the design of its basic cell are described in Section 3. Section 4 presents the hardware implementation of 3D self-replicating structures in the BioCube, our new 3D reconfigurable electronic medium for bio-inspired systems. Section 5 will conclude by opening new avenues based on the self-replication of 3D universal structures.

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 165–176, 2005.

2 The 3D Tom Thumb Algorithm

The minimal 3D structure is made up of eight cells organized as a $2 \times 2 \times 2$ array. In order to show the growth and the self-replication of this minimal structure, we introduce 2D graphical representations. In Figure 1, the eight cells of the minimal structure are organized as two levels $L = 1$ and $L = 2$ of two rows by two columns. Each cell is able to store in its four memory positions four configuration data. The original configuration information is a string of 16 data moving counterclockwise by one data at each time step ($t = 0, 1, 2, ...$).

The graphical representation as well as the hexadecimal representation of the data composing the configuration string are detailed in Figure 2. They are either *empty data* (0), *code data* (from 1 to E) or *flag data* (from 1 to 9 in addition to F). The code data is used to define the functionality of the structure. The flag data is used to build the connections between the cells of the structure and to create branches for self-replication. Furthermore, each data is given a status and corresponds eventually to a *mobile data*, indefinitely moving around the structure, or a *fixed data*, definitely trapped in a memory position of a cell.

At each time step, a data of the original configuration string is shifted from right to left and simultaneously stored in the lower leftmost cell (Figure 1). Note that the first, third, ... data of the string (i.e. each odd data) is always a flag

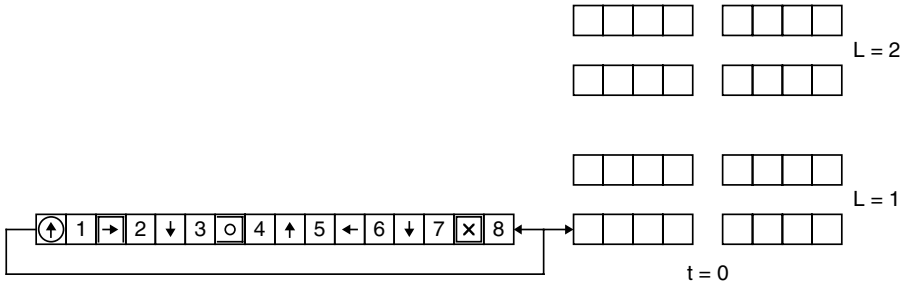


Fig. 1. 2D representation of the minimal 3D structure ($2 \times 2 \times 2$ cells) with its configuration string at the start ($t = 0$)

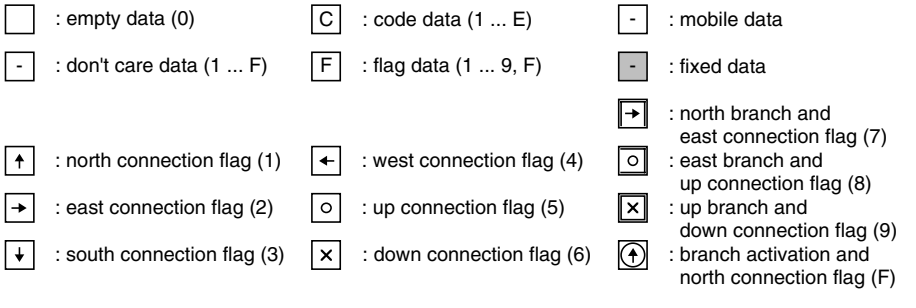


Fig. 2. Graphical and hexadecimal representations of the data

F, while the second, fourth, ... data (i.e. each even data) is always a code C. According to the Tom Thumb algorithm [5], the construction of the structure, i.e. storing the fixed data and defining the paths for mobile data, depends on two major patterns (Figure 3).

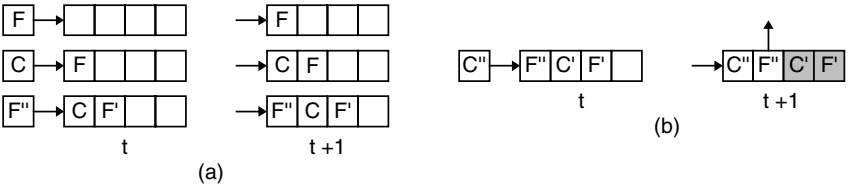


Fig. 3. Memory patterns for constructing a structure. (a) Shift data. (b) Load data.

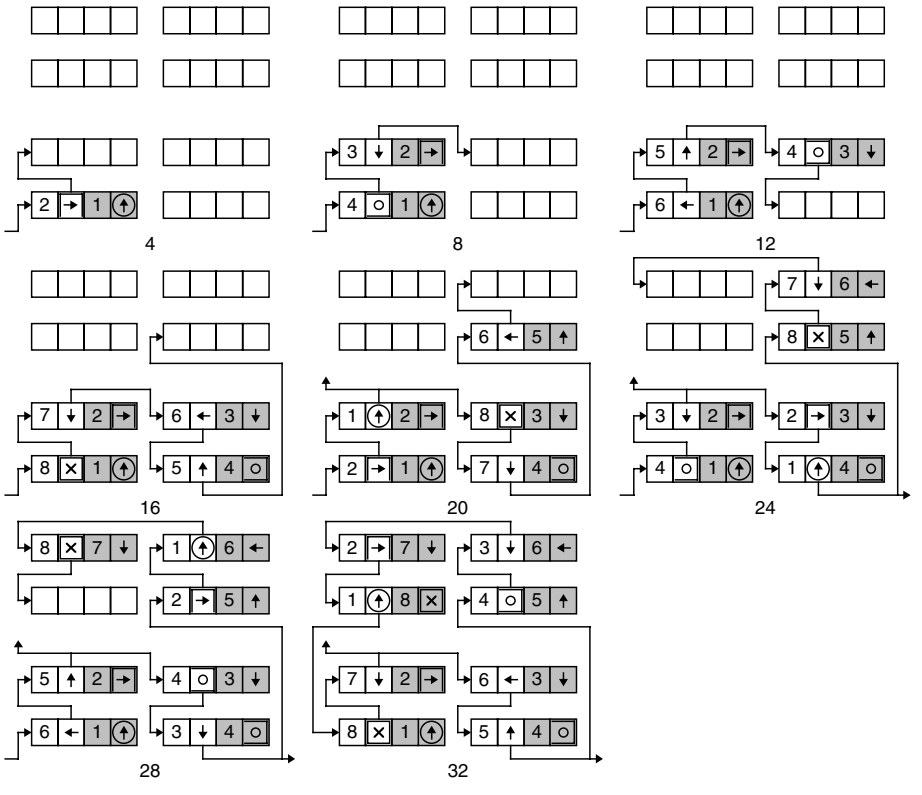


Fig. 4. Constructing the minimal structure ($t = 4$: north path, $t = 8$: east path, $t = 12$: south path, $t = 16$: up path, $t = 20$: north path ($L = 2$) and north branch ($L = 1$), $t = 24$: west path ($L = 2$) and east branch ($L = 1$), $t = 28$: south path, $t = 32$: down path and structure completion).

- If the two, three or four rightmost memory positions of a cell are empty (blank squares), the data are shifted by one position to the right (shift data).
- If the rightmost memory position is empty, the data are shifted by one position to the right (load data). In this situation, the rightmost F' and C' data are trapped in the cell (fixed data), and a new connection is established from the second leftmost position toward the northward, eastward, southward, westward, upward or downward cell, depending on the fixed flag information ($F' = 1$ or F, 2 or 7, 3, 4, 5 or 8, 6 or 9).

Applying the memory patterns of Figure 3 to our original configuration string, we get two data trapped in a cell and a new connection toward another cell of the structure every four time steps (Figure 4). At time $t = 32$, 32 data, i.e. twice the contents of the original configuration, have been stored in the 32 memory positions of the final structure. 16 data are fixed data, defining both its structure and its functionality, and the 16 remaining ones are mobile data, composing a copy of the original configuration information. *Translation* (i.e. construction of the structure) as well as *transcription* (i.e. copy of the configuration) have been therefore achieved.

In order to self-replicate, the original structure is able to trigger the construction of three copies, nothward, eastward and upward. At time $t = 19$, the pattern of data initiates the construction of the northward structure. In this pattern, the lower level upper leftmost cell is characterized by two specific flags, i.e. a fixed flag indicating a north branch ($F = 7$) and the branch activation flag ($F = \mathrm{F}$). This pattern is visible in Figure 5a (third row). The new path to the northward structure starts from the second leftmost memory position (Figure 4). At time $t = 23$ and $t = 47$, the patterns corresponding to the third row of the eastward and upward signals in Figure 5b and e initiate self-replication of the structure to the east and to the top respectively. The other patterns are needed for constructing the inner paths of the structure.

The self-replicating structure in Figure 6 is an example of a non minimal four-column, three-rows and three-level structure. All the non minimal structures can be realized according to this implementation which keeps the number of column even in order to properly close the data path. These non minimal structures involve a new flag (Figure 7) and two more construction patterns (Figure 8).

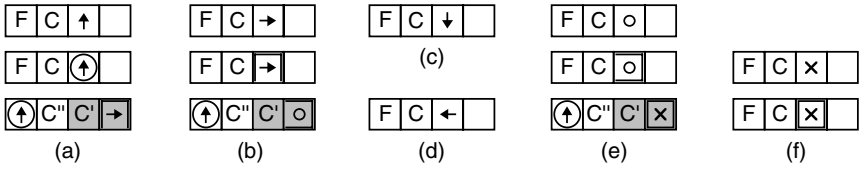


Fig. 5. Patterns of data triggering the path signals. (a) Northward. (b) Eastward. (c) Southward. (d) Westward. (e) Upward. (f) Downward.

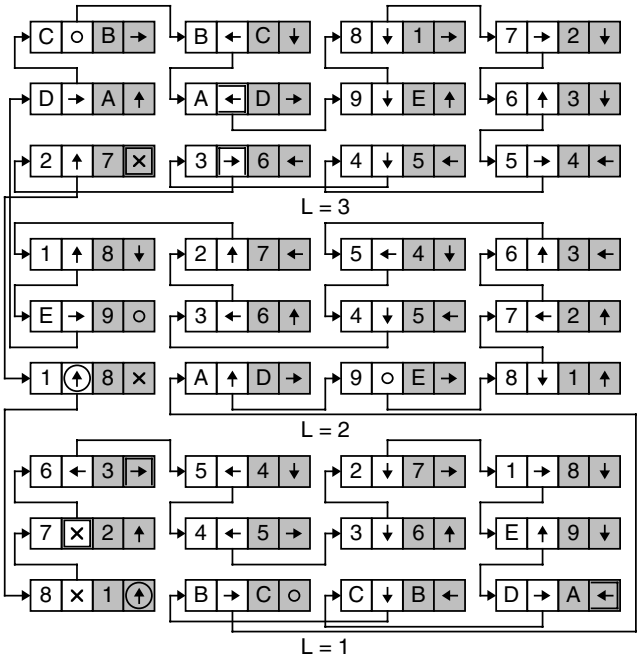


Fig. 6. Example of a non minimal structure ($4 \times 3 \times 3$ cells)

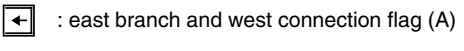


Fig. 7. Graphical and hexadecimal representations of the additional data

Fig. 8. Additional patterns of data triggering the path signals. (a) Westward. (b) Eastward.

3 The Data and Signals Cellular Automaton (DSCA)

Data and signals cellular automata (DSCA) were originally conceived to provide a formal framework for designing growing structures [8], [7]. Such an automaton is made up of an array of cells, each of which is implemented as a digital system processing both data and signals in discrete time steps. The cellular array (grid) is n-dimensional, where $n = 1, 2, 3$ is used in practice.

In growing structures, the data and the signals represents two different types of information. The *data* constitute the information that travels through the

grown structure. The *signals* constitute the information that controls the growth of the structure.

The basic cell of our three-dimensional seven-neighbor DSCA works with the northward (N), eastward (E), southward (S), westward (W), upward (U) and downward (D) directed data (D) and signals (S) (Figure 9). The cell computes its digital outputs O from its digital inputs I.

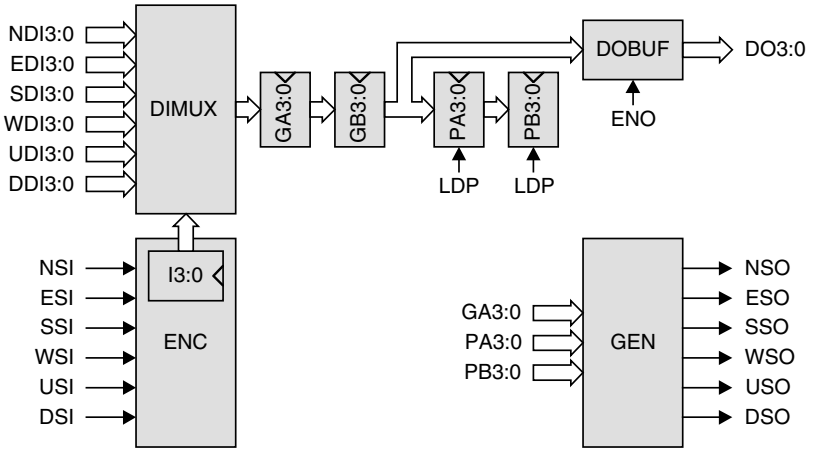


Fig. 9. Basic cell of the three-dimensional seven-neighbor DSCA

This cell is designed as a *digital system*, resulting from the interconnection of a data processing unit and a control unit. The *processing unit* handles the data. It is made up of the following resources:

- A 6-input multiplexer DIMUX for the selection of one of the six data input lines, $NDI3:0$, $EDI3:0$, $SDI3:0$, $WDI3:0$, $UDI3:0$, or $DDI3:0$.
- A 4-level stack interconnecting two 4-bit registers GA3:0 and GB3:0 for the propagation of the configuration data with two 4-bit registers PA3:0 and PB3:0 for the memorization of the configuration data.
- A buffer DOBUF to enable the data output $DO3:0$.

The *control unit* computes the signals. It combines three resources:

- A signal inputs SI encoder ENC.
- A 4-bit data input register I3:0 for the memorization of the selection operated by the multiplexer DIMUX.
- A signal outputs SO generator GEN.

The operation tables of the data input multiplexer, the stack registers, the data output buffer and the data input register are given in Appendix A. This appendix presents also the truth table of the encoder as well as the logic equations of the generator.

4 The BioCube Hardware Implementation

The BioCube is intended as a reconfigurable 3D computing structure capable of interacting with its environment by means of touch-sensitive elements coupled with LED displays. Its structure is inspired from cellular organisms, who are 3D cell arrays.

The BioCube contains 64 units, arranged as 4 layers of 4 rows by 4 columns. Each unit, integrated inside a plastic bubble, has 6 pipe links with the neighboring units. Figure 10 shows the structure of the BioCube.

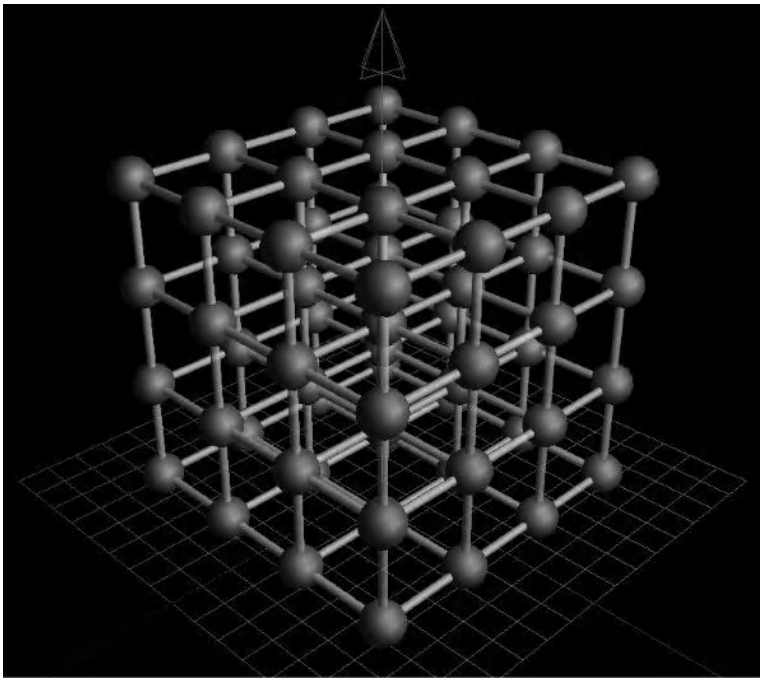


Fig. 10. The BioCube 3D computing structure

Each BioCube unit is made up the following parts:

- A proximity sensor acting as an input element able to detect a finger touching any part of the plastic bubble.
- A three color LED serving as an output element designed to illuminate the sphere uniformly with one from 16'777'216 colors (8 bits for each basic color).
- A reconfigurable computing element implemented as an FPGA (Spartan 3 XC3S200 Xilinx FPGA) with 200'000 equivalent logic gates. Some of its interesting features are the embedded 18x18 multipliers, the several 18Kb Block Ram and the four digital clock managers (DCM) who give the possibility to multiply the clock frequency.

- A flash memory and a CPLD. The flash memory can save up to four different FPGA configurations. The CPLD only serves to program the FPGA with one of these configurations.
- A 50Mhz input clock that the FPGA can multiply up to 300 Mhz with one internal DCM .
- I/O connections with the neighboring units made as 10 wire cables in order to limit the routing from bubble to bubble. Communications using serial data transfers, the signals are multiplexed in each FPGA according to these serial links.

The BioCube is placed on a table for structure robustness purpose and in order to supply the power to its 64 units. This system is controlled by a computer. A simple software and a specific interface allow the synchronization of the 64 units with an external clock driven by the PC. The software performs a second task in sending new FPGA configurations to the units that can be saved in flash memory slots. An interesting feature is the possibility to run different FPGA configurations in each bubble. With this option, we can, for example, configure all the internal bubbles with one application, and the external bubbles with another one. The BioCube logic complexity amounts to a total of about 13 mio. reconfigurable gates.

Figure 11 shows the 3D self-replication of the $2 \times 2 \times 2$ minimal loop. For demonstration purpose, a trigger was added in order to launch the self-replication process. The copy of the loop in a given direction is therefore activated by touching some specific bubbles.

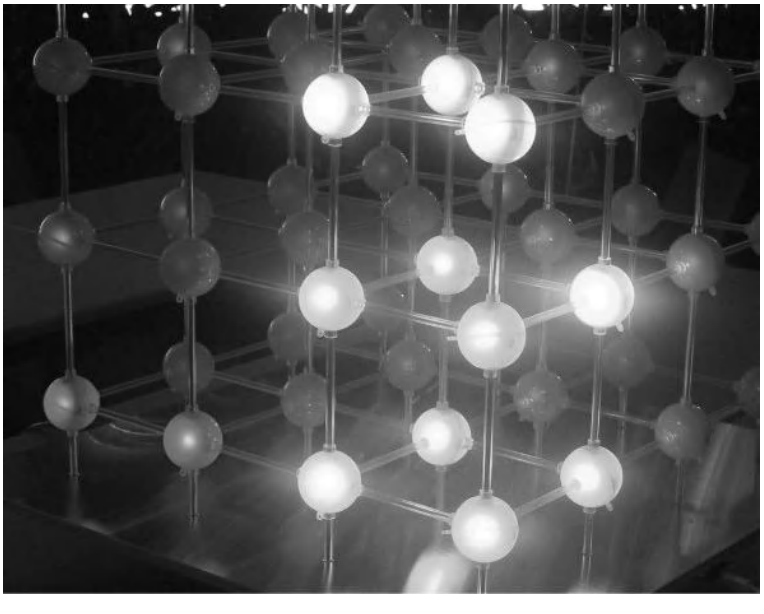


Fig. 11. The $2 \times 2 \times 2$ minimal loop and its upward 3D self-replication

5 Conclusion

Several years before the publication of the historical paper by Crick and Watson [10] revealing the existence and the detailed architecture of the DNA double helix, von Neumann was already able to point out that self-replication was a two-mode process able to both interpret (translation mode) and copy (transcription mode) a one-dimensional description, the configuration string. Self-replication will allow not only to grow, but also to repair complete 3D structures. Self-replication is now considered as a central mechanism indispensable for circuits that will be implemented through the nascent field of nanotechnologies [1] particularly when fault-tolerant properties associated with developmental approaches are taken into consideration.

A first field of application of the self-replication of 3D structures is quite naturally the classical self-replicating automata, such as three-dimensional reversible automata [2] or asynchronous cellular automata [6].

A second, and possibly more important field of application is Embryonics, where artificial multicellular organisms are based on the growth of a cluster of cells, themselves produced by cellular division and cellular differentiation [3] [4].

Other possible open avenues are in the field of *autonomic computing*. An *autonomic computer* needs to possess many characteristics of the living being, like the ability to healing itself for instance. This means that the autonomic system has to know its own resources in order to reconfigure itself and to call up redundant elements in case of a malfunction. By borrowing from living creatures three principles of organization (multicellular organization, cellular division, and cellular differentiation), we plan to design bio-inspired computing machines able to grow, to self-replicate and self-repair, thus bringing an original solution to the new era of 3D nanoscale autonomic computers.

References

1. K. E. Drexler. Nanosystems: Molecular Machinery, Manufacturing, and Computation. John Wiley, New York, 1992.
2. K. Imai, T. Hori, and K. Morita. Self-reproduction in three-dimensional reversible cellular space. Artificial Life, 8(2):155–174, 2002.
3. N. J. Macias and L. J. K. Durbeck. Self-assembling circuits with autonomous fault handling. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. S. Zebulum, editors, Proceedings of the 2002 NASA/DOD Workshop Conference on Evolvable Hardware, pages 46–55, Los Alamitos, CA, 2002. IEEE Computer Society Press.
4. D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Toward robust integrated circuits: The Embryonics approach. Proceedings of the IEEE, 88(4):516–541, April 2000.
5. D. Mange, A. Stauffer, E. Petraglio, and G. Tempesti. Self-replicating loop with universal construction. Physica D, 191(1-2):178–192, April 2004.
6. C. L. Nehaniv. Self-reproduction in asynchronous cellular automaton. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. S. Zebulum, editors, Proceedings of the 2002 NASA/DOD Workshop Conference on Evolvable Hardware, pages 201–209, Los Alamitos, CA, 2002. IEEE Computer Society Press.

7. A. Stauffer and M. Sipper. Biomorphs implemented as a data and signals cellular automaton. In W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kin, and J. Ziegler, editors, Proceedings of the 7th European Conference on Artificial Life (ECAL 2003), volume 2801 of Advances in Artificial Life, pages 235–241, Berlin Heidelberg, 2003. Springer-Verlag.
8. A. Stauffer and M. Sipper. Data and signals: A new kind of cellular automaton for growing systems. In J. Lohn, R. Zebulum, J. Steincamp, D. Keymeulen, A. Stoica, and M. I. Ferguson, editors, Proceedings of the 2003 NASA/DOD Conference on Evolvable Hardware, pages 235–241, Los Alamitos, CA, 2003. IEEE Computer Society.
9. A. Stauffer and M. Sipper. The data-and-signals cellular automaton and its application to growing structures. Artificial Life, 10(4):463–477, 2004.
10. J. D.Watson and F. H. C. Crick. A structure for desoxyribose nucleid acid. Nature, 171:737–738, 1953.

Appendix A: Tables and Equations of the DSCA Basic Cell

Figures 12 to 17 show the individual tables describing the input multiplexer, the stack registers, the output buffer, the input register, and the signal encoder.

The generator implements the output signals according to the following equations where PBZ represents an empty data (0) in the memorization register PB3:0 and GAF a branch activation and north connection flag (F) in the propagation register GA3:0.

$$\begin{aligned} NSO = {} & PBZ.PA3'.PA2'.PA1'.PA0 \\ & + PBZ.PA3.PA2.PA1.PA0 \\ & + GAF.PB3'.PB2.PB1.PB0 \end{aligned} \tag{1}$$

$$\begin{aligned} ESO = {} & PBZ.PA3'.PA2'.PA1.PA0' \\ & + PBZ.PA3'.PA2.PA1.PA0 \\ & + GAF.PB3.PB2'.PB1'.PB0' \\ & + GAF.PB3.PB2'.PB1.PB0' \end{aligned} \tag{2}$$

$$SSO = PBZ.PA3'.PA2'.PA1.PA0 \tag{3}$$

$$\begin{aligned} WSO = {} & PBZ.PA3'.PA2.PA1'.PA0' \\ & + PBZ.PA3.PA2'.PA1.PA0' \end{aligned} \tag{4}$$

$$\begin{aligned} USO = {} & PBZ.PA3'.PA2.PA1'.PA0 \\ & + PBZ.PA3.PA2'.PA1'.PA0' \\ & + GAF.PB3.PB2'.PB1'.PB0 \end{aligned} \tag{5}$$

$$\begin{aligned} DSO = {} & PBZ.PA3'.PA2.PA1.PA0' \\ & + PBZ.PA3.PA2'.PA1'.PA0 \end{aligned} \tag{6}$$

operation	description	I3:0
ZERO	DI = 0	0 0 0 0
SELECT NDI	DI = NDI	1 0 0 0
SELECT EDI	DI = EDI	1 0 0 1
SELECT SDI	DI = SDI	1 0 1 0
SELECT WDI	DI = WDI	1 0 1 1
SELECT UDI	DI = UDI	1 1 0 0
SELECT DDI	DI = DDI	1 1 0 1

Fig. 12. Operation table of the data input multiplexer DIMUX

operation	description
LOAD	GA <= DI GB <= GA

Fig. 13. Operation table of the propagation registers GA3:0 and GB3:0

operation	description	LDP
HOLD	PA <= PA PB <= PB	0
LOAD	PA <= GB PB <= PA	1

Fig. 14. Operation table of the memorization registers PA3:0 and PB3:0 ($LDP = PB3'.PB2'.PB1'.PB0'$)

operation	description	ENO
ZERO	DO = 0	0
TRUE	DO = GB	1

Fig. 15. Operation table of the data output buffer DOBUF ($ENO = PB3 + PB2 + PB1 + PB0$

ID	DSI	NSI	ESI	USI	SSI	WSI	D3:0
1	-	-	-	-	-	-	1 1 0 1
0	1	-	-	-	-	-	1 1 0 1
0	0	1	-	-	-	-	1 0 0 0
0	0	0	1	-	-	-	1 0 0 1
0	0	0	0	1	-	-	1 1 0 0
0	-	-	-	-	1	-	1 0 1 0
0	-	-	-	-	-	1	1 0 1 1

Fig. 16. Truth table of the encoder ENC ($ID = I3.I2.I1'.I0$)

operation	description	LDI
HOLD	I <= I	0
LOAD	I <= D	1

Fig. 17. Operation table of the data input register I3:0 ($LDI = NSI + ESI + SSI + WSI + USI + DSI$)

POEtic: A Prototyping Platform for Bio-inspired Hardware

J. Manuel Moreno[1], Yann Thoma[2], and Eduardo Sanchez[2]

[1] Technical University of Catalunya, Dept. of Electronic Engineering,
Campus Nord, Building C4, c/Jordi Girona 1-3, 08034-Barcelona, Spain
moreno@eel.upc.edu
[2] Swiss Federal Institute of Technology Lausanne, Logic Systems Laboratory,
IN-Ecublens, CH-1015, Switzerland
{Yann.Thoma, Eduardo.Sanchez}@epfl.ch

Abstract. This paper will present the final hardware realization of a new family of programmable devices that has specifically being conceived in order to address the prototyping of bio-inspired principles. The devices are organized around a custom 32-bit RISC microprocessor and a custom FPGA. The internal architecture devised for the devices is scalable, so that it is possible to construct a physical hardware platform whose size matches the requirements of the application to be handled. To facilitate the development of applications for this hardware platform a complete set of design tools has been developed.

1 Introduction

During the last years standard programmable devices have been extensively used to provide physical implementations for bio-inspired principles, either as a direct substrate [1] or as a supporting platform for extended architectures [2], [3], [4]. Even some custom programmable architectures [5], [6] have been developed in order to provide an efficient substrate for the realization of these principles. However, there is still a lack of an integrated system able to offer at the same time the basic features required to implement actual autonomous bio-inspired hardware:

- Partial dynamic reconfiguration, i.e., the ability to modify the functionality of a section of the design while it is in normal operation and with a delay comparable to the execution delay of the system. Even if this capability is being offered by the programmable devices offered by Xilinx [7] and Atmel [8] it is usually limited by the lack of information about the physical configuration string or by the granularity of the reconfiguration area.
- Self-configuration, i.e., the capability of the programmable device of modifying its functionality using its own resources. This feature is already present in the Cell Matrix devices [5].
- Dynamic routing, i.e., the possibility of changing in real time the connectivity between the elementary programmable cells included in the system without the need for an external compiler.

The main goal of the POEtic project [9] was the development of a flexible hardware substrate able to provide capabilities similar to those present in living beings,

J. M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 177 – 187, 2005.

like evolution, development, self-replication, self-repair and learning. One of the major outcomes of the project was an integrated programmable electronic system, the POEtic chip, that provides in a single device the three features mentioned above: partial dynamic reconfiguration, self-configuration and dynamic routing. We shall demonstrate that the combination of these capabilities in a single hardware substrate provides an efficient platform for the prototyping and development of artifacts based on bio-inspired principles.

The rest of the paper is organized as follows: in the next section we shall present the major features included in the architecture conceived for the POEtic devices. Then we shall review the actual physical implementation of the device and the development environment that has been built around it. Finally, the conclusions and our current work will be outlined.

2 The POEtic Architecture

The structural organization of a POEtic chip is represented in Fig. 1.

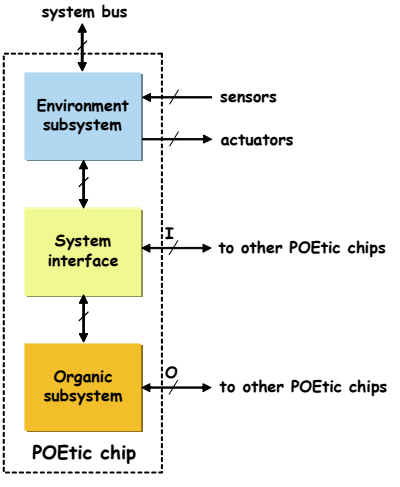


Fig. 1. Organization of the POEtic chip

As it can be deduced from this figure, a POEtic chip is constituted by three main building blocks:

- **Environment subsystem:** This is the component of the tissue that is in charge of managing the interaction with the environment. This interaction can be considered at two different time scales: on-line interaction and evolution (phylogenesis). The on-line interaction refers to the continuous process by means of which a given individual implemented in the tissue is sensitive to the input stimuli that arrive from the external environment. These stimuli may take the form of any physical magnitude (light, pressure, temperature, …), and after a conditioning and conversion processes are translated into in-

ternal signals that may be used by the individual either to extract some knowledge from the environment or to produce an output as a result of some internal processing. These output signals may be later translated, by means of a set of proper actuators, into physical magnitudes that are reverted as output actions to the environment. This on-line interaction constitutes the basic sensor-actuator loop that permits a given individual to adapt its behaviour to the specific characteristics of the environment where it is placed. The second kind of interaction with the environment acts at a population level and exceeds the life time of an individual. In this case the sensor-actuator loop is used to define the basic substrate (the genome) of the individuals that are capable of adapting its behaviour to the environment in the most efficient way according to a given fitness measure.

- **Organic subsystem:** It is in charge of implementing the behaviour of an individual, following the principles described by the innate information that has resulted from the evolutionary process. Therefore, it is the goal of this system to permit the development (ontogenesis) of a given functionality from the information stored in a genome, and also to permit the adaptation (epigenesis) of this functionality according to the stimuli received from the environment.
- **System interface:** This element will allow for an efficient communication between the environment and the organic subsystem of the tissue. It also constitutes the substrate that will provide the basic mechanisms that will permit the scalability of the tissue.

2.1 The Environment Subsystem and System Interface

The environment subsystem of the POEtic tissue has been built around a custom 32-bit microprocessor with an efficient and flexible system bus, based on the AMBA specification [10], and several custom peripherals. The reason for using a centralised system to carry out evolutionary processes is motivated by the fact that, even if evolution acts on a population of individuals, at the end there must be a global unit that should evaluate the fitness of the individuals and determine those from which the next population has to be constructed. Therefore, the functionality of the individuals will be implemented in the organic subsystem, but it is the microprocessor that constitutes the core of the environment subsystem that will drive the basic steps of the evolutionary process, as well as the interaction of the individuals with the environment. Additionally, the use of a programmable unit to implement the phylogenetic mechanisms of the tissue will permit to test and develop different evolutionary strategies, since this will imply just an update of the software executed by the microprocessor. Finally, this alternative will largely simplify the management of the acquisition/conversion units that are required to handle the sensor-actuator loop needed to complete the epigenetic and phylogenetic processes to be implemented by the tissue.

The system interface of the Poetic device plays a major role in allowing for its scalability features. This means that it is possible to connect several POEtic chips in order to construct an electronic tissue whose size can be accommodated to the actual needs of a given application without posing specific constraints neither on the system

architecture nor in the connectivity pattern among the POEtic chips that constitute the tissue.

In this way, a POEtic tissue can be constructed as a bidimensional array constituted by POEtic chips. The connectivity between these chips is based on two different buses, named organic (O) and interface (I) buses. The signals that constitute the organic bus permit to communicate (at a cellular level) the organic subsystems present in every POEtic chip. The interface bus carries those signals that permit to handle the collection of POEtic chips as a single tissue, so that from a user point of view the tissue has only one environment subsystem and one organic subsystem. This is represented in Fig. 2.

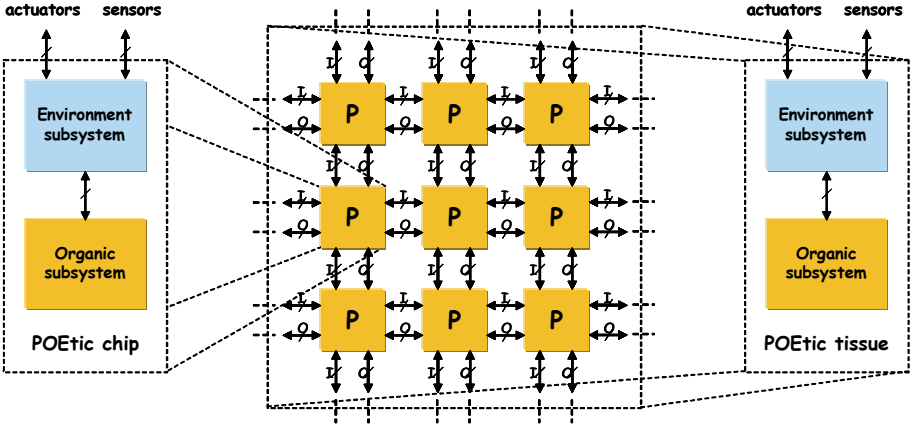


Fig. 2. Scalability properties of the POEtic tissue

2.2 The Organic Subsystem

The organic subsystem of the POEtic device is made up of 2 layers, as depicted in Fig. 3: a two-dimensional array of basic elements, called molecules, and a two-dimensional array of routing units. Each molecule is connected to its four neighbours in a regular structure. Mainly containing a 16-bit look-up table (LUT) and a flip-flop (DFF), it has the capability of accessing the routing layer that is used for inter-cellular communication. This second layer implements a dynamic routing algorithm allowing the creation of data paths between cells at runtime.

A molecule has eight different operational modes, to speed up some operations, and to use the routing plane.

- In **4-LUT** mode, the 16-bit LUT supplies an output, depending on its four inputs.
- In **3-LUT** mode, the LUT is split into two 8-bit LUTs, both supplying a result depending on three inputs. The first result can go through the flip-flop, and is the first output. The second one can be used as a second output, and is directly sent to the south neighbor (can serve as a carry in parallel operations).

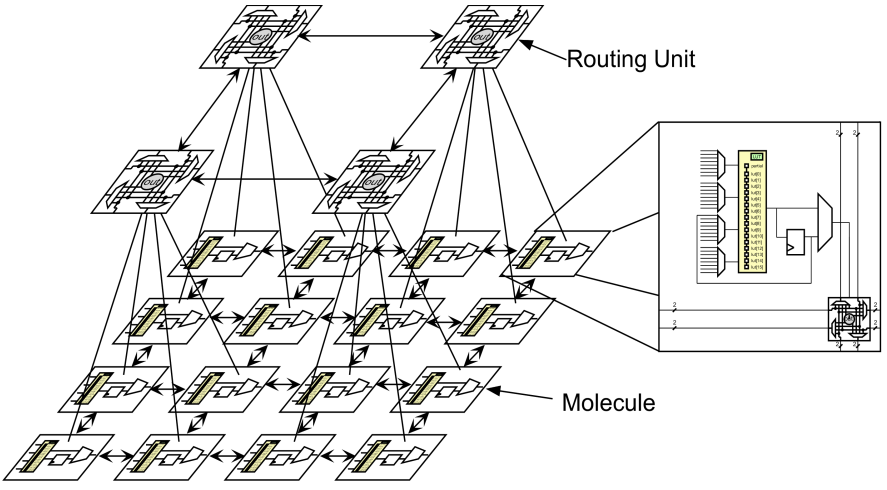


Fig. 3. Organization of the organic subsystem

- In **Comm** mode, the LUT is split into one 8-bit LUT, and one 8-bit shift register. This mode could be used to compare a serial input data with a data stored in the 8-bit shift register.
- In **Shift Memory** mode, the 16 bits are used as a shift register, in order to store data, for example a genome. One input controls the shift, and another one is the input of the shift memory.
- In **Input** mode, the molecule is a cellular input, connected to the inter-cellular routing plane.
- In **Output** mode, the molecule is a cellular output, connected to the inter-cellular routing plane.
- In **Trigger** mode, the 16-bit shift register should contain "000...01" for a 16-bit identifier system. It is used by the routing plane to synchronize the identifier decoding during the routing process.
- In **Configure** mode, the molecule can partially configure its neighborhood. One input is the configuration control signal, and another one is the configuration shifting to the neighbors.

The configuration of the device can be made in a parallel manner, through a 32-bit bus. The 76 configuration bits of a molecule are split into three 32-bit words. Additionally, the configuration system of the molecules can be seen as a shift register of 76 bits split into 5 blocks: the LUT, the selection of the LUT's input, the switch box, the mode of operation, and an extra block for all other configuration bits. Each block contains, as shown in Fig. 4, together with its configuration, one bit indicating, in case of a reconfiguration coming from a neighbour, if the block has to be bypassed. This bit can only be loaded from the microprocessor.

The special configure mode allows a molecule to partially reconfigure its neighbourhood. It sends bits coming from another molecule to the configuration of one of its neighbours. By chaining the configurations of neighbouring molecules, it is

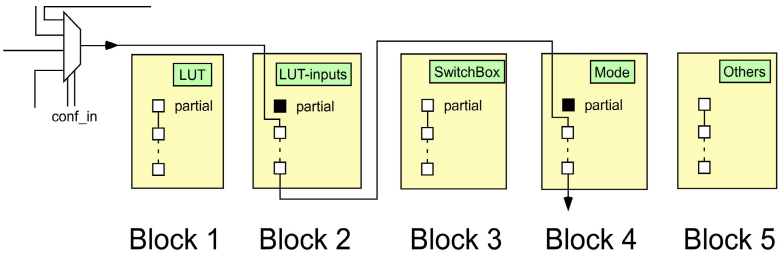


Fig. 4. Organization of the configuration bits for partial reconfiguration

possible to modify multiple molecules at the same time, allowing, for example, the synaptic weights in a neuron to be changed. Moreover, this mechanism permits to use up to 54 of the configuration bits to store information, that can be accessed serially.

2.3 Dynamic Routing

The dynamic routing system is designed to automatically connect the cells' inputs and outputs. Each output of a cell has a unique identifier. For each of its inputs, the cell stores the identifier of the source from which it needs information. A non-connected input (target) or output (source) can initiate the creation of a path by broadcasting its identifier, in case of an output, or the identifier of its source, in case of an input. The path is then created using a parallel implementation of the breadth-first search algorithm. When all paths have been created, the organism can start operation, and execute its task, until a new routing is launched, for example after a cell addition or a cellular self-repair.

Our approach has many advantages, compared to a static routing process. First of all, a software implementation of a shortest path algorithm, such as Lee's [11], is very time-consuming for a processor, while our parallel implementation requires a very small number of clock cycles to finalize a path. Secondly, when a new cell is created it can start a routing process, without the need of recalculating all paths already created. Thirdly, a cell has the possibility of restarting the routing process of the entire organism, if needed (for instance after a self-repair). Finally, our approach is totally distributed, without any global control over the routing process, so that the algorithm can work without the need of the central micro-processor.

The routing algorithm is executed in four phases:

Phase 1: Finding a Master

In this phase, every target or source that wants to and is not connected to its correspondent partner tries to become master of the routing process. A simple priority mechanism chooses the most bottom-left routing unit to be the master, as shown in Fig. 5. Note that there is no global control for this priority, every routing unit knowing whether or not it is the master. This phase is over in one clock cycle, as the propagation of signals is combinational.

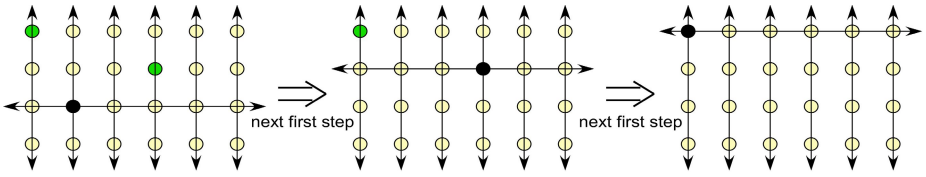


Fig. 5. Three consecutive steps of the routing algorithm. The black routing unit will be the master, and therefore will perform its routing.

Phase 2: Broadcasting the Address
Once a master has been selected, it sends its address in case of a source, or the address of the needed source in case of a target. It is sent serially, in `n` clock cycles, where `n` is the size of the address. The same path as in the first phase is used to broadcast the address, as shown in Fig. 6.

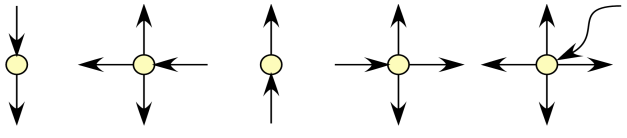


Fig. 6. The propagation direction of the address: north → south | east → south, west, and north | south → north | west → north, east, and south | routing unit → north, east, south, and west

Every routing unit, except the one that sends the address, compares the incoming value with its own address (stored in the molecule underneath). At the end of this phase, that is, after `n` clock cycles, each routing unit knows if it is involved in this path. In practice, there has to be one and only one source, and at least one target.

Phase 3: Eliminating sources and targets
In some situations, a source should start a routing process, for instance, in a developmental process. In such a process, it would be useful to have many sources and targets with the same ID. So at this stage, it is possible there is more than one source involved in the routing process. In order to avoid multiple sources, in this phase that lasts only one clock cycle, if a source is at the origin of the routing process, it sends a signal to every other routing unit, to let them know a source is at the origin. Then every other source with the same ID disabled its participation in the current process.

The same disable is performed in case a target launched the routing process. Every target that is not the master disables its participation to the current process, to ensure that the target that started the process will be the only one connected to a source.

Phase 4: Building the Shortest Path
The last phase, largely inspired by [12], creates a shortest path between the selected source and the selected targets. An example involving 8 sources and 8 targets is shown in Fig. 7, for a densely connected network.

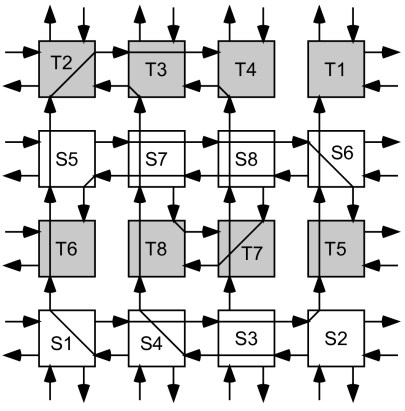


Fig. 7. Test case with a densely connected network

A parallel implementation of the breadth-first search algorithm allows the routing units to find the shortest path between a source and many targets. Starting from the source, an expansion process tries to find targets. When one is reached, the path is fixed, and all the routing resources used for the path will not be available for the next successive iterations of the algorithm.

3 Physical Realization

The POEtic chip has been implemented and fabricated as an ASIC of 54 mm^2 using a 0.35 µm CMOS process. The chip, whose layout is depicted in Fig. 8, contains 144 molecules organized as an 8x18 array and the complete environment subsystem explained previously. Even if implemented using a standard technology the ASIC implementation of the POEtic tissue demonstrates its superior integration capabilities when compared with those offered by standard prototyping platforms (the prototyping experiments performed within the framework of the project show that an FPGA with 3 million system gates capacity is able to implement the functionality of just 80 POEtic molecules).

Specific development boards have been constructed in order to test the POEtic devices and to implement practical applications on them. These are depicted in Fig. 9. Fig. 9(a) represents the master board, containing one POEtic chip, Flash and SDRAM memory blocks and a USB communication unit that permits to create an interface with an external host. Fig. 9(b) depicts the slave board, containing a 2 x 2 array of POEtic chips. The slave board can be attached to the master board, and it is also possible to connect several slave boards between them in order to create an electronic tissue with the required size for the application to be handled.

Since the complete POEtic tissue has been specified and developed using a standard hardware description language (VHDL) it can be implemented in a standard prototyping platform (though with limited functionality due to the capacity restrictions of current programmable devices). Therefore, in order to facilitate the

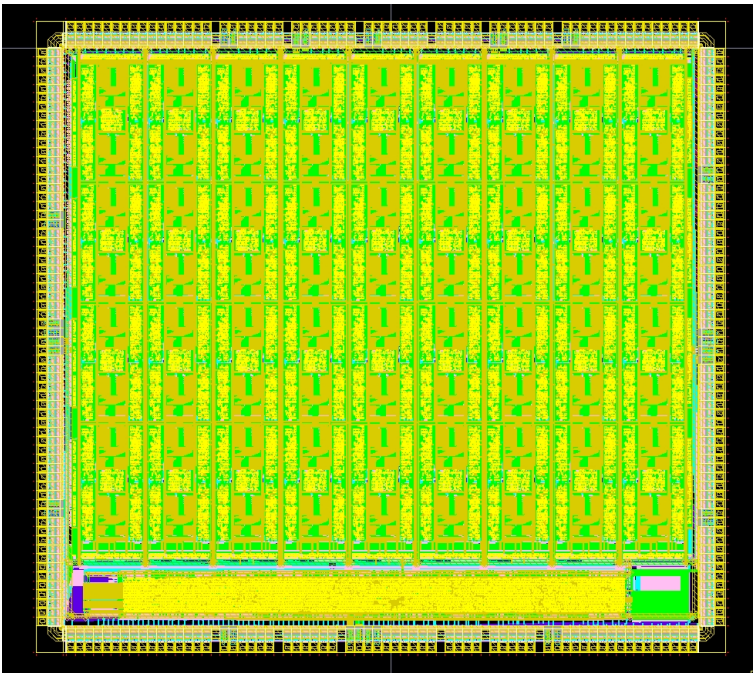


Fig. 8. Layout of the POEtic chip

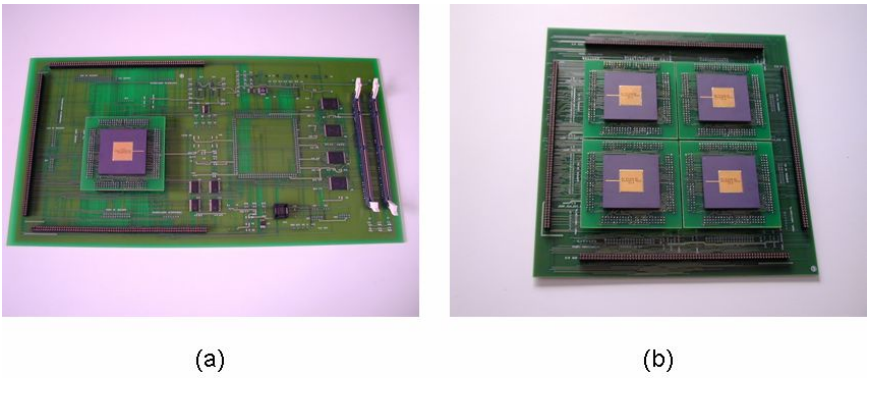


Fig. 9. Details of (a) the master and (b) slave boards developed for the POEtic devices

use of the tissue by external users a complete set of tools have been developed within the framework of the project. This set includes a schematic editor and synthesizer, a molecule-level design entry and simulation tool for the organic subsystem, a C compiler and an assembler integrated in a graphical user interface with language-sensitive editing capabilities, a graphical user interface for the simulation of programs developed for the microprocessor and a system debugger.

4 Conclusions

The POEtic project has produced at its end the first programmable integrated system with capabilities inspired in the organization principles present in living beings: evolution, development/growth, self-replication, self-repair and learning. The resulting electronic device permits to construct a multi-cellular tissue whose size can be adapted to the specific requirements of the application to be handled. The internal architecture of the device includes features, like dynamic partial reconfiguration, self-configuration or in-hardware dynamic routing, that were never combined (if not present at all) in any past electronic device.

In this paper we have presented the architecture that has been conceived for the POEtic devices, as well as the internal organization of its main constituent elements. Then the physical implementation details of the integrated systems and the development boards constructed to create applications based on these devices. The whole system has been described and developed using a standard hardware description language (VHDL). This, together with the set of tools that have been developed for the devices, will permit to test the concepts developed within the project using standard prototyping platforms.

The availability of this brand new family of programmable devices thus opens long term opportunities for the implementation of electronic systems and applications able to take profit of these new features. Among them we could consider the following list:

- Autonomous adaptive systems for deep space exploration.
- Safety critical systems in the aeronautics and the automotive domains.
- Sensor integration for distributed, highly immersive sensor and actuator environments.
- Personalized, user-adaptable assistant systems.
- User-adaptable monitoring and early warning systems for handicapped or elderly people.

Our current work is concentrated in the prototyping of large-scale spiking neural networks models with bio-inspired learning mechanisms using the prototyping platform offered by the POEtic devices.

Acknowledgements

The work presented in this paper has been funded by the grant IST-2000-28027 (POEtic) of the European Community and by grant OFES 00.0529-2 of the Swiss government. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

1. Vinger, K.A., Torresen, J.: Implementing evolution of FIR-filters efficiently in an FPGA. Proceedings of the NASA/DoD Conference on Evolvable Hardware. IEEE Computer Society (2003) 26–29

2. Haddow, P.C., Tufte, G.: Bridging the Genotype-Phenotype Mapping for Digital FPGAs. Proceedings of the NASA/DoD Conference on Evolvable Hardware. IEEE Computer Society (2001) 109-115
3. Sekanina, L: Virtual reconfigurable Circuits for Real-World Applications of Evolvable Hardware. Evolvable Systems: From Biology to hardware. Lecture Notes in Computer Science, Vol. 2606. Springer-Verlag, Berlin Heidelberg New York (2003) 186-197
4. Sekanina, L., Friedl, S.: On Routine Implementation of Virtual Evolvable Devices Using COMBO6. Proceedings of the NASA/DoD Conference on Evolvable Hardware. IEEE Computer Society (2004) 63-70
5. Macias, N.J.: The PIG Paradigm: The design and Use of a Massively Parallel Fine Grained Self-Reconfigurable Infinitely Scalable Architecture. Proceedings of the NASA/DoD Conference on Evolvable Hardware. IEEE Computer Society (1999) 175-180
6. Macias, N.J, Durbeck, L.J.K.: Self-assembling Circuits with Autonomous Fault Handling. Proceedings of the NASA/DoD Conference on Evolvable Hardware. IEEE Computer Society (2002) 46-55
7. Ullmann, M., Hübner, M., Grim, B., Becker, J.: On-Demand FPGA Run-Time System for Dynamical Reconfiguration with Adaptive Priorities. Field Programmable Logic and Applications. Lecture Notes in Computer Science, Vol. 3203. Springer-Verlag, Berlin Heidelberg New York (2003) 454-463
8. Bartosinski, R., Danek, M., Honzik, P., Matousek, R.: Dynamic Reconfiguration in FPGA-based SoC designs. Proceedings of the 2005 ACM/SIGDA 13th international Symposium on Field-programmable gate arrays (2005) 274
9. Tyrrell, A.M., Sanchez, E., Floreano, D., Tempesti, G., Mange, D., Moreno, J.M., Rosenberg, J., Villa, A.E.P.: POEtic Tissue: An Integrated Architecture for Bio-Inspired Hardware. Evolvable Systems: From Biology to Hardware. Lecture Notes in Computer Science, Vol. 2606. Springer-Verlag, Berlin Heidelberg New York (2003)129-140
10. ARM. Amba specification, rev 2.0. advanced risc machines ltd (arm). http://www.arm.com/armtech/amba_spec (1999)
11. Lee. C.Y.: An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, Vol EC-10:3 (1961) 346-365
12. Moreno, J.M., Sanchez, E., Cabestany, J: An in-system routing strategy for evolvable hardware programmable platforms. Proceedings of the NASA/DoD Conference on Evolvable Hardware. IEEE Computer Society (1999) 157-166

Implementation of Biologically Plausible Spiking Neural Networks Models on the POEtic Tissue

J. Manuel Moreno[1], Jan Eriksson[2], Javier Iglesias[2,3], and Alessandro E.P. Villa[3]

[1] Technical University of Catalunya, Dept. of Electronic Engineering,
Campus Nord, Building C4, c/Jordi Girona 1-3, 08034-Barcelona, Spain
moreno@eel.upc.edu
[2] Laboratory of Neuroheuristics, Information Systems Department INFORGE,
University of Lausanne, Lausanne, Switzerland
jan@lnh.unil.ch, Javier.Iglesias@unil.ch
[3] INSERM U318, University Joseph-Fourier Grenoble 1, Pavillon B,
CHUG Michallon, BP217, F-38043 Grenoble Cedex 9, France
Alessandro.Villa@ujf-grenoble.fr
http://www.nhrg.org

Abstract. Recent experimental findings appear to confirm that the nature of the states governing synaptic plasticity is discrete rather than continuous. This means that learning models based on discrete dynamics have more chances to provide a ground basis for modelling the underlying mechanisms associated with plasticity processes in the brain. In this paper we shall present the physical implementation of a learning model for Spiking Neural Networks (SNN) that is based on discrete learning variables. After optimizing the model to facilitate its hardware realization it is physically mapped on the POEtic tissue, a flexible hardware platform for the implementation of bio-inspired models. The implementation estimates obtained show that is possible to conceive a large-scale implementation of the model able to handle real-time visual recognition tasks.

1 Introduction

Among the different types of artificial neural networks models that have been investigated during the last decades spiking neural networks have attracted large research efforts [1], [2] because of their biological plausibility and their suitability for a physical hardware implementation. These neural paradigms usually consider a simplified model for the neuron that is based in an integration process for its inputs and the delivery of an output spike when the membrane potential exceeds a given threshold.

Among different learning mechanisms Spike Timing Dependent Plasticity (STDP), i.e., the modification of the synaptic weights depending on the time correlation between pre- and post-synaptic spikes, has raised an increasing interest [3] due to experimental evidence [4] and observations suggesting that synaptic plasticity may be based on discrete dynamics [5].

In this paper we shall consider a spiking neural network model [6] based on STDP learning rules whose learning dynamics is based on discrete variables. This model has demonstrated excellent properties for discriminating dynamic input stimuli in large-scale networks [7].

J. M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 188 – 197, 2005.

The rest of the paper is structured as follows: in the next section we shall provide a brief summary of the learning scheme proposed in the considered model. Then we shall provide the hardware implementation of this model and the procedure for its functional validation. The accuracy of the internal variables used in the model is then scaled down to allow for a compact hardware implementation. After validating this optimization the resulting model is implemented using the POEtic tissue, a prototyping platform for bio-inspired models. Finally, the conclusions and our current development work are outlined.

2 A Biologically Inspired SNN Model

The model consists of Leaky Integrate-and-Fire neuromimes connected by synapses with variable weight depending on the time correlation between pre- and post-synaptic spikes. The synaptic potentials are added until their result $V_i(t)$ overcomes a certain threshold, θ. Then a spike is produced, and the membrane value is reset. The simplified equation of the membrane value is:

$$V_i(t+1) = \begin{cases} 0 & when\ S_i(t) = 1 \\ k_{mem} \cdot V_i(t) + \sum J_{ij}(t) & when\ S_i(t) = 0 \end{cases} \quad (1)$$

where $k_{mem}=exp(-\Delta t/\tau_{mem})$, $Vi(t)$ is the value of the membrane and $S_i(t)$ is the state variable which signals the occurrence of a spike. The value of ,J_{ij} is the output of each synapse (ij) where j is the projecting neuron and i is the actual neuron.

When a spike occurs in the pre-synaptic neuron, the actual value of the synaptic output J_{ij} is added to the weight of the synapse multiplied by an activation variable A. Conversely, if there is no pre-synaptic spike then the output J_{ij} is decremented by a factor k_{syn}. Then, the value of ,J_{ij} corresponds to the following equation:

$$J_{ij}(t+1) = \begin{cases} J_{ij}(t) + (w_{RiRj} \cdot A_{RiRj}(t)) & when\ S_j(t) = 1 \\ k_{syn} \cdot J_{ij}(t) & when\ S_j(t) = 0 \end{cases} \quad (2)$$

where R is the type of the neuron, either excitatory or inhibitory.

If the actual neuron is inhibitory, the factor k_{syn} will reset the output of the synapse after a time step; if the actual neuron is excitatory, the update of the synaptic output depends on the projecting neuron and the STDP rule is applied. An inhibitory cell can not influence another inhibitory cell, i.e. assume a synaptic weight of zero between two inhibitory neurons. The basic synaptic strengths are chosen in order to maintain a balanced excitatory/inhibitory activity within the network.

The changes in strength of an excitatory-excitatory synapse depend on the variable A which is a function of on an internal variable L_{ij} given by the following equation:

$$L_{ij}(t+1)=k_{act} \cdot L_{ij}(t) + (YD_j(t) \cdot S_i(t)) - (YD_i(t) \cdot S_j(t)) \quad (3)$$

where k_{act} is a kinetic activity factor, which is the same for all the synapses and YD is a "learning" decaying variable that depends on the interval between a pre-synaptic spike and a post-synaptic spike. When there is a spike, YD reaches its maximum value

at the next time step. In the absence of a spike the value of *YD* will be decremented by the kinetic factor $k_{learn,}$ which is the same for all synapses. When a pre-synaptic spike occurs just before a post-synaptic spike, then the variable L_{ij} is increased and the synaptic strength becomes larger, thus corresponding to a potentiation of the synapse. When a pre-synaptic spike occurs just after a post-synaptic spike, the variable L_{ij} is decreased, the synaptic weight is weakened , thus corresponding to a depression of the synapse. For all kind of synapses, except the excitatory-excitatory, the activation variable is always is set to 1.

The network layout was chosen with 80% of excitatory and 20% inhibitory neurons. Each unit was fully connected within a 5x5 neighborhood, i.e. connected to 24 neurons (Fig. 1).

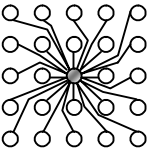


Fig. 1. Connectivity of a single neuron

3 Hardware Implementation

From a structural point of view the SNN model considered in this paper is constituted by four main building blocks: the neuron block, the decay block, the learning block and the synapse block.

The **neuron block** is in charge of implementing the dynamics of the membrane by integrating the pre-synaptic spikes, as indicated in Eq. (1). The characteristics of the parameters of this block are the following:

- The membrane potential has a resolution of 12 bits, with a range [-2048, 2047], and the threshold is kept fixed to +640.
- The membrane decay function has a time constant value of τ=20.
- The refractory period is set to 1 time unit.

The **decay block** will be used in both learning and synapse blocks. This block is aimed to implement a logarithmic decay of the input; it is obtained with a subtraction and controlling the time when it is done depending on the input value. This block is used in many parts of the design and the decaying variable has been labeled *x* in Figure 2. A new value of *x* will be the input of a shift register which is controlled by the most significant bit (*MSB*) of *x* and by an external parameter *mpar*. The output of this shift register will be subtracted from the original value of *x*. This operation will be done when the time control indicates it. The time control is implemented by the value of a counter that is compared with the result of choosing between the external value *step* and the product $(MSB–mpar)\cdot step$. The decay variable τ depends on the input parameters *mpar* and *step*.

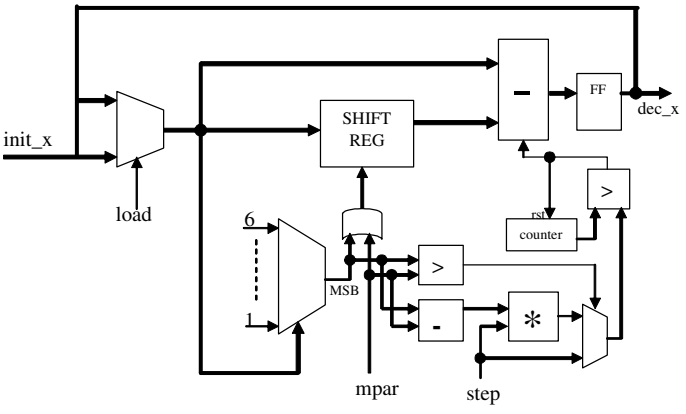


Fig. 2. Diagram of the decay block

The **learning block** "measures" the interval between a spike in the projecting neuron *j* and the actual neuron *i*. Depending on these timings and the types of the two neurons, the synaptic strength will be modified.

When a spike is produced by the projecting neuron, the variable *YD* is set to its maximum value and starts to decay. If a spike is produced by the actual neuron immediately after the presynaptic neuron the value of YD_j is added to the decaying value of *L*. Conversely, if a spike is produced at first in the actual neuron and later in the projecting neuron, then the value of YD_i is subtracted to the decaying value of *L*.

If the *L* variable overcomes a certain threshold L_{th}, positive or negative, then the activation variable *A* is increased or decreased, respectively, unless the variable had reached its maximum or minimum, respectively. If the variable *A* is increased, then *L* is reset to the value $L\text{-}2\cdot L_{th}$; if *A* is decreased, then *L* is reset to $L+2\cdot L_{th}$.

Figure 3 illustrates the organization of the learning block.

The characteristics of the parameters of the learning block are the following:

- The *YD* variable has a resolution of 6 bits.
- The time constant for the variable *YD* is τ=20.
- The learning variable *L* of 8 bits and $\cdot L_{th}$ is within the range [-128,127].
- The activation variable *A* is coded by 2 bits and takes four states.
- To improve the sensitivity of the block for long intervals between spikes the time constant for the variable *L* is set to 4000, but it can be changed depending on the network size implementation.

The **synapse block** is aimed to set the value of *J* (analogous to the the sum of all post-synaptic membrane potentials) and depends on four factors: the activation level *A* of the synapse, the spiking state of the projecting neuron S_j and the types of the pre- and post-synaptic neurons (R_i and R_j).

A given weight is set for each synapse. This weight is multiplied by the activation variable *A* by means of a shift register, such that if *A*=0, the weight is multiplied by 0, if *A*=1 it is multiplied by 1, if *A*=2 it is multiplied by 2, and if A=3 it is multiplied by 4. This weighted output is added to the decaying value of the variable *J*.

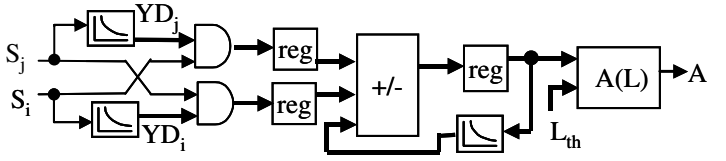


Fig. 3. Diagram of the learning block

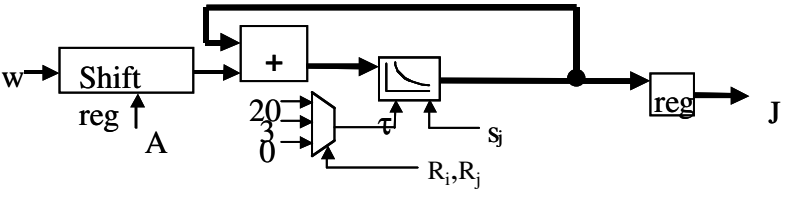


Fig. 4. Diagram of the synapse block

This operation depends on the neuronal types (R_i and R_j). In the current case study there are only two types of neurons, excitatory and inhibitory. If both neurons are inhibitory the weight of the synapse is set to 0 and the value of J is always 0 and no decay is implemented. For the other three types of synapses the time constants are multiplexed, and the multiplexer is controlled by the types of neurons (R_i,R_j). The value of J is obtained at the output of the decay block controlled by the multiplexer. Figure 4 shows the organization of the synapse block.

The characteristics of the parameters of the synapse block are the following:

- The internal resolution of the block is 10 bits, but the output resolution is 8 bits, becasue the internal value of J is divided by 4 to keep the correct scaling with the other parameters.
- The time constants used by this block are listed in Table 1.

Table 1. Time constants for different types of synapses. R=0 corresponds to an excitatory and R=1 to an inhibitory neuron.

Time Constant (τ)	Projecting Neuron Type (R_j)	Actual Neuron Type (R_i)
20	0	0
0	0	1
3	1	0
0	1	1

4 Parameters Tuning

The resolution required to represent the values of the variables and the number of operations to be performed may pose a serious limitation for the final implementation. Therefore, an important step consisted in evaluating the model and tuning its parameters in order to get a satisfactory performance. The implementation used in this study has been based on a neural network of size 15x15 with a connectivity pattern of 24 neurons corresponding to a neighborhood of 5x5 (Fig. 1). The distribution of the 20% inhibitory cells was random. The weights, *w*, and the initial activation variables, *A*, were also chosen randomly. Dynamic gradient stimuli have been applied to the neural network. A sequence of vertical bars of gradient intensity move over "strips" of neurons placed in the 2D array of the neural network (Fig. 5).

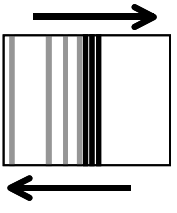
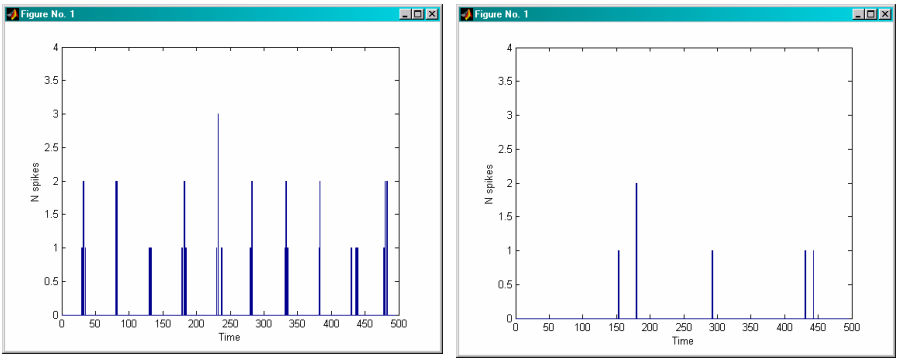


Fig. 5. Input signal applied to the neural network. The arrow to the right means forward sense and the arrow to the left means reverse sense.

The vertical bars may move at different speeds (i.e. spatial frequency). A neuron "hit" by the stimulus receives an input that is proportional to the gradient intensity. The activity of the network has been studied in a "training" condition and in a "test" condition. During training the spatial frequency of the stimulus has been incremented by discrete harmonics (2x, 4x, etc.) in one direction (the "forward" direction). During test, the stimuli were presented in both forward and reverse sense. A Gaussian noise (Mean 0, SD= 48) is applied to all neurons during all the time. The characteristics of the input applied to each neuron are the following:

- T_{CLK}: 20 ns. Maximum amplitude: 127.
- Training period: 20 us. Forward sense
- Test period: 10 us. Forward and Reverse sense

The activity calculated over a "strip" of neurons perpendicular to the direction of the movement represents a measure of "local" activity. In this case, the strip is one-column wide. In Fig. 6 the "local" activity is measured by the count of spikes produced as a function of the time steps. We can observe that in the forward sense there exists an activation pattern with a temporal correlation, but in reverse sense the network output has no such temporal correlation. This result demonstrates that the selected structure of our neural network is able to perform an implicit recognition of dynamic features based on simple unsupervised STDP rules.



(a) (b)

Fig. 6. Local activity in column 1. (a) test stimuli are applied in forward sense. (b) test stimuli are applied in reverse sense.

At a first attempt the resolution of the parameters has been reduced by 2 bits and some values and time constants have been changed to keep the correct scaling. Table 2 shows the new values of the internal parameters after this optimization process. The final organization resulting from this optimization process is depicted in Fig. 7. The simplified model resulting from this optimization has been validated again using the same input stimuli presented in Fig. 5. The results of these simulations demonstrate that the model is still capable of discriminating the input stimuli applied in the forward and in the reverse directions.

Due to the complexity of the design, the simplification of the model is very important to avoid redundancy or to use just the necessary components. For this reason, a further simplification of all the building blocks that constitute the model has been performed [8].

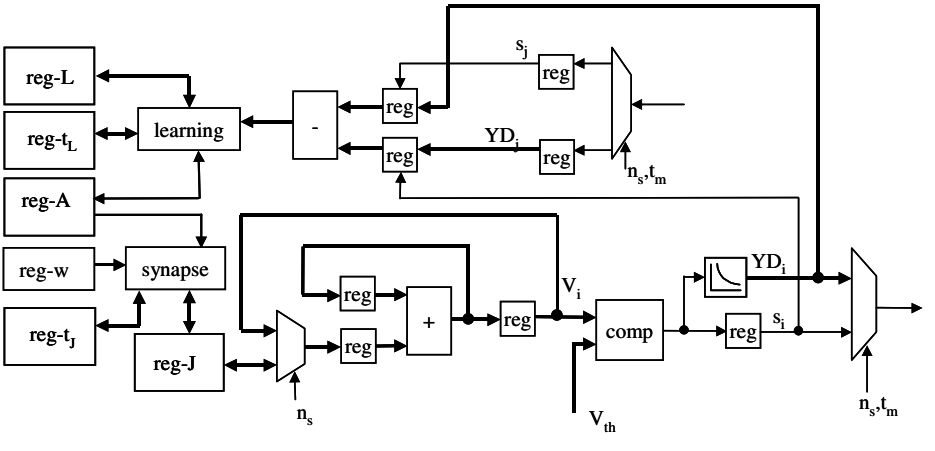


Fig. 7. Block diagram for the serial implementation of the neuron model

Table 2. Resolution of the parameters for an optimized implementation

Parameter	New value
Membrane resolution	10 bits
Threshold	+160
Input (J) resolution	6 bits
Weights (R_i,R_j) (00, 01, 10, 11)	[0:8], [64:128], [128:256], [0:0]
YD resolution	4 bits
L resolution	6 bits
Membrane decay time constant	20
YD decay time constant	20
L decay time constant	4000
J_{R_i,R_j} decay time constants (R_i,R_j) (00, 01, 10, 11)	(20, 0, 3, 0) *values not optimized*

5 Implementation on the POEtic Tissue

The POEtic tissue [9] constitutes a flexible hardware substrate that has been specifically conceived in order to permit the efficient implementation of bio-inspired models. The tissue may be constructed as a regular array composed of POEtic chips, each of them integrating a custom 32-bit RISC microprocessor and a custom FPGA with dynamic routing capabilities.

The custom FPGA included in the POEtic chip is composed of a bi-dimensional array of elementary programmable elements, called molecules. Each molecule contains a flip-flop, a 16-bit lookup table (LUT) and a switchbox that permits to establish programmable connections between molecules.

After the optimization carried out on the neural model in order to facilitate its hardware realization it has been mapped on to the molecules that constitute the POEtic device. The molecule organization shown in Fig. 8 corresponds to the actual structure of the FPGA present in the POEtic device, which is arranged as an 8x18 array of molecules.

The VHDL models developed for the POEtic tissue have been configured and simulated to validate the functionality of the neuron model designed above. After this validation stage the strategy for the simulation of large-scale SNN models has been considered. Since in its actual implementation the POEtic chip only allows for the implementation of a single neuron and the current number of POEtic chips is far less than 10,000 it will be necessary to use a smaller array of POEtic chips whose functionality should be time–multiplexed in order to emulate the entire network. This means that every POEtic chip should be able to manage a local memory in charge of storing the weights and learning variables corresponding to the different neurons it is emulating in time.

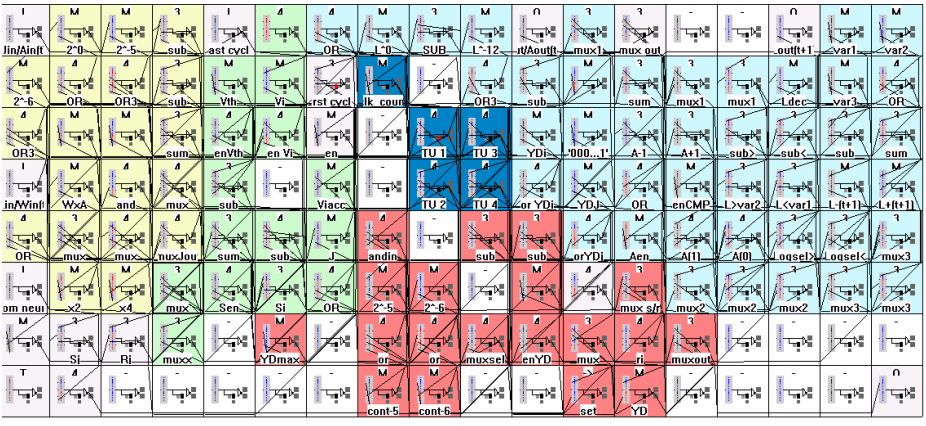


Fig. 8. Molecule-level implementation of the SNN model

A 16-neurons network organized as a 4x4 array has been constructed using this principle. This would permit the emulation of a 10,000-neurons network in 625 multiplexing cycles. Bearing in mind that each neuron is able to complete a time step in 150 clock cycles, this means that the minimum clock frequency required to handle input stimuli in real time (i.e., to process visual input stimuli at 50 frames/second) is around 5 MHz far within the possibilities of the actual clock frequency achieved by the POEtic tissue (between 50 MHz and 100 MHz).

The visual stimuli will come from an OmniVision OV5017 monochrome 384x288 CMOS digital camera. Specific VHDL and C code have been developed in order to manage the digital images coming from the camera. To test the application, artificial image sequences have been generated on a display and then captured by the camera for its processing by the network.

6 Conclusions

In this paper we have considered an unsupervised model for modifiable synapses in a Spiking Neural Network based on discrete interval variables. This model has demonstrated a good performance when used for learning and recognition tasks that involve dynamic input stimuli.

The basic parameters that define the model dynamics have been optimized in order to provide a hardware friendly implementation. The resulting model has been implemented in the POEtic tissue, a flexible hardware platform conceived for the physical realization of bio-inspired models. The results of the current implementation demonstrate that the proposed approach is capable of supporting real-time needs of large-scale spiking neural networks models.

Our current work is concentrated on the physical implementation of the real-time image recognition tasks using the development boards that have been constructed for the POEtic tissue.

Acknowledgements

The work presented in this paper has been funded by the grant IST-2000-28027 (POEtic) of the European Community and by grant OFES 00.0529-2 of the Swiss government. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

1. Maas, W.: Networks of Spiking Neurons: The Third Generation of Neural Network Models. *Neural Networks* 10 (1997) 1659–1671.
2. Hill, S.L., Villa, A.E.P.: Dynamic transitions in global network activity influenced by the balance of excitation and inhibition. *Network: Computation in Neural Systems* 8 (1997) 165-184.
3. Abbott, L.F., Nelson, S.B.: Synaptic plasticity: taming the beast. *Nature Neuroscience* 3 (2000) 1178–1183.
4. Bell, C.C., Han, V.Z., Sugawara, Y., Grant, K.: Synaptic plasticity in a cerebellum-like structure depends on temporal order. *Nature* 387 (1997) 278–281.
5. Montgomery, J.M., Madison, D.V.: Discrete synaptic states define a major mechanism of synapse plasticity. *Trends in Neurosciences* 27 (2004) 744-750.
6. Eriksson, J., Torres, O., Mitchell, A., Tucker, G., Lindsay, K., Halliday, D., Rosenberg, J., Moreno, J.M., Villa, A.E.P.: Spiking Neural Networks for Reconfigurable POEtic Tissue. Evolvable Systems: From Biology to hardware. *Lecture Notes in Computer Science* 2606 (2003) 165-173.
7. Iglesias, J., Eriksson, J., Grize, F., Tomassini, M., Villa, A.E.P.: Dynamics of pruning in simulated large-scale spiking neural networks. *Biosystems* Vol. 79 (2005) 11-20.
8. Torres, O., Eriksson, J., Moreno, J.M., Villa, A.E.P.: Hardware optimization and serial implementation of a novel spiking neuron model for the POEtic tissue. *BioSystems* 76 (2003) 201–208.
9. Moreno, J.M., Thoma, Y., Sanchez, E., Torres, O., Tempesti, G.: Hardware Realization of a Bio-inspired POEtic Tissue. *Proceedings of the NASA/DoD Conference on Evolvable Hardware*. IEEE Computer Society (2004) 237-244.

Adaptive Waveform Control in a Data Transceiver for Multi-speed IEEE1394 and USB Communication

Yuji Kasai[1], Eiichi Takahashi[1], Masaya Iwata[1], Yosuke Iijima[1,2], Hidenori Sakanashi[1], Masahiro Murakawa[1], and Tetsuya Higuchi[1]

[1] MIRAI, Advanced Semiconductor Research Center, AIST, Tsukuba Central 2, 1-1-1 Umezono, Tsukuba 305-8568, Japan
{y.kasai, e.takahashi, m.iwata, y.iijima}@aist.go.jp
http://unit.aist.go.jp/asrc/asrc-5/index_en.html
[2] Graduate School of Systems and Information Engineering, University of Tsukuba, 1-1-1 Tennoudai, Tsukuba 305-8577, Japan

Abstract. This paper proposes an adaptive waveform control in a data transceiver and demonstrates an adaptive transceiver LSI with a waveform controller. The LSI optimizes on-site transmission performance, with adjustments based on measurements for the whole transmission system, including cable properties. Utilizing genetic algorithm (GA), our adjustment method has achieved a transmission speed that is four times faster (1.6GHz) than current standards (400MHz) for IEEE1394.

1 Introduction

The demand for high-speed data transmission is rapidly increasing. As transceiver LSIs are required to operate at faster data transmission rates, however, new problems, such as waveform distortion that is known as inter-symbol interference [1], become more acute.

Conventional solutions such as pre-emphasis techniques [1] are of limited effectiveness in compensating for the cable properties of specific length transmission lines. Accordingly, we propose a new approach of adaptive waveform control, where the properties of the entire transmission system are adaptively compensated by a genetic algorithm [2-4]. This approach makes it possible to adaptively adjust for the on-site conditions where the transceiver LSI and the transmission cable are actually used. The transceiver LSI with adaptive adjustment has successfully achieved a transmission speed of 1.6GHz, which is four times faster than the current IEEE1394 standard (400MHz), and a cable length of 21m, which is four times longer than the current USB standard (5m).

2 Adaptive Waveform Control Transceiver

Fig. 1 illustrates the concept of the newly developed adaptive waveform controller. The transmitter in the transceiver LSI includes a driver that controls waveforms

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 198 – 204, 2005.

according to parameters determined by the GA. The receiver in the transceiver includes evaluation circuitry that evaluates waveform quality.

The transceiver has three operational modes; a normal-speed mode, an adjustment mode, and a high-speed mode. Adjustment is carried out in the following manner: After operating in normal-speed mode (at either IEEE1394 or USB standards), adjustment, which shifts operation to the high-speed mode, is executed in four steps (see Fig. 1).

1. Waveform-control parameters are initialized by the adjustment program, and a test signal is transmitted, once it has been pre-emphasized according to the parameters.
2. When the test signal reaches the receiver, the waveform is evaluated in terms of its quality and assigned a value (i.e., the GA fitness value).
3. This evaluation value is then fed back to the transmitter.
4. The transmitter invokes the adjustment program to obtain a better parameter setting.

By repeating these four steps, the optimal parameter setting is obtained that provide the fastest transmission.

We use a steady-state GA for the automatic waveform control on a PC, because it is suitable for hardware implementation. The GA is executed by iterating the following procedure: (1) select one individual from the population, (2) perform a crossover operation between this individual and the individual with the highest fitness value, and (3) if one of two generated individuals has a better fitness value than the worst individual in the population, then that individual replaces the worst individual.

Fig. 2 shows a block diagram of the waveform control transmitter. The circuit consists of 8 amplifiers that are current-mode driver circuits. Each driver circuit has a current-source, within the range of 0 and 12.28mA (12 bits), and a polarity switch (1 bit). These parameters are adaptively adjusted by the steady state GA.

Fig. 3 illustrates the principle behind the adjustment of the waveform control. The rectangular wave for the input data is delayed by 8 steps creating 8 waves, and these delayed waves are then converted in terms of amplitude and polarity, and are finally combined to form the output data.

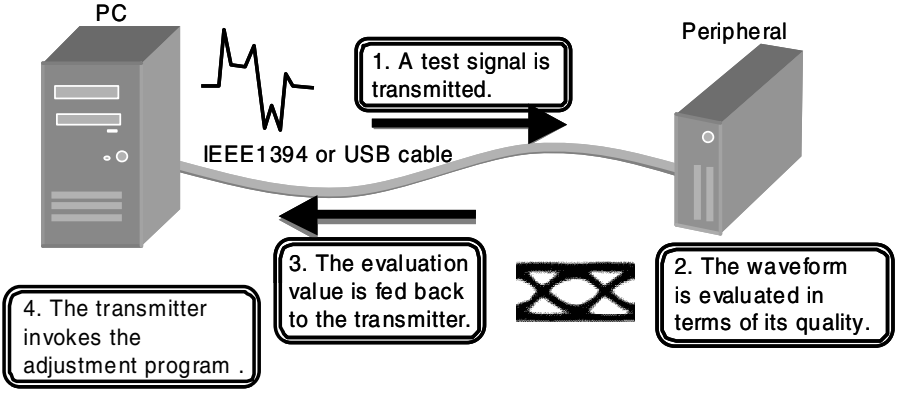


Fig. 1. Concept of adaptive waveform control

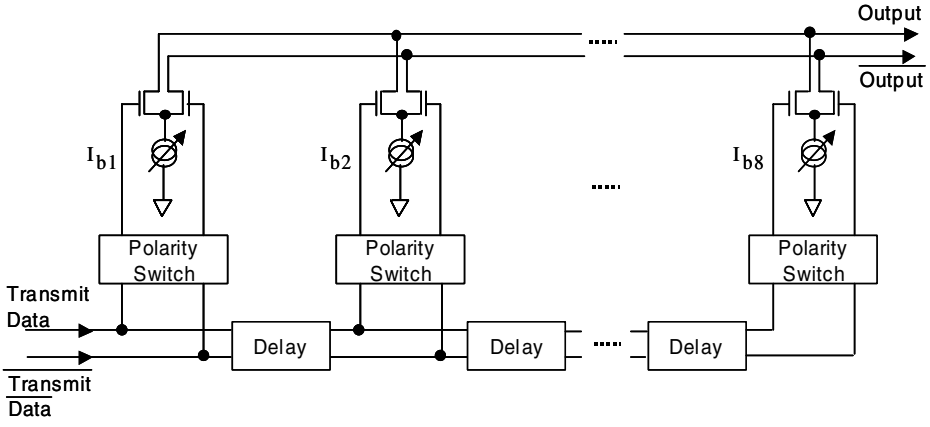


Fig. 2. Block diagram of the waveform control transmitter

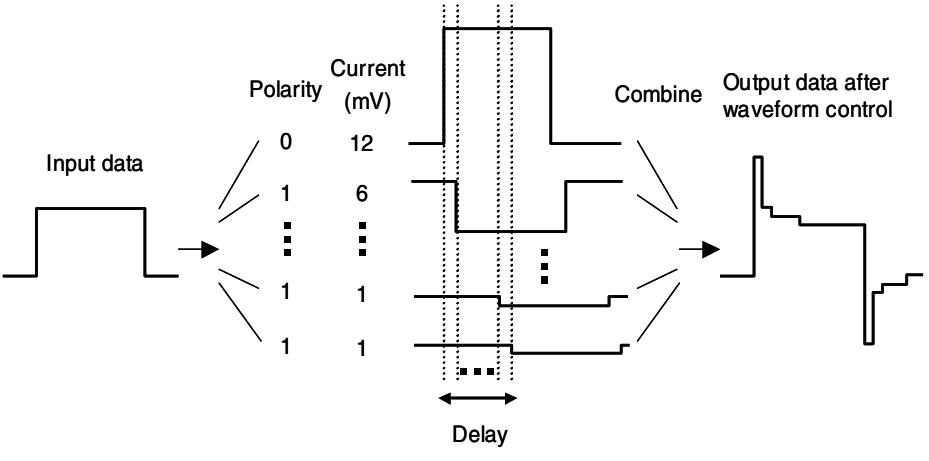


Fig. 3. The principle behind the adjustment of the waveform control

The measurement system, as shown in Fig. 4, consists of the transceiver LSI for transmission, a commercially available transmission cable (either IEEE1394 or USB), a digital oscilloscope (HP54750A) as a receiver, and a PC to execute the GA. GA evaluation values are measured by the oscilloscope.

The program on the PC controls the system and automatically adjusts the transmitter using the GA. The parameters of the transmitter are eight analog values that range between 0 and 12.28mA (12 bits / value, represented by A1, A2, … , A8) for the current-mode drivers, and a digital value (8 bits, represented by D), which determines the values of the eight polarity switches. A GA chromosome is represented with 104 bits by connecting these bit strings. The fitness function of the GA is eye height (see Fig. 4) which represents the quality of the received waveform as measured by the

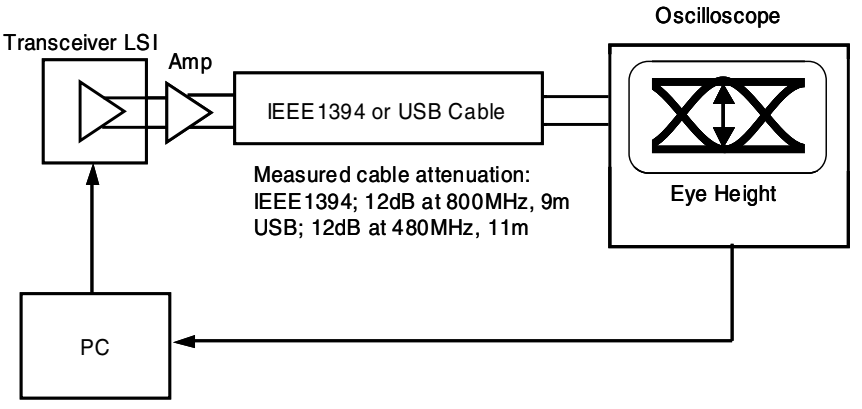


Fig. 4. Block diagram of the measurement system

oscilloscope. The population size is 30. The crossover and mutation rates are 0.8 and 0.1, respectively.

GA learning is conducted in two stages in order to accelerate adjustment. First, the GA is executed in order to learn two analog values (A1 and A8), and the digital value (D) (while the remaining six analog values are set to 0). Next, these learned values are used as initial values for the next stage of the GA, which learns all 8 analog values and the digital value. Thus, in this two-stage method, a rough waveform is initially learned and its shape is subsequently refined.

3 Results and Discussion

A chip, as shown in Fig. 5, has been designed and fabricated by 0.13μm CMOS technology. The area of the transmitter circuit, shown as a white rectangle, is 0.22mm^2. This area, however, could be reduced for practical implementation by incorporating the circuitry into the I/O buffer area.

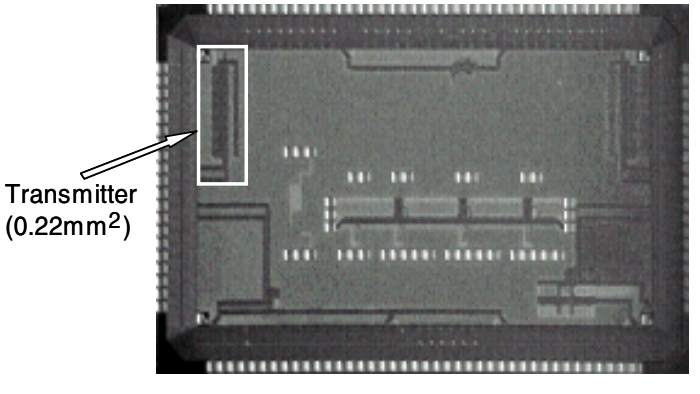


Fig. 5. Photo of the 0.13μm CMOS LSI chip

Table 1. Measured threshold margins* for (a) IEEE1394 and (b) USB

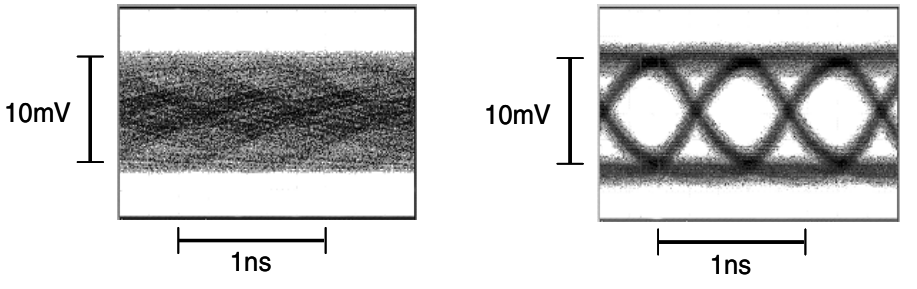
(a) IEEE1394

Cable	4.5m	9m	13.5m
400MHz	**0.81** (0.67)	**0.84** (0.62)	**0.80** (0.47)
800MHz	**0.79** (0.31)	**0.78** (0.19)	**0.71** (—)
1.2GHz	**0.77** (0.37)	**0.67** (—)	**0.51** (—)
1.6GHz	**0.84** (—)	**0.68** (—)	**0.40** (—)

(b) USB

Cable	5m	11m	21m
480MHz	**0.80** (0.56)	**0.84** (0.29)	**0.75** (0.16)
960MHz	**0.78** (0.16)	**0.64** (—)	**0.46** (—)
1.6GHz	**0.58** (—)	**0.15** (—)	— (—)

Bold: with GA adaptive adjustment waveform control
( ): without pre-emphasis, —: unable to transmit
*Threshold margin = eye height / amplitude



(a) Without pre-emphasis

(b) With GA adaptive adjustment waveform control

Fig. 6. Eye diagrams of a transmission with an IEEE1394 cable (1.6GHz, 9 m)

On average, adjustment required 200 evaluations, that is, testing of 200 transmitter configurations, for each stage. Each evaluation consists of two key processes; (1) GA execution (requiring less than 1 ms), and (2) waveform sampling (needing less than 1 ms for one million data symbols at 1GHz). Thus, each evaluation requires approximately 2 ms, indicating that the total adjustment time would normally be less than one second.

Measurements of threshold margins, as an index of waveform quality, are shown in Table 1, which also include no pre-emphasis conditions for comparison. These measurements show significant improvements at various speeds and cable lengths due to adaptive GA-adjustment waveform control. With the IEEE1394 cable (Table 1 (a)), transmission performance is four times faster and three times longer than the current standard (400MHz, 4.5m). With the USB cable (Table 1 (b)), performance is two times faster and four times longer than the current standard (480MHz, 5m). Eye dia-

grams for the IEEE1394 cable are shown in Fig. 6. Eye height as a function of the number of GA evaluations is shown in Fig. 7. The optimum parameters for the transmitter are shown in Table 2.

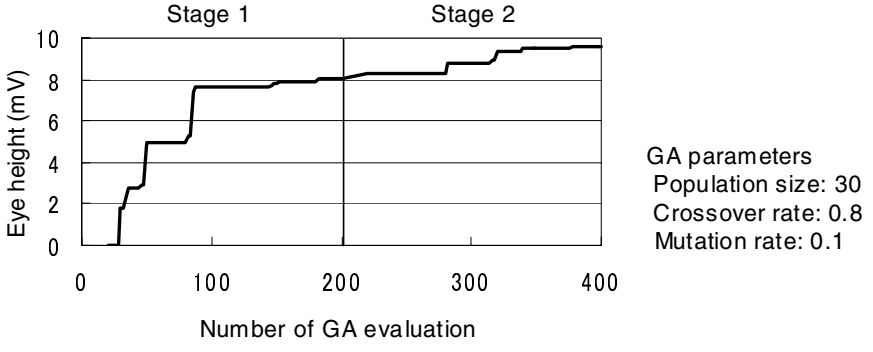


Fig. 7. Eye height as a function of the number of GA evaluations (IEEE1394, 1.6GHz, 13.5m, average of 5 executions)

Table 2. The parameters of the transmitter for IEEE1394 cables adjusted by GA

	A1	A2	A3	A4	A5	A6	A7	A8	D
Stage1	3.94	0	0	0	0	0	0	10.21	0F
Stage2	8.95	0.14	0.13	0.14	0.15	0.13	12.28	10.32	0F

4 Conclusion

We have clearly demonstrated effectiveness of GA-based adaptive waveform control using developed LSI that optimizes performance for the entire transmission system within a practically short time. Our adjustment method has achieved a transmission speed of 1.6GHz that is four times faster than the current standard (400MHz) for IEEE1394 and a cable length of 21m that is four times longer than current standard (5m) for USB.

In this paper, we have focused on data transmission for USB and IEEE1394 cables. In related work, we have proposed data transmission for on-board and back-plane connections with adaptive waveform control. Experimental results show that high speed I/O for parallel bit data transmission was realized at 2Gbps for each data bit line of 70cm in length [5].

Acknowledgments

This work was supported by NEDO. The authors would like to thank Dr. Hirose, Dr. Masuhara and Prof. Otsuka at MIRAI for their critical reading of the manuscript.

References

1. Fiedler, A., Mactaggart, R., Welch, J., Krishnan, S.: A 1.0625Gbps transceiver with 2x-oversampling and transmit signal pre-emphasis. ISSCC Digest of Technical Papers (1997) 238-239
2. Murakawa, M., et al.: An AI-calibrated IF filter: a yield enhancement method with area and power dissipation reductions. IEEE J. Solid State Circuits, Vol. 38, No. 3, (2003) 495-502
3. Holland, J. H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press (1975)
4. Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley, Reading (1989)
5. Endou, N., Kasai, Y., Iwata, M., Takahashi, E., Higuchi, T.: Galois field computation LSI: a reconfigurable chip for high-speed communication. 2005 Symposium on VLSI Circuits Digest of Technical Papers (2005) (in press).

Evolution, Re-evolution, and Prototype of an X-Band Antenna for NASA's Space Technology 5 Mission

Jason D. Lohn[1], Gregory S. Hornby[2], and Derek S. Linden[3]

[1] Computational Sciences Division, NASA Ames Research Center,
Moffett Field, CA 94035, USA
jlohn@arc.nasa.gov
http://ic.arc.nasa.gov/projects/esg/
[2] QSS Group Inc., NASA Ames Research Center,
Moffett Field, CA 94035, USA
hornby@email.arc.nasa.gov
[3] JEM Engineering, 8683 Cherry Lane,
Laurel, MD 20707
dlinden@jemengineering.com

Abstract. One of the challenges in engineering design is responding to a change of design requirements. Previously we presented a four-arm symmetric evolved antenna for NASA's Space Technology 5 mission. However, the mission's orbital vehicle was changed, putting it into a much lower earth orbit, changing the specifications for the mission. With minimal changes to our evolutionary system, mostly in the fitness function, we were able to evolve antennas for the new mission requirements and, within one month of this change, two new antennas were designed and prototyped. Both antennas were tested and both had acceptable performance compared with the new specifications. This rapid response shows that evolutionary design processes are able to accommodate new requirements quickly and with minimal human effort.

1 Introduction

One of the challenges in engineering design is responding to a change in design requirements. Previously we presented our work in using evolutionary algorithms to automatically design an X-band antenna for NASA's Space Technology 5 (ST5) spacecraft [4]. Since our original evolutionary runs and the fabrication and testing of antennas ST5-3-10 and ST5-4W-03, the launch vehicle for the ST5 spacecraft has changed resulting in a lower orbit and different antenna requirements. With traditional engineering design such a change in requirements would necessitate redoing much of the design work with a near doubling of design costs. In contrast, with an evolutionary design system for automatically creating antennas once the software has been developed, modifying it to produce antennas for a similar design problem requires only a minimal amount of human effort to implement the change a re-evolve new antennas with minimal additional cost.

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 205–214, 2005.

The ST5 mission consists of three spacecraft which will orbit at close separations in a highly elliptical geosynchronous transfer orbit and will communicate with a 34 meter ground-based dish antenna. Each spacecraft will have two antennas attached, one on each side of the spacecraft, Figure 1. Initially the spacecraft were to fly approximately 35,000 km above Earth and the requirements for the communications antenna were for a gain pattern of $\geq$0 dBic from 40° - 80° from zenith. With the change in launch vehicle and the new, lower orbit this necessitated the addition of a new requirement on the gain pattern of $\geq$-5 dBic from 0° 40° from zenith. The complete set of requirements for the antennas on the ST5 Mission are summarized in table 1. VSWR is a way to quantify reflected-wave interference, a measure of the impedance mismatch. It is the ratio between the highest voltage and the lowest voltage in the signal envelope along a transmission line, with a ratio of 1 being perfect VSWR.

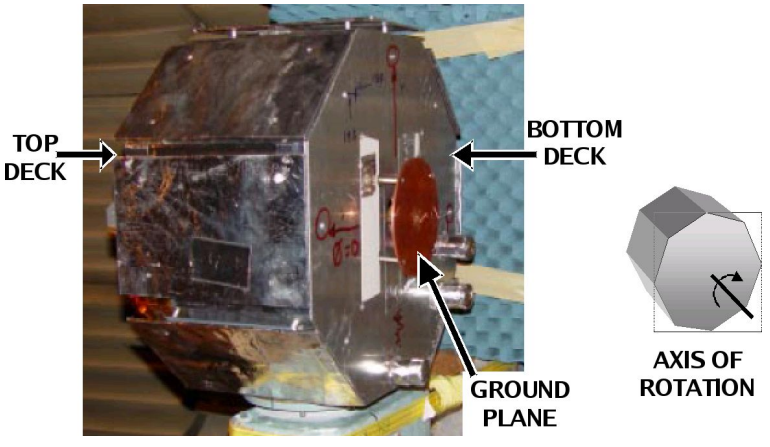


Fig. 1. Photograph of the ST5 mock-up with antennas mounted (only the antenna on the top deck is visible)

In the rest of this paper we describe the two evolutionary design systems we used for evolving the initial antennas for this mission and the changes we made to them to address the change in mission requirements. We then present the performance of the new antenna designs, both from simulation and from fabricated units. One of our newly evolved antennas, ST5-33.142.7, meets the new mission requirements and has successfully passed environmental testing. Three of these antennas are scheduled to be launched in 2006 and will be the first evolved hardware in space and the first evolved antennas to be fielded.

2 Evolutionary Antenna Design Systems

As a result of the new ST5 mission requirements we needed to change both the type of antenna we were evolving and the fitness function. The original antennas

Table 1. Key ST5 Antenna Requirements

Property	Specification
Transmit Frequency	8470 MHz
Receive Frequency	7209.125 MHz
VSWR	$< 1.2 : 1$ at Transmit Freq
	$< 1.5 : 1$ at Receive Freq
Original Gain Pattern	≥ 0 dBic, $40^\circ \leq \theta \leq 80^\circ$, $0^\circ \leq \phi \leq 360^\circ$
Additional Gain Pattern Requirement	$\geq$ -5 dBic, $0^\circ \leq \theta \leq 40^\circ$, $0^\circ \leq \phi \leq 360^\circ$
Input Impedance	50 Ω
Diameter	< 15.24 cm
Height	< 15.24 cm
Antenna Mass	< 165 g

we evolved were constrained to a monopole wire antenna with four identical arms, with each arm rotated 90° from its neighbors. There the EA evolved genotypes that specified the design for one arm and the phenotype consisted of four copies of the evolved arm. Because of symmetry, the previous four-arm design has a null at zenith that is built into the design and is unacceptable for the revised mission. To achieve an antenna that meets the new mission requirements the new antenna designs were configured to produce a single arm. In addition, because of the difficulties we experienced in fabricating branching antennas to the required precision, here we constrained our antenna designs to non-branching antennas. In the remainder of this section we describe the two evolutionary algorithms we used to evolve antennas for the ST5 mission and how we changed them to address the new requirements.

2.1 Parameterized EA for Non-branching Designs

The first EA was used in our previous work in evolutionary antenna design [3] and it is a standard genetic algorithm (GA) that evolves non-branching wire forms. With this EA the design space used a vector of real-valued triplets that specify the X, Y and Z locations of segment end-points. The fitness function for this EA used pattern quality scores at 7.2 GHz and 8.47 GHz. Unlike the second EA, VSWR was not explicitly used in this fitness calculation, rather it is included implicitly by how it affects the gain pattern. To quantify the pattern quality at a single frequency, PQ_f, the following formula was used:

$$\mathrm{PQ}_f = \sum_{\substack{0^\circ \leq \phi < 360^\circ \\ 0^\circ \leq \theta \leq 80^\circ}} (\mathrm{gain}_{\phi,\theta} - T)^2 \quad \text{if } \mathrm{gain}_{\phi,\theta} < T$$

where $\mathrm{gain}_{\phi,\theta}$ is the gain of the antenna in dBic (right-hand polarization) at a particular angle, T is the target gain (3 dBic was used in this case), ϕ is the azimuth, and θ is the elevation. To compute the overall fitness of an antenna design, the pattern quality measures at the transmit and receive frequencies were summed, lower values corresponding to better antennas:

$$F = \mathrm{PQ}_{7.2} + \mathrm{PQ}_{8.47}$$

Modifying this evolutionary design system to produce antennas for the new orbit consisted of changing the fitness function to check angles $0° \leq \theta < 40°$ as well the original range of $40° \leq \theta \leq 80°$.

2.2 Open-Ended EA

The second EA uses an open-ended, variable-length representation in which elements of the genotype specify how to construct the antenna. Each node in the tree-structured representation is an antenna-construction operator and an antenna is created by executing the operators at each node in the tree, starting with the root node. In constructing an antenna the current state (location and orientation) is maintained and operators add wires or change the current state. The operators are as follows:

- `forward(length, radius)` - add a wire with the given length and radius extending from the current location and then change the current state location to the end of the new wire.
- `rotate-x(angle)` - change the orientation by rotating it by the specified amount (in radians) about the x-axis.
- `rotate-y(angle)` - change the orientation by rotating it by the specified amount (in radians) about the y-axis.
- `rotate-z(angle)` - change the orientation by rotating it by the specified amount (in radians) about the z-axis.

Since we constrained antennas to a single, bent wire with no branching each node in the genotype has at most one child. This constructive representation for encoding antennas is an extension of our previous work in using a linear-representation for encoding rod-based robots [2]. Aside from restricting antennas to not having branches, the only changes made to this evolutionary design system to address the new mission requirements were to change the fitness function.

The fitness function used to evaluate antennas is a function of the VSWR and gain values on the transmit and receive frequencies. These three components are multiplied together to produce the overall fitness score of an antenna design:

$$F = vswr \times gain \times standard\ deviation$$

The objective of the EA is to produce antenna designs that minimize F.

The VSWR component of the fitness function is constructed to put strong pressure to evolving antennas with receive and transmit VSWR values below the required amounts of 1.2 and 1.5, reduced pressure at a value below these requirements (1.15 and 1.25) and then no pressure to go below 1.1:

$$v_r = \text{VSWR at receive frequency}$$

$$v'_r = \begin{cases} v_r + 2.0(v_r - 1.25) & \text{if } v_r > 1.25 \\ v_r & \text{if } 1.25 > v_r > 1.1 \\ 1.1 & \text{if } v_r < 1.1 \end{cases}$$

$$v_t = \text{VSWR at transmit frequency}$$

$$v'_t = \begin{cases} v_t + 2.0(v_t - 1.15) & \text{if } v_t > 1.15 \\ v_t & \text{if } 1.15 > v_t > 1.1 \\ 1.1 & \text{if } v_t < 1.1 \end{cases}$$

$$vswr = v'_r v'_t$$

The gain-penalty component of the fitness function uses the gain (in decibels) in 5° increments about the angles of interest: from $0° \leq \theta \leq 90°$ and $0° \leq \phi \leq 360°$. For each angle, the calculated gain score from simulation is compared against the target gain for that elevation and the outlier gain, which is the minimum gain value beyond which lower gain values receive a greater penalty. Gain penalty values are further adjusted based on the importance of the elevation:

```
gain_penalty (i, j):
        gain = calculated gain at  θ = 5°i , φ = 5°j;
        if (gain ≥ target[i]) {
                penalty := 0.0;
        } else if ((target[i] > gain) and (gain ≥ outlier[i])) {
                penalty := (target[i] - gain);
        } else { /* outlier[i] > gain */
                penalty := (target[i]-outlier[i]) + 3.0 * (outlier[i] - gain));
        }
        return penalty * weight[i];
```

Target gain values at a given elevation are stored in the array `target[]` and are 2.0 dBic for i equal from 0 to 16 and are -3.0 dBic for i equal to 17 and 18. Outlier gain values for each elevation are stored in the array `outlier[]` and are 0.0 dBic for i equal from 0 to 16 and are -5.0 dBic for i equal to 17 and 18. Each gain penalty is scaled by values scored in the array `weight[]`. For the low band the values of `weight[]` are 0.1 for i equal to 0 through 7; values 1.0 for i equal to 8 through 16; and 0.05 for i equal to 17 and 18. For the high band the values of `weight[]` are 0.4 for i equal to 0 through 7; values 3.0 for i equal to 8 through 12; 3.5 for i equal to 13; 4.0 for i equal to 14; 3.5 for i equal to 15; 3.0 for i equal to 16; and 0.2 for i equal to 17 and 18. The final gain component of the fitness score of an antenna is the sum of gain penalties for all angles.

To put evolutionary pressure on producing antennas with smooth gain patterns around each elevation, the third component in scoring an antenna is based on the standard deviation of gain values. This score is a weighted sum of the standard deviation of the gain values for each elevation θ. The weight value used for a given elevation is the same as is used in calculating the gain penalty.

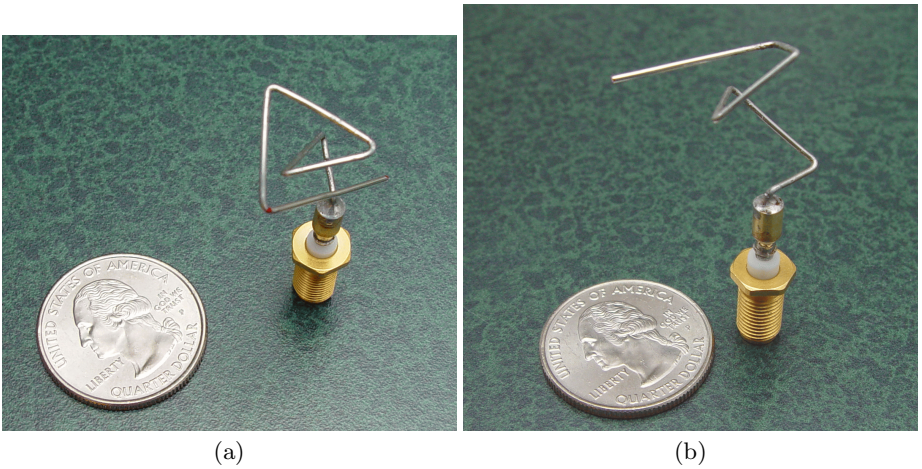
This fitness function differs from the one we used previously [4] in the fidelity to which the desired gain pattern can be specified and in explicitly rewarding for a smooth pattern. Our previous fitness function with the constructive EA had one target gain value for all elevations and weighted all elevations equal. With the new fitness function different target gain values can be set for different elevation angles and also the importance of achieving the desired gain at a given

angle is specified through setting the weight value for a given elevation. The other difference with this fitness function is that previously there was a separate penalty for "outlier" gain values whereas in the new fitness function this is included in the gain component of the fitness score and a new component that measures pattern smoothness is included.

3 Evolved Antennas

To re-evolve antennas for the new ST5 mission requirements we used the same EA setup as in our initial set of evolutionary runs, however, we did not seed the first generation with previously evolved antenna designs. For the non-branching EA, a population of fifty individuals was used, 50% of which is kept from generation to generation. The mutation rate was 1%, with the Gaussian mutation standard deviation of 10% of the value range. The non-branching EA was halted after one hundred generations had been completed, the EA's best score was stagnant for forty generations, or EA's average score was stagnant for ten generations. For the branching EA, a population size of two hundred individuals was evolved with a generational EA. Parents were selected with remainder stochastic sampling based on rank, using exponential scaling [5]. New individuals were created with an equal probability of using mutation or recombination. The Numerical Electromagnetics Code, Version 4 (NEC4) [1] was used to evaluate all antenna designs.

The best antennas evolved by the two EAs were then evaluated on a second antenna simulation package, WIPL-D, with the addition of a 6" ground plane to determine which designs to fabricate and test on the ST5 mock-up. The best antenna design from each EA was selected for fabrication and these are shown in



(a) (b)

Fig. 2. Evolved antenna designs: (a) evolved using a vector of parameters, named ST5-104.33; and (b) evolved using a constructive process, named ST5-33.142.7.

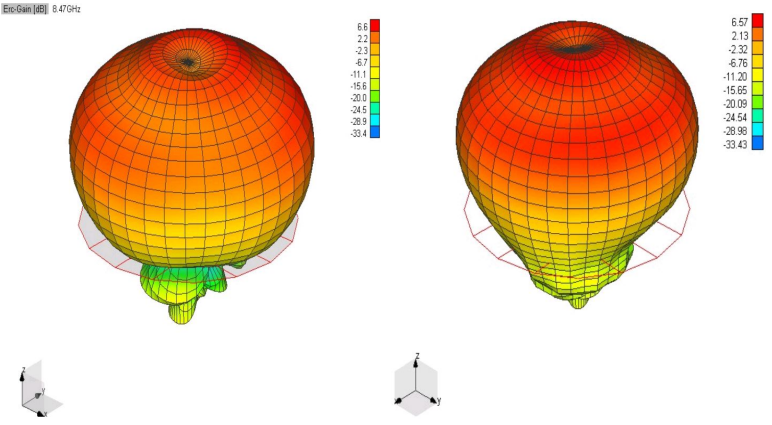


Fig. 3. Simulated 3D patterns for ST5-104.33 and ST5-33.142.7 on 6" ground plane at 8470 MHz for RHCP polarization. Simulation performed by WIPL-D. Patterns are similar for 7209 MHz.

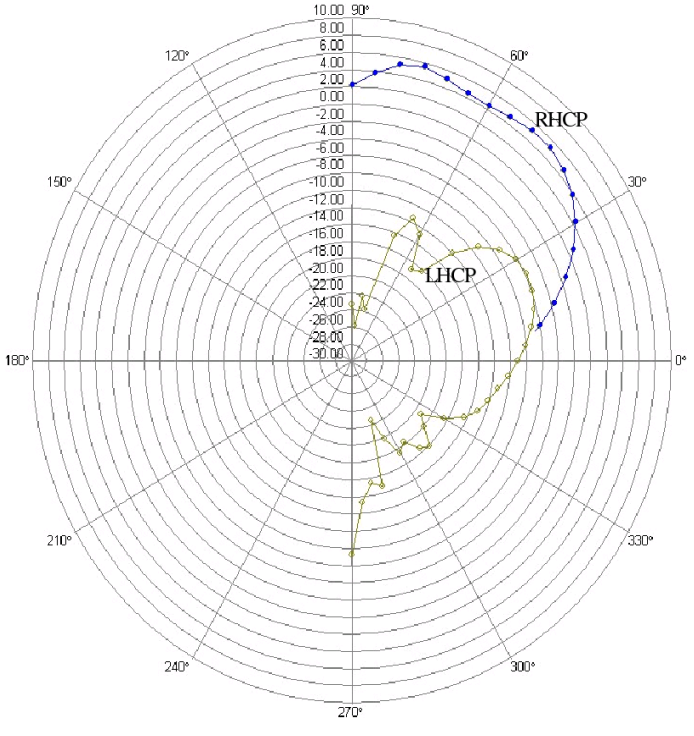


Fig. 4. RHCP vs LHCP performance of ST5-104.33. Plot has 2 dB/division.

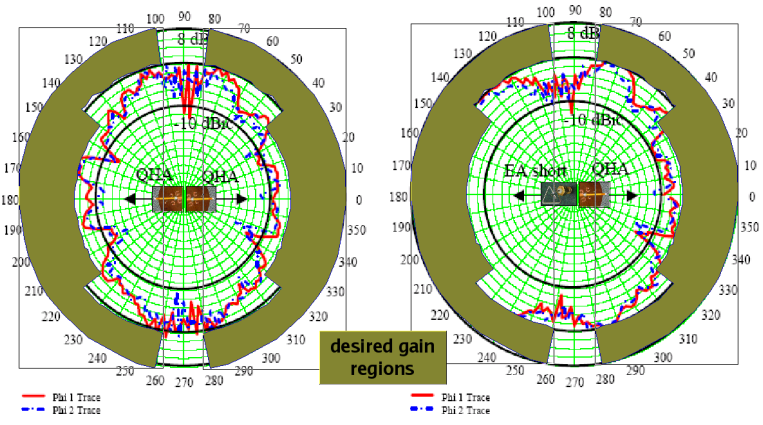


Fig. 5. Measured patterns on ST-5 mock-up of QHA antenna and ST5-104.33 plus QHA antenna. Phi 1 = 0 deg., Phi 2 = 90 deg.

Figure 2. For these runs a single antenna evaluation took a few seconds of wall-clock time to simulate and an entire run took approximately six to ten hours.

3.1 Simulated Results

Both antenna designs have excellent simulated RHCP patterns, as shown in Figure 3 for the transmit frequency. The antennas also have good circular polarization purity across a wide range of angles, as shown in Figure 4 for ST5-104.33. To the best of our knowledge, this quality has never been seen before in this form of antenna.

3.2 Measured Results

The antennas were measured on the ST5 mock-up (Figure 1), and the results are shown in Figure 5. Because each spacecraft has two antennas, one on each side of the spacecraft, of interest is the performance of pairs of antennas on the spacecraft. The evolved antennas were arrayed with a Quadrafilar Helix Antenna (QHA) developed by New Mexico State University's Physical Science Laboratory that was the original antenna for this mission. This figure shows plots of two QHA antennas together, and a QHA and an ST5-104.33 antenna. Results are similar for ST5-33.142.7, which is the design that has been selected for use on the ST5 mission. Compared to using two QHAs together, the evolved antennas have much greater gain across the angles of interest.

4 Conclusion

Previously we reported our work on evolving two X-band antennas for potential use on NASA's upcoming ST5 mission to study the magnetosphere. While those

Fig. 6. Three images of a flight antenna; the evolved wire configuration for the radiator sits on top of a 6" diameter ground plane and is encased inside a radome.

antennas were mission compliant, a change in launch vehicle resulted in a change in orbit for the ST5 spacecraft and a change in requirements for their communication antennas. In response to this change in requirements we reconfigured our evolutionary design systems and in under four weeks we were able to evolve new antenna designs that were acceptable to ST5 mission planners.

The first set of new ST5 evolved antenna flight units were delivered to Goddard Space Flight Center (GSFC) on February 25, 2005 (Figure 6). These flight units have passed all environmental testing and the current baseline plan is to fly at least three evolved antennas when the mission launches in 2006. Our ability to rapidly re-evolve new antenna designs shows that the evolutionary design process lends itself to rapid response to changing requirements, not only for automated antenna design but for automated design in general.

Acknowledgments

The work described in this paper was supported by Mission and Science Measurement Technology, NASA Headquarters, under its Computing, Information, and Communications Technology Program. The work was performed at the Computational Sciences Division, NASA Ames Research Center, Linden Innovation Research and JEM Engineering, and NASA Goddard Space Flight Center. The support of Ken Perko of Microwave Systems Branch at NASA Goddard and Bruce Blevins of the Physical Science Laboratory at New Mexico State University is gratefully acknowledged.

References

1. G. J. Burke and A. J. Poggio. Numerical electromagnetics code (nec)-method of moments. Technical Report UCID18834, Lawrence Livermore Lab, Jan 1981.
2. G. S. Hornby, H. Lipson, and J. B. Pollack. Generative representations for the automatic design of modular physical robots. *IEEE Transactions on Robotics and Automation*, 19(4):703–719, 2003.
3. D. S. Linden and E. E. Altshuler. Automating wire antenna design using genetic algorithms. *Microwave Journal*, 39(3):74–86, March 1996.

4. J. D. Lohn, G. S. Hornby, and D. S. Linden. An Evolved Antenna for Deployment on NASA's Space Technology 5 Mission. In U.-M. O'Reilly, R. L. Riolo, T. Yu, and B. Worzel, editors, *Genetic Programming Theory and Practice II*. Kluwer, in press.
5. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, 1992.

Appendix: Genotype of ST5-33.142.7

Listed below is the evolved genotype of antenna ST5-33.142.7. The format for this tree-structured genotype consists of the operator followed by a number stating how many children this operator has, followed by square brackets which start '[' and end ']' the list of the node's children. For example the format for a node which is operator 1 and has two subtrees is written: `operator1 2 [ subtree-1 subtree-2 ]`. Since antennas were constrained to be non-branching each non-leaf node in has at most one child. The different operators in the antenna-constructing language are given in section 2.2.

```
rotate-z(0.723536) 1 [ rotate-x(2.628787) 1 [ rotate-z(1.145415) 1
[ rotate-x(1.930810) 1 [ rotate-z(2.069497) 1 [ rotate-x(1.822537)
1 [ forward(0.007343,0.000406) 1 [ rotate-z(1.901507) 1 [
forward(0.013581,0.000406) 1 [ rotate-x(1.909851) 1 [
rotate-y(2.345316) 1 [ rotate-y(0.308043) 1 [ rotate-y(2.890265) 1
[ rotate-x(0.409742) 1 [ rotate-y(2.397507) 1 [
forward(0.011671,0.000406) 1 [ rotate-x(2.187298) 1 [
rotate-y(2.497974) 1 [ rotate-y(0.235619) 1 [ rotate-x(0.611508) 1
[ rotate-y(2.713447) 1 [ rotate-y(2.631141) 1 [
forward(0.011597,0.000406) 1 [ rotate-y(1.573367) 1 [
forward(0.007000,0.000406) 1 [ rotate-x(-0.974118) 1 [
rotate-y(2.890265) 1 [ rotate-z(1.482916) 1 [
forward(0.019955,0.000406) ] ] ] ] ] ] ] ] ] ] ] ] ] ] ] ] ] ] ] ]
] ] ] ] ] ] ] ]
```

Hardware Platforms for MEMS Gyroscope Tuning Based on Evolutionary Computation Using Open-Loop and Closed-Loop Frequency Response

Didier Keymeulen[1], Michael I. Ferguson[1], Wolfgang Fink[1], Boris Oks[1], Chris Peay[1], Richard Terrile[1], Yen-Cheng[2], Dennis Kim[2], Eric MacDonald[3], and David Foor[4]

[1] Jet Propulsion Laboratory, MS 303-300, 4800 Oak Grove Dr.,
Pasadena, CA 91109, USA
didier.keymeulen@jpl.nasa.gov
http://ehw.jpl.nasa.gov
[2] Mechanical and Aerospace Engineering Department, University of California,
Los Angeles, CA 90095-1597
dongj@seas.ucla.edu
[3] University of Texas at El Paso, 500 West University dr., El Paso, TX, 79968-0523
emac@ytep.edu
[4] Texas A&M University-Kingsville, 700 University Blvd, Kingsville, TX, 78363
quatro@ieee.org

Abstract. We propose a tuning method for MEMS gyroscopes based on evolutionary computation to efficiently increase the sensitivity of MEMS gyroscopes through tuning. The tuning method was tested for the second generation JPL/Boeing Post-resonator MEMS gyroscope using the measurement of the frequency response of the MEMS device in open-loop operation We also report on the development of a hardware platform for integrated tuning and closed-loop operation of MEMS gyroscopes. The control of this device is implemented through a digital design on a Field Programmable Gate Array (FPGA). The hardware platform easily transitions to an embedded solution that allows for the miniaturization of the system to a single chip.

1 Introduction

Future NASA missions would benefit tremendously from an inexpensive, navigation grade, miniaturized inertial measurement unit (IMU), which surpasses the current state-of-the art in performance, compactness (both size and mass) and power efficiency. Towards this end, under current development at JPL's MEMS Technology Group are several different designs for environment tolerant [10], high performance, low mass and volume, low power MEMS gyroscopes. The accuracy with which the rate of rotation of micro-gyros can be determined depends crucially on the properties of the resonant structure. It is both difficult and expensive to attempt to achieve these desired characteristics in the fabrication process, especially in the case of small MEMS structures, and thus one has limited overall sensor performance due to unavoidable fabrication inaccuracies.

J.M. Moreno, J. Madrenas, and J. Cosp (Eds.): ICES 2005, LNCS 3637, pp. 215 – 226, 2005.

The sensitivity of the MEMS gyroscope is maximized when the resonant frequencies of the two modes of freedom of the MEMS gyroscope are identical. Symmetry of construction is necessary to attain this degeneracy. However, despite a symmetric design, perfect degeneracy is never attained in practice. Many methods have been developed for tuning MEMS post-resonator gyroscopes. For example [1] and [2] use adaptive and closed-loop methods, while [3] changes the frame of the pick-off signal. Our approach of gyro tuning is achieved through an electrostatic biasing approach [11]. This approach consists of applying bias voltages to built-in tuning pads to electrostatically soften the mechanical springs. Because of the time consuming nature of the tuning process when performed manually, in practice any set of bias voltages that produce degeneracy is viewed as acceptable at the present time. Thus a need exists for reducing the time necessary for performing the tuning operation, and for finding the optimally tuned configuration, which employs the minimal maximum tuning voltage.

This paper describes the application of evolutionary computation to this optimization problem. Our open-loop and closed-loop methods used the following fitness function for each set of bias voltages applied to the built-in tuning pads: the frequency split between the two modes of resonance of the MEMS gyroscope. Our open-loop evaluation proceeds in two steps. First, it measures the open-loop frequency response using a dynamic signal analyzer. Second, it evaluates the frequency of resonance of both modes by fitting Lorentzian curves to the experimental data. The process of setting the bias voltages and the evaluation of the frequency split is completely computer automated. The computer controls a signal analyzer and programmable power supplies through General Purpose Interface Bus (GPIB). Our method has demonstrated that we can obtain a frequency split of 52mHz fully automatically in one hour compared with 200mHz obtained manually by humans in several hours.

The closed-loop method is based on controlling the gyro in a closed-loop along one axis and measuring the resonance frequencies along this axis at a given set of bias voltages, then swapping and driving the other axis, thereby extracting the resonant frequency of both axes. An evolutionary algorithm is then applied iteratively to modify the bias voltages until the resonant frequency of each axis is equal. A major advantage of this closed-loop approach is that the resonant frequencies can be extracted quickly (~1 second) as compared to the open-loop control system, which takes two orders of magnitude longer. The design of the closed-loop control approach is realized on an FPGA with augmented portability for future designs and implementations.

This paper is organized such that Section 2 describes the mechanics of the MEMS micro-gyro, Section 3 describes the evolutionary computation applied to open-loop measurements of the resonance frequencies, Section 4 describes the closed-loop hardware platform and the results of our preliminary experiments, and Section 5 describes future directions and summarizes the project results.

2 Mechanism of the JPL MEMS Micro-gyroscope

The mechanical design of the JPL MEMS micro-gyro can be seen in Figure 1. The JPL/Boeing MEMS post resonator gyroscope (PRG), is a MEMS analogue to the

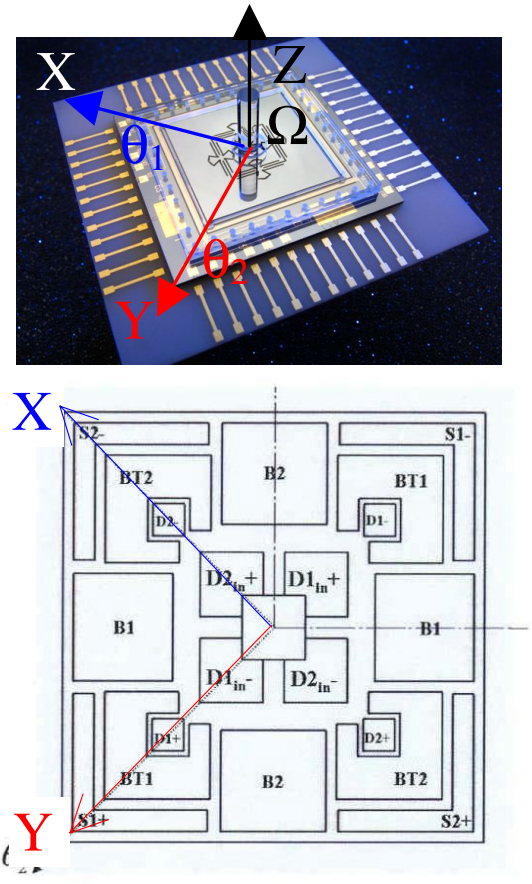


Fig. 1. A magnified picture of the JPL MEMS micro-gyroscope with sense axis Y (S2-, S2+ electrodes used to sense, D2-, D2+, D2in- and D2in+ used to drive along the sense axis) and drive axis X (D1-, D1+, D1in-, and D1in+ used to drive, S1-, S1+ electrodes used to sense along the drive axis) and the electrodes used for biasing (B1, B2, BT1, BT2) (picture courtesy of C. Peay, JPL).

classical Foucault pendulum. A pyrex post, anodically bonded to a silicon plate, is driven into a rocking mode along an axis (labeled as X in Figure 1) by sinusoidal actuation via electrodes beneath the plate. In a rotating reference frame the post is coupled to the Coriolis force, which exerts a tangential "force" on the post. Another set of electrodes beneath the device senses this component of motion along an axis (labeled as Y in the figure) perpendicular to the driven motion. The voltage that is required to null out this motion is directly proportional to the rate of rotation to which the device is subjected and the voltage scale is reduced proportionally to the *frequency split* between the two modes of resonance. A change in capacitance occurs as the top plate vibrates due to the oscillating gap variation between this plate and the electrodes underneath. This change in capacitance generates a time-varying sinusoi-

dal charge that can be converted to a voltage using the relationship V=Q/C. The post can be driven around the drive axis by applying a time-varying voltage signal to the drive petal electrodes labeled D1-, D1+, D1in-, and D1in+ in Figure 1. Because there is symmetry in the device, either of the two axes can be designated as the drive axis. Each axis has a capacitive petal for sensing oscillations as well; driving axis: labeled S1+ and S1- in Figure 1, sensing axis: labeled S2+ and S2- in Figure 1. The micro-gyro has additional plates that allow for electrostatic softening of the silicon springs, labeled B1, BT1, B2, and BT2 in Figure 1. Static bias voltages can be used to modify the amount of softening for each oscillation mode. In an ideal, symmetric device, the resonant frequencies of both modes are equal; however, unavoidable manufacturing imperfections in the machining of the device can cause asymmetries in the silicon structure of the device, resulting in a *frequency split* between the resonant frequencies of these two modes. The frequency split reduces the voltage scale used to measure the rate of rotation to which the device is subjected, and thus the sensitivity for detection of rotation is decreased. By adjusting the static bias voltages on the capacitor plates, frequencies of resonance for both modes are modified to match each other; this is referred to as the tuning of the device using an electrostatic biasing approach [11].

In order to extract the resonant frequencies of the vibration modes, there are two general methods: 1) open-loop and 2) closed-loop control [9]. In an open-loop system, we are measuring the frequency response along the drive axis over a 50Hz band and extract from the measurement the frequency split. A faster method is a closed-loop control, whereby the gyro is given an impulse disturbance and is allowed to oscillate freely between the two resonance frequencies, using a hardware platform to control the switch of the drive-angles.

3 Evolutionary Computation Using Open-Loop Measurement

3.1 Instrumentation Platform for Open-Loop Frequency Response

The open-loop measurement consists of exciting the drive axis with a sine wave at a given frequency and measuring the resulting amplitude. This is done repeatedly throughout the frequency spectrum (frequency range from 3,300Hz to 3,350Hz; 50Hz span; 800 points,). Because of cross-coupling between the different axes, two peaks in the amplitude response will appear at two different frequencies, showing the resonant frequencies of both axes (Figure 4). This takes approximately 1.4 minutes to complete using our instrumentation platform (Figure 2) and must be repeated at least three times to average out noise.

The platform includes one GPIB programmable power supply for DC voltage, a GPIB signal analyzer to extract frequency responses (from 3.3kHz to 3.35kHz) of the gyro in open-loop, and a computer (PC) to control the instruments and to execute the evolutionary optimization algorithms. The power supply DC voltage controls the electrostatic bias voltages (connected to the plates B1, BT1, B2, and BT2 in Figure 1) that are used to modify the amount of damping to each oscillation mode. The GPIB signal analyzer generates a sine wave with a variable frequency (from 3300 Hz to 3350 Hz with a stepsize of 62.5 mHz – 800 points, 50Hz span) on the drive electrode

(D1-, D1+, D1in-, and D1in+ in Figure 1) and measures the response signal on the sense electrode (S1-, S1+ in Figure 1) along the drive axis X.

A PC runs the instrument control tool, the measurement tool, and the evolutionary computation tool. The instrument control software sets up the static bias voltages using the GPIB power supply DC voltage and measures the frequency response along the X axis using the GPIB signal analyzer as shown in Figure 2. The software calculates the frequency split using peak fitting algorithms. Finally, the evolutionary computation software determines the new DC bias voltages from the frequency split. This procedure is repeated until a satisfactory (user-defined) frequency split is obtained.

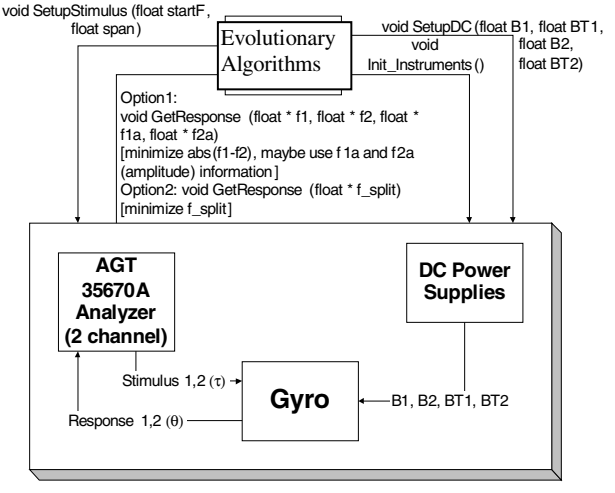


Fig. 2. Software interface between the modified Simulated Annealing/modified Genetic Algorithm (Dynamic Hill Climbing) and the Instrumentation Platform using a GPIB programmable power supply DC voltage and a signal analyzer. The modified Simulated Annealing and the modified Genetic Algorithm are running on a PC, which controls the bias voltages and receives the frequencies of both resonance modes.

3.2 Results of Evolutionary Computation

The MEMS post resonator micro-gyroscope is subject to an electro-static fine-tuning procedure, performed by hand, which is necessary due to unavoidable manufacturing inaccuracies. In order to fine-tune the gyro, 4 bias voltages applied to 8 capacitor plates have to be determined within a range of –60V to +15V. The manual tuning took several hours and obtained a frequency split of 200 mHz.

In order to fully automate the time-taking manual fine-tuning process, we have established a hardware/software interface to the existing manual gyro-tuning hardware-setup using commercial-off-the-shelf (COTS) components described in Section 3.1.

We developed and implemented two stochastic optimization techniques, for efficiently determining the optimal tuning voltages and incorporated them in the hardware/software interface: a modified simulated annealing related algorithm [7,8] and a

modified genetic algorithm with limited evaluation (Dynamic Hill Climbing) [5,6]. These optimization techniques have also been used for other space applications [4].

3.2.1 Simulated Annealing Approach

We were able to successfully fine-tune both the MEMS post-resonator gyroscope and MEMS disk-resonating gyroscope (a different gyro-design not discussed here) within one hour for the first time fully automatically. After only 49 iterations with the modified Simulated Annealing related optimization algorithm we obtained a frequency split of 125mHz within a 1V-discretization of the search space, starting with an initial split of 2.625Hz, using a 50Hz span and 800 points on the signal analyzer for the MEMS post-resonator gyroscope (Figure 3A). For the MEMS disk-resonating-gyroscope we obtained a frequency split of 250mHz/500mHz within a 0.1V-/0.01V-discretization of the search space, starting with an initial split of 16.125Hz/16.25Hz, after 249/12 iterations using a 200Hz span and 800 points on the signal analyzer (Figure 3B). All three results are better than what can be accomplished manually but worse than the results obtained by dynamic hill climbing (modified genetic algorithm). The reason for this is that instead of the peak fitting algorithm employed in the modified genetic algorithm approach a simplified, direct peak-finding procedure was used in the Simulated Annealing approach.

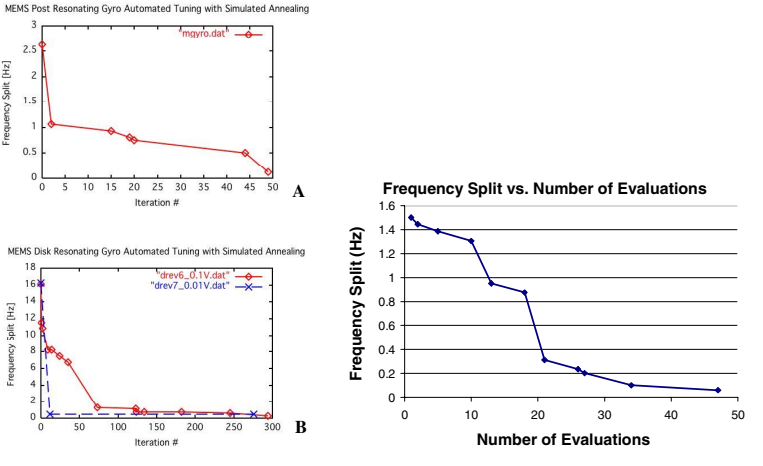


Fig. 3. Frequency split as a function of number of evaluations: Simulated Annealing iterations: (A) for the MEMS post-resonator gyroscope; (B) for the MEMS disk-resonating gyroscope. (C) Dynamic Hill Climbing algorithm (modified genetic algorithm).

3.2.2 Genetic Related Algorithm Approach

We were also able to fine-tune the MEMS post-resonator gyroscope within one hour fully automatically using a modified genetic algorithm: dynamic hill climbing.

Figure 3 (C) shows the progress of the optimization algorithm aimed at minimizing the frequency split. Each evaluation is a proposed set of bias voltages. Our optimization method only needed 47 evaluations (51 min) to arrive at a set of bias voltages that produced a frequency split of less than 100mHz.

Figures 4 and 5 show the frequency response for the unbiased micro-gyro respectively before and after tuning using the dynamic hill climbing and the peak fitting algorithm.

After optimization of the bias voltages (Figure 5), the frequency split has been minimized to less than 100mHz and the two peaks are indistinguishable on an HP spectrum analyzer at 62.5mHz / division (50Hz span, 800 points) setting, which was used during the optimization process.

The frequency split of 52mHz was verified using a higher resolution mode of the signal analyzer.

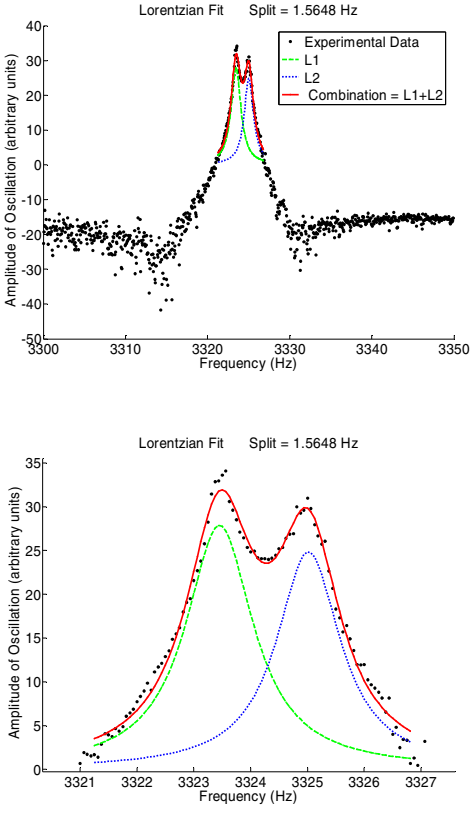


Fig. 4. Frequency response (top: 50Hz band, bottom: 6Hz band) before tuning using the modified genetic algorithm. The frequency split is 1564.8mHz. The Y axis is measured in dB. The initial values of the four bias voltages are: B1 = 14.00V, BT1 = 14.00V, B2 = 14.00V, and BT2 = 14.00V. The bottom picture shows a zoomed-in display of the frequency split over a 6Hz band.

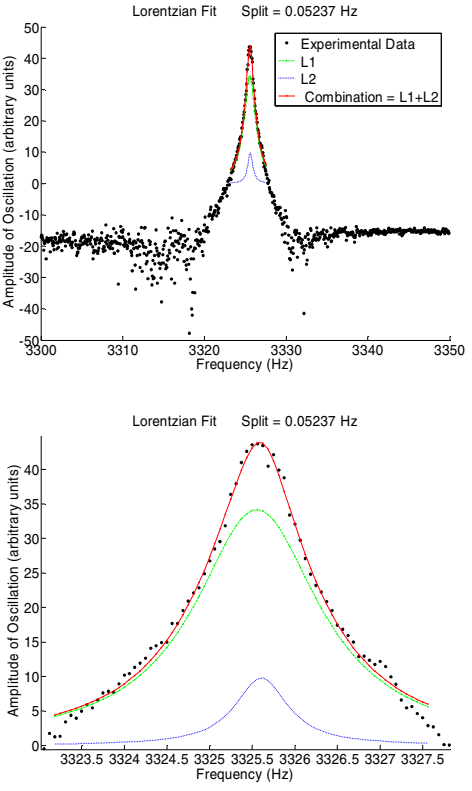


Fig. 5. Frequency response (top: 50Hz band, bottom: 5Hz band) after tuning using the modified genetic algorithm. The Y axis is measured in dB. The tuning frequency split is 52mHz. The optimized values of the four bias voltages are: B1 = 4.00V, BT1 = 4.00V, B2 = 14.00V, and BT2 = -16.00V. The bottom picture shows a zoomed-in display of the frequency split over a 4Hz band.

4 Hardware Platform Using Closed-Loop Frequency Response

The principle of closed-loop electrostatic biasing is based on measuring the resonance frequencies of the drive axis at a given set of bias voltages then swapping and driving the other axis, thereby extracting the resonant frequencies of both axes. An algorithm is then applied iteratively to modify the bias voltages until the resonant frequency of each axis is equal. A major advantage of this closed-loop approach is that the resonant frequencies can be extracted quickly (~1 second) as compared to the open-loop control system, which takes two orders of magnitude longer. The design of the electrostatic biasing approach is realized on an FPGA with augmented portability for future designs and implementations.

4.1 Control of the MEMS Micro-gyro

The closed-loop approach requires a closed-loop control whereby the gyro is given an impulse disturbance and is allowed to oscillate freely. This so-called “pinging” of the

vibration mode allows the gyroscope to immediately settle to its natural frequency. The corresponding frequency, F1, is measured from the sensing plate under the drive axis X. Because the device is relatively symmetric, the drive and sense axes are swapped and the other mode is pinged to get F2. The difference in the frequencies, i.e., frequency split, is determined very quickly using this technique, about 1.5 seconds, roughly 50 times faster than from the open-loop control method. This ability to quickly swap the drive axis with the sense axis is a feature of our FPGA Gyro Digital System (GDS).

The circuitry of the closed-loop control system includes a drive loop and a sense rebalance loop [3]. The drive loop takes the input from the "drive sense" petal (S1-, S1+ electrodes along the drive axis), and outputs the forcing signal to the "drive drive" petal electrodes (D1-, D1+, D1in- and D1in+ electrodes along the drive axis). The sense rebalance loop receives input from the "sense sense" petal (S2-, S2+ electrodes along the sense axis), and forces or rebalances the oscillations back along the drive axis with a forcing signal to the "sense drive" (D2-, D2+, D2in- and D2in+ electrodes). The magnitude of this forcing function in the rebalance loop is related to the angular rate of rotation. The closed-loop control has also several scaling coefficients, denoted as K_i, which allow for a mixing of the sensed signals from both axes and a swapping of the drive- and sense-axis, thus permitting the tuning algorithm to measure the resonance frequency along the X- or Y-axis, or, indeed, any axis between X and Y [9].

The drive loop implements an Automatic Gain Control (AGC) loop combined with finite impulse response (FIR) filters. Because the amplitude of the freely oscillating drive axis will naturally decay, the AGC is implemented in a way to lightly drive or damp, depending on the circumstance, the drive axis so that the amplitude of the driven signal is constant and the gyroscope is maintained in an oscillation mode at the natural frequency. The optimal parameters of the FIR filters and the AGC loop to maintain the oscillation of the gyroscope have been determined by the UCLA team using a DSP measurement system and a UCLA MatLab modeling tool [12].

4.2 Gyro Digital System (GDS)

The system used to implement the control, operation, and observability of the micro-gyro is referred to as the Gyro Digital System (GDS). Figure 6 illustrates the implementation of the analog and digital systems used to control the micro-gyro. The key circuit elements that allow proper operation of the micro-gyro include the audio codec (Stereo Digital to Analog Converter DAC), high voltage Analog to Digital Converters (ADCs), IEEE-1294 Enhanced Parallel Port (EPP) interface replaced by a UART interface, frequency measurement and the Digital Signal Processor (DSP) functionality integrated into a Xilinx Virtex II FPGA.

The audio codec is used to translate the analog sensing signals for both the drive and the sense axes. Its stereo capabilities allow for two inputs and two outputs. The high-voltage DACs are utilized for the setting of the electrostatic bias voltages on the gyroscope, which range from +15V to -60V. The parallel port interface allows for user input/output capabilities. The user can configure the coefficients for the finite impulse response (FIR) filters along with the scaling coefficients (K1 through K8) and automatic gain control (AGC) proportional integral (PI) coefficients (K_p and K_i). The codec is configured through this interface as well.

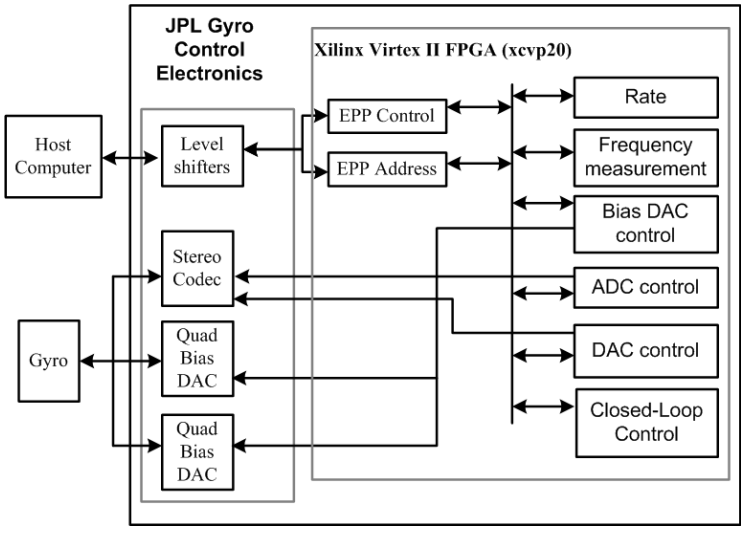


Fig. 6. Block diagram of the entire closed-loop control system

4.3 Results

Using this FPGA digital control system, the micro-gyro was operated for a period of several hours and provided a frequency measurement that was stable to 1 mHz.

This FPGA system has not yet been tested in the mode where the drive- and sense-axes are swapped, but we have performed experiments using a DSP platform controlled by a Simulink environment running on a PC that demonstrates the feasibility of the closed-loop approach. In Figure 7 we show the frequency response of a non-tuned MEMS gyroscope (B1=B2=BT1=BT2=14V) with two peaks for each of the

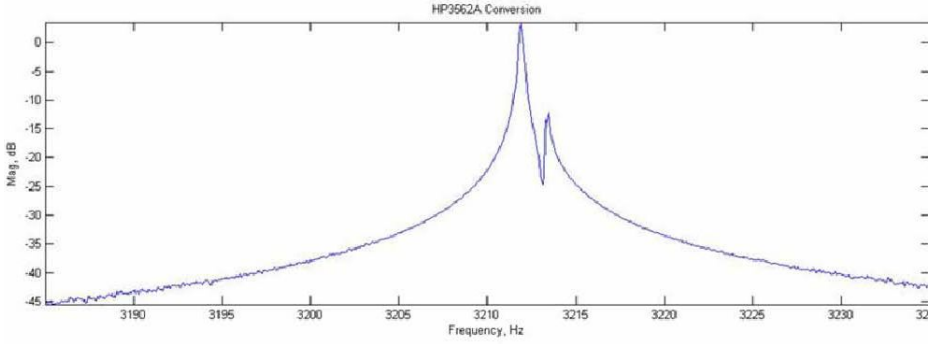


Fig. 7. Bode magnitude of the experimental frequency response data for a non-tuned MEMS micro-gyroscope (B1=B2=BT1=BT2=14V)

resonance frequencies. Using a closed-loop control, the UCLA team was able to find the correct AGC and FIR filter parameters to maintain the gyro in an oscillating mode at the natural frequency. The DSP platform measured the frequency of both modes by swapping the drive and sense axis (F1=3210.73Hz and F2=3212.2Hz). Keeping the value of the AGC and FIR parameters constant and changing the value of the DC bias voltage, we were able to maintain the gyro in an oscillation mode and to extract both resonance frequencies, which have changed due to the update DC bias voltage. The next step is to couple the FPGA frequency measurement with the genetic algorithm (GA) and the simulated annealing (SA) running on the PC. The ultimate goal is to implement the GA and the SA on a microprocessor integrated into a FPGA.

5 Conclusion

The tuning method for MEMS micro-gyroscopes based on evolutionary computation shows great promise as a technology to replace the cumbersome, manual tuning process. We demonstrated, using an open-loop measurement, that we can, for the first time fully automatically, obtain a four times smaller frequency split at a tenth of the time, compared to human performance. We also showed that the closed-loop system has the option of swapping the drive- and sense-axes, thus decreasing the time required for tuning by more than a factor of fifty compared to the open-loop approach. Additionally, a future design will include a microprocessor on-chip to allow for in-situ retuning of the MEMS micro-gyroscope if there is an unexpected change in the behavior due to radiation, temperature shift, or other faults.

The novel capability of fully automated gyro tuning, integrated in a single device next to the gyro, enables robust, low-mass and low-power high-precision Inertial Measurement Unit (IMU) systems to calibrate themselves autonomously during ongoing missions, e.g., Mars Ascent Vehicle.

References

1. Leland, R.P., “Adaptive mode tuning vibrational gyroscopes”, IEEE Trans. Control Systems Tech., vol. 11, no. 2, pp242-247, March 2003.
2. Painer C.C., Shkel A.M., “Active structural error suppression in MEMS vibratory rate integrating gyroscopes”, IEEE Sensors Journal, vol.3, no.5, pp. 595-606, Oct. 2003.
3. Y. Chen, R. M’Closkey, T. Tran and B. Blaes. “A control and signal processing integrated circuit for the JPL-Boeing micromachined gyroscopes” (submitted to IEEE)
4. R. J. Terrile, et al., “Evolutionary Computation Technologies for Space Systems”, in Proceedings of the IEEE Aerospace Conference, Big Sky, March 2005
5. J.H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, Michigan, 1975.
6. D. Yuret, M. de la Maza, “Dynamic Hill Climbing – Overcoming limitations of optimization teqniques”, AI Laboratory, MIT, Cambridge, MA 02139, USA
7. N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, “Equation of State Calculation by Fast Computing Machines,” *J. of Chem. Phys.*, **21**, 1087--1091, 1953.
8. S. Kirkpatrick, C.D. Gelat, M.P. Vecchi,, “Optimization by Simulated Annealing,” *Science*, **220**, 671--680, 1983.

9. K. Hayworth, “Continuous Tuning and Calibration of Vibratory Gyroscopes”, In NASA Tech Brief, Oct 2003 (NPO-30449)
10. M. I. Ferguson, D. Keymeulen, C. Peay, K. Yee, D. Li, “Effect of Temperature on MEMS Vibratory Rate Gyroscope”, in Proceedings of the IEEE Aerospace Conference, Big Sky, March 2005.
11. K. Hayworth, K. Shcheglov, T. Humphreys, A. Challoner, “Electrostatic Spring Softening in Redundant Degree of Freedom resonators”, patent US 6,823,734 B1, JPL and Boeing, Nov. 30, 2004.
12. R. M’Closkey and D. Kim, “Real-time tuning of JPL-Boeing MEMS gyro”, personal communication, JPL, March 2005.

Author Index