How to Evolve Complex Combinational Circuits From Scratch?

Zdenek Vasicek and Lukas Sekanina
Brno University of Technology, Faculty of Information Technology
Brno, Czech Republic
Email: vasicek@fit.vutbr.cz, sekanina@fit.vutbr.cz

***Abstract*—One of the serious criticisms of the evolutionary circuit design method is that it is not suitable for the design of complex large circuits. This problem is especially visible in the evolutionary design of combinational circuits, such as arithmetic circuits, in which a perfect response is requested for every possible combination of inputs. This paper deals with a new method which enables us to evolve complex circuits from a randomly seeded initial population and without providing any information about the circuit structure to the evolutionary algorithm. The proposed solution is based on an advanced approach to the evaluation of candidate circuits. Every candidate circuit is transformed to a corresponding binary decision diagram (BDD) and its functional similarity is determined against the specification given as another BDD. The fitness value is the Hamming distance between the output vectors of functions represented by the two BDDs. It is shown in the paper that the BDD-based evaluation procedure can be performed much faster than evaluating all possible assignments to the inputs. It also significantly increases the success rate of the evolutionary design process. The method is evaluated using selected benchmark circuits from the LGSynth91 set. For example, a correct implementation was evolved for a 28-input frg1 circuit. The evolved circuit contains less gates (a 57% reduction was obtained) than the result of a conventional optimization conducted by ABC.**

I. Introduction

Evolutionary design is an approach which could provide us with new and efficient implementations of digital circuits. We will only deal with combinational circuits in this paper. Unfortunately, the evolutionary design shows several limitations where the most serious ones are that: (a) the evaluation time of a candidate circuit grows exponentially with the number of inputs; and (b) corresponding search spaces are large and complex.

Various techniques have been proposed to reduce the search space [1]. The enormous evaluation time can partially be reduced by using advanced fitness calculation techniques in the case that the goal is to minimize the number of gates (or delay) of an already functional, but unoptimized circuit [2]. However, it is unknown how to evolve complex combinational circuits (with more than about 15 inputs) from scratch.

This paper deals with an efficient method capable of reducing the evaluation time. The goal is to evolve complex circuits (tens of inputs, thousands of gates) from a randomly seeded initial population and without providing any information about the circuit structure to the search method. Problems related to constructing of efficient circuit representations and efficient search algorithms are left untouched in this paper.

The proposed solution is based on a new fitness function transforming every candidate circuit to a corresponding *binary decision diagram* (BDD), functional equivalence checking against the specification given as another BDD and measuring a similarity between these BDDs if they are not functionally equivalent. The proposed fitness function is embedded into *Cartesian genetic programming* (CGP), which is a well-established method for digital circuit evolution [3]. The method is evaluated using selected benchmark circuits from the LGSynth91 set.

The rest of the paper is organized as follows. Section II surveys the state of the art in the area of evolvable hardware and determines the domain of our contribution. The principles and variants of binary decisions diagrams are presented in Section III. The proposed method based on embedding BDD into CGP is described in Section IV. Section V summarizes the obtained results and analyzes advantages and disadvantages of the proposed method. Conclusions are given in Section VI.

II. State of the Art

This section briefly surveys the main subareas developed within evolvable hardware and identifies the domain of evolvable hardware which is relevant for this paper.

A. Subareas of Evolvable Hardware

Evolvable hardware is usually understood as an approach which utilizes evolutionary algorithms (and other bio-inspired algorithms) in order to automatically design, optimize, adapt and/or repair various types of hardware [4]. Two main subareas of evolvable hardware are *evolutionary hardware design* and *adaptive hardware* [1]. The former one deals with the scenario in which the evolutionary algorithm is used only in the design phase of hardware. Its goal is to increase the level of design automation and produce new designs automatically. Candidate solutions are usually evaluated using simulators. A typical evolved solution would exhibit a better quality with respect to existing designs of the same category. For example, the solution would occupy a smaller area on a chip, compute faster, provide better precision, reduce power consumption, increase reliability etc. In the case of adaptive hardware, the role of bio-inspired methods is to ensure the autonomous adaptation and/or repair which partly involves the concept of evolutionary hardware design. Candidate solutions are evaluated in situ, i.e. in reconfigurable devices.

We will only deal with the evolutionary design of digital circuits in this paper. For purposes of this paper, the applications of digital circuit evolution can be divided into two main classes which we will call the approximate design and the accurate design.

In the case of *approximate design* it is sufficient to evolve a circuit responding correctly for a reasonable subset of all possible input vectors. The problem is that specifications are in principle incomplete since a correct output is not explicitly defined for every input vector. The design of filters, classifiers, predictors or hash functions falls into this class. The fitness value is usually calculated on the basis of circuit responses obtained for a carefully chosen training set, a subset of all possible input vectors. The behavior of evolved solution has to be validated using a test set at the end of evolution, i.e. using some vectors unseen during the evolution.

In the case of *accurate design*, the goal is to obtain a circuit responding *perfectly* for all possible assignments to the inputs, i.e. no error is accepted. The evolution of arithmetic circuits is a typical example of this class. Let us further focus on the evolutionary design of accurately working gate-level combinational circuits in the rest of the paper.

B. Evolutionary Design of Combinational Circuits

The most common approach to the evolutionary design of combinational circuits, which one can find in the literature (e.g. [3]), is as follows: Requested circuit behavior is provided in the form of truth table together with a set of available gates and constraints. A suitable evolutionary algorithm (EA) is then chosen and applied with the aim of discovering a fully functional solution in the first phase and optimizing additional criteria such as the number of gates, area or power consumption in the second phase. A candidate circuit is evaluated using a circuit simulator by applying all possible assignments to the inputs and comparing the obtained outputs with desired values. It is important that the initial population is randomly seeded as the approach is called the evolutionary design and it is expected that a solution is discovered from "nothing". However, the aforementioned approach suffers from various limitations:

1) The circuit evaluation time grows exponentially with the number of inputs. Hence the method is only applicable to the evolution of relatively small circuits.
2) If a direct gate-level encoding is employed, complex circuits are represented by long chromosomes, which usually lead to complex and difficult search spaces.
3) Obtaining a fully functional solution from a randomly seeded population consumes a considerable time because the approach exploits the "generate and test" principle and no additional knowledge about the problem is used.
4) Because the specification is given in the form of truth table, it is impossible to specify complex circuits in practice.

The problems 1) – 3) are known as the scalability problems of evolutionary circuit design, often referred to as the scalability of evaluation and the scalability of representation [5]. Several methods have been proposed to eliminate these problems, see an overview in [1]. The most complex circuits evolved in this category are, for example, 6-bit multipliers, 9-bit adders, and 17-bit parity circuits [6], [7].

The fourth problem, which has not been reflected in the literature at all, could also be understood as the scalability problem. We will call it the *scalability of specification problem* in this paper. Despite some successes with the evolution of small accurate circuits (such as compact multipliers [8]), the aforementioned evolutionary approach has not fully been accepted by industry, mainly because it is not scalable and evolutionary algorithms are too non-deterministic and time consuming in comparison with common circuit design and optimization methods.

However, there are good reasons why it makes sense to develop the aforementioned evolutionary approach. Firstly, the circuit design problem can serve as a useful test problem for the performance evaluation and comparison of genetic programming systems. Secondly, the approach seems to be suitable for adaptive embedded systems. If a new (but relatively simple) logic function has to be implemented in a reconfigurable device, employing an EA which utilizes a training set (i.e. a truth table) in the fitness function could be a good approach because running standard circuit design packages is usually impossible in embedded systems. Moreover, the reconfigurable device is seen as a black box in the context of evolvable hardware. A fully functional solution can be obtained even if some (unknown) parts of the reconfigurable device are faulty; which is impossible using conventional synthesis, placement and routing algorithms which expect fault-free chips.

C. Evolutionary Optimization of Complex Circuits

In order to bring the evolutionary circuit design methodology closer to the conventional design flow, it is important to accept how circuits are specified in conventional tools and enable the evolution of really complex circuits. The conventional circuit synthesis and optimization process usually starts with an unoptimized fully functional circuit which is supplied by a designer. Truth tables are usually utilized to describe functions of small circuits having up to twenty primary inputs. Large circuits are described using (a) a hierarchical description such as BLIF or (b) a high level hardware description language such as Verilog or VHDL.

If a fully functional (unoptimized) circuit is available then the fitness calculation based on applying all possible input vectors can be replaced by a more efficient procedure. The method exploits the fact that efficient algorithms were developed in the field of formal verification which enable us to relatively quickly decide whether two circuits are functionally equivalent. In our context, the task is to decide whether the parent and its offspring (created by a mutation operator) are functionally equivalent, assuming that the evolutionary algorithm was seeded by a fully functional solution and the current parent is also fully functional. If the equivalence holds, the fitness of the offspring is given by the number of gates if the task is to minimize the number of gates. An evolutionary circuit optimization method was introduced in [2], which employs a satisfiability problem solver (SAT solver) in the fitness function. An average gate reduction of 25% was reported for benchmark circuits containing thousands of gates and having tens of inputs in comparison with the state of the art academia as well as commercial tools [9]. While the scalability of evaluation problem has been partly eliminated, the method still suffers from long execution times (tens of minutes for the aforementioned circuits) and non-deterministic behavior. The method should be called the evolutionary optimization rather than the evolutionary design because circuit's function is known in advance and only the number of gates is optimized. Using the word *optimization* is also consistent with terminology developed for digital design [10].

An open question is whether complex accurately working circuits can be evolved from randomly seeded initial populations, i.e. whether a truly evolutionary circuit design is possible for complex circuits. Another motivation is that better (more compact) solutions could be obtained because the EA starting with a randomly generated population is not biased by conventional circuits.

Unfortunately, the approach based on equivalence checking [2] can not be used to solve this task. The reason is that the outcome of equivalence checking is a Boolean value, but no additional information showing to what extent the candidate circuit fulfills the specification is provided if the circuits are not functionally equivalent. Such information has to be available in order to construct a fitness function which can distinguish small differences in the quality of phenotypes.

III. Binary Decision Diagrams

Decision diagrams, and especially reduced binary decision diagrams, are the most frequently used data structure for representation and manipulation of Boolean functions in the area of digital circuit design. Among others, verification, test generation, fault simulation, sat-solving and logic synthesis represent their typical applications.

A Binary Decision Diagram (BDD) is a directed acyclic graph with one root and two terminal nodes that are referred to as '0' and '1'. All other nodes are called non-terminal nodes. Each non-terminal node is associated with a primary input variable and has exactly two outgoing edges, called the E-edge (*else* edge) corresponding to assigning the variable a *false* truth value, and the T-edge (*then* edge) corresponding to assigning the variable a *true* truth value, respectively. Every path in a BDD is unique (i.e. no variable appears more than once in the path). This means that if we arbitrarily trace out a path from root to the terminal node '1', then we have automatically found a value assignment to function variables for which the function will be evaluated to 1 regardless of the values of the other variables.

An Ordered Binary Decision Diagram (OBDD) is a BDD where variables occur along every path from the root to a terminal node in strictly ascending order, with regard to fixed ordering. A Reduced Ordered Binary Decision Diagram (ROBDD) is an Ordered BDD where each node represents a unique logic function, i.e. it contains neither isomorphic subgraphs nor nodes with isomorphic descendants. The most important property of ROBDD is the canonicity of the representation. If there are two logic functions representing the same Boolean function, the canonicity of the representation implies that the corresponding ROBDDs are isomorphic. This represents the well known property that is useful especially when checking the equivalence of two Boolean functions (digital circuits) represented as a BDD. Implicit sharing of the nodes represents another feature of ROBDDs. For example, when two or more Boolean functions are simultaneously represented using ROBDD (i.e. there co-exist several root nodes), the merging of isomorphic subgraphs, which represents the basic reduction operation on Ordered BDDs, is applied on all of them. Hence, a node can be reused by several Boolean functions.

Although the ROBDDs offer an efficient way of representing Boolean functions and provide a tool for solving many practical problems in digital circuit design, there are situations in which BDDs perform unsatisfactory. It is the requirement of canonicity which makes BDDs inefficient in representing certain classes of functions. For example, multipliers are known for their exponential memory requirements for any variable ordering [11]. It is also a well known fact that the size of BDD (i.e. the number of non-terminal nodes) for a given function is very sensitive to the chosen variable order. Depending on the actual variable order, there are Boolean functions for which the size of the ROBDD can be either linear or exponential in the number of nodes [12].

A. Operations and Algorithms on ROBDDs

One of the advantages of ROBDDs is the possibility to efficiently peform many of the operations needed for the manipulation of Boolean functions.

The synthesis in general and Boolean operations in particular probably represent the most important operations since they can be used to construct ROBDDs. In order to treat the synthesis in a unified way, so-called If-Then-Else operator (ITE) was introduced [13]. ITE is a ternary Boolean function with inputs f, g, h that computes the function: if f then g else h. Result of ITE is a binary decision diagram for function $f(g, h)$. As one of the inputs of ITE for binary operations is always a terminal node, the overall time complexity of ITE is $O(|F| \cdot |G|)$, where $|F|$, $|G|$ stands for the size of BDD corresponding with function f, g, respectivelly. The implementation of the synthesis operator depends on a particular BDD package. For example, Buddy package offers not only ITE operator as function $ite(f, g, h)$ which takes three ROBDDs as its arguments, but also a ternary function $apply(op, a, b)$ which is optimized for binary operations. This function takes a binary operator op and two ROBDDs a and b as arguments and returns a ROBDD corresponding with the result of $a \ op \ b$ [14].

Satisfiability and equivalence test represent another class of operations that can be efficiently computed due to the canonicity and compactness of ROBDDs. The satisfiability problem (sometimes denoted as *Sat-One*) is defined as follows. Let f be a Boolean function. Then, the goal is to decide whether $Onset(f) \neq \varnothing$ and if this is the case, determine $a \in Onset(f)$. It means that for an OBDD F representing function f one has to find an input assignment a for which $f(a) = 1$ or inform that no such a exists. As it is sufficient to consider a single path, a satisfying assignment can be easily computed in linear time $O(n)$ with respect to the number of variables n. One operation closely related to the satisfiability is *Sat-Count* which computes the number of input assignments for which $f(a) = 1$, i.e. it determines $|Onset(f)|$. *Sat-Count* can be performed in time $O(|F|)$. The equivalence test of two functions f and g is even easier because in most cases it is sufficient to check whether pointers for f and g lead to the same node. This can be done in constant time.

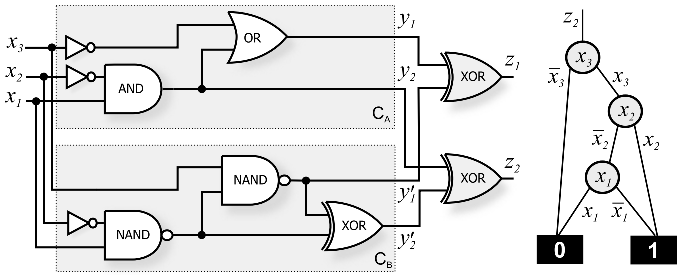

Fig. 1. Principle of equivalence checking of two combinational circuits C_A and C_B using ROBDD. As the ROBBD for z_2 (shown on the right side) contains a path from root node z_2 to the terminal node '1', the circuits are not equivalent. ROBDD for z_1 contains only terminal nodes which means that the outputs y_1 and y'_1 capture the same Boolean function.

B. Equivalence of Boolean Functions

Let C_A and C_B be combinational circuits, both with k inputs denoted $x_1 \dots x_k$ and m outputs denoted $y_1 \dots y_m$ and $y'_1 \dots y'_m$, respectively. In order to check whether the circuits are functionally equivalent, i.e. perform the equivalence test, the following method can be used. Corresponding primary inputs of both circuits are aligned and corresponding primary outputs y_i and y'_i are connected using the XOR gates. The goal is to obtain one circuit that has only k primary inputs $x_1 \dots x_k$ and m primary outputs $z_1 \dots z_m$, $z_i = y_i$ XOR y'_i. In order to disprove the equivalence, it is sufficient to identify at least one output z_i whose $Onset(z_i)$ is not empty, i.e. to find an input assignment x for which the corresponding outputs y_i and y'_i provide different values and thus $z_i = 1$. In order to achieve this goal, *Sat-One* operation can be performed for each output. This approach is illustrated in Figure 1.

C. Towards a Fitness Function Based on ROBDDs

The aforementioned method returns a single Boolean value. However, in evolutionary algorithms we require fitness functions returning a rich scale of values in order to distinguish small differences among candidate solutions. Instead of utilizing the *Sat-One* function, we propose to apply the *Sat-Count* function on every output z_i and count up all the results. The resulting value represents the Hamming distance between C_A and C_B. In the example shown in Fig. 1, *Sat-Count* will return 3 for z_2 and 0 for z_1, i.e. the Hamming distance is $0 + 3 = 3$. It can easily be checked that if $x \in \{001, *11\}$, C_A and C_B provide different output values.

Contrasted to the evaluation of circuit responses for all input combinations, which represents a typical approach in the area of evolutionary circuit design, the method based on ROBDDs allows us to assess the similarity of two circuits efficiently even for complex circuits. The similarity between two circuits, expressed as the Hamming distance, can be calculated in linear time with respect to the size of a ROBDD representing those circuits, which is a significant improvement over the current (exponential) approach. The most complex operation is the construction of a ROBDD tree. However, the construction process can be optimized using hashing and node sharing. Another fact, supporting the efficiency of the ROBDD-based approach, is that circuits to be evaluated (C_A and C_B) are usually very similar, because C_B is created by a genetic operation (e.g. mutation) from its parent C_A.

Getting the most out of any BDD package is not always easy in practice. Some knowledge about the optimal order of the BDD variables and internal implementation of the BDD package is required. At least, it is very beneficial to be familiar with the principle of construction of hash functions. For example, Buddy package uses a triple (L, R, op) to compute a hash value of a binary operation op over BDDs L and R. The hash $H(L, R, op)$ is, however, constructed in such a way that $H(L, R, op) \neq H(R, L, op)$ even if the binary operations corresponding with basic logic expressions such as AND, OR, XOR are commutative.

IV. Proposed Method

The proposed method is based on CGP operating at the gate level. The specification is given in the form of BDD. All operations over BDDs are performed using the Buddy package [14]. Candidate circuits (phenotypes) are represented as directed acyclic graphs in a two-dimensional array of processing nodes (gates). CGP is characterized by [3]:

- a simple encoding system in which a phenotype is encoded in a constant-size array of integers;
- a simple search method based on $(1 + \lambda)$ evolution strategy;
- a single genetic operator – a point mutation.

A. Circuit Encoding

The CGP parameters which the user has to define are as follows:

- $n_r \cdot n_c$ – the number of rows and columns of the grid of nodes in which phenotypes are embedded to;
- n_i – the number of primary inputs;
- n_o – the number of primary outputs;
- n_a – the number of inputs of a node (maximum);
- Γ – a set of functions implemented by each node;
- l – the levels-back parameter.

Primary inputs and processing node outputs are labeled $0, 1, \dots, n_i - 1$ and $n_i, n_i + 1, \dots, n_i + n_c \cdot n_r - 1$, respectively. Each node input can be connected either to the output of a node placed in previous l columns or to one of the primary circuit inputs. A candidate solution consisting of two-input nodes is represented in the chromosome by $n_r \cdot n_c$ triplets (x_1, x_2, ψ) determining for each processing node its function ψ ($\psi \in \Gamma$), and addresses of nodes x_1 and x_2 which its inputs are connected to. The last part of the chromosome contains n_o integers specifying the nodes where the primary outputs are connected to. The chromosome size s is

$$s = n_r n_c (n_a + 1) + n_o. \quad (1)$$

While all the genes are always included in the chromosome, the phenotype size is variable as some nodes (gates) can be disconnected. The nodes involved in the phenotype are called *active nodes*.

B. Fitness Function

Before a fitness value is assigned to a candidate circuit C (represented using CGP), ROBDD for the reference circuit is constructed firstly. Note that the construction of this ROBDD can be performed only once during the initialization. Then, a new ROBDD, D^C, which is functionally equivalent with C has to be constructed. In order to do so, the Apply function (available in the Buddy implementation) is called for every active gate N_j of circuit C. It consumes the logic function performed by N_j and two operands of N_j which are interpreted as pointers to appropriate ROBDD nodes. Depending on the logic function of N_j, one or several new ROBDD nodes are thus included into D^C by means of one call of Apply. The active nodes of C have to be processed from left to right in order to construct the ROBDD correctly. Another ROBDD (let us denote it D^S) is constructed for the specification in the case that the specification is not directly provided in the form of ROBDD.

Corresponding outputs of D^C and D^S are connected to a set of exclusive-or gates, i.e. $z_i = y_i^{D^C}$ xor $y_i^{D^S}$. By means of the *Sat-Count* function, which is available in the Buddy package, one can obtain the number of assignments b_i to the inputs which evaluate z_i to 1. Finally, the fitness function f is defined as

$$f = \sum_{i=1}^{n_o} b_i. \tag{2}$$

Obtaining $f = 0$ in the course of evolution means that a fully functional solution was discovered. The fitness function is then modified to reflect the number of gates which has to be minimized.

C. Search Algorithm

The initial population is randomly generated, but it is ensured that all chromosomes are legal, i.e. the values for (x_1, x_2, ψ) are not pointing out outside the allowed ranges. After evaluation of the population, the highest scored individual (parent) is selected and λ offspring individuals are generated using the mutation operator which modifies h randomly chosen genes (integers) of the parent. However, if two or more individuals can serve as the parent, the individual which has not served as the parent in the previous generation will be selected. This strategy is important because it ensures the diversity of population. The whole process is repeated for a predefined number of generations.

V. Experimental Setup and Results

This section provides the experimental setup, reports the experimental results and discusses various aspects of the proposed method.

A. Optimized Fitness Function

In order to maximize the efficiency of the proposed method, several additional optimizations have been enabled:

- If a mutation is detected as neutral, the fitness value is not calculated and the offspring obtains the same fitness as its parent.
- A new ROBDD is not constructed from scratch, but the *cone of influence*, which is a reduction abstraction technique aiming at simplifying the model in hand by only referring to variables that are of interest, is employed.
- *Sat-Count* is called only for the outputs that are influenced by the mutation.
- The node's indices are swapped during the BDD construction in order to ensure that $\text{index}_1 < \text{index}_2$.
- In order to avoid unwanted penalties during evolutionary design, dynamic variable reordering is disabled.
- ROBDD representing the reference circuit (specification) is minimized before the evolution is started.

TABLE I. Parameters of selected LGSynth91 benchmark circuits

Circuit	PI	PO	Gates	Levels	CE	|BDD|	Gain
x2	10	7	41	7	6.6×10^{2}	38	44%
cm151a	12	2	28	8	1.8×10^{3}	32	96%
9sym	9	1	228	14	1.8×10^{3}	33	5%
ex5	8	63	505	10	2.0×10^{3}	293	4%
cm162a	14	5	41	8	1.0×10^{4}	43	46%
cu	14	11	47	8	1.2×10^{4}	52	19%
apex4	9	19	2868	17	2.3×10^{4}	1011	1%
b12	15	9	60	7	3.1×10^{4}	72.0	11%
cm163a	16	5	42	7	4.3×10^{4}	31	41%
t481	16	1	66	11	6.8×10^{4}	32	0%
tcon	17	16	33	3	6.8×10^{4}	24	21%
alu4	14	8	1059	22	2.7×10^{5}	769	48%
table5	17	15	1500	24	3.1×10^{6}	753	20%
cordic	23	2	63	11	8.3×10^{6}	83	6%
frg1	28	3	103	12	4.3×10^{8}	91	46%
C499	41	32	185	12	6.4×10^{12}	31699	36%

B. Benchmark Circuits and Their Properties

Table I contains benchmark circuits (and their parameters) that were selected from the LGSynth91 set. Their selection followed the aim of finding a threshold in circuit complexity for which it makes sense to employ a BDD-based fitness function instead of the standard evaluation of all assignments to the inputs. The number of primary inputs (PI) and primary outputs (PO) are given. First, we applied a conventional optimization tool ABC [15] and obtained the number of gates (in the column *Gates*) and the longest path from inputs to outputs (in the column *Levels*). The CE column gives the computational effort needed to simulate all input combinations on a 64-bit processor for a circuit consisting of k gates. Because a parallel simulation is utilized, $CE = k \cdot 2^B$, where $B = PI - 6$ if $PI > 6$, and $B = 1$ otherwise. Circuits are listed according to their expected complexity for evolutionary design (i.e. CE) in Table I.

Parameters of corresponding ROBDDs are the size ($|BDD|$) and *Gain*. Each ROBDD encodes PO Boolean functions using $|BDD|$ nodes. If a node is shared by more functions, it is counted just once. Except circuit C499 and compared to the number of gates (produced by ABC), the BDDs can be seen as a more compact representation. The C499 circuit is probably one of the circuits whose ROBDD representations are exponential with respect to standard logic networks. It should be noted that Buddy can still manipulate over circuits of this size.

It was discussed in Section III that the variable order significantly influences the size of BDD. In order to minimize the size of ROBDD representing a benchmark circuit, BDDs (constructed according to resulting logic networks produced by ABC) were further optimized by a SIFT algorithm, which is available in Buddy. The *Gain* column shows the node reduction obtained with respect to the original variable order $x_1 \prec x_2 \prec x_3 \prec ... \prec x_{PI}$. It can be seen that applying the SIFT algorithm is very useful, especially for cm151a circuit (96% reduction). On the other hand, no improvement can be seen for t481 circuit.

Enabling all the aforementioned optimizations led to the average construction time of 0.2 s, with the worst case of 1.8 s for t481 circuit. Logic networks (in terms of the number of gates) as well as the corresponding ROBDDs (the number of nodes) can be seen as carefully pre-optimized by means of the conventional methods (ABC and Buddy) in order to fairly evaluate the contribution of the evolutionary design/optimization methods.

C. The Obtained Speedup: CGP vs. CGP-BDD

In this subsection, we will compare the standard evaluation used in CGP and the BDD-based evaluation (CGP-BDD) in two scenarios. In the first scenario, the initial population is seeded by the circuits obtained from ABC. In the second one, the initial population is seeded randomly. We will study the impact of neutral mutations and active nodes on the evaluation time. The results were obtained from 50 independent runs using the following experimental setup: $\lambda = 4$, $h = 5$, $n_r = 1$, $\Gamma = \{\mathrm{AND}, \mathrm{OR}, \mathrm{XOR}, \mathrm{NAND}, \mathrm{NOR}, \mathrm{XNOR}, \mathrm{NOT}, \mathrm{BUF}\}$. The number of CGP columns n_c was chosen to be equal to the size of a circuit syntesized using ABC (see Table I). As the circuits synthesized by ABC are known to be usually far from an optimal result, the number of available resources should be sufficient to provide some degree of neutrality for CGP. The evolution is terminated when a predefined number of generations $g_{max} = 10^4$ is spent. This value was chosen so that the runtime of a single evolutionary run is within tens of seconds. All the experiments were performed on a cluster of computation nodes equipped with Intel Xeon processors running at 3 GHz.

TABLE II. THE AVERAGE EVALUATION TIME FOR ONE POPULATION WHEN CGP AND CGP-BDD ARE SEEDED BY CONVENTIONAL DESIGNS.

	$t_{evalpop}$ (ms)			Averages		
Circuit	CGP	CGP-BDD	speedup	α	DIFF	CEef
x2	0.04 ± 0.2	0.02 ± 0.1	2	93.5%	28.4%	6.1×10^2
cm151a	0.12 ± 0.3	0.02 ± 0.2	5	85.9%	35.5%	1.5×10^3
9sym	0.11 ± 0.3	0.23 ± 0.5	0.5	94.1%	15.0%	1.7×10^3
ex5	0.14 ± 0.4	0.16 ± 0.4	0.9	93.7%	7.0%	1.9×10^3
cm162a	0.75 ± 0.5	0.03 ± 0.2	24	88.5%	27.5%	9.3×10^3
cu	0.83 ± 0.4	0.04 ± 0.2	20	95.5%	27.2%	1.1×10^4
apex4	1.55 ± 0.5	0.54 ± 0.6	3	99.6%	2.3%	2.3×10^4
b12	2.26 ± 0.4	0.03 ± 0.2	82	96.0%	21.6%	2.9×10^4
cm163a	3.24 ± 1.3	0.02 ± 0.1	147	86.1%	24.2%	3.7×10^4
t481	2.44 ± 0.9	0.03 ± 0.2	73	40.4%	10.0%	2.7×10^4
tcon	3.43 ± 0.7	0.02 ± 0.1	223	72.6%	14.8%	4.9×10^4
alu4	18.09 ± 2.0	0.96 ± 0.8	19	96.9%	6.5%	2.6×10^5
table5	207.41 ± 19.8	0.60 ± 0.6	347	99.3%	5.7%	3.1×10^6
cordic	* 5.78×10^2	0.14 ± 0.4	4.0×10^4	81.5%	23.3%	6.7×10^6
frg1	* 3.02×10^4	0.41 ± 0.6	7.3×10^5	90.3%	18.1%	3.9×10^8
C499	* 4.45×10^8	78.84 ± 33.7	5.6×10^7	99.3%	40.8%	6.3×10^{12}

Table II gives the average evaluation time of one population (standard CGP vs. CGP-BDD) when the initial population is seeded by circuits pre-optimized by ABC. The averages are calculated using the last 100 generations in which the whole process is considered to be stable and the resulting values are trustworthy. It can be seen that for the standard CGP the order of magnitude of $t_{evalpop}$ is corresponding with the CE given in Table I. The average duration of one operation performed during the evaluation is 70 ps, while minimum is 44.5 ps and maximum 94.4 ps. This spread is caused by changes in the active nodes count and cache misses. Only estimated numbers are given for circuits with more than 20 primary inputs because conducting real experiments with such complex circuits would be extremely time and memory consuming (see * in Tables). In the case of CGP-BDD, the evaluation time of one population is lower than 1 ms; the only exception is C499 circuit which was discussed earlier. The speedup of CGP-BDD against CGP is given in column *speedup*.

The average number of active gates with respect to available gates (n_c) is denoted α. Considering the fact that the evolution started with well pre-optimized circuits by ABC, one would expect α close to 100%. However, CGP is still capable of reducing the circuit size even if a relative small number of generations was allowed (as reported in [2]). For example, a 60% reduction was obtained for t481 and a 28% reduction in the case of tcon. The *DIFF* column gives the average number of BDD nodes (in %) with respect to the total number of available gates which has to be included into BDD in order to calculate the Hamming distance to its parent (i.e. to its parent's BDD). One can observe that this percentage represents just a small fraction of the total number of nodes and hence this operation has only a small overhead. The last column gives the average computational effort $CEef = CE \cdot \alpha$. The $CEef$ values are similar to CE from Table I because almost all nodes are active (α is relatively high). Hence the values shown in *speedup* column can be considered as a good estimate of the real speedup.

TABLE III. THE AVERAGE EVALUATION TIME FOR ONE POPULATION WHEN CGP AND CGP-BDD ARE RANDOMLY SEEDED.

	$t_{evalpop}$ (ms)			Averages		
Circuit	CGP	CGP-BDD	speedup	α	DIFF	CEef
x2	0.03 ± 0.2	0.01 ± 0.1	2	36.7%	10.1%	2.4×10^2
cm151a	0.06 ± 0.2	0.01 ± 0.1	5	37.5%	12.9%	6.7×10^2
9sym	0.01 ± 0.1	0.02 ± 0.1	0.6	8.2%	0.7%	1.5×10^2
ex5	0.08 ± 0.3	0.06 ± 0.2	1	29.0%	3.3%	5.9×10^2
cm162a	0.37 ± 0.5	0.01 ± 0.1	32	34.5%	7.5%	3.6×10^3
cu	0.50 ± 0.5	0.02 ± 0.1	29	34.2%	9.0%	4.1×10^3
apex4	0.11 ± 0.3	0.17 ± 0.4	0.6	4.0%	0.1%	9.3×10^2
b12	1.09 ± 0.6	0.01 ± 0.1	89	29.6%	5.8%	9.1×10^3
cm163a	1.23 ± 0.7	0.01 ± 0.1	114	27.6%	5.9%	1.2×10^4
t481	0.60 ± 0.7	0.01 ± 0.1	73	8.7%	1.4%	5.9×10^3
tcon	2.47 ± 1.1	0.01 ± 0.1	184	17.9%	5.9%	1.2×10^4
alu4	1.71 ± 1.7	0.08 ± 0.3	21	6.1%	0.2%	1.7×10^4
table5	18.14 ± 18.2	0.11 ± 0.3	163	5.3%	0.2%	1.6×10^5
cordic	* 1.86×10^2	0.01 ± 0.1	1.4×10^5	10.8%	2.0%	9.0×10^5
frg1	* 9.72×10^3	0.02 ± 0.1	6.4×10^6	8.5%	1.1%	3.7×10^7
C499	* 1.43×10^8	1.84 ± 2.0	7.8×10^8	21.1%	2.2%	1.3×10^{12}

Table III shows the average evaluation time for one population in the case that the initial populations are randomly seeded. The obtained speedup (CGP-BDD vs. CGP) is very close to the values reported in Table II despite the fact that the number of active gates (α) is low. Randomly seeded CGP typically operates with small phenotypes whose evaluation is less time consuming and CE is thus lower in comparison with the previous scenario. For example, in the case of apex4, CGP is faster than CGP-BDD and the corresponding computational effort is one order of magnitude smaller than for seeded initial populations. In addition to low α (4%), the reason is that the size of a BDD used as reference for apex4 is relatively high (1011 nodes).

Low α values together with a careful analysis of CGP log files confirmed us that randomly seeded evolution typically begins with a small phenotype which is enriched by additional gates in the process of evolution. This means that the standard CGP evaluation will be slower in the course of evolution, which corresponds with increasing CE.

In comparison with the first scenario, the ROBDDs which are constructed to evaluate candidate circuits require a lower number of additional nodes (see the *DIFF* column in Table II). This phenomenon was actually expected because phenotypes are smaller. The consequence is the evaluation time is significantly reduced. For example, $t_{evalpop}$ is 10 times shorter for alu4 or C499 (see CGP-BDD in Table II vs. Table III).

One can observe that employing CGP-BDD is useful if CE is higher than 3.0×10^3 independently of the method used to generate the initial population. In practice, this CE corresponds with circuits having about **14 primary inputs**. The achievable speedup then significantly increases with the number of inputs. The standard CGP is hardly applicable for circuits with over 20 inputs because the target truth table requires a considerable memory and many operations have to be performed to obtain the fitness values.

D. Evolutionary Design Using CGP-BDD

The standard CGP and CGP-BDD were utilized to evolve selected combinational circuits from scratch. We employed the setting introduced in the previous section. A single run was terminated after $t_{max} = 3$ hours. Statistical results which are summarized in Table IV are presented from 100 independent runs. Columns CGP and CGP-BDD give the average number of generations that can be performed within the available time t_{max}. The corresponding speedup is shown as *speedup*. The *success rate* represents the number of runs in which a fully functional (correct) solution was obtained. Employing CGP-BDD has led to increasing the number of evaluated candidate circuits as well as the success rate. For example, the 113 times faster evaluation allowed increasing the success rate by 85% in the case of tcon. A relatively small increase in the number of generations (8x) allowed us to evolve a fully functional circuit alu4 in 58% of runs, while the standard CGP did not provide any correct implementation within the same time. Despite the increased number of evaluations, no correct solution was obtained for circuits table5, cordic and C499. Similarly to large multipliers, these circuits seem to be difficult for the evolutionary design.

Parameters of evolved circuits are summarized in Table V. f_{bst} is the best obtained fitness (in %) across all the runs. If $f_{bst} = 100\%$ for a given circuit, g_{bst} gives the number of gates in the most compact solution. If $f_{bst} < 100\%$, g_{bst} is the average number of gates in the last 100 generations across all the runs. If $f_{bst} < 100\%$ and g_{bst} is close to the number of available gates (n_c), we supposed that the evolution failed because the number of gates is insufficient. However, this is not the case of the circuits for which the CGP-BDD did not provide any correct solution, because about 80% of available gates are unused. The CGP-BDD was very close to desired solutions, but get stuck in a local optimum because of the overall hardness of the task.

Evolved circuits are much more compact (a 17.9 – 93.4% reduction obtained) than the circuits produced by ABC (see the *Improv* column). CGP-BDD provided better results than CGP in most cases, with the exception of smaller circuits when CGP-BDD is slower than CGP.

Figure 2 shows convergence curves (25 runs) of both methods when applied on the tcon circuit for which they provided the same size of a correct solution. The progress of fitness values is very similar in both cases; however, CGP-BDD is approximately 100 times faster than CGP.

Figure 2 also shows the average number of active gates in all phenotypes. One can observe the spread in the number of active gates among the runs and estimate the minimum and maximum evaluation time of a population. The first quarter of the runtime is characterized by a very variable number of active gates. In the progress of evolution, the number of active gates is growing until a correct solution is reached. Later, the number of active gates is slightly decreasing which corresponds with the optimization phase toward the most compact circuit.

TABLE IV. EVOLUTIONARY CIRCUIT DESIGN USING CGP AND CGP-BDD.

	# generations			success rate	
Circuit	CGP	CGP-BDD	speedup	CGP	CGP-BDD
x2	1.6×10^8	1.6×10^8	1.0	4.0%	4.0%
cm151a	5.8×10^7	1.2×10^8	2.1	97.0%	**98.0%**
9sym	1.9×10^8	8.0×10^7	0.4	100.0%	100.0%
ex5	6.1×10^7	4.1×10^7	0.7	**13.0%**	7.0%
cm162a	1.2×10^7	1.2×10^8	10.4	44.0%	**100.0%**
cu	1.2×10^7	1.6×10^8	13.2	0.0%	**7.0%**
apex4	1.5×10^7	9.9×10^6	0.7	0.0%	0.0%
b12	4.6×10^6	1.3×10^8	28.1	0.0%	**19.0%**
cm163a	3.4×10^6	1.6×10^8	47.8	0.0%	**20.0%**
t481	3.7×10^6	1.7×10^8	45.9	92.0%	**100.0%**
tcon	1.6×10^6	1.8×10^8	113.0	15.0%	**100.0%**
alu4	2.0×10^6	1.8×10^7	8.5	0.0%	**58.0%**
table5	3.0×10^5	2.7×10^7	90.2	0.0%	0.0%
cordic	n.a.	2.3×10^8	n.a.	0.0%	0.0%
frg1	n.a.	1.1×10^8	n.a.	0.0%	**54.0%**
C499	n.a.	6.6×10^6	n.a.	0.0%	0.0%

VI. CONCLUSION

In this paper, we proposed a new method for an efficient evaluation of candidate circuits in randomly seeded CGP. The method exploits the fact that the specification can be given in the form of BDD, every candidate circuit can be transformed to BDD and the similarity/equivalence of these BDDs can be calculated in a relatively short time. In order to construct BDDs and perform various operations over these BDDs, Buddy package was utilized. Compared to the standard CGP, the proposed method enabled us to significantly accelerate the whole evolutionary design process and increase the success rate, especially if the number of primary inputs of target circuits is greater than 14.

TABLE V. PARAMETERS OF EVOLVED CIRCUITS

	CGP			CGP-BDD		
Circuit	f_{bst}	g_{bst}	Improv.	f_{bst}	g_{bst}	Improv.
x2	100.0%	27.0	34.1%	100.0%	26.0	**36.6%**
cm151a	100.0%	23.0	17.9%	100.0%	23.0	17.9%
9sym	100.0%	23.0	89.9%	100.0%	23.0	89.9%
ex5	100.0%	152.0	**69.9%**	100.0%	155.0	69.3%
cm162a	100.0%	26.0	36.6%	100.0%	26.0	36.6%
cu	99.8%	24.9	n.a.	**100.0%**	34.0	**27.7%**
apex4	**89.7%**	648.1	n.a.	89.2%	594.0	n.a.
b12	99.3%	35.7	n.a.	**100.0%**	45.0	**25.0%**
cm163a	99.4%	23.6	n.a.	**100.0%**	26.0	**38.1%**
t481	100.0%	22.0	66.7%	100.0%	21.0	**68.2%**
tcon	100.0%	25.0	24.2%	100.0%	25.0	24.2%
alu4	99.3%	178.2	n.a.	**100.0%**	70.0	**93.4%**
table5	98.1%	134.3	n.a.	**99.4%**	157.5	n.a.
cordic	n.a.	n.a.	n.a.	**99.4%**	12.2	n.a.
frg1	n.a.	n.a.	n.a.	**100.0%**	44.0	**57.3%**
C499	n.a.	n.a.	n.a.	**99.6%**	36.2	n.a.

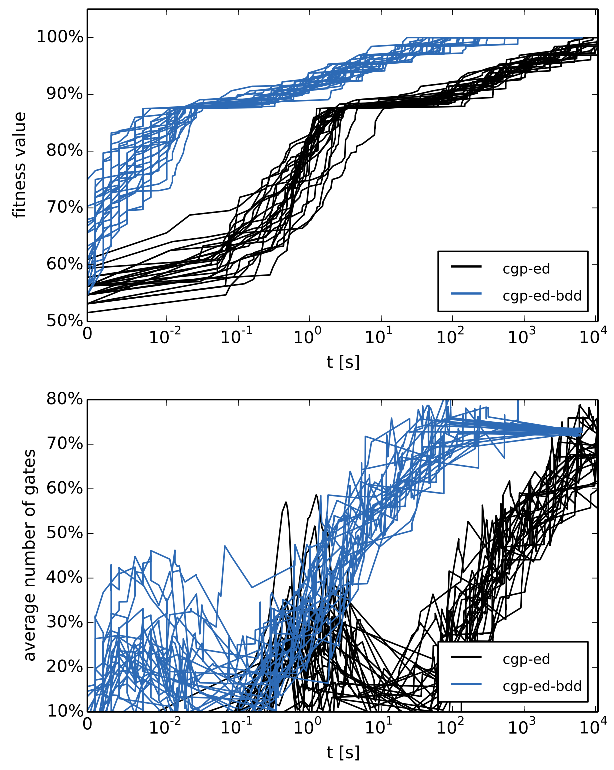

Fig. 2. Convergence curves for *tcon* benchmark circuit.

Correct implementations were evolved for problem instances which are fairly out of the scope of well-optimized standard CGP implementations. For example, a correct implementation was evolved for a 28-input frg1 circuit. The evolved circuit contains less gates than the result of a conventional optimization conducted by ABC (a reduction of 57% obtained). A detailed comparison was performed with the standard CGP. The results indicate that if even more complex circuits should be evolved, the proposed fast evaluation method would be accompanied with a suitable decomposition scheme (such as the divide and conquer) in order to reduce the size of genotype and thus to restrict the search space. Our future work will be devoted to a further development and detailed analysis of the proposed method. A research combining the proposed method with a suitable automated decomposition scheme is expected.

ACKNOWLEDGMENT

This work was supported by the Czech science foundation project 14-04197S – Advanced Methods for Evolutionary Design of Complex Digital Circuits.

REFERENCES

[1] L. Sekanina, "Evolvable hardware," in *Handbook of Natural Computing*. Springer Verlag, 2012, pp. 1657–1705.

[2] Z. Vasicek and L. Sekanina, "Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 305–327, 2011.

[3] J. F. Miller, *Cartesian Genetic Programming*. Springer-Verlag, 2011.

[4] G. Greenwood and A. M. Tyrrell, *Introduction to Evolvable Hardware*. IEEE Press, 2007.

[5] X. Yao and T. Higuchi, "Promises and Challenges of Evolvable Hardware," in *First International Conference on Evolvable Systems: From Biology to Hardware*, ser. LNCS, vol. 1259. Springer, 1996, pp. 55–78.

[6] R. Hrbacek and L. Sekanina, "Towards highly optimized cartesian genetic programming: From sequential via simd and thread to massive parallel implementation," in *Proceeding of Genetic and Evolutionary Computation Conference, GECCO*. ACM, 2014, pp. 1015–1022.

[7] E. Stomeo, T. Kalganova, and C. Lambert, "Generalized disjunction decomposition for evolvable hardware," *IEEE Transaction Systems, Man and Cybernetics, Part B*, vol. 36, no. 5, pp. 1024–1043, 2006.

[8] V. Vassilev, D. Job, and J. F. Miller, "Towards the Automatic Design of More Efficient Digital Circuits," in *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*. IEEE, 2000, pp. 151–160.

[9] Z. Vasicek and L. Sekanina, "A global postsynthesis optimization method for combinational circuits," in *Proc. of the Design, Automation and Test in Europe, DATE*. EDAA, 2011, pp. 1525–1528.

[10] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.

[11] R. E. Bryant, "On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication," *IEEE Transaction on Computers*, vol. 40, no. 2, pp. 205–213, 1991.

[12] R. Ebendt, G. Fey, and R. Drechsler, *Advanced BDD Optimization*. Springer, 2000.

[13] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design*, 1st ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1998.

[14] J. Lind-Nielsen and H. Cohen. BuDDy - A Binary Decision Diagram Package. [Online]. Available: http://buddy.sourceforge.net

[15] A. Mishchenko, "ABC: A system for sequential synthesis and verification, Berkley logic synthesis and verification group," 2012. [Online]. Available: http://www.eecs.berkeley.edu/˜ alanmi/abc/