Transistor-Level Evolution of Digital Circuits Using a Special Circuit Simulator

Luděk Žaloudek and Lukáš Sekanina

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

Abstract. An evolutionary algorithm is used to design digital circuits at the transistor level. In particular, various static CMOS circuits with up to four inputs were evolved. The increase in the complexity of evolved circuits wrt existing circuits evolved at the transistor level is primarily caused by two phenomena: the usage of a specialized circuit simulator and restriction of the search space. Because we restricted the search space to the set of "reasonable designs" we could employ imperfect, but very fast circuit simulation. The usage of proposed simulator allowed exploring more candidate designs than a conventional Spice-based approach. However, in some cases, an incorrect behavior was detected after validation of evolved circuits using Spice simulator.

1 Introduction

A typical design flow of a digital circuit can be viewed as a set of transformations starting from a specification and then creating behavioral design, RTL design, gate level design, transistor level design and physical design. In this paper, our attention will be focused on the gate-level design and transistor-level design only.

1.1 Evolutionary Design of Digital Circuits

Various methods have been developed for synthesis of digital circuits at the gate level. As Miller noticed, conventional methods can handle large numbers of input variables but they are not able to adapt to new non-canonical building blocks. They perform logically correct transformations over canonical representations but they are not able to escape from the space of these logically correct representations [1]. On the other hand, evolutionary design methods work in a much larger space of possible solutions. It was shown that evolutionary algorithm can generate innovative implementations of various digital circuits [1, 2, 3]. However, evolutionary circuit design is not competitive with conventional methods in many areas. The scalability of representation and scalability of fitness calculation were identified as major problems of the evolutionary approach [4].

In conventional methodologies, a gate-level circuit is mapped to transistor-level circuit in such a way that gates are replaced by their transistor-level implementations which are available in a particular fabrication technology. Unfortunately, this

G.S. Hornby et al. (Eds.): ICES 2008, LNCS 5216, pp. 320–331, 2008.

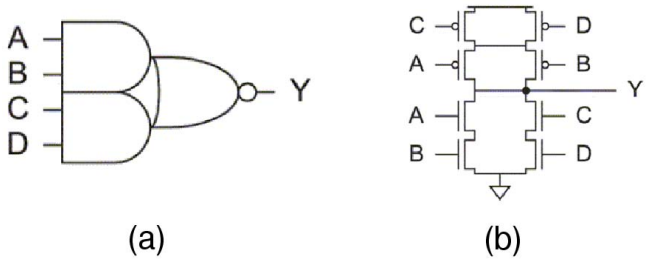

Fig. 1. And-Or-Invert Circuit: a) at the gate level, b) CMOS implementation

fact is not reflected in most papers dealing with evolutionary design of gate-level circuits. Instead of minimizing the number of transistors (or the area, more precisely), a typical objective is usually to minimize the number of gates, see e.g. [1, 5, 6, 7]. The following example shows that some transistor-level implementations can be more efficient than their gate-level counterparts: Consider logic expression $Y = (AB + CD)'$ (a corresponding circuit is shown in Fig. 1a) which seems to be optimal at the gate level. It requires two AND gates (each of them costs 6 transistors in ordinary CMOS technology) and a single NOR gate (4 transistors), which costs 16 transistors in total. However, Y can be implemented using 8 transistors only when a special AND-OR-Invert circuit is employed (see Fig. 1b). Similar observation is valid for some other circuits, for example adders [8], majority circuits [9] etc.

In comparison with the gate-level evolution, only relatively simple circuits (such as two-input ordinary gates) were evolved at the transistor level. For extrinsic evolution, direct representations or generative encodings are of major interest. When a direct representation is used, the genes directly represent the types, values and connections of circuit components in the chromosome [2, 10]. By generative encodings we mean such approaches in which the chromosome contains a program for construction of target circuit (e.g. NAND gate was evolved using developmental genetic programming in [11]). Intrinsic evolution was performed on various reconfigurable platforms, including PAMA [2], FPTA-2 [12] and FPTA-Heidelberg [13]. Mixtrinsic evolution which combines extrinsic and intrinsic approach was demonstrated for the AND gate design problem in [14]. Finally, Stoica's group evolved various polymorphic gates, i.e. multifunctional gates whose logic function is controlled by an external factor, for example, by the level of the power supply voltage, temperature or external voltage [15, 16]. These gates exhibit quite unconventional structure. Only the NAND/NOR gate controlled by Vdd was fabricated so far [16]. The goal of some of these experiments was not only to demonstrate that the evolutionary design of gates really works but also to provide new, robust, portable and fault tolerant implementations of the gates. A general observation is that the evolutionary design of the XOR and NXOR gate is the most difficult task in this area.

The limit in the complexity of evolved transistor-level circuits is primarily caused by the problem of scalability of representation and a very time consuming

fitness calculation. When the extrinsic evolution is carried out, precise simulations of a candidate circuit must be performed in order to detect possible malfunctions in its design. For these purposes, simulators of the Spice family are usually used. Various techniques allowing to eliminate typical problems of evolved gates (such as incorrect transient response, insufficient driving capabilities, incorrect operation at different timescales) were presented in [16]. This strongly contrasts with the extrinsic evaluation of gate-level designs which can be simply accelerated using, for example, parallel simulation and other techniques.

1.2 The Objectives of This Paper

The goal of this paper is to show that non-trivial digital circuits can be evolved directly at the transistor level. In particular, we will deal with static CMOS designs of logic functions of four inputs in maximum. Various techniques were employed to make the evolution faster and more robust: (1) In order to quickly evaluate candidate circuits, we developed a simulation tool allowing the speedup of two orders of magnitude in comparison with the conventional Spice simulator. At the end of evolution, behavior of resulting circuits is precisely simulated using Spice simulator. (2) Some expert knowledge was included to the representation of candidate circuits (see Section 2.1). Although it reduces the space of possible designs, more reliable solutions can be obtained. (3) Special techniques, such as providing direct input values together with their complementary values, were introduced for evaluation of candidate circuits. They can make the problem easier for evolution.

The paper is organized as follows. Proposed representation and evolutionary algorithm are presented in Section 2. Section 3 introduces the circuit simulator we developed to quickly evaluate candidate circuits. Various evolved circuits are presented in Section 4. Conclusions are given in Section 5.

2 Proposed Method

Evolutionary algorithm is used to search for digital circuits composed of NMOS and PMOS transistors, i.e. CMOS circuits. At the most abstract level, the PMOS transistor might be viewed as a three node component (with nodes denoted as gate – G, source - S and drain – D) which exhibits either a zero impedance between D and S when logic 0 is present at G or an infinite impedance between D and S when logic 1 is present at G. In case of NMOS transistors, the impedance between D and S is controlled by complementary level at G wrt to PMOS transistors. The goal is to generate digital circuits directly at this level and possibly to discover novel implementations for chosen problems.

This section describes circuit representation, evolutionary algorithm, fitness calculation process and various techniques used to make the problem easier for evolution.

2.1 Circuit Representation

The representation is loosely inspired by Cartesian Genetic Programming (CGP) [1]. A candidate n_i-input/n_o-output circuit is represented as a string of integers using a direct genotype-phenotype mapping. In the chromosome, ground terminal (GND) is encoded as 0 and the power supply voltage terminal (Vcc) is encoded as 1. The j-th primary input is encoded using value 1+j. Terminals of all transistors are uniquely numbered, starting from $n_i + 3$. The chromosome is divided to two parts:

- The first part is composed of K quadruples; each of them is devoted to a single transistor. The first value of the quadruple denotes the type of transistor (–2 for NMOS, –1 for PMOS). The second value defines the terminal connected to transistor's gate, the third value refers to the terminal connected to transistor's source and the fourth value is the terminal connected to the drain.
- The second part consists of n_o integers which determine the terminals connected to the primary outputs.

Figure 2 shows a typical 2-input NAND gate and its representation in the chromosome. This representation is designed for evolution, so we did not want to limit the circuits to conventional design patterns. However, experience of designers suggest that some connection patterns should be forbidden in order to avoid really wrong solutions and make the evolution easier. The following techniques were implemented:

- At least one transistor must be connected to Vcc and one to GND.
- The source and drain terminals are allowed to be connected only either to other sources/drains or Vcc or GND clamps.
- It is not possible to connect a PMOS transistor to GND and it is not possible to connect an NMOS transistor to Vcc. The reason is that PMOS degrades logic 0 signal and NMOS degrades logic 1 signal.

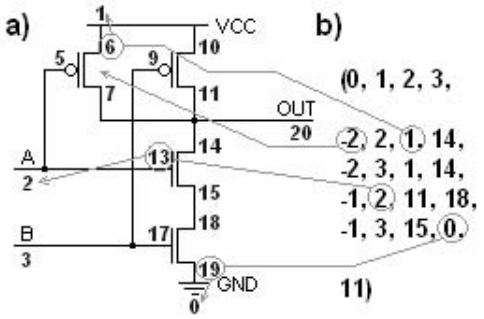

Fig. 2. a) Typical 2-input NAND gate. b) Chromosome representation of 2-input NAND where the quadruples in the middle represent individual transistors - grey circles connect selected genes and their corresponding transistor terminals while arrows indicate the connection of the terminals according to genes' values.

- It is not possible to connect any output directly to one of the voltage clamps (Vcc or GND).
- Gate electrodes can be connected to primary inputs only. Note that in standard CMOS circuits, the gates which are not connected to the primary inputs are used almost only in the output or input inverters.
- There is also a fourth transistor terminal which we completely omitted so far – the body. We simply assume that the body of PMOS transistors is connected to Vcc and the body of NMOS transistors is connected to GND. Then, PMOS transistors will be open for logic 0 and NMOS transistors will be open for logic 1 signal at the gates.

2.2 Evolutionary Algorithm

The algorithm is based on the $(1 + \lambda)$ evolutionary strategy which means that every new population consists of the best individual of the previous population and its λ mutants (λ is usually 8–40). Crossover is not used at all. The initial population is generated randomly. Again, this process must respect the rules mentioned in Section 2.1.

Every offspring is generated by mutating m genes of the parent (usually, $m = 1 - 3$). The mutation function randomly selects one gene and randomly changes its value. The new value is limited by the mentioned rules. In addition, the following rules must be followed:

- The source/drain can not be connected to the drain/source of the same transistor.
- A terminal can not be connected to itself.
- The only connection to Vcc or GND existing in the circuit can not be removed.

The mutation is attempted m times and if it sometimes fails, a new gene is selected for mutation. This is repeated until m mutations are really performed. If the fitness value reaches the maximum value or the maximum number of generations is exhausted, the algorithm stops and returns current best individual.

2.3 The Fitness Function

At the beginning of evaluation, the fitness value is cleared. In order to evaluate a candidate circuit, all possible binary combinations are generated and applied at the primary inputs (i.e. 2^{n_i} test vectors). This gives us $n_o 2^{n_i}$ output values which are compared with the reference output values defined in the specification. If the simulated output value matches the reference output, the fitness value is incremented by 1. If the value matches only partially (e.g. there is weak 1 instead of strong 1 at the output) only 0.75 is added to the fitness value. If the output is Z (high impedance) only 0.5 is added and if the value is X (undefined) then 0.5 is subtracted from the fitness value. Otherwise, the fitness value is not updated. The maximum fitness value is therefore $n_o 2^{n_i}$.

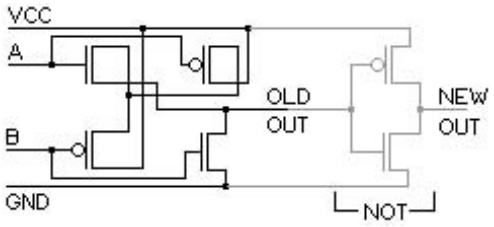

Fig. 3. Evolved AND circuit with explicitly added inverter (shown in grey)

2.4 Special Techniques for the Fitness Function

Sometimes, it is hard to reach a solution when there is only one different value among lots of identical values in the required truth table, e.g. for the 4-input NOR gate design problem. The evolution easily finds a circuit outputting the 15 zero bits but is not encouraged to find the remaining logic 1 for the input combination 0000. This can be avoided by adding one "fake" constant input and increasing the weight for the problematic input combinations in the fitness function. The fake input is ignored in the resulting solution.

Another problem is that the model does not allow connections from gate electrodes to the terminals which are different from the primary inputs, so the evolution would not consider inverters. This can be avoided by explicit specifying the outputs which will be equipped by invertors. Figure 3 shows the AND circuit which was evolved using an explicit inverter connected at the output.

Experiments also suggest that in some cases it is easier to evolve required solutions when the primary input signals are provided together with their complementary signals explicitly. For example, instead of using two inputs (i.e. four test vectors), four inputs consisting of two original inputs and their complementary values are employed (i.e. there are sixteen test vectors).

3 Circuit Simulator

In order to quickly evaluate a candidate circuit, a circuit simulator was developed which works at the level of simplified models of PMOS and NMOS transistors. The simulator operates with six logic levels: Strong 1, weak 1, strong 0, weak 0, high impedance and undefined value. The simulator works directly with the proposed circuit representation.

At the beginning of simulation, all values of source/drain/output terminals are set to high impedance and the primary inputs are set according to a given training vector. First, the path of the strong 1 signal is followed from Vcc clamp through all the connections. The values on the terminals on the way are updated to strong 1. When the signal reaches a transistor source or drain (it does not matter which side, both work identically from the viewpoint of microelectronics), the algorithm checks the transistor state (logic signal on the gate) and updates its state according to the rule table developed for the simulator. The rules reflect the

fact, that strong 1 degrades on NMOS or poorly open transistors. On the other hand, strong 0 degrades on PMOS or poorly open transistors. If the algorithm recognizes that the transistor is open in some way, it propagates the signal to the other side if it is possible. Note that there is no need to propagate it when the signal on the other side is stronger or identical. If the algorithm finds the opposite value or undefined valued on the other side, a short circuit is encountered. That means, that the output is set to undefined value and the process proceeds with another training vector. Otherwise, the signal is followed until it propagates. Then, other signals are processed in the same way (of course, logic 0 propagates from GND) in the following order: weak 1s, strong 0s and weak 0s. At the end, the values on the output terminals are updated.

Evolved circuits are validated using Spice 3f5 (a part of ngspice package) which utilizes level 8 model of transistors. Typical parameters of transistors are 1.2 um for length and 3.6 um for width in PMOS and 1.2 um for length and 1.8 um for width in NMOS. In particular, static and transient analysis are performed for different timescales and loads.

4 Results

The first part of this section deals with elementary CMOS gates that were evolved just to verify whether proposed method really works. Then, XOR gates were investigated in the second set of experiments. In the third part, results obtained for more complex problems are reported.

4.1 Evolution of Elementary Gates

In this part, we evolved the simplest gates: 2-input NAND, 3-input NOR, 4-input NOR and 3-input AND. Table 1 summarizes parameters of evolutionary algorithm and obtained results.

The 2- and 3-input gates were relatively easy to evolve with the exception of AND, where it was necessary to utilize an explicit output inverter. The hardest problem of this group was the evolution of 4-input NOR, because the truth table of this logic gate contains 15 zero values and only a single non-zero value. The evolution simply could not find the solution because it solved the 15 zero cases and was not encouraged enough to solve the remaining case because any change

Table 1. Evolved elementary gates – parameters of EA and results obtained using proposed simulator

Circuit	Inputs	Max. Trans.	Max. geners.	Mutation	Pop. size	Runs	Succ. Runs
NAND-2	2	4	100000	1	10	70	100%
NOR-3	3	6	200000	2	20	90	100%
NOR-4	4	8	200000	2	20	90	100%
AND-3	3	8	150000	2	20	90	100%
XOR-2	2	10	200000	1	20	100	100%

would break the zeros and decrease the fitness value. Surprisingly, this problem did not occur so much with the 3-input logic gates, however, it worsened the average results. We employed a simple trick mentioned earlier to solve the problem which resulted in 100% success rate of the EA. We added a false input and copied the combination with the non-zero value into the extended training vectors set to obtain a problem with more non-zero values. In summary, the evolutionary algorithm rediscovered well-known implementations for all problems in this category.

The experiments were performed on Intel Xeon 5345 (2.32 GHz) processor. On average, 8.08 seconds are needed to evaluate 10000 populations for NOR-4 problem which is roughly 209 times faster than evaluation on the Spice simulator. While 175.6 generations are required on average to evolve NAND-2, 3808.2 generations are needed to evolve NOR-4.

4.2 Evolution of XORs

Last row of Table 1 summarizes parameters of the evolutionary algorithm and results for the XOR-2 gate. In order to evolve the XOR-2 gate, 2.17 times more generations are needed in average than for the NAND-2 problem. Figure 4 shows one of evolved correct solutions.

Evolved XOR gates were carefully tested in Spice simulator. The testing included different time scales (duration of input signals is 15 us, 15 ns, 1.5 ns, 150 ps

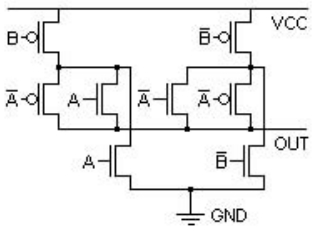

Fig. 4. Evolved two-input XOR gate

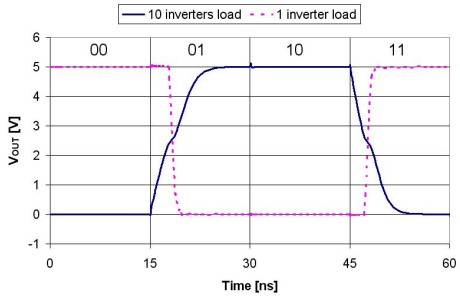

Fig. 5. The output of evolved XOR-2 gate with different load for combinations of input signals 00, 01, 10 and 11

Table 2. Evolved gates – parameters of EA and results obtained using proposed simulator. Last column shows the success rate in Spice simulator.

Circuit	Max. trans.	Max. generes.	Mut.	Pop. size	Runs	Succ. runs	Succ. Spice
NAND/NOR	10	500000	1	20	100	100%	33%
MUX-2	15	200000	1	20	100	100%	49%
MUX-2	8	200000	1	20	100	95%	19%
FA	30	200000	1	20	20	85%	20%
FA	25	300000	1	20	20	40%	5%
FA	23	400000	1	20	20	10%	0%
FA	21	400000	1	20	20	5%	0%

and 15 ps) and different loads (1–10 gates). Figure 5 shows the output signal of evolved XOR gate taken from Spice simulations when 1 gate and 10 gates are connected at the gate output. No significant output degradation was noticed for duration of the input signals down to 1.5 ns and when 1–10 gates are connected at the output.

4.3 Evolution of More Complex Circuits

The goal of the next series of experiments was to evolve more complex circuits, including NAND/NOR gate controlled by an external logic signal (which is, in fact, the inverted majority function), 2-to-1 multiplexer and 1-bit full adder (FA). Table 2 shows that a solution with the maximum fitness value was obtained for all problems. The success rate decreases with growing complexity of problems. Also the number of generations needed to find a solution is much higher than in the previous set of experiments.

Especially for these more complex circuits we observed that although proposed simulator can indicate that a particular solution is correct (fitness = 100%), more precise simulations conducted using the Spice simulator can show the opposite – some circuits do not work exactly according to the specification. Figure 6 shows two evolved NAND/NOR circuits which obtained the perfect fitness score; however, the first one does not work correctly in Spice for one input combination (see Fig. 7). The second one works perfectly also in Spice (Fig. 8). As this circuit consists of 10 transistors, it can be considered as a better solution than its gate-level variant consisting of NAND (4 transistors), NOR (4 transistors) and multiplexer (6 transistors), i.e. 14 transistors in total.

Some solutions (also working in Spice simulator) were evolved for the multiplexer problem and the full adder problem. However, these solutions contain more transistors than conventional solutions [8].

The overall difference between the proposed simulator and Spice simulator may be seen in Table 2 where the amount of evolved solutions working in Spice is shown next to the amount of solutions working with the proposed simulator. Note that the amount of solutions working in Spice decreases with the complexity of the circuits.

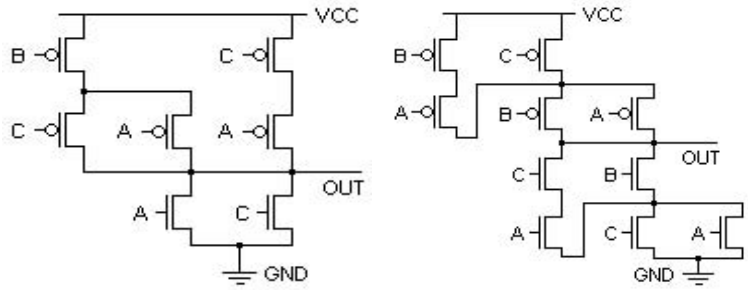

Fig. 6. Examples of evolved structures (both with the 100% fitness value according to proposed simulator) when the goal is to obtain the NAND/NOR gate which should perform NAND(A, B) for C = 0 and NOR(A, B) for C = 1. Left: Problematic behavior in Spice (see Fig. 7) Right: Correct behavior in Spice (see Fig. 8).

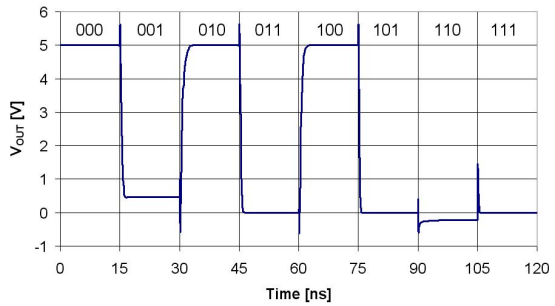

Fig. 7. Spice simulations: The output of evolved NAND/NOR gate for combinations of input signals CAB = 000, ..., 111, where A and B are data inputs and C is the control signal. The output is wrong for CAB=001.

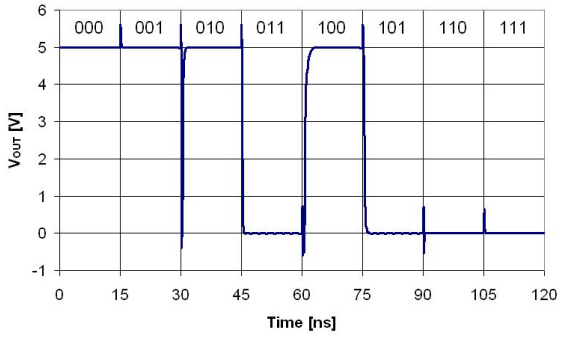

Fig. 8. Spice simulations: The output of evolved NAND/NOR gate for combinations of input signals CBA = 000, ..., 111, where A and B are data inputs and C is the control signal. The output is correct for all input combinations.

In 33% runs of the EA, proposed simulator gives the same results as Spice simulator for the NAND/NOR circuit. 55% runs lead to circuits whose outputs differ from Spice simulations only for one combination of input signals; two differences are observable in 11% runs and four differences in 1% runs.

5 Conclusions

The method used in this work can be characterized as evolutionary design using a direct genotype-phenotype mapping in which indispensable domain knowledge is included in genetic operators and fitness calculation. The use of imperfect but very fast circuit simulator allowed us to evolve working solutions only for some of target problems. Experiments with more complex problems have shown the limitations of proposed simulator: The output signals in Spice resemble the desired signal shape but at some input combinations, the outputs are severely degraded. The reason for this is that the simulator does not cover transistors more elaborate physical characteristics. After examining evolved circuits and the reasons why the circuits fail in Spice, we plan to redesign and improve our simulation algorithm.

In most cases, ordinary implementations of standard gates were re-discovered by the evolutionary algorithm (only the implementation of NAND/NOR gate could be considered as innovative). That is not a surprising outcome when one considers that these implementations have been optimized and used in micro-electronics for tens of years. In future work, we plan to evolve digital circuits at the transistor level which will include a built-in self-test mechanism, similar to Garvies's gate-level circuits reported in [17]. It is assumed that more area-efficient self-checking circuits could be evolved at the transistor level than at the gate level.

Acknowledgements

This work was partially supported by the Grant Agency of the Czech Republic under contract No. 102/06/0599 *Methods of polymorphic digital circuit design* and the Research Plan No. MSM 0021630528 - *Security-Oriented Research in Information Technology.*

References

[1] Miller, J., Job, D., Vassilev, V.: Principles in the Evolutionary Design of Digital Circuits – Part I. Genetic Programming and Evolvable Machines 1(1), 8–35 (2000)
[2] Zebulum, R., Pacheco, M., Vellasco, M.: Evolutionary Electronics – Automatic Design of Electronic Circuits and Systems by Genetic Algorithms. The CRC Press International Series on Computational Intelligence (2002)
[3] Higuchi, T., Liu, Y., Yao, X.: Evolvable Hardware. Springer, Heidelberg (2006)
[4] Yao, X., Higuchi, T.: Promises and Challenges of Evolvable Hardware. IEEE Transactions on Systems, Man, and Cybernetics 29(1), 87–97 (1999)

[5] Vassilev, V., Job, D., Miller, J.: Towards the Automatic Design of More Efficient Digital Circuits. In: Lohn, J., Stoica, A., Keymeulen, D., Colombano, S. (eds.) Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware, pp. 151–160. IEEE Computer Society Press, Los Alamitos (2000)
[6] Chen, D., Aoki, T., Homma, N., Terasaki, T., Higuchi, T.: Graph-Based Evolutionary Design of Arithmetic Circuits. IEEE Trans. on Evolutionary Computing 6(1), 86–100 (2002)
[7] Zhao, S., Jiao, L.: Multi-objective evolutionary design and knowledge discovery of logic circuits based on an adaptive genetic algorithm. Genetic Programming and Evolvable Machines 7(3), 195–210 (2006)
[8] Weste, N., Harris, D.: CMOS VLSI Design: A Circuits and Systems Perspective, 3rd edn. Addison-Wesley, Reading (2004)
[9] Quintana, J.M., Avedillo, M.J., Jiménez, R., Rodríguez-Villegas, E.: Practical low-cost cpl implementations of threshold logic functions. In: Proc. of the 11th ACM Great Lakes Symposium on VLSI 2001, pp. 139–144. ACM Press, West Lafayette (2001)
[10] Zebulum, R., Vellasco, M., Pacheco, M.: Evolutionary design of logic gates. In: Proceedings of the workshop in Evolutionary Design, Artificial Intelligence in Design Conference, Lisbon, Portugal (1998)
[11] Keane, A., Streeter, M.J., Mydlowec, W.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Springer, New York (2004)
[12] Stoica, A., Zebulum, R.S., Keymeulen, D., Ferguson, M.I., Duong, V., Guo, X.: Evolvable hardware techniques for on-chip automated reconfiguration of programmable devices. Soft Computing 8(5), 354–365 (2004)
[13] Langeheine, J.: Intrinsic Hardware Evolution on the Transistor Level. PhD thesis, Rupertus Carola University of Heidelberg (2005)
[14] Stoica, A., Zebulum, R., Keymeulen, D.: Mixtrinsic Evolution. In: Miller, J.F., Thompson, A., Thompson, P., Fogarty, T.C. (eds.) ICES 2000. LNCS, vol. 1801, pp. 208–217. Springer, Heidelberg (2000)
[15] Stoica, A., Zebulum, R.S., Keymeulen, D.: Polymorphic electronics. In: Liu, Y., Tanaka, K., Iwata, M., Higuchi, T., Yasunaga, M. (eds.) ICES 2001. LNCS, vol. 2210, pp. 291–302. Springer, Heidelberg (2001)
[16] Stoica, A., Zebulum, R., Guo, X., Keymeulen, D., Ferguson, I., Duong, V.: Taking Evolutionary Circuit Design From Experimentation to Implementation: Some Useful Techniques and a Silicon Demonstration. IEE Proc. Comp. Digit. Tech. 151(4), 295–300 (2004)
[17] Garvie, M.: Reliable Electronics through Artificial Evolution. PhD thesis, University of Sussex (2005)