An Evolvable Hardware Tutorial

Jim Torresen

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

Abstract. Evolvable Hardware (EHW) is a scheme - inspired by natural evolution, for automatic design of hardware systems. By exploring a large design search space, EHW may find solutions for a task, unsolvable, or more optimal than those found using traditional design methods. During evolution it is necessary to evaluate a large number of different circuits which is normally most efficiently undertaken in reconfigurable hardware. For digital design, FPGAs (Field Programmable Gate Arrays) are very applicable. Thus, this technology is applied in much of the work with evolvable hardware. The paper introduces EHW and outlines how it can be applied for hardware design of real-world applications. It continues by discussing the main problems and possible solutions. This includes improving the scalability of evolved systems. Promising features of EHW will be addressed as well, including run-time adaptable systems.

1 Introduction

The number of transistors becoming available for designers continue to increase as Moores law seems to be valid for the development of new computer hardware. Earlier we have seen a limit in the *size* of hardware devices. However, we may very well soon see a limit in *designability*. That is, designers are not able to apply all the transistors in the largest integrated circuits becoming available. To overcome this problem, new and more automatic design schemes would have to be invented. One such method is evolvable hardware (EHW).
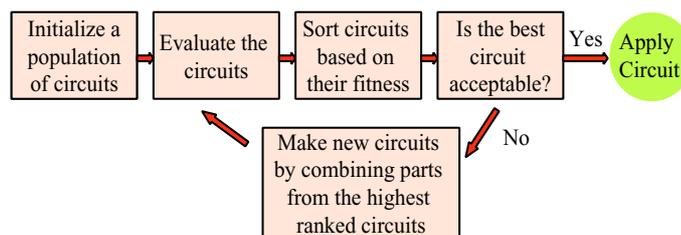

Fig. 1. The algorithm for evolving circuits.

It was introduced for about ten years ago as a new way of designing electronic circuits [4]. Instead of manually designing a circuit, only input/output-relations are specified. The

circuit is automatically designed using an adaptive algorithm inspired from natural evolution. The algorithm is illustrated in Fig. 1. In this algorithm, a set (population) of circuits – i.e. circuit representations, are first randomly generated. The behavior of each circuit is evaluated and the best circuits are combined to generate new and hopefully better circuits. Thus, the design is based on incremental improvement of a population of initially randomly generated circuits. Circuits among the best ones have the highest probability of being combined to generate new and possibly better circuits. The evaluation is according to the behavior initially specified by the user. After a number of iterations, the fittest circuit is to behave according to the initial specification. The most commonly used evolutionary algorithm is genetic algorithm (GA) [3]. The algorithm – which follows the steps described above, contains important operators like crossover and mutation of the circuit representations for making new circuits. The operations are very similar to those found in natural evolution as seen in Fig. 2.
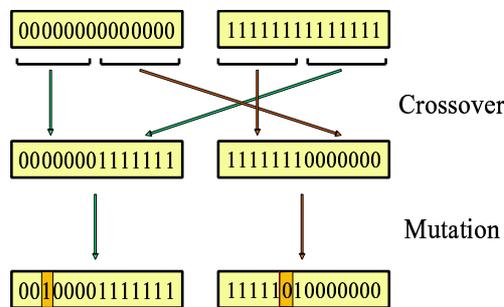

Fig. 2. The genetic algorithm operators.

Each individual – representing a circuit description, in the population is often named *chromosome* or genotype and is represented by an array of bits. Each bit in the array is often called a *gene*. Thus, each chromosome contains a representation of a circuit with a set of components and their interconnections. In crossover, the parameters of the pairwise selected circuits are exchanged to generate – for each couple, two new offspring – preferably fitter than the parents. As an alternative to crossover operation, there is usually some probability for conducting cloning instead. Then, the two offspring circuits are equal to the two parent circuits. Further, the *best* circuit may as well be directly copied into the next generation (called elitism). Mutations may also occur and involves randomly inverting a few genes in the chromosome. This make the chromosomes slightly different from what could be obtained by only *combining* parent chromosomes.

When the number of offspring circuits equals the number of circuits in the parent population, the new offspring population is ready to become the new parent population. The original parent population is deleted. Thus, one loop in Fig. 1 is named one *generation*. Randomness is introduced in the selection of parents to be mated. Not only the fittest circuits are selected. However, the probability of a circuit being selected for breeding decreases with decreasing fitness score.

A circuit can be represented in several different ways. For digital circuits however, gate level representation is most commonly used. That is, the representation contains a
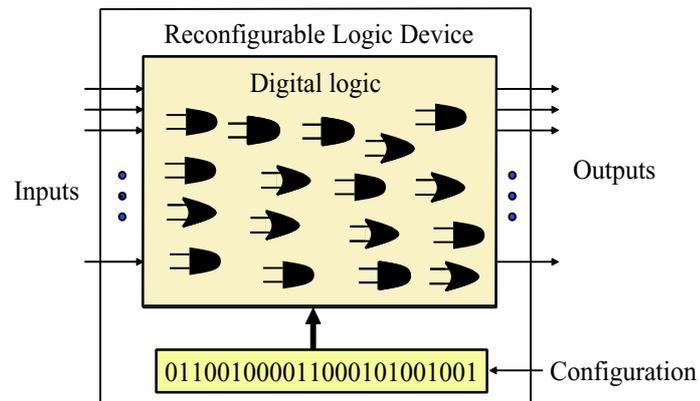
Fig. 3. Illustration of an Field Programmable Logic Device (FPLD).

description of what kind of gates are applied and their interconnections. This is coded into a binary configuration bitstream applied to configure a reconfigurable logic device as seen in Fig. 3. This is usually either a commercial device like an Field Programmable Gate Array (FPGA) or a part of an Application Specific Integrated Circuit (ASIC). Each new circuit would have to be evaluated for each generation. This can be undertaken more efficiently by measuring the performance on a real reconfigurable device compared to using simulation. Thus, reconfigurable technology is an important technology for the development of systems based on evolvable hardware.

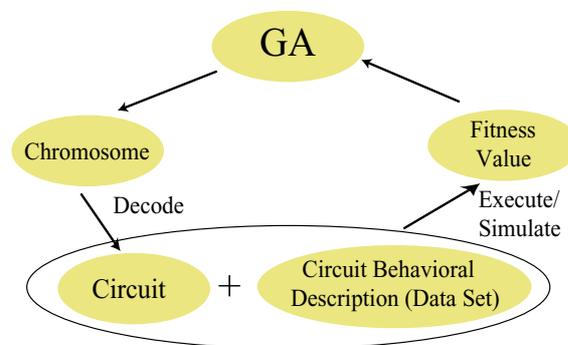**Fig. 4.** The cycle of evolving a circuit.

In addition to the evolutionary algorithm (GA), a circuit *specification* would have to be available. This is often a set of training vectors (input/output mappings) assembled into a *data set*. The operation of GA together with the data set are given in Fig. 4. The most computational demanding part of GA is usually the evaluation of each circuit – typically

named fitness computation. This involves inputing data to each circuit and computing the error given by the deviation from the specified correct output.

There are a number of aspects to consider when evolving hardware. Many roads lead to evolved systems. This paper will give an overview of the large variety of schemes, parameter settings and architectures available. However, evolving systems have a number of limitations. One of the main problems with evolution seems to be the scalability of the systems. As the complexity of a system increases, the chromosome string length must be increased. However, a larger number of generations is required as the string length increases. This often makes the search space too large to be able to find a well performing system. Thus, this paper will go into this important problem and present how it is tried to be solved.

On the other hand, there are several new features provided with evolution. Since the systems are automatically designed, we are able to have the evolution going on at run-time in parallel with normal operation. This could be applied to modify the system if the environment changes or errors occur in the system. Work on such run-time adaptable systems will be described. Evolvable hardware has been applied to a number of real-world applications showing the applicability of the promising approach of evolving hardware systems.

The next section contains a classification of EHW research based on a given classification framework [32]. This is followed by a discussion of the main problems in Section 3. Section 4 includes a presentation of the online adaptivity provided with EHW. Conclusions are given in Section 5.

2 A Framework for Classifying EHW

EHW research is rapidly diverging. Thus, to understand the EHW field of research, a classification framework would be beneficial. This is presented below. The many degrees of freedom in EHW could be represented in a multi-dimensional space. However, here a list format is prefered.

Evolutionary Algorithm (EA). A set of major algorithms exists:
- **Genetic Algorithm (GA)**
- **Genetic Programming (GP)**
- **Evolutionary Programming (EP)**

The major difference between GA and GP is the chromosome representation. GA organizes the genes in an array, while GP applies a tree of genes. Both schemes apply both crossover and mutation, while EP – which has no contraints on the representation, uses mutation only.

Technology (TE). Technology for the target EHW:
- **Digital**
- **Analog**

Application	EA	TE	AR	BB	THW	FC	EV	SC
Adaptive Equalizer [18]	GA	D	CD	Neuron	Custom	ONL	On-chip	S
Ampl. and Filter Design [15]	GA	A	CD	T/R/L/C	Custom	OFL	Off-chip	S
Analog Circuit Synthesis [12]	GP	A	CD	R/L/C	Custom	OFL	Off-chip	S
Character Recognition [31]	GA	D	CD	Gate	Comm.	OFL	Off-chip	S
Clock Adjustment [27]	GA	D	CT	Gate	Custom	ONL	Off-chip	S
Digital Filter Design [16]	GA	D	CD	Gate	–	OFL	Off-chip	S
Gene Finding [39]	GA	D	CD	Gate	Comm.	OFL	Off-chip	S
IF Filter Tuning [17]	GA	A	CT	Filter	Custom	ONL	Off-chip	S
Image Compression [22]	GA	D	CT	Pixel	Custom	ONL	On-chip	D
Image Compression [23]	GA	D	CD	Gate	Comm	ONL	On-chip	D
Multi-spect. Image Rec. [20]	GA	D	CT	Function	Comm.	OFL	Off-chip	S
Number Recognition [7]	GA	D	CD	Gate	Comm.	OFL	Off-chip	S
Prosthetic Hand [9]	GA	D	CD	Gate	Custom	ONL	Complete	S
Road Image Rec. [33]	GA	D	CD	Gate	Comm.	OFL	Off-chip	S
Robot Control [10]	GA	D	CD	Gate	Comm.	ONL	Complete	D
Robot Control [28]	GA	D	CD	Gate	Comm.	ONL	Off-chip	S
Sonar Classification [38]	GA	D	CD	Gate	Comm.	OFL	Off-chip	S

Table 1. Characteristics of EHW applied to real-world applications.

Architecture (AR). The architecture applied in evolution:
– **Complete Circuit Design (CD)** Complete circuit evolution where building block functions (see below) and their interconnections are evolved.
– **Circuit Parameter Tuning (CT)** The architecture is designed in advance and only a set of configurable parameters are evolved.

Building Block (BB). The evolution of a hardware circuit is based on connecting basic units together. Several levels of complexity in these building blocks are possible:
– **Analog comp. level.** E.g. transistors, resistors, inductors and capacitors.
– **Gate level** E.g. OR and AND gates.
– **Function Level** E.g. sine generators, adders and multipliers.

Target Hardware (THW). In EHW, the goal is to evolve a circuit. The two major alternatives for target hardware available today are:
– **Commercially available devices.** FPGAs are most commonly used. Field-Programmable Analog Arrays (FPAA) are available as well. They use the same programming principle as FPGAs, but they consist of reconfigurable analog components instead of digital gates.
– **Custom hardware.** ASIC (Application Specific Integrated Circuit) is a chip fully designed by the user.

Fitness Computation (FC). Degree of fitness computation in hardware:
– **Offline Fitness Computation (OFL).** The evolution is simulated in software, and only the elite chromosome is written to the hardware device (sometimes named extrinsic evolution).
– **Online Fitness Computation (ONL).** The hardware device gets configured for each chromosome for each generation (sometimes named intrinsic evolution).

Evolution (EV). Degree of evolution undertaken in hardware:
- **Off-chip evolution.** The evolutionary algorithm is performed on a separate processor.
- **On-chip evolution.** The evolutionary algorithm is performed on a separate processor incorporated into the chip containing the target EHW.
- **Complete HW evolution.** The evolutionary algorithm is implemented in special hardware – i.e. it is not running on a processor.

Scope (SC). The scope of evolution:
- **Static evolution.** The evolution is finished before the circuit is put into normal operation. No evolution is applied during normal operation. The evolution is used as a circuit optimizing tool.
- **Dynamic evolution.** Evolution is undertaken while the circuit is in operation and this makes the circuit online adaptable.

Table 1 summarizes the characteristics of the published work on EHW applied to real-world applications. The applications are mainly in the areas of classification and control when complete circuit design is applied. There are also some examples of circuit parameter tuning. A major part of them are based on digital gate level technology using GA as the evolutionary algorithm. However, promising results are given for analog designs, where evolution is used to find optimal parameters for analog components. About half of the experiments are based on custom hardware – or simulation of such. It is more common to put only the fitness computation (ONL), than the whole evolution (On-chip/Complete), on the *same* chip as the target EHW. This is reasonable, since the fitness computation is – as mentioned earlier, the most computational demanding part of the evolution.

Even though there are promising results in evolving systems, there are several obstackles as well that will be discussed in the next section.

3   Evolvable Hardware Scalability

There are several problems concerning evolution and scalability. In this section, two of the major ones will be discussed.

3.1   Chromosome String Length

As mentioned in the introduction, there has been a lack of schemes to overcome the limitation in the chromosome string length [14, 37]. A long string is required for representing a complex system. However, a larger number of generations are required by genetic algorithms as the string length increases. This often makes the search space *too* large and explains why only small circuits have been evolvable so far. Thus, work has been undertaken trying to diminish this limitation. Various experiments on *speeding up* the GA computation have been undertaken [1]. The schemes involve fitness computation in parallel or a partitioned population evolved in parallel. Experiments are focussed on speeding up the GA computation, rather than dividing the application into subtasks. This approach assumes that GA finds a solution if it is allowed to compute enough generations. When

small applications require weeks of evolution time, there would probably be strict limitations on the systems evolvable even by parallel GA.

Other approaches to the problem have used variable length chromosomes [7]. Another option, called function level evolution, is to apply building blocks more complex than digital gates [19]. Most work is based on fixed functions. However, there has been work in Genetic Programming for *evolving* the functions — called Automatically Defined Functions (ADF) [11].

An improvement of artificial evolution — called co-evolution, has been proposed [6]. In co-evolution, a part of the data which defines the problem co-evolves simultaneously with a population of individuals solving the problem. This could lead to a solution with a better generalization than a solution based only on the initial data. A variant of co-evolution — called cooperative co-evolutionary algorithms, has been proposed by De Jong and Potter [8, 21]. It consists of parallel evolution of sub-structures which interact to perform more complex higher level structures. Complete solutions are obtained by assembling representatives from each group of sub-structures together. In that way, the fitness measure can be computed for the top level system. However, by testing a number of individuals from each sub-structure population, the fitness of individuals in a sub-population can be sorted according to their performance in the top-level system. Thus, no explicit local fitness measure for the sub-populations is applied in this approach. However, a mechanism is provided for initially seeding each GA population with user-supplied rules. Darwen and Yao have proposed a co-evolution scheme where the subpopulations are divided without human intervention [2].

Incremental evolution for EHW was first introduced in [30]. The approach is a divide-and-conquer on the evolution of the EHW system, and thus, named *increased complexity evolution*. It consists of a division of the *problem* domain together with incremental evolution of the hardware system. Evolution is first undertaken individually on a set of basic units. The evolved units are the building blocks used in further evolution of a larger and more complex system. The benefits of applying this scheme is both a *simpler* and *smaller* search space compared to conducting evolution in one single run [35].

When considering *manual* hardware design, much *a priori* knowledge is applied [29]. E.g. designing a large circuit would be almost impossible if the designer had to design each sub-circuit from scratch every time it is used [13]. Further, connections are with a mixture of buses and bit-wires. Thus, in the future it is expected to see *intelligent* (or controlled) evolution as an alternative to the normal unconstrained evolution.

3.2 Fitness Computation

To be able to conduct online fitness computation, fast switching between different configurations in a population should be possible. With the available FPGA technology, only a single configuration can be stored at a time within the device. Reloading a new configuration takes too much time to be of interest for evolution. Thus, there has been work on implementing a *user defined FPGA* inside an ordinary FPGA [25, 30]. By providing a set of different configuration registers, it is possible to store a population of configurations within an FPGA and perform configuration switching in a single clock cycle [34].

4 Online Adaptation in Real-Time.

Most living creatures are able to learn to live in an environment and adapt if some change occur. Artificial systems are far from such an adaptivity. Research on evolutionary methods has – with a few exceptions, been based on one-time evolution. However, there is a wish of being able to design online adaptable evolvable hardware. The hardware would then have to be able to reconfigure its configuration dynamically and autonomously, when operating in its environment [5]. This would be by dynamic evolution.

One possible approach to such a system is to use *two* parallel units. A primary unit is applied for normal runtime operation of an application, while a secondary unit keeps evolving in parallel. If the performance of the secondary unit becomes better than the primary unit, they are exchanged. Thus, the unit giving the best performance at the moment is enabled.

A system for autonomous evolution of robot control has been proposed [10]. A mobile robot learns to track a red ball without colliding with obstacles through dynamic evolution. Thus, all information about the environment is collected concurrently with evolution. This is undertaken through building a model of the environment. Another application with dynamic evolution is image compression [22]. During compression of a given image, evolution search for an optimal set of templates for doing pixel prediction.

So far there is not much work on applying dynamic evolution. This is mainly due to the problem of evolution speed and fitness computation. A set of individuals in the population would have to be evaluated – one at a time, for each generation which is normally time consuming. To make fitness computation you need a feedback from the environment on the performance. This could be difficult to obtain. However, if these problems can be overcome, there is a large potential in being able to provide online-adaptable systems.

5 Conclusions

This paper has introduced EHW and contained a study of the characteristics of EHW applied to real-world applications. Further, problems and new features related to evolution are discussed. For more information about evolvable hardware, there are a couple of special journal issues [36, 26] and some books [40, 24]. The two main conferences in the field are International Conference on Evolvable Systems: From Biology to Hardware (Springer LNCS publisher) and NASA/DoD Conference on Evolvable Hardware (IEEE publisher).

References

1. E. Cantu-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171, 1998.
2. P. Darwen and X. Yao. Automatic modularization by speciation. In *Proc. of 1996 IEEE International Conference on Evolutionary Computation*, pages 88–93, 1996.
3. D. Goldberg. *Genetic Algorithms in search, optimization, and machine learning*. Addison–Wesley, 1989.
4. T. Higuchi et al. Evolvable hardware: A first step towards building a Darwin machine. In *Proc. of the 2nd International Conference on Simulated Behaviour*, pages 417–424. MIT Press, 1993.

5. T. Higuchi, M. Iwata, I. Kajitani, H. Iba, Y. Hirao, B. Manderick, and T. Furuya. Evolvable hardware and its applications to pattern recognition and fault-tolerant systems. In E. Sanchez and M. Tomassini, editors, *Towards Evolvable Hardware: The evolutionary Engineering Approach*, volume 1062 of *Lecture Notes in Computer Science*, pages 118–135. Springer-Verlag, 1996.

6. W.D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42(1-3):228–234, 1990.

7. M. Iwata, I. Kajitani, H. Yamada, H. Iba, and T. Higuchi. A pattern recognition system using evolvable hardware. In *Proc. of Parallel Problem Solving from Nature IV (PPSN IV)*, volume 1141 of *Lecture Notes in Computer Science*, pages 761–770. Springer-Verlag, September 1996.

8. K.A. De Jong and M.A. Potter. Evolving complex structures via co-operative coevolution. In *Proc. of Fourth Annual Conference on Evolutionary Programming*, pages 307–317. MIT Press, 1995.

9. 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.

10. D. Keymeulen et al. On-line model-based learning using evolvable hardware for a robotics tracking systems. In *Genetic Programming 1998: Proc. of the Third Annual Conference*, pages 816–823. Morgan Kaufmann, 1998.

11. J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. The MIT Press, 1994.

12. J. R. Koza et al. *Genetic Programming III*. San Francisco, CA: Morgan Kaufmann Publishers, 1999.

13. J.R. Koza et al. The importance of reuse and development in evolvable hardware. In J. Lohn et al., editor, *Proc. of the 2003 NASA/DoD Conference on Evolvable Hardware*, pages 33–42. IEEE, 2003.

14. W-P. Lee, J. Hallam, and H.H. Lund. Learning complex robot behaviours by evolutionary computing with task decomposition. In A. Birk and J. Demiris, editors, *Learning Robots: Proc. of 6th European Workshop, EWLR-6 Brighton*, volume 1545 of *Lecture Notes in Artificial Intelligence*, pages 155–172. Springer-Verlag, 1997.

15. J.D. Lohn and S.P. Colombano. A circuit representation technique for automated circuit design. *IEEE Trans. on Evolutionary Computation*, 3(3):205–219, September 1999.

16. J. F. Miller. Digital filter design at gate-level using evolutionary algorithms. In W. Banzhaf et al., editors, *Proc. of the Genetic and Evolutionary Computation Conference (GECCO'99)*, pages 1127–1134. Morgan Kaufmann, 1999.

17. M. Murakawa et al. Analogue EHW chip for intermediate frequency filters. In M. Sipper et al., editors, *Evolvable Systems: From Biology to Hardware. Second International Conference, ICES 98*, pages 134–143. Springer-Verlag, 1998. Lecture Notes in Computer Science, vol. 1478.

18. M. Murakawa et al. The grd chip: Genetic reconfiguration of dsps for neural network processing. *IEEE Transactions on Computers*, 48(6):628–638, June 1999.

19. M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, and T. Higuchi. Hardware evolution at function level. In *Proc. of Parallel Problem Solving from Nature IV (PPSN IV)*, volume 1141 of *Lecture Notes in Computer Science*, pages 62–71. Springer-Verlag, September 1996.

20. R. Porter et al. An applications approach to evolvable hardware. In *Proc. of the First NASA/DoD Workshop on Evolvable Hardware*, 1999.

21. M.A. Potter and K.A. De Jong. Evolving neural networks with collaborative species. In *Proc. of Summer Computer Simulation Conference*. The Society for Computer Simulation, 1995.

22. Sakanashi et al. Evolvable hardware chip for high precision printer image compression. In *Proc. of 15th National Conference on Artificial Intelligence (AAAI-98)*, 1998.

23. L. Sekanina. Toward uniform approach to design of evolvable hardware based systems. In R.W. Hartenstein et al., editors, *Field-Programmable Logic and Applications: 10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, pages 814–817. Springer-Verlag, 2000. Lecture Notes in Computer Science, vol. 1896.

24. L. Sekanina. *Evolvable Components: From theory to Hardware Implementations*. Springer-Verlag, 2004. ISBN 3-540-40377-9.

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

26. M. Sipper and D. Mange. Special issue on from biology to hardware and back. *IEEE Trans. on Evolutionary Computation*, 3(3):165–250, September 1999.

27. E. Takahashi et al. An evolvable-hardware-based clock timing architecture towards gigahz digital systems. In *Proc. of the Genetic and Evolutionary Computation Conference*, 1999.

28. A. Thompson. Exploration in design space: Unconventional electronics design through artificial evolution. *IEEE Trans. on Evolutionary Computation*, 3(3):171–177, September 1999.

29. J. Torresen. Exploring knowledge schemes for efficient evolution of hardware. In *Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware*.

30. J. Torresen. A divide-and-conquer approach to evolvable hardware. In M. Sipper et al., editors, *Evolvable Systems: From Biology to Hardware. Second International Conference, ICES 98*, volume 1478 of *Lecture Notes in Computer Science*, pages 57–65. Springer-Verlag, 1998.

31. J. Torresen. Increased complexity evolution applied to evolvable hardware. In Dagli et al., editors, *Smart Engineering System Design: Neural Networks, Fuzzy Logic , Evolutionary Programming, Data Mining, and Complex Systems, Proc. of ANNIE'99*, pages 429–436. ASME Press, November 1999.

32. J. Torresen. Possibilities and limitations of applying evolvable hardware to real-world application. In R.W. Hartenstein et al., editors, *Field-Programmable Logic and Applications: 10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, volume 1896 of *Lecture Notes in Computer Science*, pages 230–239. Springer-Verlag, 2000.

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

34. J. Torresen and K.A. Vinger. High performance computing by context switching reconfigurable logic. In *Proc. of the 16th European Simulation Multiconference (ESM2002)*, pages 207–210. SCS Europe, June 2002.

35. Jim Torresen. A scalable approach to evolvable hardware. *Journal of Genetic Programming and Evolvable Machines*, 3(3):259–282, 2002.

36. X. Yao. Following the path of evolvable hardware. *Communications of the ACM*, 42(4):47–79, 1999.

37. X. Yao and T. Higuchi. Promises and challenges of evolvable hardware. In T. Higuchi et al., editors, *Evolvable Systems: From Biology to Hardware. First International Conference, ICES 96*, volume 1259 of *Lecture Notes in Computer Science*, pages 55–78. Springer-Verlag, 1997.

38. M. Yasunaga et al. Evolvable sonar spectrum discrimination chip designed by genetic algorithm. In *Proc. of 1999 IEEE Systems, Man, and Cybernetics Conference (SMC'99)*, 1999.

39. M. Yasunaga et al. Gene finding using evolvable reasoning hardware. 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.

40. R. Zebulum et al. *Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*. CRC Press, 2001. ISBN 0849308658.