Emergence, Complexity and Computation ECC

Susan Stepney
Andrew Adamatzky *Editors*

Inspired by Nature

Essays Presented to Julian F. Miller on the Occasion of his 60th Birthday

Springer

Emergence, Complexity and Computation

Volume 28

The Emergence, Complexity and Computation (ECC) series publishes new developments, advancements and selected topics in the fields of complexity, computation and emergence. The series focuses on all aspects of reality-based computation approaches from an interdisciplinary point of view especially from applied sciences, biology, physics, or chemistry. It presents new ideas and interdisciplinary insight on the mutual intersection of subareas of computation, complexity and emergence and its impact and limits to any computing based on physical limits (thermodynamic and quantum limits, Bremermann's limit, Seth Lloyd limits…) as well as algorithmic limits (Gödel's proof and its impact on calculation, algorithmic complexity, the Chaitin's Omega number and Kolmogorov complexity, non-traditional calculations like Turing machine process and its consequences,…) and limitations arising in artificial intelligence field. The topics are (but not limited to) membrane computing, DNA computing, immune computing, quantum computing, swarm computing, analogic computing, chaos computing and computing on the edge of chaos, computational aspects of dynamics of complex systems (systems with self-organization, multiagent systems, cellular automata, artificial life,…), emergence of complex systems and its computational aspects, and agent based computation. The main aim of this series it to discuss the above mentioned topics from an interdisciplinary point of view and present new ideas coming from mutual intersection of classical as well as modern methods of computation. Within the scope of the series are monographs, lecture notes, selected contributions from specialized conferences and workshops, special contribution from international experts.

More information about this series at http://www.springer.com/series/10624

Susan Stepney · Andrew Adamatzky
Editors

Inspired by Nature

Essays Presented to Julian F. Miller on the Occasion of his 60th Birthday

Springer

Editors
Susan Stepney
Department of Computer Science
University of York
York
UK

Andrew Adamatzky
Unconventional Computing Centre
University of the West of England
Bristol
UK

ISSN 2194-7287 ISSN 2194-7295 (electronic)
Emergence, Complexity and Computation
ISBN 978-3-319-67996-9 ISBN 978-3-319-67997-6 (eBook)
https://doi.org/10.1007/978-3-319-67997-6

Library of Congress Control Number: 2017952894

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Julian Francis Miller

This book is a tribute to Julian Francis Miller's breadth of ideas and achievements in computer science, evolutionary algorithms and genetic programming, electronics, unconventional computing, artificial chemistry, and theoretical biology. Well-known for both Cartesian Genetic Programming and evolution *in materio*, Julian has further interests from quantum computing to artificial chemistries. He has over 200 refereed publications (http://www.cartesiangp.co.uk/jfm-publications.html); here, we highlight just a few of his major accomplishments.

Julian started his life in science as mathematical physicist working on the interaction of solitons in various nonlinear partial differential equations such as the sine-Gordon equation [3, 5], and the modified Korteweg-de Vries equation [4]. He entered classical computer science with his paper on synthesis and optimisation of networks implemented with universal logic modules [1, 16, 30]. Julian's interest in optimisation led him to genetic algorithms, which he employed for optimisation of field-programmable arrays [26], Reed-Muller logical functions [15], finite-state machines [2], and evolving combinatorial logic circuits [18, 22, 23] and non-uniform cellular automata [27, 28].

Julian combined his interests in physics and computer science in work on constant complexity algorithm for solving Boolean satisfiability problems on quantum computers, and quantum algorithm for finding multiple matches [31]. Julian's ideas in optimisation of circuits and quantum computing are reflected in Younes' Chapter "Using Reed-Muller Expansions in the Synthesis and Optimization of Boolean Quantum Circuits".

Julian's interest in combining natural processes and computation expanded from physics to include the exciting world of biological processes, such as evolution and morphogenesis. He used principles of morphogenesis to evolve computing circuits and programs [14, 17, 19]. These aspects of Julian's work are reflected in Chapters "Evolvable Hardware Challenges: Past, Present and the Path to a Promising Future" by Haddow and Tyrell, "Artificial Development" by Kuyucu et al., and Banzhaf's "Some Remarks on Code Evolution with Genetic Programming".

In 2000, Julian, together with Peter Thomson, presented a fully developed concept of Cartesian Genetic Programming (CGP) [24]. There, a program is genetically represented as a directed graph, including automatically defined functions [29] and self-modifying operators [10]. This approach has become very popular, because it allows the discovery of efficient solutions across a wide range of mathematical problems and algorithms. Several chapters of the book manifest the success of CGP in diverse application areas: "Designing Digital Systems Using Cartesian Genetic Programming and VHDL" by Henson et al.; "Breaking the Stereotypical Dogma of Artificial Neural Networks with Cartesian Genetic Programming" by Khan and Ahmad; "Approximate Computing: An Old Job for Cartesian Genetic Programming?" by Sekanina; "Medical Applications of Cartesian Genetic Programming" by Smith and Lones; "Multi-step Ahead Forecasting Using Cartesian Genetic Programming" by Dzalbs and Kalganova; "Cartesian Genetic Programming for Control Engineering" by Clarke; "Bridging the Gap Between Evolvable Hardware and Industry Using Cartesian Genetic Programming" by Vasicek; "Combining Local and Global Search: A Multi-objective Evolutionary Algorithm for Cartesian Genetic Programming" by Kaufmann and Platzner.

In 2001, Miller and Hartman published "Untidy evolution: Evolving messy gates for fault tolerance" [21]. Their ideas of exploiting of "messiness" to achieve "optimality"—"natural evolution is, par excellence, an algorithm that exploits the physical properties of materials"—gave birth to a new field of unconventional computing: evolution *in materio* [7, 12, 20]. The evolution *in materio* approach has proved very successful in discovering logical circuits in liquid crystals [11–13], disordered ensembles of carbon nanotubes [6, 7, 25] (and Chapter "Evolution in Nanomaterio: The NASCENCE Project" by Broersma), slime mould (Chapter "Discovering Boolean Gates in Slime Mould" by Harding et al.), living plants (Chapter "Computers from Plants We Never Made: Speculations" by Adamatzky et al.), and reaction-diffusion chemical systems ("Chemical Computing Through Simulated Evolution" by Bull et al.).

Julian's inspiration from nature has not neglected the realm of chemistry: he has exploited chemical ideas in the development of a novel form of artificial chemistry,

used to explore emergent complexity [8, 9]. Chapter "Sub-Symbolic Artificial Chemistries" by Faulkner et al. formalises this approach.

The book will be a pleasure to explore for readers from all walks of life, from undergraduate students to university professors, from mathematicians, computers scientists, and engineers to chemists and biologists.

York, UK
Bristol, UK
June 2017

Susan Stepney
Andrew Adamatzky

References

1. Almaini, A.E.A., Miller, J.F., Xu, L.: Automated synthesis of digital multiplexer networks. IEE Proc. E (Comput. Dig. Tech.) **139**(4), 329–334 (1992)
2. Almaini, A.E.A., Miller, J.F., Thomson, P., Billina, S.: State assignment of finite state machines using a genetic algorithm. IEE Proc. Comp. Dig. Tech. **142**(4), 279–286 (1995)
3. Bryan, A.C., Miller, J.F., Stuart, A.E.G.: A linear superposition formula for the sine-Gordon multisoliton solutions. J. Phys. Soc. Japan **56**(3), 905–911 (1987)
4. Bryan, A.C., Miller, J.F., Stuart, A.E.G.: Superposition formulae for multisolitons. II.—The modified Korteweg-de Vries equation. Il Nuovo Cimento B **101**(6), 715–720 (1988)
5. Bryan, A.C., Miller, J.F., Stuart, A.E.G.: Superposition formulae for sine-Gordon multisolitons. Il Nuovo Cimento B **101**(6), 637–652 (1988)
6. Dale, M., Miller, J.F., Stepney, S.: Reservoir computing as a model for *in materio* computing. In: Advances in Unconventional Computing, pp. 533–571. Springer, Berlin (2017)
7. Dale, M., Miller, J.F., Stepney, S., Trefzer, M.A.: Evolving carbon nanotube reservoir computers. In: International Conference on Unconventional Computation and Natural Computation, pp. 49–61. Springer, Berlin (2016)
8. Faulconbridge, A., Stepney, S., Miller, J.F., Caves, L.: RBN-world: The hunt for a rich AChem. In: ALife XII, Odense, Denmark, August 2010, pp. 261–268. MIT Press, Cambridge (2010)
9. Faulconbridge, A., Stepney, S., Miller, J.F., Caves, L.S.D.: RBN-World: A sub-symbolic artificial chemistry. In: ECAL 2009, Budapest, Hungary, September 2009, vol. 5777 of LNCS, pp. 377–384. Springer, Berlin (2011)
10. Harding, S., Miller, J.F., Banzhaf, W.: Developments in *Cartesian Genetic Programming*: Self-modifying CGP. Genet. Program. Evolvable Mach. **11**(3/4), 397–439 (2010)
11. Harding, S., Miller, J.: Evolution in materio: Initial experiments with liquid crystal. In: Proceedings of NASA/DoD Conference on Evolvable Hardware, 2004, pp. 298–305. IEEE (2004)
12. Harding, S., Miller, J.F.: A scalable platform for intrinsic hardware and in materio evolution. In: Proceedings of NASA/DoD Conference on Evolvable Hardware, 2003, pp. 221–224. IEEE (2003)
13. Harding, S., Miller, J.F.: Evolution in materio: Evolving logic gates in liquid crystal. In: Proc. Eur. Conf. Artif. Life (ECAL 2005), Workshop on Unconventional Computing: From Cellular Automata to Wetware, pp. 133–149. Beckington, UK (2005)
14. Liu, H., Miller, J.F., Tyrrell, A.M.: A biological development model for the design of robust multiplier. In: Workshops on Applications of Evolutionary Computation, pp. 195–204. Springer, Berlin, Heidelberg (2005)
15. Miller, J., Thomson, P., Bradbeer, P.: Ternary decision diagram optimisation of reed-muller logic functions using a genetic algorithm for variable and simplification rule ordering. In: Evolutionary Computing, pp. 181–190 (1995)

16. Miller, J.F., Thomson, P.: Highly efficient exhaustive search algorithm for optimizing canonical Reed-Muller expansions of boolean functions. Int. J. Electr. **76**(1), 37–56 (1994)
17. Miller, J.: Evolving a self-repairing, self-regulating, french flag organism. In: Genetic and Evolutionary Computation—GECCO 2004, pp. 129–139. Springer, Berlin, Heidelberg (2004)
18. Miller, J., Thomson, P.: Restricted evaluation genetic algorithms with tabu search for optimising boolean functions as multi-level and-exor networks. In: Evolutionary Computing, pp. 85–101 (1996)
19. Miller, J.F.: Evolving developmental programs for adaptation, morphogenesis, and self-repair. In: European Conference on Artificial Life, pp. 256–265. Springer, Berlin, Heidelberg (2003)
20. Miller, J.F., Downing, K.: Evolution in materio: looking beyond the silicon box. In: Proceedings of NASA/DoD Conference on Evolvable Hardware, pp. 167–176. IEEE (2002)
21. Miller, J.F., Hartmann, M.: Untidy evolution: evolving messy gates for fault tolerance. In: International Conference on Evolvable Systems, pp. 14–25. Springer, Berlin, Heidelberg (2001)
22. Miller, J.F., Job, D., Vassilev, V.: Principles in the evolutionary design of digital circuits—part I. Genetic Prog. Evolvable Mach. **1**(1–2), 7–35 (2000)
23. Miller, J.F., Thomson, P.: Combinational and sequential logic optimisation using genetic algorithms. In: GALESIA. First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995 (Conf. Publ. No. 414), pp. 34–38. IET (1995)
24. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: European Conference on Genetic Programming, pp. 121–132. Springer, Berlin Heidelberg (2000)
25. Mohid, M., Miller, J.F., Harding, S.L., Tufte, G., Massey, M.K., Petty, M.C.: Evolution-in-materio: solving computational problems using carbon nanotube–polymer composites. Soft Comput. **20**(8), 3007–3022 (2016)
26. Thomson, P., Miller, J.F.: Optimisation techniques based on the use of genetic algorithms (gas) for logic implementation on fpgas. In: IEE Colloquium on Software Support and CAD Techniques for FPGAs, pp. 4–1. IET (1994)
27. Vassilev, V., Miller, J., Fogarty, T.: The evolution of computation in co-evolving demes of non-uniform cellular automata for global synchronisation. In: Advances in Artificial Life, pp. 159–169 (1999)
28. Vassilev, V.K., Miller, J.F., Fogarty, T.C.: Co-evolving demes of non-uniform cellular automata for synchronisation. In: Proceedings of the First NASA/DoD Workshop on Evolvable Hardware, 1999, pp. 111–119. IEEE (1999)
29. Walker, J.A., Miller, J.F.: The automatic acquisition, evolution and re-use of modules in Cartesian genetic programming. IEEE Trans. Evol. Comput. **12**, 397–417 (2008)
30. Xu, L., Almaini, A.E.A., Miller, J.F., McKenzie, L.: Reed-Muller universal logic module networks. IEE Proc. E-Computers Dig. Tech. **140**(2), 105–108 (1993)
31. Younes, A., Rowe, J., Miller, J.: A hybrid quantum search engine: A fast quantum algorithm for multiple matches. arXiv preprint quant-ph/0311171 (2003)

Contents

Part I
Evolution and Hardware

Evolvable Hardware Challenges: Past, Present and the Path to a Promising Future

Pauline C. Haddow and Andy M. Tyrrell

Abstract The ability of the processes in Nature to achieve remarkable examples of complexity, resilience, inventive solutions and beauty is phenomenal. This ability has promoted engineers and scientists to look to Nature for inspiration. Evolvable Hardware (EH) is one such form of inspiration. It is a field of evolutionary computation (EC) that focuses on the embodiment of evolution in a physical media. If EH could achieve even a small step in natural evolution's achievements, it would be a significant step for hardware designers. Before the field of EH began, EC had already shown artificial evolution to be a highly competitive problem solver. EH thus started off as a new and exciting field with much promise. It seemed only a matter of time before researchers would find ways to convert such techniques into hardware problem solvers and further refine the techniques to achieve systems that were competitive (better) than human designs. However, almost 20 years on, it appears that problems solved by EH are only of the size and complexity of that achievable in EC 20 years ago and seldom compete with traditional designs. A critical review of the field is presented. Whilst highlighting some of the successes, it also considers why the field is far from reaching these goals. The chapter further redefines the field and speculates where the field should go in the next 10 years.

P.C. Haddow (✉)
CRAB Lab, Department of Computing and Information Science,
Norwegian University of Science and Technology, Trondheim, Norway
e-mail: pauline@ntnu.no

A.M. Tyrrell
Intelligent Systems & Nano-Science Group, Department of Electronic Engineering,
University of York, York, UK
e-mail: andy.tyrrell@york.ac.uk

S. Stepney and A. Adamatzky (eds.), *Inspired by Nature*, Emergence,
Complexity and Computation 28, https://doi.org/10.1007/978-3-319-67997-6_1

1 Introduction

Yao and Higuchi published a paper in 1999 entitled "Promises and Challenges of Evolvable Hardware" which reviewed the progress and possible future direction of what was then a new field of research, Evolvable Hardware [1]. A little more than ten years on, this chapter[1] considers the progress of this research field, both in terms of the successes achieved to date and also the failure to fulfil the promises of the field highlighted in the 1999 paper. Through a critical review of the field, the authors' intention is to provide a realistic status of the field today and highlight the fact that the challenges remain, to redefine the field (what should be considered as Evolvable Hardware) and to propose a revised future path.

In the mid 1990s, researchers began applying Evolutionary Algorithms (EAs) to a computer chip that could dynamically alter the hardware functionality and physical connections of its circuits [2–10]. This combination of EAs with programmable electronics—e.g. Field Programmable Gate Arrays (FPGAs) and Field Programmable Analogue Arrays (FPAAs)—spawned a new field of EC called Evolvable Hardware.

The EH field has since expanded beyond the use of EAs on simple electronic devices to encompass many different combinations of EAs and biologically inspired algorithms (BIAs) with various physical devices or simulations thereof. Further, the challenges inherent in EH have led researchers to explore new BIAs that may be more suitable as techniques for EH.

In this chapter we define the field of EH and split the field into the two related but different sub-fields: Evolvable Hardware Design (EHD) and Adaptive Hardware (AH).

Evolvable Hardware, as the name suggests, should have a connection to embodiment in a real device. However, in a number of EH papers, results produced are interpreted as hardware components e.g. logic gates, illustrated as a circuit diagram and justified as a hardware implementation, despite the lack of a realistic hardware simulator. In this chapter such work is not considered as Evolvable Hardware. The lack of grounding in real hardware, either physical or through realistic simulators, defies their inclusion in the field of EH. In other cases, authors attempt to speed up their EC process by implementing part or all of the BIA in hardware. In other words, hardware accelerators are certainly not, as defined in this chapter, Evolvable Hardware.

In this chapter we define **the field of Evolvable Hardware (EH) as the design or application of EAs and BIAs for the specific purpose of *creating*[2] physical devices and novel or optimised physical designs.**

[1]This chapter is a revised and updated version of: Pauline C. Haddow, Andy M. Tyrrell (2011) Challenges of evolvable hardware: past, present and the path to a promising future. Genetic Programming and Evolvable Machines 12(3):183–215.

[2]"creating" refers to the creation of a physical entity.

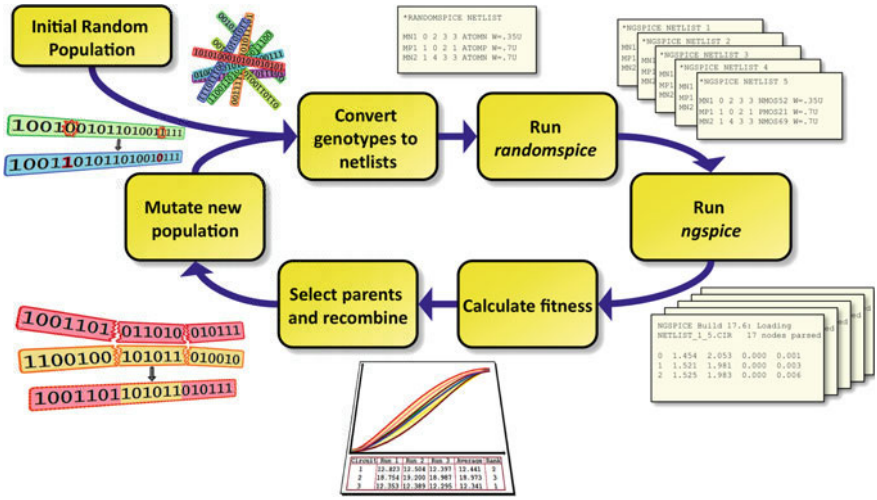

Fig. 1 An evolvable hardware cycle, for circuit design where high-fidelity simulation is used [13]

With this in mind, there are some successes in the field including analogue and digital electronics, antennas, MEMS chips, optical systems as well as quantum circuits. Section 3 presents an overview of some of these successes.

However, let us step back for a minute and consider what evolving hardware should consist of. Figure 1 illustrates an example of EH where an accurate model of the device physics is applied to produce the fitness function used to evaluate the efficacy of the circuits. So while the evolutionary loop is using no hardware, the simulation models used are very accurate with respect to the final physical system (in this case transistors with variable characteristics). Other forms of realistic hardware designs may include device simulators, as in [11], or actual physical devices, as in [12].

Such work can be classed under the subfield of Evolvable Hardware Design. The goal is novel or optimised non-adaptive hardware designs, either on physical hardware or as solutions generated from realistic simulators.

The sub-field of Adaptive Hardware can be defined as the application of BIAs to endow real hardware, or simulations thereof, with some *adaptive characteristics*. These adaptive characteristics enable changes within the EH design/device, as required, so as to enable them to continue operating correctly in a changing environment. Such environmental pressure may consist of changes in the operational requirements i.e. functionality change requirements, or maintenance of functionality e.g. robustness, in the presence of faults. Examples of such Adaptive Hardware include an FPAA that can change its function as operational requirements change or a circuit on an FPGA that can "evolve" to heal from radiation damage.

The remainder of the chapter is structured as follows: Sect. 2 provides a definition of the field together with the inherent advantages one can expect from such a field. Some actual success stories are reviewed in a little more detail in Sect. 3.

Section 4 considers many of the challenges that still face the community. Some newer approaches that are, or might be, applied to evolvable hardware are discussed in Sect. 5. Section 6 provides some thoughts on the future of evolvable hardware and Sect. 7 concludes with a short summary.

2 Defining Evolvable Hardware

2.1 Evolvable Hardware Characteristics

In the literature, it is difficult to find a set of characteristics that the community has agreed upon for what characterises an EH system should have. Indeed most work does not address this issue at all. However, there are a number of fundamental characteristics that should be included in such a definition:

- The system will be evolved not designed (an obvious one, but worth pointing out) N.B.: The term "evolved" is used to encompass any BIA and does not assume that an EA is used.
- It is often not an optimal design, but will be fit for the purpose (typical of biological systems in general—due to the evolution process).
- Typically, evolved systems do show levels of fault tolerance not seen in designed systems (again typical of biological systems).
- They may be designed to be adaptable to environment change.
  Such adaptivity is dependent upon when the evolutionary cycle stops. This is not necessarily true for all evolved systems, for example those that stop once the goal has been reached (again this is a characteristic of all biological systems).
- They produce systems that are most often unverifiable formally; indeed in many cases it is difficult to analyze the final system to see how it performs the task.

What is EH?

From considering the characteristics involved in EH one can start to be more precise as to what is and is not EH. Evolvable hardware will include a form of evolutionary loop, a population (even very small ones), variation operators and selection mechanisms. These will vary from one implementation to another but will be present. In addition to this, there are a number of other characteristics that are important to consider:

- The final embodiment of the evolutionary process (the final phenotype) should be hardware, or at least have the potential to be implemented in hardware. An obvious example would be evaluating the evolving design on a given FPGA device and for the evolved design to be implemented on that FPGA. A less obvious example might be the evolution of part or all of a VLSI device where the fabrication of such a device may not be feasible at that point in time i.e. in terms of cost, but could later be fabricated.

- The evolutionary process may be applied to the investigation of some future technology medium (non electronics).
- The evolution can be either intrinsic ("on-chip") or extrinsic ("off-chip", applying a realistic simulator).
- The evolved design should solve or illustrate a viable approach to solving a "useful" problem—application or device—not one that could be solved by traditional means e.g. *n*-bit multipliers.
- The final embodied implementation includes features that traditional designs could not produce e.g. inherent fault-tolerance.

What is not EH?

The following are not considered to be evolvable hardware in this chapter:

- Simple optimisation processes (where "simple" refers to nothing more than when the optimization process itself is trivial).
- Where there is never any intension or reason ever to embed the final solution into hardware (whatever the platform).
- Where there are no benefits from evolving a solution.
- Where hardware is applied purely as an accelerator.

Taking a few examples from the literature and placing these into this context, we can summaries these in Table 1.

2.2 Possible Advantages of Evolvable Hardware

Evolvable Hardware is a method for circuit or device design (within some media) that uses inspiration from biology to create techniques that enable hardware designs to emerge rather than be designed. At first sight, this would seem very appealing. Consider two different systems, one from nature and one human engineered: the human immune system (nature design) and the computer that we are writing this chapter on (human design). Which is more complex? Which is most reliable? Which is optimised the most? Which is most adaptable to change?

The answer to almost all such questions is the human immune system. This is usually the case when one considers most natural systems. Possibly the only winner from the engineered side might be the question, "Which is optimised the most?" However, this will depend on how you define optimised. While this is appealing, anyone who has used any type of evolutionary system knows that it is not quite that straightforward. Nature has a number of advantages over current engineered systems, not least of which are time (most biological systems have been around for

Table 1 Comparison of different proposed evolvable hardware systems

	Analogue/Digital	Intrinsic/Extrinsic	Real hardware	Realistic simulator	Hardware accelerator?	EH?
Higuchi's team e.g. (2, 11, 52)	Digital	Intrinsic	Yes	No	No	Yes
JPL team e.g. (16, 32, 47)	Analogue	Intrinsic	Yes	No	No	Yes
N-bit digital circuit evolution e.g. (39, 48, 71)	Digital	Extrinsic	No	No	No	No
Lohn's team e.g. (34, 83, 91)	Analogue	Extrinsic	Yes	Yes	No	Yes
NanoCMOS project e.g. (13)	Analogue (for digital)	Extrinsic	Potentially	Yes	No	Yes
EvoDevo work e.g. (59, 69, 73)	Digital	Both	Yes	No	No	Yes
Koza's team e.g. (23, 26, 28)	Analogue	Extrinsic	No	Yes	No	Yes
Sekanina's team e.g. (101)	Digital	Both	Yes	No	No	Yes
Neural network (NN) projects e.g. (5, 6, 61)	Both	Both	Sometimes	Sometimes	Yes	No
Liquid crystal evolution e.g. (84, 85, 86)	Analogue	Intrinsic	Yes	No	No	Yes

If the structure and connections of an NN project are actually evolved and may be developed then it could be considered as EH

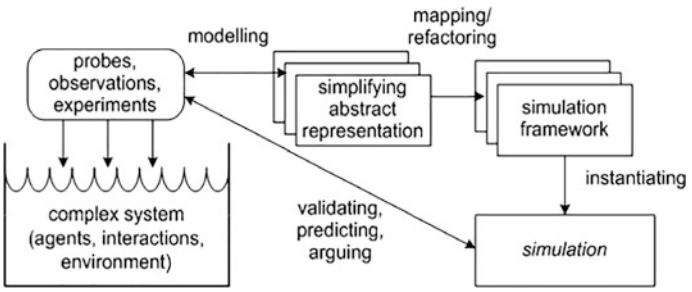

Fig. 2 Conceptual framework [14]

quite a long time) and resources (most biological systems can produce new resources as they need them, e.g. new cells). In addition, we need to be very careful as practitioners of bio-inspired techniques that we pay due regard to the actual biological systems and processes. All too often researchers read a little of the biology and charge in applying this to an engineering system without really understanding the biology or the consequences of the simplifications they make. On the other hand, we are not creating biological systems but rather artificial systems. An exact replication of a complex biological process may be neither needed nor appropriate for the hardware design in question. Improvements in BIAs for EH cannot be justified purely by their "improvements in biological realism", as may be seen in the literature. Such improvements should be reflected in improvements in the efficiency of the BIA or/and the hardware solutions achievable.

It is important to note that biological-like characteristics will be observed in an engineering system that is referred to as bio-inspired but that has in fact little or no real resemblance to the biology that "inspired" it. It is thus important to have a proper framework when using bio-inspired techniques. A framework for such systems is suggested in [14] and illustrated in Fig. 2. We are, however, not suggesting that every EH researcher needs to turn to biological experimentation. However, the biological inspiration and validation does need to come from real biological knowledge.

To summarise, use evolvable hardware where appropriate, associate evolution with something that has some real connection with the hardware and if using a simulator, make sure the simulator is an accurate one for the technology of choice and ensure that your biological inspiration is not only biologically inspired but also beneficial to the evolved hardware.

3 Platforms for Intrinsic EH

Evolvable hardware has, on the whole, been driven by the hardware/technology available at a particular time, and usually this has been Electronic hardware. While this chapter does not go into great detail on the different hardware platforms, it is at

least an important enough driver for the subject to mention a few of the main platforms/devices. These are presented under Analogue and Digital Devices.

(A) **Analogue Devices**

Field Programmable Analogue Arrays (FPAA)

The Lattice Semiconductor ispPAC devices are typical of what is currently available from manufacturers that allow reconfiguration of "standard" analogue blocks (in this case based around OpAmps). The isp family of programmable devices provide three levels of programmability: the functionality of each cell; the performance characteristics for each cell and the interconnect at the device architectural level. Programming, erasing, and reprogramming are achieved quickly and easily through standard serial interfaces. The device is depicted in Fig. 3. The basic active functional element of the ispPAC devices is the PACell, which, depending on the specific device architecture, may be an instrumentation amplifier, a summing amplifier or some other elemental active stage. Analogue function modules, called PACblocks, are constructed from multiple PACells to replace traditional analogue components such as amplifiers and active filters, eliminating the need for most external resistors and capacitors. Requiring no external components, ispPAC devices flexibly implement basic analogue functions such as precision filtering,

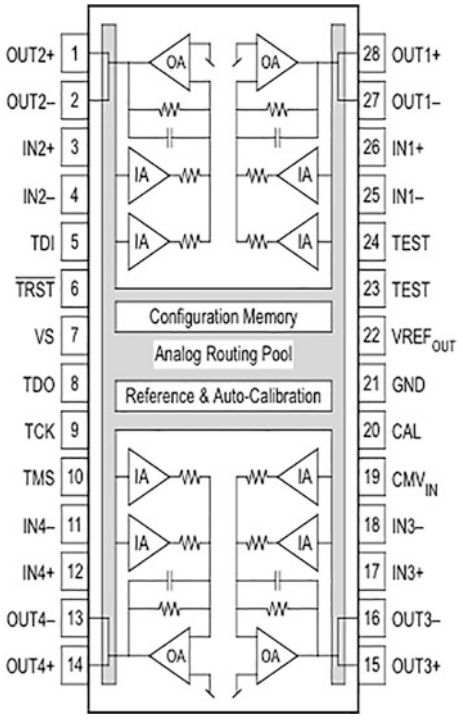

Fig. 3 PACell structure [15]

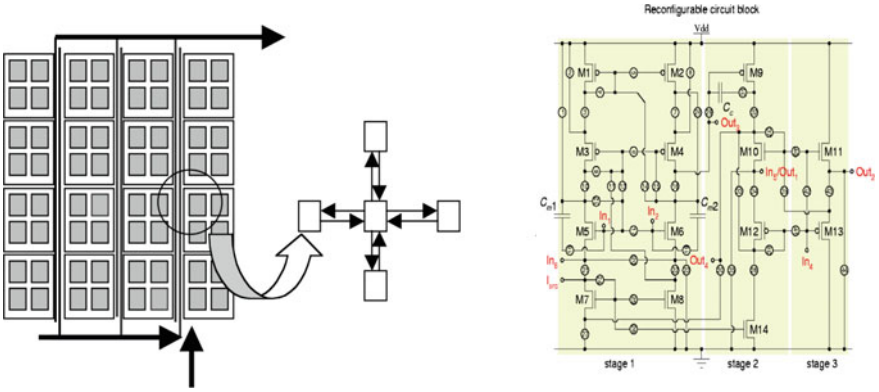

Fig. 4 The FPTA2 architecture. Each cell contains additional capacitors and programmable resistors (not shown) [16]

summing/differencing, gain/attenuation and conversion. An issue for someone wishing to undertake evolution on these devices is that the changes that can be made are at a relatively high functional level i.e. at the OpAmp level. This, together with the overhead for changing functionality, have limited their use in the EH field.

JPL Field Programmable Transistor Array

The Field Programmable Transistor array, designed by the group at NASA, was the first such analogue device specifically designed with evolvable hardware in mind. The FPTA has transistor level reconfigurability and supports any arrangement of programming bits without danger of damage to the chip (as is the case with some commercial devices). Three generations of FPTA chips have been built and used in evolutionary experiments. The latest chip, the FPTA-2, consists of an 8 × 8 array of reconfigurable cells (see Fig. 4). The chip can receive 96 analogue/digital inputs and provide 64 analogue/digital outputs. Each cell is programmed through a 16-bit data bus/9-bit address bus control logic, which provides an addressing mechanism to download the bit-string of each cell. Each cell has a transistor array (reconfigurable circuitry shown in Fig. 4), as well as a set of other programmable resources (including programmable resistors and static capacitors). The reconfigurable circuitry consists of 14 transistors connected through 44 switches and is able to implement different building blocks for analogue processing, such as two- and three-stage Operational Amplifiers, logarithmic photo detectors and Gaussian computational circuits. It includes three capacitors, Cm1, Cm2 and Cc, of 100 fF, 100 fF and 5 pF respectively.

Heidelberg Field Programmable Transistor Array (FPTA)

The FPTA consists of 16 × 16 programmable transistor cells. As CMOS transistors come in two types, namely N- and P-MOS, half of the transistor cells are designed as programmable NMOS transistors and half as programmable PMOS

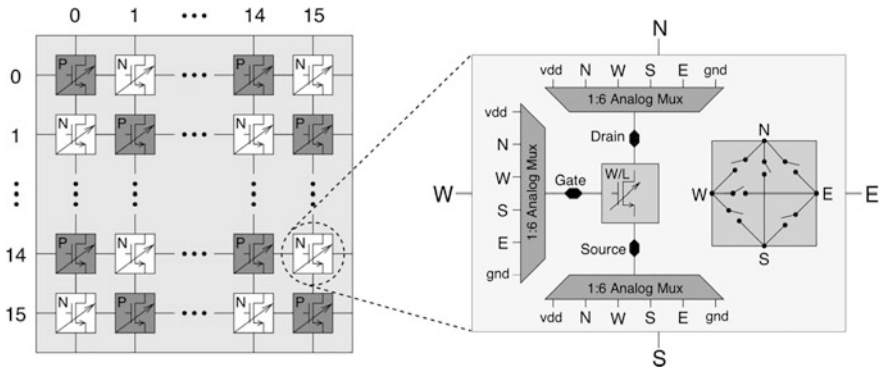

Fig. 5 Schematic diagram of the FPTA [17]

transistors. P- and N-MOS transistor cells are arranged in a checkerboard pattern, as depicted in Fig. 5. Each cell contains the programmable transistor itself, three decoders that allow the three transistor terminals to be connected to one of the four cell boundaries, Vdd or Gnd and six routing switches. Width W and Length L of the programmable transistor can be chosen to be 1, 2, … 15 μm and 0.6, 1, 2, 4, or 8 μm respectively. The three terminals—Drain, Gate and Source—of the programmable transistor can be connected to either of the four cell edges (N, S, E, W), as well as to Vdd or Gnd. The only means of routing signals through the chip is via the six routing switches that connect the four cell borders with each other. Thus, in some cases, it is not possible to use a transistor cell for routing and as a transistor.

(B) **Digital Devices**

Field Programmable Gate Arrays (FPGA)

The FPGA is a standard digital device and is organised as an array of logic blocks, as shown in Fig. 6. Devices from both Xilinx and Alter a have been applied to EH. Programming an FPGA requires programming three tasks: (1) the functions implemented in logic blocks, (2) the signal routing between logic blocks and (3) the characteristics of the input/output blocks i.e. a tri-state output or a latched input. All of these, and hence the complete functionality of the device (including interconnect), are defined by a bit pattern. Specific bits will specify the function of each logic block, other bits will define what is connected to what. To date FPGAs have been the workhorse of much of EH that has been achieved in the digital domain.

The CellMatrix MOD 88

As with analogue devices, the other path to implement evolvable hardware is to design and build your own device, tuned to the needs of Evolvable Hardware. The Cell Matrix, illustrated in Fig. 7, is a fine-grained reconfigurable device with an 8×8 array of cells, where larger arrays may be achieved by connecting multiple

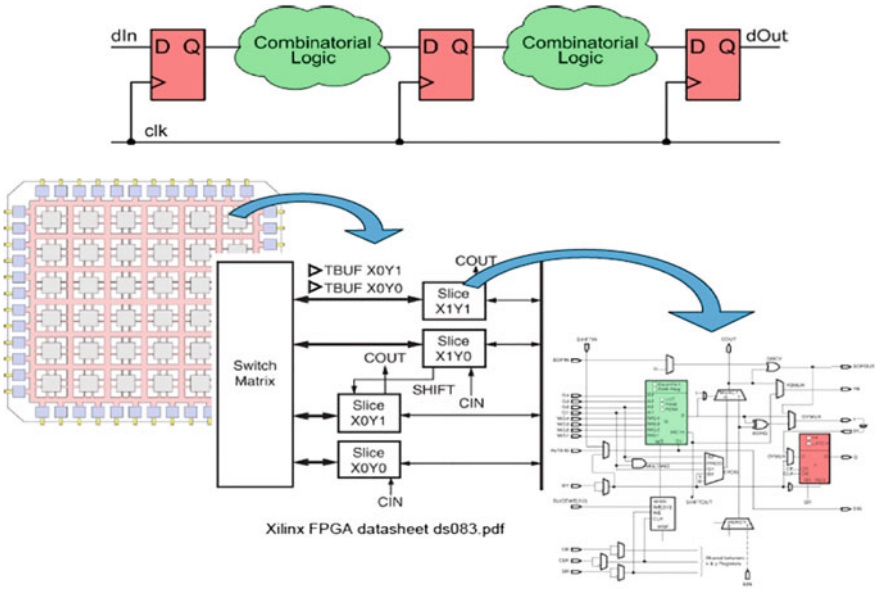

Fig. 6 Generic FPGA block diagram [18]

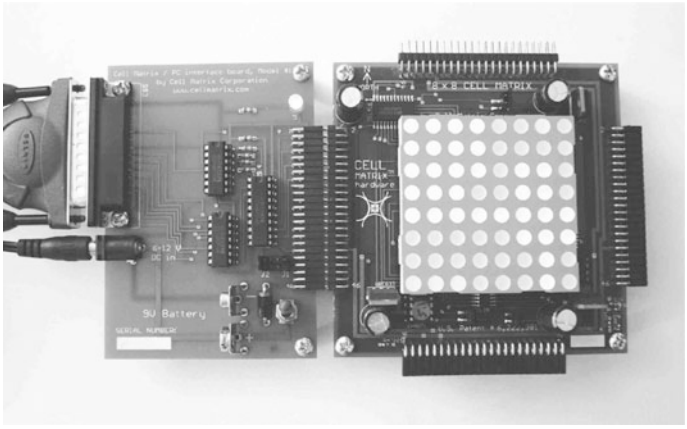

Fig. 7 The cellmatrix MOD 88, 8 × 8 Array [19]

8 × 8 arrays. Unlike most other reconfigurable devices, there is no built-in configuration mechanism. Instead, each cell is configured by its nearest neighbour, thus providing a mechanism not only for configuration but also for self-reconfiguration and the potential for dynamic configuration. Further, the Cell matrix cannot be accidently damaged by incorrect configurations, as is the case for FPGAs if the standard design rules are not followed.

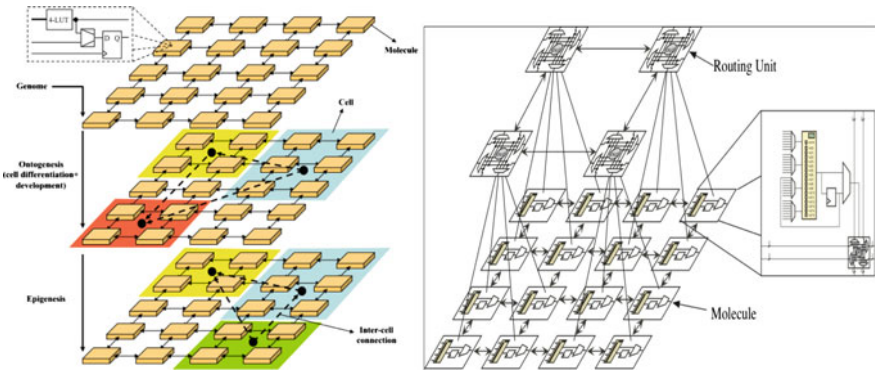

Fig. 8 Organic subsystem, schematic and organic views [21]

The POEtic Device

The goal of the "Reconfigurable POEtic Tissue" ("POEtic") [20], completed under the aegis of the European Community, was the development of a flexible computational substrate inspired by the evolutionary, developmental and learning phases in biological systems. POEtic applications are designed around molecules, which are the smallest functional blocks (similar to FPGA CLBs). Groups of molecules are put together to form larger functional blocks called cells. Cells can range from basic logic gates to complete logic circuits, such as full-adders. Finally, a number of cells can be combined to make an organism, which is the fully functional application. Figure 8 shows a schematic view of the organic subsystem.

The organic subsystem is constituted from a regular array of basic building blocks (molecules) that allow for the implementation of any logic function. This array constitutes the basic substrate of the system. On top of this molecular layer there is an additional layer that implements dynamic routing mechanisms between the cells that are constructed from combining the functionality of a number of molecules. Therefore, the physical structure of the organic subsystem can be considered as a two- layer organization (at least in the abstract), as depicted in Fig. 8.

The RISA Device

The structure of the RISA cell enables a number of different system configurations to be implemented. Each cell's FPGA fabric may be combined into a single area and used for traditional FPGA applications. Similarly, the separate microcontrollers can be used in combination for multi-processor array systems, such as systolic arrays [10]. However, the intended operation is to use the cell parts more locally, allowing the microcontroller to control its adjoining FPGA configuration. This cell structure is inspired by that of biological cells. As illustrated in Fig. 9, the microcontroller provides functionality similar to a cell nucleus. Each cell contains a microcontroller and a section of FPGA fabric. Input/output (IO) Blocks provide interfaces between FPGA sections at device boundaries. Inter-cell communication is provided by dedicated links between microcontrollers and FPGA fabrics.

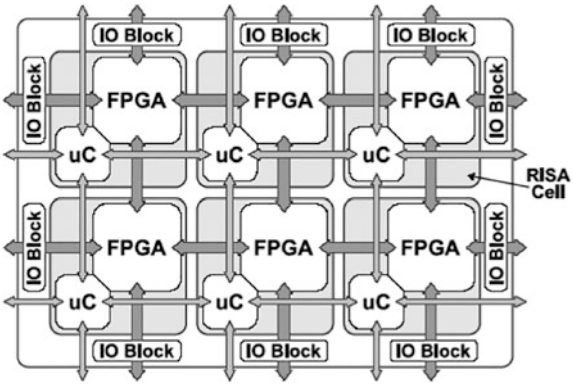

Fig. 9 The RISA architecture comprising an array of RISA Cells [22]

4 Success Stories

There have been some real successes within the community over the past 15 years. This section gives a short review of some of these.

Extrinsic Analogue

Some of the earliest work on using evolutionary algorithms to produce electronic circuits was performed by Koza and his team [23–27]. The majority of this work focuses on using Genetic Programming (GP) to evolve passive, and later active, electronic circuits for "common" functions such as filters and amplifiers. All of this work is extrinsic, that is, performed in software with the results generally presented as the final fit individual. Much of the early work considers the design of passive filters with considerable success [25]. Figure 10 illustrates such results. The interesting aspect in that particular paper (which is carried on and developed in subsequent work) is the use of input variables (termed *free variables*) and conditional statements to allow circuit descriptions to be produced that are solutions to multiple instances of a generic problem, in this case filter design. In Fig. 10 this is illustrated by specifying, with such input variables, whether a low-pass or high-pass filter is the required final solution of the GP.

A more sophisticated example of evolving electronic circuits is shown in Fig. 11 [28]. Here the requirement is to create a circuit that mimics a commercial amplifier circuit (and even improve on it). There are a number of both subtle and complex points that come out of that paper that are of interest and use to others trying to use evolvable hardware. The paper illustrates how the use of domain knowledge, both general and problem-specific, is important as the complexity of the problem increases. New techniques, or improvements in actual GP techniques, are developed to solve this problem. Finally, and potentially most important, the use of multi-objective fitness measures are shown to be critical for the success of evolution.

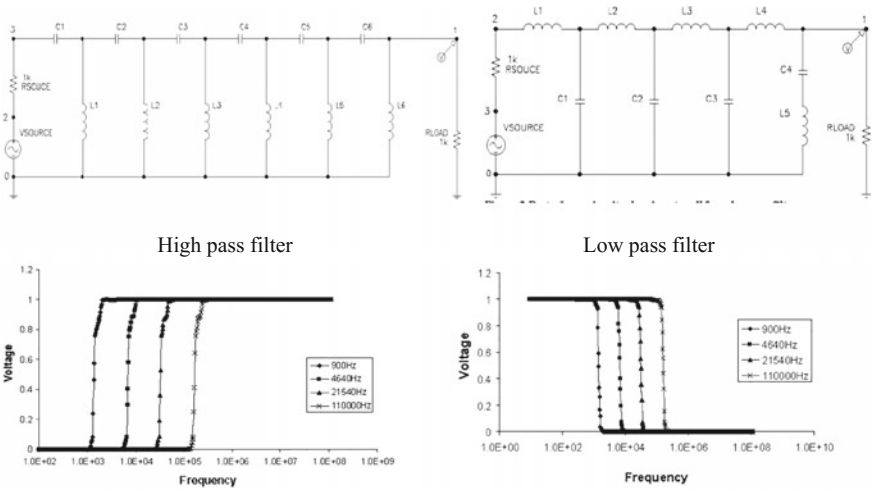

Fig. 10 Examples of filter circuits and frequency plots obtained by evolutionary processes [25]

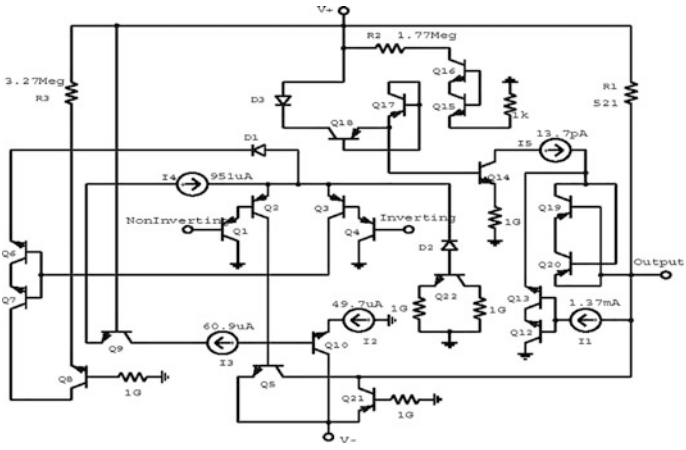

Fig. 11 Best run of an amplifier design [28]

In that case, 16 objective measures are incorporated into the evolutionary process, including: gain, supply current, offset voltage, phase margin and bias current. When evolving for real systems in real environments the use of multi-objective fitness criteria is shown to be extremely important.

Koza and his group continue to evolve electronic circuits in an extrinsic manner, and have many results where the evolutionary process has replicated results of previously patented human designs.

Intrinsic Digital

One of the very first research teams to apply evolvable hardware in an intrinsic manner was Higuchi and his group in Japan [1, 2, 29–31]. Their work includes applying evolvable hardware (almost always intrinsically) to myoelectric hands, image compression, analogue IF filters, femto-second lasers, high-speed FPGA I/O and clock-timing adjustment. Here we give a little more detail on the application of clock-timing adjustment as an illustration of the work conducted by Higuchi and his group.

As silicon technology passed the 90 nm process size new problems in the fabrication of VLSI devices started to appear. One of these is the fact that, due to process variations cause by the fabrication process, the clock signals that appear around the chip are not always synchronised with each other. This clock skew can be a major issue in large complex systems. This is a difficult issue for chip designers since it is a post-fabrication problem where there is a significant stochastic nature to the issues. One solution would, of course, be to slow the clocks down, but that is generally an unacceptable solution. Another possibility is to treat this as a post-fabrication problem, where actual differences in paths and actual speeds can be measured.

Figure 12 illustrates the basic philosophy behind the idea of using evolution to solve the problem. Typically, clock signals are transmitted around complex VLSI devices, in a hierarchical form, often known as clock-trees, as illustrated in the left-hand picture in Fig. 12. At the block level (middle picture), sequential circuits should receive clock signals at the same time for the circuit to perform correctly. The idea behind this work is to introduce programmable delay circuits (right-hand picture) that are able to fine-tune the delay on all the critical path lengths that the clock propagates. The issue is, what delay should be programmed into each individual path on each individual VLSI device? Each device is likely to be different due to the fabrication process and we will not know what this is until after fabrication. However, the delay can be controlled (programmed) by a short bit pattern

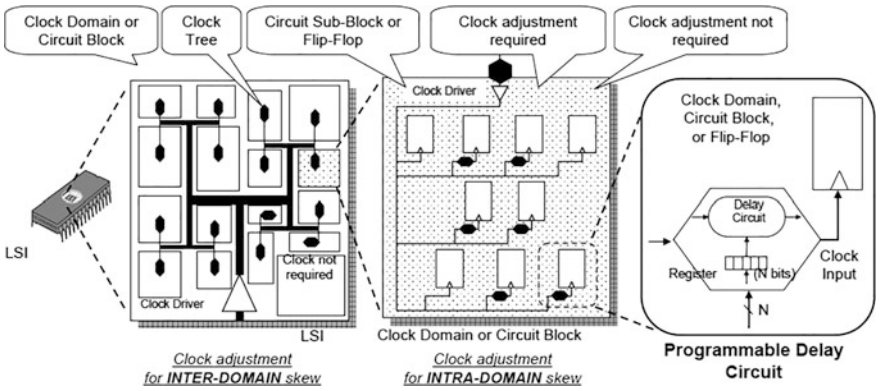

Fig. 12 Basic process of clock skew adjustment using evolvable hardware [29]

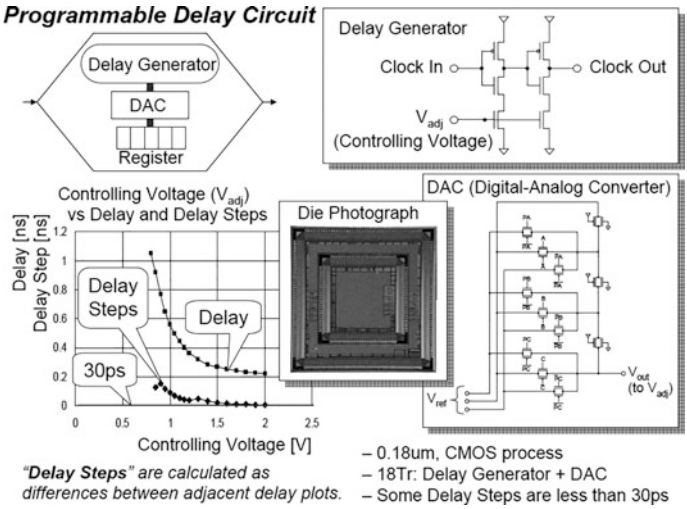

Fig. 13 Realisation of clock skew adjustment on VLSI device [29]

stored in a register, on-chip. This bit pattern can form part of a genome that is used within an evolutionary loop that is using a Genetic Algorithm (GA) to optimise the delay in each of the critical paths in the circuit. Figure 13 illustrates in more detail the actual circuit elements required, and the final chip design, to achieve the required functionality for this evolvable chip. The results suggest that not only can the clock skew be optimised, and consequently the frequency that the device can run at be increased (in the paper by 25%) but that also the supply voltage can be reduced while maintaining a functioning chip (in some cases by more than 50%).

Intrinsic Analogue

Section 3 has given a brief outline of the work performed by the team at NASA JPL on the design and manufacture of a number of analogue programmable devices aimed at assisting with intrinsic analogue evolution, Field Programmable Transistor Arrays (FPTAs) [16]. Here, one of these devices is presented to illustrate, not only the evolution of useful circuits (in real environments), but that through the continued use of evolution throughout the lifetime of the system, fault tolerance is increased. In this case, the functionality considered is that of a 4-bit digital-to-analogue converter (4-bit DAC). The basic evolvable block used in these experiments is shown in Fig. 4, and the results obtained in the work are summarised in Fig. 14.

Hardware experiments use the FPTA-2 chip, with the structure illustrated in Fig. 4. The process starts by evolving first a 2-bit DAC—a relatively simple circuit. Using this circuit as a building block, a 3-bit DAC is then evolved and, again reusing this, a 4-bit DAC is evolved. The total number of FPTA cells used is 20. Four cells map a previously evolved 3-bit DAC (evolved from a 2-bit DAC), four

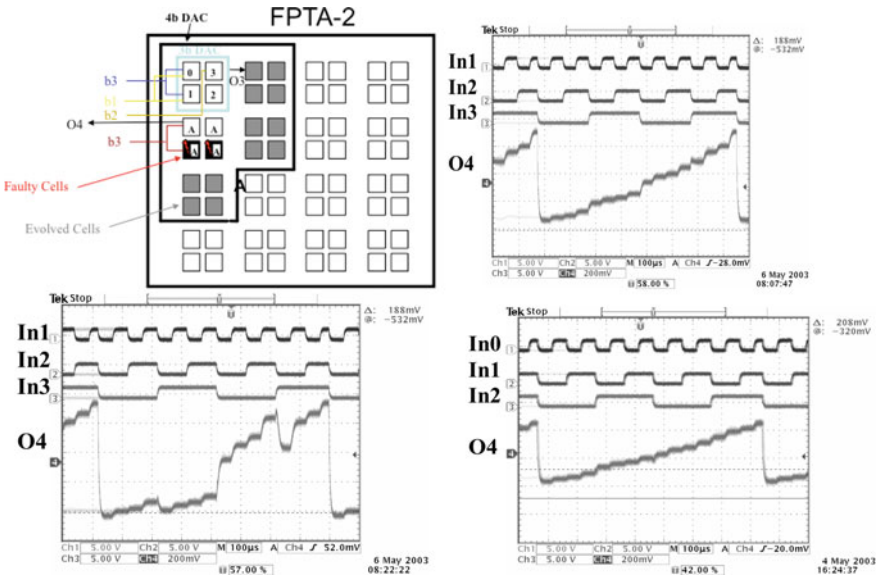

Fig. 14 FPTA topology (top-left), evolved 4-bit DAC response (top-right), evolved 4-bit DAC response with two faulty cells (bottom-left) and recovered evolved 4-bit DAC response with two faulty cells (bottom-right) [32]

cells map a human designed Operational Amplifier (buffering and amplification) and 12 cells have their switches' states controlled by evolution. When the fault is injected into the operating circuit, the system opens all the switches of the 2 cells of the evolved circuit.

Figure 14 (top-left) shows the faulty cells in black. In these cells all switches are opened (stuck-at-0 fault). The 3-bit DAC, cells '0', '1', '2' and '3' map the previously evolved 3-bit DAC, whose output is O3. The Operational Amplifier cell (Label 'A') is constrained to OpAmp functionality only. The evolved cell (in grey) switches' states are controlled by evolution. O4 is the final output of the 4-bit DAC.

The top-right plot in Fig. 14 illustrates the initial evolved 4-bit DAC with inputs 1, 2 and 3 (4 is missing due to number of oscilloscope channels) and output O4 which can be seen to be functioning correctly. The bottom left plot in Fig. 14 shows the response of the 4-bit DAC when two cells are made faulty. Finally, the plot at the bottom-right of Fig. 14 illustrates the response of the DAC after further evolution (after fault injection) has taken place. This response is achieved after only 30 generations of evolution. Again, the only cells involved in the evolutionary process are those in grey in Fig. 14. This example shows very well that recovery by evolution, in this case for a 4-bit DAC, is possible. The system makes available 12 cells for evolution. Two cells are constrained for the implementation of the OpAmp. Four cells are constrained to implement the simpler 3-bit DAC. Two faulty cells are implemented by placing all their switches to the open position.

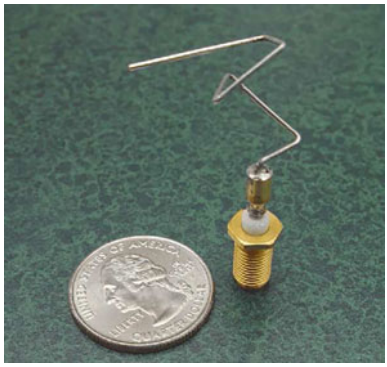

Fig. 15 Prototype evolved antenna, which on March 22, 2006 was successfully launched into space [34]

Extrinsic and Intrinsic Analogue

As a final example in this section, the domain of evolved antenna design [33] is presented. In the work presented in [34], a GA is used in conjunction with the Numerical Electromagnetic Code, Version 4 (NEC, standard code within this area) as the simulator to create and optimise wire antenna designs. These designs not only produce impressive characteristics, in many cases similar or better to those produced by traditional methods, but their structure is often very different from traditional designs (see Fig. 15). The 'crooked-wire' antennas consist of a series of wires joined at various locations and with various lengths (both these parameters determined by the GA). Such unusual shapes, unlikely to be designed using conventional design techniques, demonstrate excellent performance both in simulation and physical implementation [34].

Apart from the difference in domain, an interesting feature of this application of EH is the relatively large number of constraints (requirements) that such systems impose on a design. Requirements of a typical antenna include [34]: Transmit Frequency; Receive Frequency; Antenna RF Input; VSWR; Antenna Gain Pattern; Antenna Input Impedance, and Grounding. From this list, which is not exhaustive, it can be seen that this is certainly a non-trivial problem requiring a complex objective function to achieve appropriate results.

Given the nature of the problem and the solution (wire antenna), a novel representation and construction method is chosen. The genetic representation is a small "programming language" with 3 instructions: place wire, place support, and branch. The genotype specifies the design of one arm in a 3D-space. The genotype is a tree-structured program that builds a wire form. Commands to undertake this process are: Forward (length radius); rotate_x (angle); rotate_y (angle) and rotate_z (angle). Branching in genotype equates to branching in wire structure. The results are very successful (see Fig. 15) and the characteristics (electrical and mechanical) of the final antenna more than meet the requirements list.