

# Evolution of Self-Diagnosing Hardware

Miguel Garvie and Adrian Thompson

Centre for Computational Neuroscience and Robotics, School of Cognitive and  
Computing Sciences, University of Sussex, Brighton BN1 9QH, UK.  
mmg20, adrianth@cogs.susx.ac.uk

**Abstract.** The evolution of digital circuits performing built-in self-test behaviour is attempted in simulation for a one bit adder and a two bit multiplier. Promising results show evolved designs can perform a better diagnosis using less resources than hand-designed equivalents. Future extensions of the approach could allow the self-diagnosis of analog circuits under failure and abnormal operating conditions.

## Motivation

Self-diagnosis is important especially in mission critical systems exposed to radiation. Built-in self-test (BIST) is widely used yet commonly requires more than 100% overhead as in voting based systems [3, 18, 15] or off-line testing [24, 33].

Evolutionary methods [10, 7, 13] applied to hardware have produced circuits comparable to those designed by experts [21, 31, 19, 14] and also unconventional circuits [27, 16] in which hardware resources are used extremely efficiently. Moreover, many evolved systems in nature exhibit self-diagnostics such as the lymphatic system [2].

All this leads to the possibility that evolutionary methods could explore areas of design space which reuse hardware components so that they contribute both to the circuit's main functionality and its BIST, leading to a low overhead on-line solution.

## 1 Introduction

Traditional approaches to BIST use the Test Pattern Generation (TPG) - Design Under Test (DUT) - Test Response Evaluation (TRE) model, with the first and last being implemented using Linear Feedback Shift Registers (LFSR) [6]. Variations exist such as hierarchic, test-per-scan, BILBO (built-in logic block observer), PRPG-MISR (pseudo random pattern generator - multi input signature register), circular BIST, and Reconfigurable Matrix Based Built-In Test Processor (RMBITP) [24, 33, 8]. Even though techniques such as RMBITP are successful at providing BIST for designs as large as 5 million gates with only around 11% overhead, they all suffer from two main disadvantages. The first is that they require the circuit's operation to go off-line periodically to feed in the test patterns. The second is a bootstrapping problem: if the testing logic fails we will never know if the rest of the circuit is functioning properly.

Voting systems solve both these problems because on one hand faults are detected immediately during on-line operation and on the other hand the testing logic is usually small – all at the expense of complete redundant copies of the main circuit. Redundancy with on-line checking can be at the level of cells in a VLSI array [18, 15] or at the level of circuit modules, which could even be diverse designs of the same module, failing in different ways [3]. These, and other techniques for on-line diagnosis [22, 5, 4], share the problem that the benefits of self-checking must outweigh the higher fault rate due to increased silicon area.

There are examples in the evolutionary electronics literature of designs that operate in surprising or intricate ways [9, 20, 23, 14, 21, 25, 17]. This paper makes a first attempt to apply this creativity to the design of circuits having BIST. To establish a proof of principle two simple digital design problems were chosen, which may illuminate the following issues:

1. Can evolutionary search reach solutions to the BIST problem?
2. Do these solutions reuse logic for the main task and BIST?
3. Are these solutions competitive with conventionally designed ones?
4. Are there any principles of operation we could extract from them, perhaps to add to our own conventional design toolset?
5. Does this method scale up for larger problems?

Section 2 will describe the GA, the simulator, the fault model, the tasks to be evolved and the fitness evaluation mechanism for BIST. Section 3 presents the results achieved while section 4 discusses what was learned and future avenues.

## 2 Method

### 2.1 The Genetic Algorithm

A generational Genetic Algorithm (GA) is used with a population size of 32 with 2 elites where 60% of the next generation is created through mutation and the rest by single-point crossover. Following from earlier work [27] we adopted the model of a small genetically semi-converged population evolving for many generations. Fitness ranges from 0 (worst) to 1 (best). An adaptive mutation rate is used, analogous to a *Simulated Annealing* strategy moving from “exploring” to “exploiting”. For each individual exactly  $m$  mutations are made, where  $m = \lfloor k_r \times \ln(1/\bar{f}) \rfloor + m_{\text{floor}}$ ,  $k_r = \frac{m_{\text{roof}}}{\ln(1/f_{\text{min}})}$ ,  $\bar{f}$  is the current average fitness,  $f_{\text{min}}$  is minimum possible fitness above 0,  $m_{\text{floor}} = 1$  is the number of mutations applied as  $\bar{f}$  reaches 1, and  $m_{\text{roof}} = 10$  is the number of mutations that would be applied if  $\bar{f} = f_{\text{min}}$ . This assumes that  $\bar{f} > 0$  which is safe under the settings used in this paper. Informal preliminary experiments indicated solutions were found in less generations than with a constant mutation rate. Linear rank selection is used, such that the elite has twice the selective advantage of the median of the population.

The genotype-phenotype mapping used is similar to [21] excepting that: the locations of circuit outputs are fixed, there are no limitations on connectivity

allowing sequential circuits, and the genotype is encoded in binary. The genotype length depends on the task being evolved. To allow implementation across a network of processors, an island based model was used [26] with a low migration rate. This population structure may have aided the search, but the details are not thought to be crucial to the results.

## 2.2 The Simulator

The simulator used is a simple version of an event driven digital logic simulator in which each logic unit is in charge of its own behaviour when given discrete time-slices and the state of its inputs. Logic units are Look-Up Tables (LUT) of two inputs capable of representing any two input logic gate. Any unit can be connected to any other allowing sequential circuits, so care must be taken to update all units “simultaneously”. This is achieved by sending the time-slices to the logic units in two waves: the first to read their inputs and the second to update their outputs. During each evaluation, circuit inputs were kept stable for 20 time-slices and the outputs were read in the second half of this cycle allowing them time to settle.

Gate delays are simulated in time-slice units and are randomized with a Gaussian distribution ( $\mu = 1.5, \sigma^2 = 0.5$ ). This amounts to a noisy fitness evaluation with the intention to facilitate transfer of sequential circuits to real hardware as in a “Minimal Simulation” [11]. At the start of each generation,  $e$  different sets of logic delays are generated, simulating  $e$  different variations to the circuit delays from manufacturing or environmental variation. Each individual’s performance is then evaluated in each of the  $e$  conditions, and its fitness is the mean value. The number  $e$  is occasionally adjusted by hand during the run, such that more evaluations are used as the population leaves the “exploring” stage and enters the “exploiting” stage. It was set such that twice the standard error of the series of fitness trials is significantly smaller than the difference in fitness between adjacent individuals in the rank table with non-equal fitnesses.

The Single-Stuck-At (SSA) Fault model was chosen because it simulates the most common type of on-line failure – that produced by radiation hazards in the absence of mishandling or manufacturing defects [24, 1]. It is also an industry standard and it has been shown that tests generated for SSA faults are also good at detecting other types of faults. SSA faults can be introduced at any of the logic units of the simulator simply by setting its output always to 0 or 1.

## 2.3 The Tasks

Two simple problems were chosen: a one bit adder with carry and a two bit multiplier. The adder was chosen as a small combinational circuit complex enough to be a good starting point for attempting to evolve BIST. It has three inputs  $A, B, C_{in}$  and two outputs  $S, C_{out}$  such that  $S = A \oplus B \oplus C_{in}$  and  $C_{out} = A \cdot B + A \cdot C_{in} + B \cdot C_{in}$ . The multiplier, chosen as a step up from the adder, has four inputs  $A_1 A_0 B_1 B_0$  and four outputs  $P_3 P_2 P_1 P_0$  where  $P = A \times B$ .

The evolving networks may be recurrent and could show an unwanted dependence on the order in which inputs are presented, and on the networks' internal state. To demand insensitivity to input ordering, the same approach was taken as for the randomization of logic delays (above): at the start of each generation,  $e$  (the same number  $e$  defining the number of evaluations with random gate delays above) different orderings of the full set of possible inputs for that task were generated, and the individuals of that generation evaluated on all of them. On each of the  $e$  evaluations the circuit state was reset, then the ordering of the full set of inputs was presented twice in sequence, to prevent dependence on initial conditions.

The task evaluation score was measured as follows. Let  $Q_r$  be the concatenation of the series of values at the  $r^{\text{th}}$  output bit for the final 10 time-slices of the presentation of each input vector during an evaluation, and  $Q'_r$  the desired response. We take the modulus of the correlation of  $Q_r$  and  $Q'_r$ , averaged over all  $N$  outputs:

$$f_t = \frac{\sum_{r=0}^{N-1} |corr(Q_r, Q'_r)|}{N} \quad (1)$$

## 2.4 Evolving BIST

An extra output  $E$  was recorded from circuits with the aim that it would go high whenever a fault affected any other output. The performance of a circuit at its main task  $f_t$  and at BIST behaviour  $f_b$  were evaluated separately. BIST behaviour itself was evaluated with two fitness measures:

1. BIST per fault  $f_{b_F}$ : Let  $u_f$  be the number of faults affecting task performance for which none of the possible input vectors raises  $E$ . Then  $f_{b_F}$  encourages faults to be "detectable":  $f_{b_F} = 1 / (1 + u_f \times k_f)$  where  $k_f$  was chosen to be 25, to give  $f_{b_F}$  good sensitivity when  $u_f$  is small.
2. BIST per instance  $f_{b_i}$ : Let  $u_i$  be the number of instances out of all possible combinations of SSA faults and input vectors, for which the task output is incorrect but  $E$  is low. Then  $f_{b_i}$  encourages immediate detection of faults:  $f_{b_i} = 1 / (1 + u_i \times k_i)$  where  $k_i$  was chosen to be 200.

Notice that having a high  $f_{b_F}$  but low  $f_{b_i}$  is similar to off-line BIST solutions while having a high  $f_{b_i}$  is like an on-line BIST detecting faults at the first instance they affect circuit behaviour.

$u_f$  is measured by evaluating task fitness  $f_t$  separately under all SSA faults to every unit able to affect the task outputs. The same set of  $e$  evaluation conditions chosen for the current generation is used. If  $f_t$  falls by at least 0.001 due to a fault, then it is considered to affect task performance.  $u_i$  is measured by comparing the output of the circuit under each fault and input vector to its output for the same input vector under no faults. The output is deemed unaffected if it is the same at steps 10, 15 and 20 of the 20 time steps for which inputs are stable. Hence most faults that induce oscillations will be detected. If during the simulated time  $E$  goes high for more than  $eSize$  consecutive time slices then it is considered to

have gone high.  $eSize$  is set at 7: greater than average race conditions yet not excluding a wide range of behaviours. A circuit exhibiting a high  $E$  when no faults were in place was deemed to have  $f_{b_F} = f_{b_I} = 0$ .

It seems foolish to try to detect faults at every instance when some faults are not detected at any instance. Early runs (§3.1) incorporated the objectives into one overall fitness value, but with more emphasis on  $f_{b_F}$  than on  $f_{b_I}$  and more on  $f_t$  than on  $f_b$ :  $f = w_t \times f_t^2 + w_b \times f_b$  where  $f_b = w_{b_F} \times f_{b_F}^2 + w_{b_I} \times f_{b_I}$  and weights  $w_t \approx w_{b_F} \approx 0.85$ ,  $w_b = 1 - w_t$  and  $w_{b_I} = 1 - w_{b_F}$ . Later runs (§3.2) exploited our use of rank selection. The objectives were given the priority  $f_t > f_{b_F} > f_{b_I}$ , and when sorting the individuals the comparison operator only considered an objective if higher priority objectives were equal. An extra objective encouraging parsimony was also used, having the lowest priority of all.

### 3 Results

#### 3.1 One Bit Adder with Carry

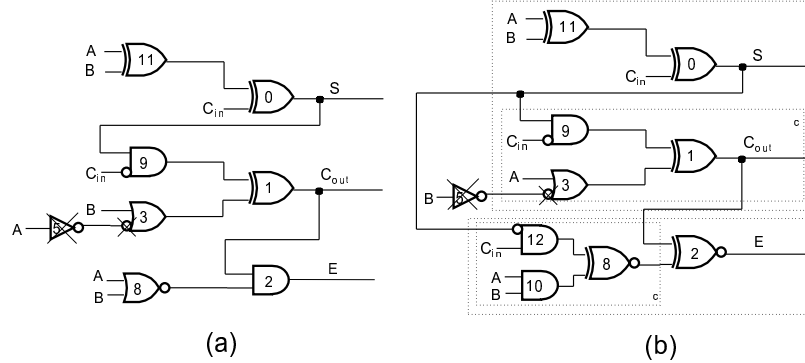
The maximum number of logic units was constrained to 13 and the genotype length was 156 bits. This circuit can be implemented with a minimum of five logic units. It is worth mentioning that the straightforward TPG-DUT-TRE BIST architecture (§1) would require 3 flip-flops and 2 Xor gates for the TPG LFSR and 2 flip-flops and 1 Xor for the TRE LFSR totalling 13 extra gates. Even if unusual to apply this method to circuits of this size, it is still valuable as a comparison of BIST quality and underlying operation principles because it is an industry standard. The minimal on-line BIST solution – a voter with two copies – would require 8 extra gates, 5 for the extra copy and 3 for the comparison logic.

**A) Hybrid Minimalist** This run was seeded with the elite of generation 10100 from a previous run which was started from a population of random individuals and did not take  $f_{b_I}$  into account. This elite was bred because it provided 82% off-line fault coverage being only 3 gates larger than the minimum adder.

After 4000 generations the elite was the circuit shown in Fig. 1a that can be trivially pruned to 7 gates by skipping unit 5 as shown. This circuit detects all SSA faults affecting task performance save unit 1 SA0 (clearly since  $E = u_1 \wedge u_8$ ) with either input test vector  $(A, B, C_{in}) = (0, 0, 0)$  detecting all SSA-1 faults or  $(1, 0, 0)$  detecting all SSA-0. Note these are the *only* input vectors detecting faults since  $E$  is low whenever  $B$  or  $C_{in}$  are high. Due to its short test pattern this could be considered as a hybrid of the on-line and off-line approaches using the best of both worlds.

The formula for  $E$  is  $\overline{B \vee C_{in}} \wedge C_{out}$  and acts as a consistency check: “ $C_{out}$  cannot be high if both  $B$  and  $C_{in}$  are low”. The use of Xor gates – which propagate any bit flip – in the calculation of  $S$ , which is used for  $C_{out}$ , ensures all faults raise  $C_{out}$  when  $B$  and  $C_{in}$  are low and  $A$  is high or low, thus checking  $S$  implicitly. This method seems like a cross between *arithmetic coding* “checksums” and *state monitoring*.

Table 1 compares this to conventional solutions.



**Fig. 1.** Evolved BIST solutions for the adder: (a) nearly full off-line fault coverage with 2 extra gates and (b) full on-line solution with 3 extra gates. Unit 5 can be trivially removed in both – presumably the two necessary mutations would have eventually occurred. LUTs are displayed as their equivalent two input logic gate.

**B) On-line Full** About 15000 generations later in the same run the full on-line BIST solution shown in Fig. 1b was the elite and can again be trivially pruned to 9 gates. The design can be decomposed into modules  $\alpha$  (adder) and  $\beta$  (BIST). Sub-modules  $\alpha_c$  and  $\beta_c$  calculate  $C_{out}$  and  $\overline{C_{out}}$  respectively while unit 2 (whose output is  $E$ ) acts as a voter. Errors in  $S$  are detected because they affect  $\alpha_c$  and  $\beta_c$  differently. Here evolution has arrived at the mission critical system design concept of *design diversity* [3] and gone beyond by aiming for circuits having minimal re-design necessary for maximal fault-coverage. Even though it is not surprising that  $\alpha_c$  and  $\beta_c$  are different, because there was no gene copy operator, the fact that their differences are exploited to evaluate the correctness of  $S$  and  $C_{out}$  simultaneously and under any fault and input vector is quite remarkable.

$E$  also went high whenever the circuit failed due to pathological gate delays (*delay faults*). Hence this method could also be useful to catch out deviations of standard behaviour due to abnormal operating conditions (*parametric faults*) which could be useful in unconventional evolved designs such as [27]. Sequential faults are caught since the circuits are tested under more than ten different random orderings of the full input test sequence at each evaluation. Being an on-line solution transient faults will also be detected as they look like a temporary SSA.

Table 1 compares this to conventional solutions.

**Adders discussed** Evolutionary methods have found better solutions in terms of standard BIST design criteria than those of conventional methods. It is up to

**Table 1.** Comparison of test quality attributes between conventional and evolved BIST solutions to the adder problem. Fault types: C=combinatorial, S=sequential, T=transient, D=delay. Test controller logic is not taken into account for off-line solutions.

Quality attribute	TPG-DUT-TRE	Evolved A	Voting	Evolved B
Fault Coverage	100%	90%	100%	100%
Test Scope	module	module	module	module
Fault types	C	CS	CST	CSTD
Area overhead	13 units	3 units	8 units	4 units
Pin overhead	2 (start/error)	2	1	1
Perf. penalty	0	1 gate delay	0	1 gate delay
Test time	8 test vectors	2 test vectors	0 (on-line)	0 (on-line)

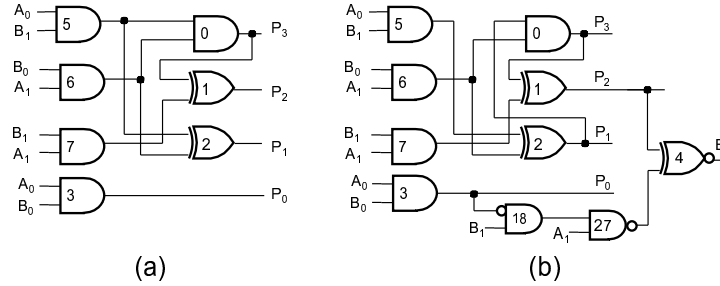
hardware designers to consider it worthwhile to have BIST on circuits of this size. Other runs minimized a hand-designed voting BIST solution from 8 overhead gates disappointingly just down to 7, arrived at a nearly full on-line solution with 8 overhead gates which included a low-pass filter ironing out race conditions because it was evolved with a small *eSize* (meaning it could be clocked faster when deployed), evolved a full on-line solution with 6 overhead gates exhibiting a linear relation between the minimum *eSize* it would work with and the standard deviation  $\sigma^2$  of its delays, and evolved a circuit very similar to the one in Fig. 1b from a random initial population.

### 3.2 Two Bit Multiplier

The maximum size allowed for these circuits was 28 LUTs and the genotype length was 392 bits. The smallest implementations of this circuit use 7 two input logic gates [21], such as shown in Fig. 2a. Even though the difference in size between the adder and multiplier is not great, there is twice the number of outputs to check: a TPG-DUT-TRE BIST approach would require 11 gates for each of the LFSRs totalling 22 extra gates while a voter would require 7 gates for the comparators again totalling 14 extra gates.

Note that evolving this kind of circuit, even without any BIST functionality uses a considerable amount of CPU time as this seems to grow exponentially with the complexity of the specification [31].

**Off-line Full** This run was seeded with the conventionally designed multiplier shown in Fig. 2a. After around 150000 generations the circuit of Fig. 2b was the elite (after pruning as in Fig. 1). This is nearly equal to the seed except that one input of unit 0 comes from unit 2 instead of unit 5, which implements a similar strategy to that used in §3.1: one output is influenced by others through their reuse in a cascading fashion – here output  $P_2$  is calculated using  $P_3$  which is calculated using  $P_1$  – so only this output need be checked. This is achieved by unit 14, which compares  $P_2$  to a recalculation of its inverse by units 18 and 27. Only these gates (14, 18, 27) are used for BIST



**Fig. 2.** (a) Hand designed two bit multiplier extracted from Fig. 15a of [21] is (b) post-evolved to incorporate BIST with 100% off-line coverage using only 3 gates overhead.

behaviour achieving full off-line fault coverage with the input vector pattern set  $(A_1, A_0, B_1, B_0) = \{(0, 1, 1, 1), (1, 0, 1, 1), (1, 1, 1, 1)\}$ , reducing the size of the LFSR to 5 gates while the signature analyzer LFSR is replaced by the error line. This run has shown that this method can also be used to add BIST behaviour to existing hand designed circuits without substantial modifications to circuit structure thus leaving its properties unaltered.

## 4 Conclusion

A proof of principle has been established and the questions set out in §1 can be answered: circuits with BIST behaviour are within the reach of evolutionary search, can reuse components for the main task and BIST functionality, and they are competitive in overhead both to off-line and on-line solutions. Low overhead on-line and hybrid solutions have been found. These solutions would function correctly in real hardware because the slightly unconventional simulator does capture the processes and variations influencing combinatorial circuits. Evolution has explored design space containing established methods for BIST design and beyond. Some solutions use a “checksum” formula on the current circuit state to evaluate correctness (comparable to the “Immunotronics” [4] approach for sequential systems which evaluates correct circuit state transitions), others increase testability by cascading outputs so that errors are propagated to a single output and others exploit design diversity to minimize redundancy in a voting system. These methods could prove useful to be adopted by designers. For circuits larger than those considered here, it is still unclear whether the enhanced BIST strategies produced by evolution would be worth the computational effort needed to produce them since the evolution of large BIST circuits faces the same problems – and perhaps, solutions – as with other circuits [12, 29, 30, 32]. However, Vassilev et al. [31] have suggested that when large circuits can be evolved (perhaps from a hand-designed seed), their size leads to greater scope for evolutionary optimization.

Most cases arrived at modular designs (as in [28, 21]). Further understanding of this effect may aid future experiments. The modularity of the circuits can be useful in identifying design patterns or principles.

Any single circuit will have its own characteristics for which a particular BIST strategy is most suited. Evolution through blind variation and selection is capable of searching for this strategy without constraints. The evolved BIST circuits seem well suited to uncovering *delay faults* and may be useful for *parametric faults* such as those occurring in [27]. The evolution of self-diagnosing analog hardware is an interesting possibility where the *E* line could give an estimate of *how* wrong the outputs are. Future work could also include the adoption of a more comprehensive fault model where the memory of the LUTs is affected – as in FPGAs – and other common faults are simulated, and tackling more complex tasks such as sequential circuits – synchronous and asynchronous – perhaps using techniques set out in [12, 29, 30, 32].

#### 4.1 Acknowledgments

Thanks to Matthew Quinn for helpful comments and to the COGS bursary that supports Michael Garvie's research.

#### References

1. The NASA/GSFC Radiation Effects and Analysis Home Page. <http://radhome.gsfc.nasa.gov/>.
2. A. Avizienis. Design diversity and the immune system paradigm: Cornerstones for information system survivability, 2000.
3. A. Avizienis and John P. J. Kelly. Fault-tolerance by design diversity: Concepts and experiments. *Computer*, 17(8):67–80, August 1984.
4. D.W. Bradley and A.M. Tyrrell. Immunotronics: Novel finite state machine architectures with built in self test using self-nonsel self differentiation. *IEEE Transactions on Evolutionary Computation*, 6(3):227–238, 2001.
5. R O Canham and A M Tyrrell. A multilayered immune system for hardware fault tolerance within an embryonic array. In J Timmis and P J Bentley, editors, *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)*, volume 1, pages 3–11, University of Kent at Canterbury, September 2002. University of Kent at Canterbury Printing Unit.
6. C. Dufaza. Theoretical properties of LFSRs for built-in self test. *INTEGRATION, the VLSI journal*, 25:17–35, 1998.
7. D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison Wesley, 1989.
8. H. Gollnabi and J. Provence. RMBITP: A reconfigurable matrix based built-in self-test processor. *Microelectronics Journal*, 28:115–127, 1997.
9. T. Higuchi, M. Iwata, and L. Weixin, editors. *Proc. 1st Int. Conf. on Evolvable Systems: From Biology to Hardware*, volume 1259 of *LNCS*. Springer-Verlag, 1997.
10. J. H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, 1975.

11. N. Jakobi. Half-baked, ad-hoc and noisy: Minimal simulations for evolutionary robotics. In Phil Husbands and Inman Harvey, editors, *Proc. 4th Eur. Conf. on Artificial Life (ECAL'97)*, pages 348–357. MIT Press, 1997.
12. T. Kalganova. Bidirectional incremental evolution in extrinsic evolvable hardware. In J. Lohn, A. Stoica, and D. Keymeulen, editors, *The Second NASA/DoD workshop on Evolvable Hardware*, pages 65–74, Palo Alto, California, 13-15 2000. IEEE Computer Society.
13. J. R. Koza. *Genetic Programming: On the programming of computers by means of natural selection*. MIT Press, Cambridge, Mass., 1992.
14. J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane. Reuse, parameterized reuse, and hierarchical reuse of substructures in evolving electrical circuits using genetic programming. In T. Higuchi, M. Iwata, and L. Weixin, editors, *Proc. 1st Int. Conf. on Evolvable Systems: From biology to hardware (ICES-96)*, number 1259 in LNCS, pages 312–326. Springer-Verlag, 1996.
15. J. Lach, W. Mangione-Smith, and M. Potkonjak. Low overhead fault-tolerant FPGA systems, 1998.
16. P. Layzell. A new research tool for intrinsic hardware evolution. In M. Sipper, D. Mange, and A. Pérez-Urbe, editors, *Proc. 2nd Int. Conf. on Evolvable Systems (ICES'98)*, volume 1478 of LNCS, pages 47–56. Springer-Verlag, 1998.
17. J. Lohn, A. Stoica, D. Keymeulen, and S. Colombano, editors. *Proc. 2nd NASA/DoD workshop on Evolvable Hardware*. IEEE Computer Society, 2000.
18. D. Mange, A. Stauffer, and G. Tempesti. Embryonics: A microscopic view of the molecular architecture. In A. Perez-Urbe M. Sipper, D. Mange, editor, *Proc. 2nd Int. Conf. on Evolvable Systems (ICES1998: From biology to hardware)*, volume 1478 of LNCS, pages 285–195. Springer-Verlag, 1998.
19. J. Miller. On the filtering properties of evolved gate arrays. In A. Stoica, J. Lohn, and D. Keymeulen, editors, *The First NASA/DoD Workshop on Evolvable Hardware*, pages 2–11, Pasadena, California, 1999. Jet Propulsion Laboratory, California Institute of Technology, IEEE Computer Society.
20. J. Miller, A. Thompson, P. Thomson, and T. Fogarty, editors. *Proc. 3rd Int. Conf. on Evolvable Systems (ICES2000): From Biology to Hardware*, volume 1801 of LNCS. Springer-Verlag, 2000.
21. J. F. Miller, D. Job, and Vesselin K. Vassilev. Principles in the evolutionary design of digital circuits - part I. *Genetic Programming and Evolvable Machines*, 1(3), 2000.
22. W. Mangione-Smith N. Shnidman and M. Potkonjak. On-line fault detection for bus-based field programmable gate arrays. *IEEE Transactions on VLSI systems*, 6(4):656–666, 1998.
23. M. Sipper, D. Mange, and A. Pérez-Urbe, editors. *Proc. 2nd Int. Conf. on Evolvable Systems (ICES98)*, volume 1478 of LNCS. Springer-Verlag, 1998.
24. A. Steininger. Testing and built-in self-test - a survey. *Journal of Systems Architecture*, 46:721–747, 200.
25. A. Stoica, D. Keymeulen, and J. Lohn, editors. *Proc. 1st NASA/DoD workshop on Evolvable Hardware*. IEEE Computer Society, 1999.
26. R. Tanese. Distributed genetic algorithms. In J.D. Schaffer, editor, *Proc. of the Third International Conference of Genetic Algorithms*, pages 434–439. Morgan Kauffmann, 1989.
27. A. Thompson, I. Harvey, and P. Husbands. Unconstrained evolution and hard consequences. In E. Sanchez and M. Tomassini, editors, *Towards Evolvable Hardware: The evolutionary engineering approach*, volume 1062 of LNCS, pages 136–165. Springer-Verlag, 1996.

28. A. Thompson and P. Layzell. Analysis of unconventional evolved electronics. *Communications of the ACM*, 42(4):71–79, April 1999.
29. P. Thomson. Circuit evolution and visualisation. In J. Miller, A. Thompson, P. Thomson, and T. Fogarty, editors, *Proc. 3rd Int. Conf. on Evolvable Systems (ICES2000): From biology to hardware*, volume 1801 of *LNCS*, pages 229–240. Springer-Verlag, 2000.
30. J. Torresen. A divide-and-conquer approach to evolvable hardware. *Lecture Notes in Computer Science*, 1478:57–??, 1998.
31. V. Vassilev, D. Job, and J. Miller. Towards the automatic design of more efficient digital circuits. In J. Lohn, A. Stoica, and D. Keymeulen, editors, *The Second NASA/DoD workshop on Evolvable Hardware*, pages 151–160, Palo Alto, California, 13-15 2000. IEEE Computer Society.
32. V. Vassilev and J. Miller. Scalability problems of digital circuit evolution: Evolvability and efficient designs. In J. Lohn, A. Stoica, and D. Keymeulen, editors, *The Second NASA/DoD workshop on Evolvable Hardware*, pages 55–64, Palo Alto, California, 13-15 2000. IEEE Computer Society.
33. H. Wunderlich. BIST for systems-on-a-chip. *INTEGRATION, the VLSI journal*, 26:55–78, 1998.