Genetic Programming and Evolvable Machines manuscript No.
(will be inserted by the editor)

Formal Verification of Candidate Solutions for Post-Synthesis Evolutionary Optimization in Evolvable Hardware

Zdenek Vasicek · **Lukas Sekanina**

Received: date / Accepted: date

Abstract We propose to utilize a formal verification algorithm to reduce the fitness evaluation time for evolutionary post-synthesis optimization in evolvable hardware. The proposed method assumes that a fully functional digital circuit is available. A post-synthesis optimization is then conducted using Cartesian Genetic Programming (CGP) which utilizes a satisfiability problem solver to decide whether a candidate solution is functionally correct or not. It is demonstrated that the method can optimize digital circuits of tens of inputs and thousands of gates. Furthermore, the number of gates was reduced for the LGSynth93 benchmark circuits by 37.8% on average with respect to results of the conventional SIS tool.

Keywords Cartesian genetic programming · circuit optimization · SAT solver · evolvable hardware

1 Introduction

In the evolvable hardware field, evolutionary algorithms (and other bio-inspired algorithms) are applied either for automated hardware design or dynamic hardware adaptation or repair [53, 20, 16, 54, 39, 30]. According to Gordon and Bentley, the field of evolvable hardware originates from the intersection of computer science, electronic engineering and biology and typically includes aspects of hardware design and optimization techniques, particularly logic synthesis, technology mapping, placing and routing [14].

In this article we will only deal with evolvable hardware as a method for automated design, i.e. with a scenario in which the evolutionary algorithm is used only

Zdenek Vasicek
Faculty of Information Technology, Brno University of Technology, Czech Republic
E-mail: vasicek@fit.vutbr.cz

Lukas Sekanina
Faculty of Information Technology, Brno University of Technology, Czech Republic
E-mail: sekanina@fit.vutbr.cz

in design and optimization phase of a product. In this context, evolvable hardware potentially offers promising solutions to logic synthesis and optimization where new problems have recently been identified. It was shown that commonly used logic synthesis algorithms are not capable of efficient synthesis and optimization for some circuit classes, especially for large circuits and circuits containing hard-to-synthesize substructures [5, 10]. In some cases the size of synthesized circuits is of orders of magnitude greater than the optimum.

The *scalability problem* has been identified as one of the most difficult problems the researchers are faced with in the evolvable hardware field. The scalability problem means such situation in which the evolutionary algorithm is able to provide a solution to a small problem instance; however, only unsatisfactory solutions can be generated for larger problem instances. Although various methods have been proposed to eliminate the scalability problem (see Section 2), only a partial success has been achieved in some domains.

We will consider a subarea of the scalability problem – the *scalability of evaluation*, in the context of optimization problems. We will show that it can reasonably be eliminated in a task of *gate-level post-synthesis optimization* of complex combinational circuits consisting of thousands of gates and having tens of inputs and outputs. The method assumes that a fully functional circuit is available in a standard netlist format which can be obtained using a conventional synthesis algorithm. The main goal is to reduce the number of gates.

We propose to use modern formal verification methods that have been overlooked by the evolvable hardware community so far. The proposed method utilizes equivalence checking algorithms (those used by conventional synthesis algorithms) that allow a significant acceleration of the fitness evaluation procedure. Particularly, the method is based on a post-synthesis optimization of combinational circuit conducted using Cartesian genetic programming (CGP) [34] which evaluates candidate solutions using the satisfiability (SAT) solver [9]. The technique relies on functional correctness of an initial solution (a seed for CGP). Note that not all applications of evolvable hardware fall into this category because such a seed is not generally available. We have also introduced some techniques that explore the CGP representation and operators to reduce the number of clauses for the SAT solver and thus further shorten the evaluation time.

Optimized circuits are compared with the most compact circuits that we obtained from iterative application of decomposition and re-synthesis process which is conducted by conventional synthesis tools such as ABC and SIS.

The plan for this article is as follows. Section 2 introduces the concept of evolvable hardware and surveys the scalability problems. In Section 3, the proposed method is explained. The key contribution of this article, the construction of the fitness function on the basis of formal verification techniques is introduced in Section 4. The experimental evaluation of the proposed method represents the content of Section 5. Some practical aspects of the method are discussed in Section 6. Section 7 gives our conclusions from the experimental evaluation and also some suggestions for further work.

2 Evolvable Hardware and Its Scalability

2.1 Motivation for Circuit Evolution

Figure 1 explains the concept of evolvable hardware: Electronic circuits that are encoded as finite strings of symbols are constructed and optimized by the evolutionary algorithm to obtain a circuit implementation satisfying a specification given by designer. Since the introduction of evolvable hardware at the beginning of nineties [21, 11], the main motivation for circuit evolution can be seen in the fact that evolutionary approach can lead to fully functional designs without being instructed how to construct them. Hence one of the goals is to evolve as complex circuit as possible with a minimum computational effort and domain knowledge supplied [51,43,41]. A typical application could be a reactive robot controller which is evolved in a sufficiently large reconfigurable device where there is no need to optimize the number of gates and delay [27].

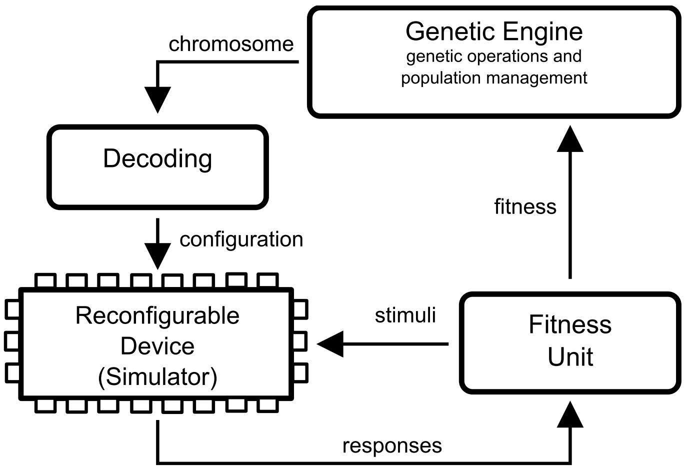


Fig. 1 The principle of evolvable hardware

In many applications a perfect circuit response must by obtained for all requested assignments to the inputs. The fitness function is usually constructed in such a way that all requested assignments are applied to the inputs of a candidate circuit and the fitness value is defined as the number of bits that the candidate circuit computes correctly. When target functionality is obtained additional criteria can be optimized. Evolution of arithmetic circuits is a typical example of that class [49,32]. To give examples where partially imperfect solutions are acceptable we can mention evolution of image filters, classifiers or predictors [19,38,12]. In addition to functionality, another goal can be to obtain a solution which exhibits a better quality in some aspects with respect to existing designs of the same category. For example, a solution would occupy a smaller area on a chip, compute faster, provide a better precision, reduce the energy consumption, increase the reliability etc.

2.2 Scalability of Fitness Evaluation

In case of combinational circuit evolution, the evaluation time of a candidate circuit grows exponentially with the increasing number of inputs (assuming that all possible input combinations are tested in the fitness function). This fitness calculation method is currently applicable for circuits with up to 10-20 inputs (depending on a particular target function) [43, 51, 37, 49, 41]. In order to reduce the time of evaluation, various techniques can be adopted:

- Only a subset of all possible input vectors is utilized. That is typical for synthesis of filters, classifiers or robot controllers. Unfortunately, the approach is not applicable for synthesis of arithmetic circuits as it does not ensure that correct responses will be obtained for those input combinations which were not used during evolution [23].
- In some cases it is sufficient to evaluate only some structural properties (not the functionality!) of candidate circuits which can be done with a reasonable time overhead. For example, because testability of a candidate circuit can be calculated in the quadratic time complexity, very large benchmark circuits with predefined testability properties (more than 1 million gates) were evolved [36].
- In case that a target system is linear, it is possible to perfectly evaluate a candidate circuit using a single input vector independently of the circuit complexity. Multiple-constant multipliers composed of adders, subtractors and shifters were evolved for a 16-bit input and tens of 16-bit outputs [48].

An obvious conclusion is that the evaluation time becomes the main bottleneck of the evolutionary approach when complex digital circuits with many inputs are evolved or optimized.

2.3 Scalability of Representation

From the viewpoint of the *scalability of representation*, the problem is that long chromosomes which are usually required to represent complex solutions imply large search spaces that are typically difficult to search. In order to evolve large designs and simultaneously keep the size of chromosome small, various techniques have been proposed, including functional-level evolution [35, 39], incremental evolution [44, 45, 43], modularization [51, 26] and their combinations [41, 12]. Despite the fact that a new field of computational development has attracted a lot of attention in this area and brought some theoretical as well as practical results [18, 29, 22, 42, 15, 17, 47, 31, 55] the problem of scalability is still an open issue.

3 Proposed Method

The goal of proposed method is to minimize the number of gates in a functionally correct combinational circuit that is typically obtained using a conventional synthesis tool. The method consists of three main steps that will be described in detail in the following sections:

1. Perform the synthesis/optimization using a conventional synthesis algorithm.
2. Convert resulting circuit to the CGP representation and use it to seed the initial population of CGP.
3. Run CGP that uses a formal verification method that will be described in Section 4 to reduce the number of gates. CGP is terminated if either the maximum allowed number of generations has been exhausted or a solution that fulfills the requirements has been discovered.

3.1 Conventional Circuit Synthesis

Combinational logic functions are commonly specified by PLA files where PLA stands for programmable logic array. The PLA file is an abbreviated truth table where all inputs are specified. However, it does not list products for which all the outputs are zero or undefined combinations. A circuit can also be represented as a netlist of gates in BLIF (Berkeley Logic Interchange Format) format. BLIF lists all interconnected combinational gates (and latches in case of sequential circuits).

Since proposed method is intended for a gate-level optimization, other steps of the circuit design process such as mapping, routing, placement and subsequent technology-specific optimizations are not considered in this paper. From conventional and routinely used synthesis methods we have chosen the SIS [40] tool (version sis1.2) which provided in most cases better results than other tools such as ABC [3] (version abc70930) or Espresso [4].

Implementations of synthesis tools support various operations with circuits, for example, it is possible to convert PLA to BLIF and vice versa. Circuits specified in BLIF can also be mapped on a chosen set of gates or look-up tables. The ABC and SIS tools are deterministic. They attempt to apply various circuit decomposition and re-synthesis techniques to transform a circuit under optimization and generate optimized netlist. We have used them with recommended (standard) setting which is represented by synthesis scripts given in Table 1. In order to improve their results we applied them on their own results iteratively as suggested in [3]. That technique will be discussed in Section 5.5.

3.2 Cartesian Genetic Programming

Cartesian Genetic Programming is a widely-used method for evolution of digital circuits [34, 32]. CGP was originally defined for gate-level evolution; however, it can easily be extended for functional level evolution [38]. In its basic version, candidate circuits are directly represented in the chromosome. The following paragraphs describe how we have used CGP in the proposed method.

3.2.1 Representation

A candidate entity (circuit) is modeled as an array of n_c (columns) $\times$ n_r (rows) of programmable nodes (gates). The number of inputs, n_i, and outputs, n_o, is fixed. Each

Table 1 Synthesis scripts for the SIS and ABC method

SIS	ABC
read PLA file script_rugged map	read PLA file script_choice map

script_rugged:
sweep; eliminate -1
simplify -m nocomp
eliminate -1
sweep; eliminate 5
simplify -m nocomp
resub -a
fx
resub -a; sweep
eliminate -1; sweep
full_simplify -m nocomp
	script_choice:
	fraig_store;
	resyn; fraig_store;
	resyn2; fraig_store;
	resyn2rs; fraig_store;
	share; fraig_store;
	fraig_restore

node input can be connected either to the output of a node placed in the previous l columns or to one of the circuit inputs. The l-back parameter, in fact, defines the level of connectivity and thus reduces/extends the search space. For example, if l=1 only neighboring columns may be connected; if $n_r = 1$ and $n_c = l$, full connectivity is enabled. Feedback is not allowed. Each node is programmed to perform one of n_a-input functions defined in the set Γ (n_f denotes $|\Gamma|$). As Figure 2 shows, while the size of chromosome is fixed, the size of phenotype is variable (i.e. some nodes are not used). Every individual is encoded using $n_c \times n_r \times (n_a + 1) + n_o$ integers.

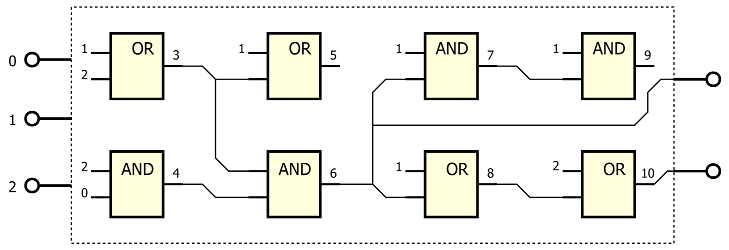


Fig. 2 Example of a candidate circuit. CGP parameters are as follows: $l = 3$, $n_c = 4$, $n_i = 3$, $n_o = 2$, $n_r = 2$, Γ = {AND (0), OR (1)}. Nodes 5,7 and 9 are not utilized. Chromosome: 1,2,**1**, 2,0,**0**, 1,3,**1**, 3,4,**0** 1,6,**0**, 1,6,**1**, 1,7,**0**, 2,8,**1**, 6, 10. The last two integers indicate the outputs of circuit. The function of a gate is typed in bold.

3.2.2 Search Algorithm

CGP operates with the population of $1+\lambda$ individuals (typically, $\lambda = 4$). The initial solution (the seed) is constructed by means of mapping of the circuit obtained from conventional synthesis and specified in the BLIF format to the CGP representation. The mapping is straightforward since the CGP representation is in fact a netlist. If the initial circuit consists of m gates, each of them possessing up to γ inputs, then

CGP will operate with parameters $n_c = m, n_r = 1, l = n_c, n_a = \gamma$. The circuit is also transformed into the conjunctive normal form in order to create a reference solution for the formal verification (see the method in Section 4 and Figure 5).

The seed together with λ offspring created using a point mutation operator form the initial population which has to be evaluated. Every new population consists of the best individual of the previous population and its λ offspring. In case when two or more individuals have received the same highest fitness score in the previous generation, one of individuals which have not served as a parent in the previous population will be selected as a new parent. This strategy is important because it contributes to ensuring the diversity of population [33].

3.2.3 Fitness function

When the objective is to minimize the number of gates the fitness value of a candidate circuit may be defined in CGP as [24]:

$$fitness = B + (n_c n_r - z) \tag{1}$$

where B is the number of correct output bits obtained as response for all possible assignments to the inputs, z denotes the number of gates utilized in a particular candidate circuit and $n_c.n_r$ is the total number of gates available. The last term $n_c n_r - z$ is considered only if the circuit behavior is perfect, i.e. $B = n_o 2^{n_i}$.

The fitness calculation carried out by the proposed method differs from equation 1. Instead of evaluating all possible assignments to the inputs, a candidate circuit is verified against a reference circuit as described in Section 4. The result of the verification algorithm is a Boolean value. If the value is negative then the fitness score is the worst-possible value. If the value is positive, the fitness value is just the number of utilized gates (assuming that the goal is to minimize here) which can easily be obtained from the CGP representation of a candidate solution.

3.2.4 Acceleration Techniques for Standard CGP

We will also utilize fitness calculation according to equation 1) in order to compare the results with the formal verification-based fitness calculation. However, two modifications are incorporated to the implementation of equation 1 to reduce the computational overhead:

(i) Because the initial population already contains a fully functional solution and the elitism is implicit for CGP, there will be at least one perfectly working solution in each population. Hence we can now consider CGP as a circuit optimizer rather than a tool for discovering new circuit implementations from a randomly generated initial population. The fitness evaluation procedure which probes every assignment to the inputs (i.e., $0 \ldots 2^{n_i} - 1$ test cases) is time consuming. In order to make the evaluation of a candidate circuit as short as possible, it is only tested whether a candidate circuit is working correctly or incorrectly. In case that a candidate circuit does not produce a correct output value for the j-th input vector, $j \in \{0 \ldots 2^{n_i} - 1\}$, during the evaluation, the remaining $2^{n_i} - j - 1$ vectors are not evaluated and the circuit gets the worst possible score (0). Experimental results show that this technique reduces the computational overhead (see Table 3), but it does not significantly contribute to solving the scalability problems. Note that this technique cannot be applied for the randomly initialized CGP because we have to know the fitness score as precisely as possible (i.e. the exact number of bits has to be calculated that can be generated by a particular candidate circuit) in order to obtain a reasonably smooth fitness landscape.

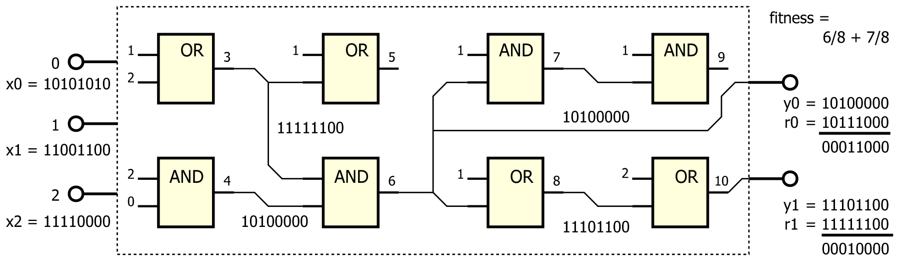


Fig. 3 Parallel simulation of a combinational circuit. The values y_0 and y_1 are the results of simulation; r_0 and r_1 are the required outputs

(ii) Parallel simulation is another technique that can be used to accelerate circuit evaluation [32,37]. The idea of parallel simulation is to utilize bitwise operators operating on multiple bits in a high-level language (such as C) to perform more than one evaluation of a gate in a single step. For example, when a combinational circuit under simulation has three inputs and it is possible to concurrently perform bitwise operations over $2^3 = 8$ bits in a simulator then the circuit can completely be simulated by applying a single 8-bit test vector at each input (see the encoding in Figure 3). In contrast, when it is impossible then eight three-bit test vectors must be applied sequentially. Current processors allow us to operate with 64 bit operands, i.e. it is possible to evaluate a truth table of a six-input circuit by applying a single 64-bit test vector at each input. Therefore, the obtained speedup is 64 against the sequential simulation. In case that the circuit has more than 6 inputs then the speedup is constant, i.e. 64.

4 Formal Verification Approach in the Fitness Function

We propose to replace the fitness calculation approach based on testing of all possible assignments to the inputs by a functional equivalence checking algorithm. In order to specify the problem, a set of Boolean functions $F = \{f_1, f_2, \ldots, f_n\}$ can be used. Let each function f_i represent Boolean function of the i-th output of a candidate circuit. Then the set F can be used to check whether a candidate solution meets the specification or not.

4.1 Functional Equivalence Checking

Determining whether two Boolean functions are functionally equivalent represents a fundamental problem in formal verification. Although the functional equivalence checking is an NP-complete problem, several approaches have been proposed so far to reduce the computational requirement for practical circuit instances.

Most of proposed techniques are based on representing a circuit by means of its canonical representation. Generally, two Boolean functions are equivalent if and only if canonical representations of their output functions are equivalent. The Reduced Ordered Binary Decision Diagrams (ROBDD) represent a widely used canonical representation in formal verification [52]. ROBDD is a directed acyclic graph that can be obtained by applying certain transformations on the ordered binary decision diagram. Determining whether two circuits represent the same Boolean function is equivalent to determining whether two ROBDDS are isomorphic. Some of methods developed to determine whether two ROBDDS are isomorphic are based on graph-based algorithms. Other methods are based on the combination of ROBDDs with the XOR operation (see Fig. 4) and checking whether the resulting ROBDD is a constant node (zero). And-or-invert graphs represent another canonic representation with similar properties. All these graph-based approaches rely on the fact, that the number of nodes in the resulting graph will be relative small, otherwise, the time of the ROBDD construction as well as the time of comparison will be enormous. In practice, these methods are rarely implemented directly without any further circuit preprocessing. The main problems are the need for high memory resources due to a huge number of BDD nodes and significant time requirements. Although many functions in practice can be represented by polynomial number of BDD nodes with respect to the number of inputs, there are functions (e.g. multipliers) that always have the number of nodes exponentially related to the number of inputs [7]. The verification of such functions still represents a challenge.

High consumption of memory resources has motivated researchers to look for alternative methods. Since the satisfiability (SAT) solvers were significantly improved during the last few years, the SAT-based equivalence checking becomes to be a promising alternative to the BDD-based checking. In this case, circuits to be checked are transformed into one Boolean formula which is satisfiable if and only if the circuits are functionally equivalent [13]. In this article we will use the SAT-based equivalence checking because: (i) combinational circuits represented by CGP can be converted to Boolean formula in linear time with respect to the number of CGP nodes, (ii) several optimization techniques specific for the evolutionary design can be applied and (iii) the SAT-based checking becomes to be a preferred method as it outperforms the BDD-based approaches.

SAT solvers assume that the equivalence checking problem is expressed using Boolean formula in conjunctive normal form (CNF). CNF formula φ consists of a conjunction of clauses denoted as ω. Each clause contains a disjunction of literals. A literal is either variable x_i or its complement $\neg x_i$. Each clause can contain up to n literals providing there exists exactly n variables.

For our purposes, the most suitable transformation of the circuit to CNF is represented by Tseitin's algorithm proposed in [46] that works as follows: Let us consider a combinational circuit C_A with k inputs that is composed of n interconnected logic gates. Without loss of generality, let us restrict the set of all possible gates to the following one-input and two-input gates: NOT, AND, OR, XOR, NAND, and NOR only. Let $y_i = \Omega(x_{i1}, x_{i2})$ denote a gate i of C_A with function Ω, output y_i and two inputs x_{i1} and x_{i2} ($1 \leq i1, i2 \leq k+n$). The Tseitin transformation is based on the fact that the CNF representation φ captures the valid assignments between the primary inputs and outputs of a given circuit. This can be expressed using a set of valid assignments for every gate. In particular, $\varphi = \omega_1 \wedge \omega_2 \wedge \cdots \wedge \omega_n$ where $\omega_i(y_i, x_{i1}, x_{i2}) = 1$ if and only if the corresponding predicate $y_i = \Omega(x_{i1}, x_{i2})$ holds true. During the transformation a new auxiliary variable is introduced for every signal of C_A. Hence CNF contains exactly $k+n$ variables and the size of the resulting CNF is linear with respect to the size of C_A. Table 2 contains the CNF representation for the gates utilized in this article.

Table 2 CNF representation of some common gates

Gate	Corresponding CNF representation
$y = \mathrm{NOT}(x_1)$	$(\neg y \vee \neg x_1) \wedge (y \vee x_1)$
$y = \mathrm{AND}(x_1, x_2)$	$(y \vee \neg x_1 \vee \neg x_2) \wedge (\neg y \vee x_1) \wedge (\neg y \vee x_2)$
$y = \mathrm{OR}(x_1, x_2)$	$(\neg y \vee x_1 \vee x_2) \wedge (y \vee \neg x_1) \wedge (y \vee \neg x_2)$
$y = \mathrm{XOR}(x_1, x_2)$	$(\neg y \vee \neg x_1 \vee \neg x_2) \wedge (\neg y \vee x_1 \vee x_2) \wedge$ $(y \vee \neg x_1 \vee x_2) \wedge (y \vee x_1 \vee \neg x_2)$
$y = \mathrm{NAND}(x_1, x_2)$	$(\neg y \vee \neg x_1 \vee \neg x_2) \wedge (y \vee x_1) \wedge (y \vee x_2)$
$y = \mathrm{NOR}(x_1, x_2)$	$(y \vee x_1 \vee x_2) \wedge (\neg y \vee \neg x_1) \wedge (\neg y \vee \neg x_2)$

In order to check whether two circuits are functionally equivalent, the following scheme is usually used. Let C_A and C_B be combinational circuits, both with k inputs denoted as $x_1 \ldots x_k$ and m outputs denoted as $y_1 \ldots y_m$ and $y'_1 \ldots y'_m$ respectively. For SAT based equivalence checking of two circuits, corresponding primary outputs y_i and y'_i are connected using the XOR-gate. This gate is denoted as a *miter*. The corresponding primary inputs are connected as well. The goal is to obtain one circuit that has only k primary inputs $x_1 \ldots x_k$ and m primary outputs $z_1 \ldots z_m$, $z_i = XOR(y_i, y'_i)$. In order to disprove the equivalence, it is necessary to identify at least one miter which evaluates to 1 for an input assignment, i.e. it is necessary to find an input assignment for which the corresponding outputs y_i and y'_i provide different values and thus $z_i = 1$. This can be done by checking one miter after another (i.e. a CNF is created and solved for each miter output separately) or by the all outputs approach (all miter outputs are connected using the m-input OR gate; thus one CNF is created and solved only). Note that both approaches are used in practice. Figure 4 shows the all output approach adopted in this article.

4.2 Proposed Fitness Function

Assume that C is a k-input/m-output circuit composed of n logic gates and the goal is to reduce the number of gates. The first step involves creating a reference solution by converting C to the corresponding CNF φ_1 using the approach described above. Let $X = \{x_1, x_2, \ldots, x_N\}$ be a set containing the variables used within φ_1 and $|X| =$

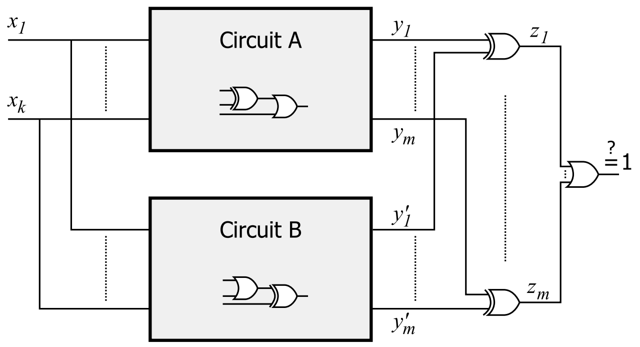
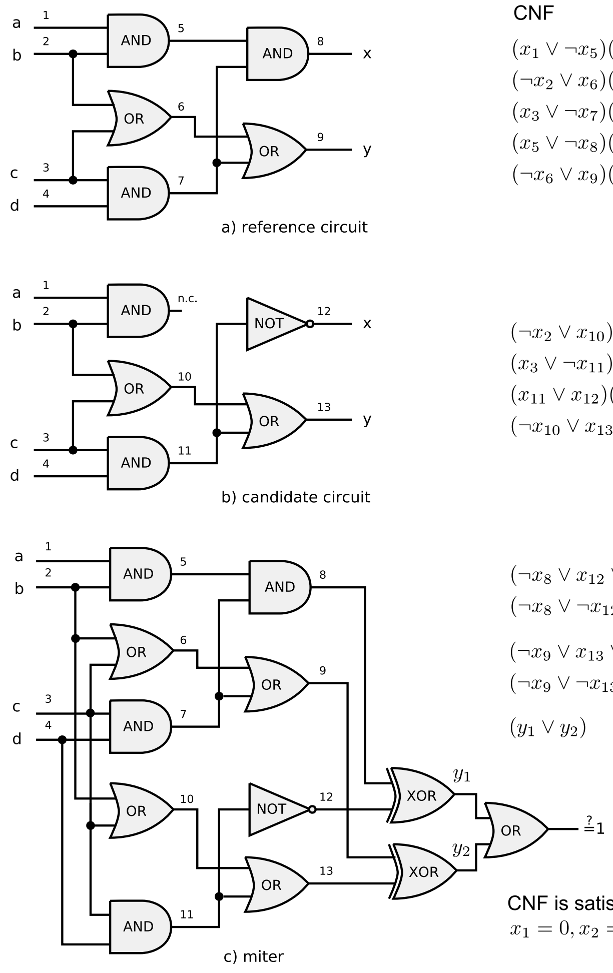


Fig. 4 Equivalence checking of two combinational circuits using the all outputs approach

$N = k + n$. The variables corresponding with the primary inputs will be denoted as $x_1, \dots, x_k$ and the auxiliary variables generated during the transformation process will be denoted as $x_{k+1}, \dots, x_{k+n}$. Let the last m variables $x_{N-m+1}, \dots, x_N$ correspond with the primary outputs of C (see Fig. 5a).

The fitness calculation consists of the following steps:

1. A new instance of the SAT solver is created and initialized with the reference circuit. This comprises creating of N new variables and submitting all clauses of φ_1 into the SAT solver.
2. A candidate solution is transformed to a list of clauses that are submitted into the SAT solver (see Fig. 5b). The transformation includes reading the CGP representation according to the indexes of the nodes. If a CGP node contributes to the phenotype, it is converted to the corresponding CNF according to Table 2, otherwise it is skipped. In particular, for each node a new variable is created and a list of corresponding CNF clauses is submitted into the SAT solver. The following input mapping is used in order to form a CNF: If an input of the node situated in row i_r and column i_c is connected to the primary input i, variable x_i is used; otherwise variable x_{N+i} is used where $i = (i_c - 1).n_r + i_r$ denotes the index of the corresponding node. Let variables corresponding with the primary outputs of a candidate solution be denoted $x_{N'-m+1}, \dots, x_{N'}$ where N' is the number of converted CGP nodes.
   Note that although it is possible to include unused gates to CNF without affecting the reasoning, it is preferred to minimize the number of clauses and variables of the resulting CNF since it can decrease the decision time.
3. Miters are created. The XOR gates are applied to each output pair which means that m new variables denoted as $y_1, \dots, y_m$ have to be created and CNFs of XOR gates $y_i = \text{XOR}(x_{N-i}, x_{N'-i})$, $i = 0 \dots m-1$ have to be submitted to the SAT solver (see Fig. 5c).
4. In order to guarantee that the resulting CNF will be satisfiable if and only if at least one miter is evaluated to 1, the outputs of the miters generated in the previous step have to be combined together. The solution is based on combining the outputs by m-input OR gate $z = \text{OR}(y_1, \dots, y_m)$. The corresponding CNF representation has the form of $(\neg z \vee x_1 \vee \dots \vee x_k) \wedge \bigwedge_{i=1}^{k} (\neg x_i \vee z)$. In order to provide a CNF



CNF

$(x_1 \vee \neg x_5)(x_2 \vee \neg x_5)(\neg x_1 \vee \neg x_2 \vee x_5)$
$(\neg x_2 \vee x_6)(\neg x_3 \vee x_6)(x_2 \vee x_3 \vee \neg x_6)$
$(x_3 \vee \neg x_7)(x_4 \vee \neg x_7)(\neg x_3 \vee \neg x_4 \vee x_7)$
$(x_5 \vee \neg x_8)(x_7 \vee \neg x_8)(\neg x_5 \vee \neg x_7 \vee x_8)$
$(\neg x_6 \vee x_9)(\neg x_7 \vee x_9)(x_6 \vee x_7 \vee \neg x_9)$

$(\neg x_2 \vee x_{10})(\neg x_3 \vee x_{10})(x_2 \vee x_3 \vee \neg x_{10})$
$(x_3 \vee \neg x_{11})(x_4 \vee \neg x_{11})(\neg x_3 \vee \neg x_4 \vee x_{11})$
$(x_{11} \vee x_{12})(\neg x_{11} \vee \neg x_{12})$
$(\neg x_{10} \vee x_{13})(\neg x_{11} \vee x_{13})(x_{10} \vee x_{11} \vee \neg x_{13})$

$(\neg x_8 \vee x_{12} \vee y_1)(x_8 \vee \neg x_{12} \vee y_1)$
$(\neg x_8 \vee \neg x_{12} \vee \neg y_1)(x_8 \vee x_{12} \vee \neg y_1)$

$(\neg x_9 \vee x_{13} \vee y_2)(x_9 \vee \neg x_{13} \vee y_2)$
$(\neg x_9 \vee \neg x_{13} \vee \neg y_2)(x_9 \vee x_{13} \vee \neg y_2)$

$(y_1 \vee y_2)$

CNF is satisfiable if
$x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0$

Fig. 5 Example of transformation of reference circuit, candidate circuit and miter to CNF

instance capable of the equivalence checking, it is necessary to append the clause (z) that implies $z = 1$, thus $(\neg z \vee y_1 \vee \cdots \vee y_k) \wedge \bigwedge_{i=1}^{k} (\neg y_i \vee z) \wedge (z) = (y_1 \vee \cdots \vee y_k)$. So, in order to finish the CNF, clause $(y_1 \vee \cdots \vee y_k)$ has to be submitted to the SAT solver (see last clause in Fig. 5c).

5. The SAT solver determines whether the submitted set of clauses is satisfiable or not. If the CNF is satisfiable, the fitness function returns 0 (the candidate circuit and the reference circuit are not equivalent); otherwise the number of utilized gates is returned.

4.3 Time of Candidate Circuit Evaluation

In order to compare the time of evaluation for the common fitness function (eq. 1) and the proposed SAT based fitness function, the parity circuit optimization problem has been chosen. The design of a parity circuit consisting of AND, OR and NOT gates only is considered as a standard benchmark problem for genetic programming [28]. The relevant CGP parameters are as follows: $\lambda = 4$, $\Gamma = \{\text{AND}, \text{OR}, \text{NOT}, \text{Identity}\}$, $l = N_g$, $n_c = N_g$ and $n_r = 1$ where N_g is the number of gates of the reference circuit. One gene of the chromosome undergoes the mutation only. The CGP implementation uses the parallel evaluation described in Section 3.2.4. The initial circuit (seed) has been obtained by mapping a parity circuit consisting of XOR gates (parity tree) to the 2-inputs gates using ABC. Table 3 gives the mean evaluation time (out of 100 runs) for three fitness functions – the standard fitness function of CGP (t_{cgp}), the optimized and accelerated evaluation (t_{ocgp}, see Section 3.2.4) and the SAT-based method (t_{sat}). Last two columns contain the achieved speedup of proposed approach against the common and accelerated CGP. The experiments were carried out on Intel Core 2 Duo 2.26 GHz processor. For $n_i \geq 26$ only extrapolated values are given as running the experiments is not practical. The MiniSAT 2 (version 070721) has been used as a SAT solver [9] because it can be effectively embedded into a custom application.

Table 3 The mean evaluation time for the standard fitness function of CGP t_{cgp}, CGP with optimized and accelerated evaluation t_{ocgp} and the SAT-based CGP t_{sat}. Symbol '*' denotes extrapolated values.

n_i	seed [gates]	t_{cgp} [ms]	t_{ocgp} [ms]	t_{sat} [ms]	t_{cgp}:t_{sat} speedup	t_{ocgp}:t_{sat} speedup
12	69	0.13	0.04	0.348	0.3	0.1
14	87	0.54	0.16	0.438	1.2	0.4
16	103	2.54	0.27	0.531	4.8	0.5
18	115	11.45	1.20	0.722	15.9	1.7
20	125	51.44	5.17	0.776	66.3	6.7
22	135	220	25.11	0.804	273.6	31.2
24	145	1328	139	0.903	1471	153.9
26	171	5962*	626*	1.028	5799	608
28	181	26748*	2820*	1.195	22383	2359
30	199	119996*	12703*	1.211	99088	10489
32	215	538327*	57207*	1.348	399352	42438

Since t_{cgp} increases exponentially with the increasing number of circuit inputs, the standard CGP approach provides a reasonable evaluation time for parity circuits that contain up to 22 inputs. The optimized evaluation is applicable for up to 24 inputs. In case of the SAT-based approach the evaluation time is almost similar independently of the number of candidate circuit inputs.

4.4 CGP-Specific Performance Improvement Techniques

Although the system can be used directly as it was proposed in the previous section, we have introduced some techniques allowing the SAT solver even to increase the performance.

The speed of the SAT-based equivalence checking depends mainly on the number of paths that have to be traversed in order to prove or disprove the satisfiability. The number of paths increases with the increasing number of outputs to be compared, i.e. more outputs to be compared more time the SAT-solver needs for the decision. In order to simplify the decision problem and increase the performance, CNF reduction based on finding structural similarities were proposed in literature.

In our case we can apply a very elegant and simple solution. Since every fitness evaluation is preceded by a mutation, a list of nodes that are different for the parent and its offspring can be calculated. This list can be used to determine the set of outputs that have to be compared with the reference circuit and only these outputs are included into CNF. This can be achieved by omitting the unnecessary outputs during the miter creation phase.

In order to decrease the number of variables as well as the number of clauses in NOT-intensive circuits, the following approach is proposed. Let $y_i = NOT(x_i)$, then the NOT gate can be subsumed to CNF of every gate that is connected directly to output y_i. Using literal $\neg x_i$ instead of y_i and literal x_i instead of $\neg y_i$ respectively solves the problem.

Note that proposed approach can easily be combined with other methods designed to speedup the SAT-based equivalence checking, e.g. circuit preprocessing, incremental approach or improved CNF transformation [6,8,2,50].

Table 4 The mean time needed to evaluate a candidate solution for plain and optimized SAT-based fitness method

circuit	n_i	n_o	seed [gates]	t_{sat} [ms]	t_{osat} [ms]	$t_{sat} : t_{osat}$ speedup
apex1	45	45	1408	49.80	15.52	3.21
apex2	39	3	235	3.54	2.52	1.40
apex3	54	50	1407	34.56	13.93	2.48
apex5	117	87	784	17.45	5.07	3.44

In order to evaluate the impact of proposed improvements, four complex circuits have been selected for experiments from the LGSynth93 benchmark set. This benchmark set includes nontrivial circuits specified in BLIF format that are traditionally used by engineers to evaluate quality of synthesis algorithms. The benchmark circuits were mapped to 2-input gates using SIS. Parameters of selected circuits as well as obtained results are summarized in Table 4. It can be seen that even if the circuits exhibit higher level of complexity in comparison with parity circuits, the average time needed to perform the fitness evaluation remains still reasonable. Note that the same experimental setup mentioned in Section 4.3 has been utilized. Obtained results show that the average time needed to evaluate a candidate solution has been reduced three times in average by means of applying the proposed steps during the transformation of a candidate solution to corresponding CNF.

5 Results

This section surveys experiments performed to further evaluate the proposed method. In particular, the effect of population sizing, CGP grid sizing, mutation rate and time allowed to evolution are analyzed for benchmark circuits. In all experiments we used the optimized SAT-based fitness function.

5.1 Population Size

Table 5 surveys the best (minimum) and mean number of gates obtained for $\lambda = 1$ and $\lambda = 4$ out of 100 independent runs. The number of evaluations was limited to 400,000 which corresponds with 100,000 generations for ES(1+4) and 400,000 generations for ES(1+1). The mutation operator modified 1 gene of the chromosome, $l = n_c$ and $\Gamma = \{\text{Identity}, \text{AND}, \text{OR}, \text{NOT}, \text{XOR}, \text{NAND}, \text{NOR}\}$. The best values as well as mean values indicate that ES(1+1) performs better than ES(1+4) which corresponds with our intuitive assumption of very rugged fitness landscape.

Table 5 The best and mean number of gates for different population sizing.

			seed	ES 1+4		ES 1+1	
circuit	n_i	n_o	[gates]	best	mean	best	mean
apex1	45	45	1408	1240	1267	1201	1255
apex2	39	3	235	138	155	132	146
apex3	54	50	1407	1336	1350	1331	1347
apex5	117	87	784	736	746	730	743
mean			959	863	880	849	873

5.2 Mutation Rate and CGP Grid Size

Table 6 gives the best (minimum) and mean number of gates obtained for different mutation rates (1, 2, 5, 10, 15 genes) and CGP grid setting ($n_c \times 1$ versus $n_c \times n_r^{(i)}$). It will be seen below that the number of rows $n_r^{(i)}$ is variable. The number of evaluations was limited to 400,000 and results were calculated out of 100 independent runs of ES(1+1). Table 6 also includes the mean number of bits that were included to create miters and the mean time of a candidate circuit evaluation.

The best results were obtained for the lowest mutation rate. The higher mutation rate the higher mean number of gates in the final circuit. While the mean number of miters grows with increasing of the mutation rate, the mean evaluation time is reduced. This phenomenon can be explained by the fact that higher mutation rate implies more changes that are performed in circuits and thus more miters have to be considered. On the other hand, because of many (mostly harmful) changes in a circuit it is easier to disprove the equivalence for SAT solver and so reduce the evaluation time.

The settings $n_c \times 1$ or $n_c \times n_r$ do not have a significant impact on the resulting number of gates on average. Recall that the values of n_c and n_r are given by the circuit topology which is created by the SIS tool. The number of rows ($n_r^{(i)}$) is considered as variable for a given circuit in order to represent the circuit optimally. For example, the 1408 gates of the apex1 benchmark is mapped on the array of 19x189 nodes; however only 1, 5, 7, 14, 17, 26, 43, 57, 84, 117, 142, 177, 189, 187, 139, 89, 51, 27, 40 gates are utilized in columns $i = 1 \dots 19$. The advantage of using $n_r > 1$ is that delay of the circuit is implicitly controlled to be below a given maximum value.

Table 6 The best (minimum) and mean number of gates, the mean number of miters and the mean evaluation time for different mutation rates (1–20 genes) and CGP grid setting ($n_c \times 1$ versus $n_c \times n_r^{(i)}$)

	mutated genes ($n_c \times 1$)						mutated genes ($n_c \times n_r^{(i)}$)					
	1	2	5	10	15	20	1	2	5	10	15	20
	apex1 - 1408x1						apex1 - 19x189					
best	1240	1290	1351	1377	1382	1393	1260	1290	1351	1379	1385	1392
mean	1269	1313	1367	1387	1396	1399	1287	1326	1369	1390	1395	1399
mean (miters)	3.8	5	8.2	12.3	15.3	17.6	3.6	4.8	8	12.2	15.2	17.6
mean t_{osat} [ms]	15.8	11.2	8.8	7.7	7.7	7.2	11.8	11.5	9.7	7.8	7.9	6.7
	apex2 - 235x1						apex2 - 22x23					
best	164	159	166	181	195	200	165	167	172	186	194	201
mean	170	172	181	195	203	209	171	174	182	195	205	209
mean (miters)	1.8	2.1	2.5	2.7	2.8	2.9	1.8	2	2.5	2.7	2.8	2.9
mean t_{osat} [ms]	1.7	1.7	1.4	1.2	1.1	0.9	1.7	1.6	1.4	1.2	1.0	1.0
	apex3 - 1407x1						apex3 - 24x193					
best	1341	1358	1383	1392	1395	1396	1345	1362	1383	1392	1396	1398
mean	1354	1369	1389	1397	1399	1400	1357	1372	1390	1397	1400	1401
mean (miters)	2.6	3.6	6.2	9.4	12	14	2.6	3.5	6.1	9.4	11.9	14.1
mean t_{osat} [ms]	10.5	10.1	9.0	11.4	8.3	8.0	10.5	10.3	9.8	8.8	9.8	7.2
	apex5 - 784x1						apex5 - 34x117					
best	740	741	755	765	767	774	741	750	757	767	768	771
mean	748	753	764	773	775	779	751	757	766	773	775	777
mean (miters)	4.6	6.4	11.1	18.1	23.7	28.4	4.6	6.4	11.2	18.1	23.7	28.4
mean t_{osat} [ms]	3.3	3.1	3.0	2.9	2.9	2.7	3.1	3.2	2.9	3.0	3.2	2.9

5.3 Parity Benchmarks

In Section 4.3 we compared the evaluation time of the standard fitness function and the SAT-based fitness function in the task of parity circuits optimization. Table 7 shows concrete results - the minimum number of gates that were obtained for 12–38 input parity circuits by running the proposed method for 3, 6, 9 and 12 hours on a 2.4 GHz processor. The results are averaged from 100 independent runs of CGP with the following setting: ES(1+1), 1 mutated gene/chromosome, $\Gamma = \{\text{Identity}, \text{AND}, \text{OR}, \text{NOT}\}$, and CGP array of $n_c \times 1$ nodes where n_c is the number of gates in the seed – the initial circuit created by SIS. Column TG denotes the number of gates of the optimal solution which is known in this case. It can be calculated as $4w$ where w is the number of XOR gates in the optimized parity tree and 4 denotes the number of gates from Γ needed to form a single XOR gate.

We can observe that the proposed method provides an optimal solution for $n_i \leq 20$ and almost optimal solution for larger problem instances. Last column shows that the proposed method improves the original solution of SIS by 28–42 %.

Table 7 The minimum number of gates that were obtained for parity circuits by running the proposed method for 3, 6, 9 and 12 hours. TG gives the optimum solution.

	seed	run-time				TG	relative
n_i	[gates]	3h	6h	9h	12h	[gates]	improv.
12	69	45	44	44	44	44	36 %
14	87	54	53	52	52	52	40 %
16	103	64	61	60	60	60	42 %
18	115	74	70	69	69	68	40 %
20	125	82	79	77	76	76	39 %
22	135	95	91	88	87	84	36 %
24	145	110	101	98	96	92	34 %
26	171	134	120	114	111	100	35 %
28	181	151	132	124	121	108	33 %
30	199	165	140	132	129	116	35 %
32	215	186	169	159	143	124	33 %
34	227	214	187	172	160	132	30 %
36	237	220	192	168	162	140	32 %
38	247	235	219	193	177	148	28 %

5.4 LGSynth93 Benchmarks

Table 8 shows the minimum and mean number of gates that were obtained for real-world benchmark circuits of the LGSynth93 suite (we have selected those with more than 20 inputs) by running the proposed method for 3, 6, 9 and 12 hours on a 2.4 GHz processor. The results are averaged from 100 independent runs of CGP with the following setting: ES(1+1), 1 mutated gate/chromosome, $\Gamma = \{\text{Identity}, \text{AND}, \text{OR}, \text{NOT}, \text{XOR}, \text{NAND}, \text{NOR}\}$, and CGP array of $n_c \times 1$ nodes where n_c is the number of gates in the seed circuit. The initial circuit was obtained by converting the PLA files of LGSynth93 circuits to the 2-input gates of Γ and optimizing them by SIS. Last column shows that the proposed method improves the original solutions obtained from SIS by 22–58%.

5.5 Seeding the Initial Population

In order to investigate the role of seeding of the initial population we have used two seeds obtained after 1 and 1000 iterations of the SIS script. Figure 6 shows that con-

Table 8 The minimum (even rows) and mean number (odd rows) of gates for LGSynth93 circuits obtained from the proposed method after 3, 6, 9 and 12 hours.

circuit	n_i	n_o	seed [gates]	run-time 3h	6h	9h	12h	relative improv.
apex1	45	45	1408	1179	1083	1026	990	30 %
				1230	1108	1042	1001	29 %
apex2	39	3	235	104	101	99	98	58 %
				119	102	100	98	58 %
apex3	54	50	1407	1280	1223	1189	1167	17 %
				1333	1240	1202	1175	16 %
apex5	117	87	784	675	649	640	633	19 %
				692	661	644	636	19 %
cordic	23	2	67	32	32	32	32	52 %
				33	32	32	32	52 %
cps	24	109	1128	870	788	737	698	38 %
				909	806	757	713	37 %
duke	22	29	430	286	274	270	268	38 %
				296	279	272	269	37 %
e64	65	65	192	133	130	129	129	33 %
				139	131	129	129	33 %
ex4p	128	28	500	404	399	396	394	21 %
				414	401	397	395	21 %
misex2	25	18	111	76	73	72	70	37 %
				82	74	72	71	36 %
vg2	25	8	95	79	75	74	74	22 %
				83	77	74	74	22 %

vergence curves for two selected benchmark circuits - apex1 (the largest one) and ex4p (the highest number of inputs) - are very similar for those seeds. We can also observe how the progress of evolution is influenced by restarting CGP (every 3 hours; using the best solution out of 100 independent runs) which can be also considered as a new seeding. Figure 6 shows that repeating the synthesis scripts (SIS and ABC are compared) quickly lead to a small reduction of the circuit size; however, no further improvements have been observed in next 1 hour.

6 Discussion

Applying the SAT solver in the fitness function allowed us to significantly reduce the computational requirements of the fitness function for such combinational circuit optimization problems for which a fully functional initial solution exists before the optimization is started. In this category of problems, we were able to optimize much larger circuit instances than standard CGP. Furthermore, we reduced the number of gates in solutions that can be delivered by conventional synthesis methods. However, proposed method requires significantly more computational time in comparison to conventional synthesis tools.

Although the results for LGSynth93 benchmarks are very encouraging, the SAT-based combinational equivalence checking can definitely perform unsatisfactory for some problem instances, for example for multipliers where the number of paths traversed by the SAT solver grows enormously with the increasing number of inputs.

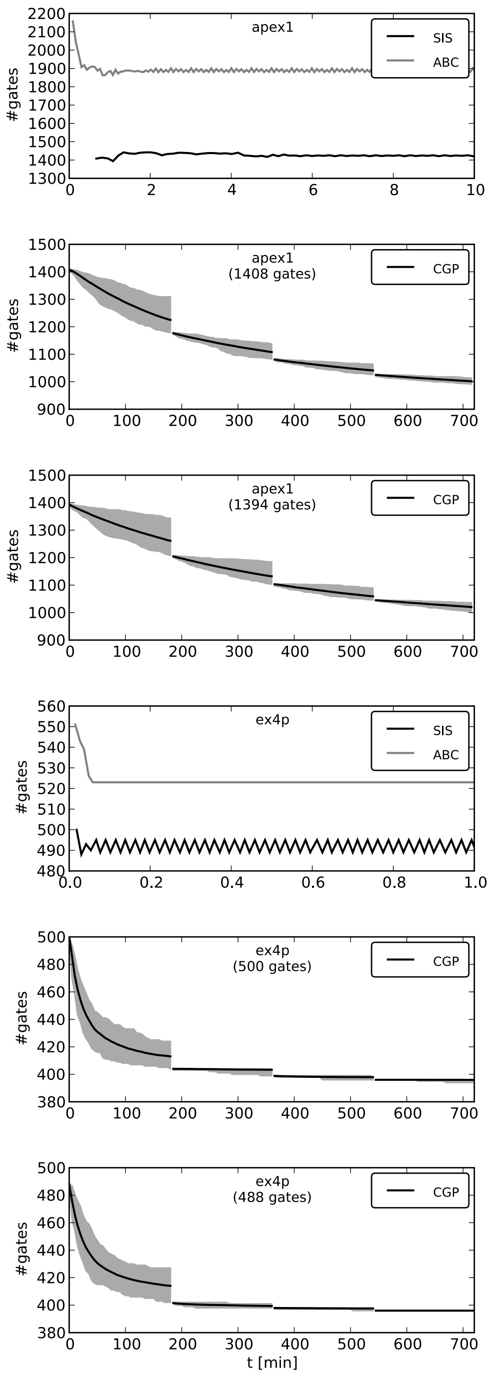


Fig. 6 Convergence curves for the apex1 and ex4p benchmarks. The mean, minimum and maximum number of gates from 100 independent runs of CGP when seeded using the result of the 1st iteration and the best result out of 1000 iterations of the SIS tool. ABC and SIS were repeated until stable results observed.

In order to improve performance of the SAT solver in this particular case, various techniques have been proposed to reduce the equivalence checking time [1,2]. The proposed method is assumed to be able to handle large-scale multipliers optimization if more advanced version of SAT solver is utilized. Other techniques exist that can be employed to improve the proposed fitness function, e.g. CNF preprocessing, BDD-based checking, hierarchical equivalence checking etc. [25].

7 Conclusions

We demonstrated that some applications of evolvable hardware could benefit from formal verification techniques. The main advantage of our method is that the time of evaluation can significantly be reduced in comparison to the standard fitness function in cases when a fully functional solution exists before optimization is started. Consequently, we demonstrated that the circuit post-synthesis optimization conducted by CGP is applicable on complex digital circuits. CGP reduced the number of gates for the LGSynth93 benchmark circuits by 37.8% on average with respect to the SIS tool. Future research will be oriented towards improving the formal verification module by using more sophisticated verification algorithms and applying the proposed method in various domains, including software evolution, developmental CGP and numerous real-world evolvable hardware problems.

Acknowledgements This work was partially supported by the Czech Science Foundation under projects *Natural Computing on Unconventional Platforms* P103/10/1517 and *Mathematical and Engineering Approaches to Developing Reliable and Secure Concurrent and Distributed Computer Systems* GD102/09/H042 and by the research programme *Security-Oriented Research in Information Technology* MSM 0021630528.

References

1. Andrade, F.V., Oliveira, M.C.M., Fernandes, A.O., Coelho, C.J.N.: Sat-based equivalence checking based on circuit partitioning and special approaches for conflict clause reuse. Design and Diagnostics of Electronic Circuits and Systems pp. 1–6 (2007)
2. Andrade, F.V., Silva, L.M., Fernandes, A.O.: Improving SAT-based combinational equivalence checking through circuit preprocessing. In: 26th International Conference on Computer Design, ICCD 2008, pp. 40–45 (2008)
3. Berkley Logic Synthesis and Verification Group: ABC: A System for Sequential Synthesis and verification. URL http://www.eecs.berkeley.edu/˜ alanmi/abc/
4. Brayton, R.K., Hachtel, G.D., McMullen, C.T., Sangiovanni-Vincentelli, A.L.: Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers, Boston, MA, USA (1984)
5. Cong, J., Minkovich, K.: Optimality Study of Logic Synthesis for LUT-Based FPGAs. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems **26**(2), 230–239 (2007)
6. Disch, S., Schollm, C.: Combinational equivalence checking using incremental SAT solving, output ordering, and resets. Asia and South Pacific Design Automation Conference pp. 938–943 (2007)
7. Ebendt, R., Fey, G., Drechsler, R.: Advanced BDD Optimization. Springer (2000)
8. Een, N., Mishchenko, A., Sorensson, N.: Applying logic synthesis for speeding up SAT. Lecture notes in computer science p. 272 (2007)
9. Een, N., Sorensson, N.: MiniSAT. URL http://minisat.se
10. Fiser, P., Schmidt, J.: Small but nasty logic synthesis examples. In: Proc. 8th Int. Workshop on Boolean Problems, pp. 183–190 (2008)

11. de Garis, H.: Evolvable Hardware – Genetic Programming of a Darwin Machine. In: International Conference on Artificial Neural Networks and Genetic Algorithms ICANNGA'93. Innsbruck, Austria (1993)
12. Glette, K., Torresen, J., Yasunaga, M.: An online EHW pattern recognition system applied to face image recognition. In: Applications of Evolutinary Computing, EvoWorkshops 2007, *LNCS*, vol. 4448, pp. 271–280. Springer (2007)
13. Goldberg, E., Prasad, M., Brayton, R.: Using SAT for combinational equivalence checking. In: DATE '01: Proceedings of the conference on Design, automation and test in Europe, pp. 114–121. IEEE Press, Piscataway, NJ, USA (2001)
14. Gordon, T.G.H., Bentley, P.J.: Evolving hardware. In: Zomaya, A.Y. (ed.) Handbook of Nature-Inspired and Innovative Computing, pp. 387–432. Springer (2006)
15. Gordon, T.G.W., Bentley, P.J.: Towards development in evolvable hardware. In: Proc. of the 2002 NASA/DoD Conference on Evolvable Hardware, pp. 241–250. IEEE Computer Society Press, Washington D.C., US (2002)
16. Greenwood, G., Tyrrell, A.M.: Introduction to Evolvable Hardware. IEEE Press (2007)
17. Haddow, P.C., Tufte, G., van Remortel, P.: Shrinking the genotype: L-systems for EHW? In: Proc. of the 4th Int. Conference on Evolvable Systems: From Biology to Hardware, *LNCS*, vol. 2210, pp. 128–139. Springer (2001)
18. Harding, S., Miller, J.F., Banzhaf, W.: Self modifying cartesian genetic programming: Parity. In: 2009 IEEE Congress on Evolutionary Computation, pp. 285–292. IEEE Press (2009)
19. Higuchi, T., Iwata, M., Keymeulen, D., Sakanashi, H., Murakawa, M., Kajitani, I., Takahashi, E., Toda, K., Salami, M., Kajihara, N., Otsu., N.: Real-World Applications of Analog and Digital Evolvable Hardware. IEEE Transactions on Evolutionary Computation **3**(3), 220–235 (1999)
20. Higuchi, T., Liu, Y., Yao, X.: Evolvable Hardware. Springer (2006)
21. Higuchi, T., Niwa, T., Tanaka, T., Iba, H., de Garis, H., Furuya, T.: Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine. In: Proc. of the 2nd International Conference on Simulated Adaptive Behaviour, pp. 417–424. MIT Press (1993)
22. Hornby, G., Globus, A., Linden, D., Lohn, J.: Automated Antenna Design with Evolutionary Algorithms. In: Proc. 2006 AIAA Space Conference, p. 8. AIAA, San Jose, CA (2006)
23. Imamura, K., Foster, J.A., Krings, A.W.: The test vector problem and limitations to evolving digital circuits. In: Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware, pp. 75–79. IEEE Computer Society Press (2000)
24. Kalganova, T., Miller, J.F.: Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In: The First NASA/DoD Workshop on Evolvable Hardware, pp. 54–63. IEEE Computer Society, Pasadena, California (1999)
25. Katebi, H., Markov, I.L.: Large-scale boolean matching. In: Design, Automation and Test in Europe, DATE 2010, pp. 771–776. IEEE (2010)
26. Kaufmann, P., Platzner, M.: Advanced techniques for the creation and propagation of modules in cartesian genetic programming. In: Proc. of Genetic and Evolutionary Computation Conference, GECCO 2008, pp. 1219–1226. ACM (2008)
27. Keymeulen, D., Durantez, M., Konaka, K., Kuniyoshi, Y., Higuchi, T.: An evolutionary robot navigation system using a gate-level evolvable hardware. In: Higuchi, T., Iwata, M., Liu, W. (eds.) Proc. of the 1st International Conference on Evolvable Systems: From Biology to Hardware ICES'96, *LNCS*, vol. 1259, pp. 195–209. Springer, Tsukuba, Japan (1997)
28. Koza, J.R.: Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge, MA (1994)
29. Koza, J.R., Bennett, F.H., Andre, D., Keane, M.A.: Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann Publishers, San Francisco, CA (1999)
30. Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers (2003)
31. Mange, D., Sipper, M., Stauffer, A., Tempesti, G.: Towards Robust Integrated Circuits: The Embryonics Approach. Proceedings of IEEE **88**(4), 516–541 (2000)
32. Miller, J.F., Job, D., Vassilev, V.K.: Principles in the Evolutionary Design of Digital Circuits – Part I. Genetic Programming and Evolvable Machines **1**(1), 8–35 (2000)
33. Miller, J.F., Smith, S.L.: Redundancy and Computational Efficiency in Cartesian Genetic Programming. IEEE Transactions on Evolutionary Computation **10**(2), 167–174 (2006)
34. Miller, J.F., Thomson, P.: Cartesian Genetic Programming. In: Proc. of the 3rd European Conference on Genetic Programming EuroGP2000, *LNCS*, vol. 1802, pp. 121–132. Springer (2000)

35. Murakawa, M., Yoshizawa, S., Kajitani, I., Furuya, T., Iwata, M., Higuchi, T.: Evolvable Hardware at Function Level. In: Parallel Problem Solving from Nature PPSN IV, *LNCS*, vol. 1141, pp. 62–71. Springer (1996)
36. Pecenka, T., Sekanina, L., Kotasek, Z.: Evolution of Synthetic RTL Benchmark Circuits with Predefined Testability. ACM Transactions on Design Automation of Electronic Systems **13**(3), 1–21 (2008)
37. Poli, R., Page, J.: Solving high-order boolean parity problems with smooth uniform crossover, submachine code gp and demes. Genetic Programming and Evolvable Machines **1**(1–2), 37–56 (2000)
38. Sekanina, L.: Image Filter Design with Evolvable Hardware. In: Applications of Evolutionary Computing – Proc. of the 4th Workshop on Evolutionary Computation in Image Analysis and Signal Processing EvoIASP'02, *LNCS*, vol. 2279, pp. 255–266. Springer Verlag, Kinsale, Ireland (2002)
39. Sekanina, L.: Evolvable Components: From Theory to Hardware Implementations. Natural Computing Series, Springer Verlag (2004)
40. Sentovich, E.M., Singh, K.J., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P.R., Brayton, R.K., Sangiovanni-vincentelli, A.: Sis: A system for sequential circuit synthesis. Tech. rep., University California, Berkeley (1992)
41. Shanthi, A.P., Parthasarathi, R.: Practical and scalable evolution of digital circuits. Applied Soft Computing **9**(2), 618–624 (2009)
42. Stanley, K.O., Miikkulainen, R.: A taxonomy for artificial embryogeny. Artificial Life **9**, 93–130 (2003)
43. Stomeo, E., Kalganova, T., Lambert, C.: Generalized disjunction decomposition for evolvable hardware. IEEE Transaction Systems, Man and Cybernetics, Part B **36**(5), 1024–1043 (2006)
44. Torresen, J.: A Divide-and-Conquer Approach to Evolvable Hardware. In: Sipper, M., Mange, D., Perez-Uribe, A. (eds.) Proc. of the 2nd International Conference on Evolvable Systems: From Biology to Hardware ICES'98, *LNCS*, vol. 1478, pp. 57–65. Springer, Lausanne, Switzerland (1998)
45. Torresen, J.: A scalable approach to evolvable hardware. Genetic Programming and Evolvable Machines **3**(3), 259–282 (2002)
46. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Studies in Constructive Mathematics and Mathematical Logic, Part II, pp. 115–125 (1968)
47. Tufte, G., Haddow, P.C.: Towards development on a silicon-based cellular computing machine. Natural Computing **4**(4), 387–416 (2005)
48. Vasicek, Z., Zadnik, M., Sekanina, L., Tobola, J.: On evolutionary synthesis of linear transforms in FPGA. In: Proc. of the 8th Conference on Evolvable Systems: From Biology to Hardware, *LNCS*, vol. 5216, pp. 141–152. Springer Verlag (2008)
49. Vassilev, V., Job, D., Miller, J.F.: 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, Los Alamitos, CA, USA (2000)
50. Velev, M.N.: Efficient translation of boolean formulas to CNF in formal verification of microprocessors. Asia and South Pacific Design Automation Conference pp. 310–315 (2004)
51. Walker, J.A., Miller, J.F.: The Automatic Acquisition, Evolution and Re-use of Modules in Cartesian Genetic Programming. IEEE Transactions on Evolutionary Computation **12**(4), 397–417 (2008)
52. Yanushkevich, S., Miller, D.M., Shmerko, V.P., Stankovic, R.S.: Decision Diagram Techniques for Micro- and Nanoelectronic Design Handbook. CRC (2006)
53. Yao, X., Higuchi, T.: Promises and Challenges of Evolvable Hardware. IEEE Transactions on Systems, Man, and Cybernetics – Part C **29**(1), 87–97 (1999)
54. 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)
55. Zhan, S., Miller, J.F., Tyrrell, A.M.: A developmental gene regulation network for constructing electronic circuits. In: Proc. of the 8th Int. Conference on Evolvable Systems: ¿From Biology to Hardware, *LNCS*, vol. 5216, pp. 177–188. Springer Verlag, Berlin (2008)