Generalized Disjunction Decomposition for Evolvable Hardware

Emanuele Stomeo, *Student Member, IEEE*, Tatiana Kalganova, and Cyrille Lambert

***Abstract*—Evolvable hardware (EHW) refers to self-reconfiguration hardware design, where the configuration is under the control of an evolutionary algorithm (EA). One of the main difficulties in using EHW to solve real-world problems is scalability, which limits the size of the circuit that may be evolved. This paper outlines a new type of decomposition strategy for EHW, the "generalized disjunction decomposition" (GDD), which allows the evolution of large circuits. The proposed method has been extensively tested, not only with multipliers and parity bit problems traditionally used in the EHW community, but also with logic circuits taken from the Microelectronics Center of North Carolina (MCNC) benchmark library and randomly generated circuits. In order to achieve statistically relevant results, each analyzed logic circuit has been evolved 100 times, and the average of these results is presented and compared with other EHW techniques. This approach is necessary because of the probabilistic nature of EA; the same logic circuit may not be solved in the same way if tested several times. The proposed method has been examined in an extrinsic EHW system using the $(1 + \lambda)$ evolution strategy. The results obtained demonstrate that GDD significantly improves the evolution of logic circuits in terms of the number of generations, reduces computational time as it is able to reduce the required time for a single iteration of the EA, and enables the evolution of larger circuits never before evolved. In addition to the proposed method, a short overview of EHW systems together with the most recent applications in electrical circuit design is provided.**

***Index Terms*—Adaptive system, evolutionary computation, evolvable hardware (EHW), problem decomposition.**

I. Introduction

EVOLVABLE hardware (EHW) [1]–[12], also known as evolutionary electronics and hardware evolution, is a technique to automatically design circuits (digital [5], [9], [13] and analog [5]–[7], [14], [15], antennas [16]–[18], and robots [1], [19], [20]) using methods inspired by natural evolution. The circuit configuration is carried out under the control of evolutionary algorithms (EAs) [21]–[25]. These techniques began to be treated with increasing interest in the 1960s when Holland introduced the concept of genetic algorithms (GAs) [26], [27], which are the most general methods of solving search and optimization problems. Research in this area has introduced other EAs, such as genetic programming (GP) [28]–[30], evolution strategy (ES) [24], grammatical evolution (GE) [31], evolutionary programming (EP) [32], Cartesian GP (CGP) [33], [34], adaptive GA (AGA) [35], parallel GA (PGA) [36]–[38], compact GA [39], etc. The basic schema of an EHW system, adapted for the evolution of logic circuits defined by truth tables, is illustrated in Fig. 1. Initially, EHW was intended for real-world applications [41], but due to its limitations in scalability (see Section II), to date few real-world applications have been developed. Table I outlines the most important applications for electrical circuits developed with EHW systems so far. The performance of EHW has been actively studied on the evolution of multipliers. Both programmable logic array (PLA)-based and field-programmable gate array (FPGA)-based circuits have been considered. For example, the three-bit multiplier containing 26 logic gates [40] (the best solution obtained to date) has been evolved for an FPGA structure after 3 000 000 generations using the gate-level EHW approach, introduced in [42] and [43]. In gate-level evolution, the design of digital circuits is based on primitive hardware gates such as AND and OR. Analysis of the complexity of evolved logic circuits in one run revealed that the most complex multiplier evolvable is the four-digit multiplier [44]. The four-bit multiplier was designed using logic gates as building blocks for an FPGA target structure after 643 274 721 generations [44].

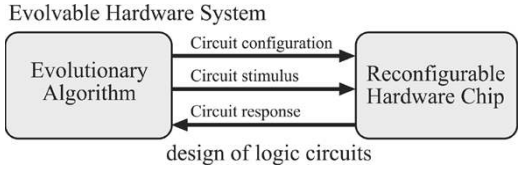

Fig. 1. Schema of a basic EHW system. The EA sends the circuit configurations (the chromosome) to the chip, which configures itself with the configuration received. The EA is also responsible for sending test vectors to stimulate the design inputs and to check the circuit's response received against the expected values. Based on those responses, the EA modifies the chromosomes and supplies a new configuration to the chip. In the case of digital logic circuits, the stimuli are the input combinations of the truth table.

This paper presents a new type of decomposition strategy for the evolutionary design of relatively large combinational circuits. The proposed method involves reducing the number of inputs prior to evolving logic circuits by introducing a new function, which was previously briefly discussed in [45]. Here, the authors intend to investigate more thoroughly the behavior of the proposed technique and to show the following.

- It is capable of evolving larger circuits never before evolved with any other evolutionary computational technique in a reasonably short time. The most complex circuits evolved are the 17-bit parity, the six-bit multiplier, and the alu4, which is a circuit with 14 inputs and eight

Manuscript received May 4, 2005; revised July 25, 2005 and December 27, 2005. This work was supported in part by the Engineering and Physical Sciences Research Council under Grant GR/S17178. This paper was recommended by Associate Editor H. Takagi.

The authors are with the School of Engineering and Design, Brunel University, UB8 3PH Middlesex, U.K. (e-mail: stomeo@ieee.org; tatiana.kalganova@brunel.ac.uk).

Digital Object Identifier 10.1109/TSMCB.2006.872259

TABLE I
MAIN ELECTRICAL CIRCUITS DESIGNED WITH EHW SYSTEMS

Author	Year	Target device	Approach	Design
Koza et al.	1992-present	Computer program presented as trees.	Genetic programming [28]–[30].	Synthesize design for complex structures as analog electrical circuits, controllers and mathematical algorithm [28]–[30], [106], [107].
Xu et al.	1995	An Intel Hypercube iPSC™, with 16 microprocessor [63].	Parallel genetic algorithm [36]–[38].	Digital filter [63].
Higuchi et al.	1999	Analog EHW chip [5].	Function Level Evolution [42], [43], [64].	Chip for cellular phones [5].
				Clock timing for Giga hertz systems [5].
		Digital chip which includes GA hardware, reconfigurable logic gates and a CPU core.	Function Level Evolution [42], [43], [64].	Neural network chip [5].
			Gate level evolution [42], [43].	Data compression chip for electrophotographic printers [5].
				Prosthetic hand [5].
				Mobile robot navigation [5].
Thompson et al.	1999	Dynamic state machine [66].	Genetic algorithm.	Robot controller [66].
		Xilinx XC6216 FPGA [100].		Tone discriminator [66].
Miller et al.	2000	Software based with array of AND, XOR [44].	Cartesian genetic programming [33], [34] together with (1+λ) Evolution Strategy [24], [25]	4-bit multiplier [44].
Kalganova	2000	Software simulation: array of AND, OR, XOR, NOT gates and multiplexer.	Bi-directional incremental evolution [62], [84].	Logic functions from MCNC benchmark [81]. Best circuit evolved was with 7 inputs and 10 outputs.
			Extrinsic Function Level evolution [67].	3-bit multiplier.
Seok et al.	2000	Xilinx XC6216 FPGA [100].	Context switching [68].	Controllers for autonomous mobile robot [68].
Keymeulen et al.	2000	The GA runs in UNIX workstation. The circuit configurations are downloaded from the workstation to a PC board with 4 FPTA [97]–[99].	Evolutionary algorithm with: fitness and population based approach [6], [96].	Fault tolerant digital (XNOR) and analog (2-bit multiplier) circuit [6].
Stoica et al.	2003	The evolutionary algorithm is implemented on DSP and the reconfigurable chip is a JPL FPTA [97]–[99].	Evolutionary algorithm.	Half-wave rectifier circuit [7].
Vinger et al.	2003	Xilinx Virtex FPGA XCV1000 [101].	Genetic algorithm.	Digital Filter: 8 tap 12 bit [69].
Torresen	2003	Software simulation: array of AND-gates and XOR-gates [70].	Incremental evolution with training partitioning vector [70].	5-bit multiplier [70].
	2004	Software simulation: FPGA-based [71].	Data bus architecture [71].	Multipliers (max 4 bits).
		Software simulation. The proposed architecture fits into most FPGA [72].	Classic GA with incremental evolution [72].	Traffic signs classifier [72].
Hartmann et al.	2004	Software simulator of digital circuits including analogue noise.	Tournament selection genetic algorithm [104], [105].	2-bit multiplier with noise [13].
				2-bit adders with noise [13].
				3-bit multiplier with noise [13].
Zhang et al.	2004	Xilinx Virtex FPGA XCV1000 [101]. It is reconfigured by an external hardware GA.	Cartesian Genetic programming [33], [34]	Digital image filter [73].
Gallagher et al.	2004	FPGA XC400 [102] on BORG board and VirtexII XC2V1000 [101].	Modified compact genetic algorithm (CGA) [39].	De Jong test suite [74], [76].
				Dynamic benchmark designed by Gallagher et al. [74].
				Locomotion controller [74].
Sekanina	2003	Xilinx FPGA XC4028XLA [102].	Cartesian Genetic [33], [34] Programming operating at function level [42], [43], [64].	Several image digital noise filter and edge detectors [75].
	2005	Software simulation with polymorphic gate-level circuits.	Cartesian Genetic Programming [33], [34].	Polymorphic circuits [77], [78].

outputs taken from the Microelectronics Center of North Carolina (MCNC) library [81].

- It is able to significantly reduce the number of generations required to evolve digital logic circuits and to reach higher fitness values, which result in better optimized circuits, although the research presented in this paper concentrates only on the evolution of fully functional circuits rather than on the optimization.

The method presented here therefore breaks through the scalability barrier and opens up new opportunities in the design and application of evolvable combinational electrical circuits. The significance of this paper lies in the potential for EHW to contribute to the design of electrical circuits by removing human intervention and associated costs. Not only electrical circuits but also antennas and robot controllers could be automatically designed without the need for human inputs. Furthermore, EHW, because of its basis in EAs, could adapt itself to change task requirements and optimize its performance, which would be particularly desirable where human intervention is unfeasible or very expensive. Despite recent advancements in research and technology, a number of issues remain unresolved, including the following.

- Reducing the number of logic gates, which is quite high if compared with hand design circuits.
- Increasing the evolvability [46]–[49]. Evolvability, as defined in [52] is the ability of the genetic operator/representation scheme to produce offsprings that are fitter than their parents.

- Fault tolerance [6], [13], [49]–[51].
- Maintainability and comprehensibility of the evolved circuit [53]. Maintainability as in [54] is the system's ability to preserve and improve its performance and fault tolerance properties and to adjust them to the varying environment.

This paper is organized as follows. The next section considers the problems of scalability and stalling effect in the fitness function. In Section III, a brief description of the benchmarks used to carry out the simulations is presented. Section IV illustrates the basis of an extrinsic EHW system (first introduced in [55]), together with the EA implemented, the chromosome encoded, and the fitness function applied to the system. Furthermore, a short introduction on bidirectional incremental evolution (BIE) is given. Section V outlines the limitations of an EHW system by testing the evolution of several logic circuits of different complexities. Section VI proposes the generalized disjunction decomposition (GDD) for EHW, together with a study case. Section VII gives the parameter's setting for the experiments carried out with the proposed method. Section VIII illustrates the experimental results of the proposed method together with BIE. The proposed method is compared against other techniques and the advantages and disadvantages are outlined. Section IX concludes this paper and provides a summary of key conclusions.

II. Scalability Problems and Stalling Effect in the Fitness Function

In this section, the problem of scalability [1], [49], [60], [85]–[87] and the stalling effect in the fitness function for an EHW system are outlined. The word scalable, or scalability, as written in [93], has been used to describe how the size of the problem will influence the performance of algorithms. EHW systems are not scalable because of the genotype length, which increases with problem size [61], and the time required for fitness evaluation, which increases rapidly with the size of the desired evolvable circuits.

The length of the genotype increases with the number of logic gates used during the evolution and the level of permitted connectivity between logic gates. The time necessary for fitness evaluation is not scalable because it is exponentially dependent on the number of inputs of the system that should be evolved. If the number of inputs increases linearly, the number of input–output combinations, which represent the description of the digital logic circuits problem, increases by a power of 2. Consequently, as the number of inputs increases, the system needs more time to produce new potential solutions, to evaluate them, and to select new individuals. The time required for the evolutionary process is given by the time for a single iteration (μ) multiplied by the total number of iterations N_{gen} needed to solve the problem, i.e.,

$$T_{\mathrm{tot}} = \mu N_{\mathrm{gen}}. \tag{1}$$

A possible method to reduce the total time for the evolutionary process is to introduce a new algorithm to reduce the number of generations needed or to reduce the time for a single iteration. The method proposed in this paper is able to accomplish both reductions.

Recently, the scalability problem has been investigated mainly in the following areas.

- Introducing and/or improving existing evolutionary processes [19], [74], [90].
- Developing multievolutionary processes using the principles of problem decomposition [68], [79], [84].
- Improving the genotype–phenotype mapping on biological development [85], [86], [88], [89], [91], [92]. This approach is achieving good results, for example, Gordon [86] was able to evolve a 12-bit parity function.

One approach to tackling scalability is the function-level EHW proposed in [42], [43], [64], and [65]. In function-level evolution, the synthesis of circuits is done based on higher functions as sin, adders, etc., instead of primitive gates such as AND and OR. Function-level evolution has proven to be successful in achieving the evolution of relatively complex tasks [64]. However, one of the main weaknesses of this approach is that it still requires human intervention to select the most appropriate functions for specific problems. Function-level evolution has been further extended in [67]. Although the proposed approach significantly reduces the number of generations required to obtain a fully functional solution, the evolvability [46], [48], [49] of logic circuits with a higher number of inputs remains the central issue.

Using decomposition strategies, a number of approaches intended to overcome scalability have been introduced, such as divide-and-conquer for EHW [79] and BIE [62], [84].

The divide-and-conquer method, also called increased complexity evolution [80], has been introduced to reduce the search space, which allows complete evolution of logic circuits with ten inputs by introducing the training vector and partitioned training set [70]. However, a significant weakness is also present, that is, the difficulties in defining the fitness function for the initial stages of evolution, which makes it less suitable for completely automatic systems. Furthermore, this method creates an unconditional constraint on the system: the top-down solving approach that does not allow the discovery of new designs. BIE [62], [84] is a completely automatic decomposition method that does not require any knowledge from the designer. However, it is not scalable to really large circuits due to the limitations of EHW-oriented output and Shannon decompositions (see Fig. 2). The first attempt to use BIE in EHW was achieved by the evolution of a seven-input ten-output logic function from MCNC benchmark [81]. BIE has been further improved by the introduction of new assembling techniques [82]. Although BIE and increased complexity evolution have proven to be successful in the evolution of logic circuits, the scalability problem remains to be one of the main issues in achieving the evolution of relatively large logic circuits in a reasonably short time. This paper addresses that issue.

The problem of stalling effect in fitness functions is related to the nonimprovement of the fitness values during the evolutionary process. Fig. 3 shows the stalling effect of the fitness function during an evolutionary process for evolving a

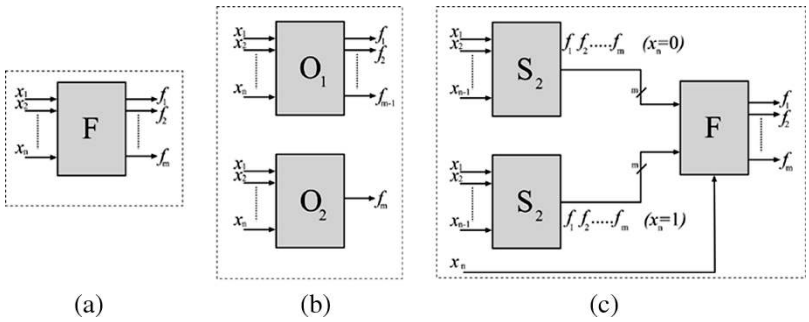

Fig. 2. Decomposition of logic circuits. (a) Initial system with n inputs and m outputs. (b) Output decomposition. (c) Shannon decomposition [62].

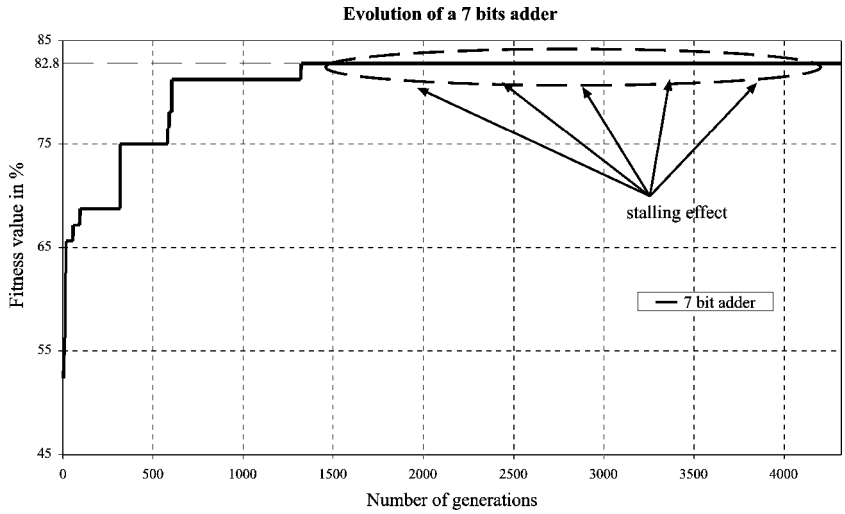

Fig. 3. Stalling effect in fitness function. Stalling effect in fitness function refers to a nonimprovement of the fitness value during the evolution process. In this graph, the stalling effect during an evolution of a seven-bit adder is shown.

seven-bit adder using $(1 + \lambda)$ ES. It may be observed that when the fitness function reaches 82.8% the stalling effect occurs. It means that the EA chosen and/or the initial configuration for solving that particular problem are not suitable.

III. Benchmarks

In this paper, several combinational circuits have been considered for use in simulations. First of all, the benchmarks usually used within the EHW community were considered, i.e., multipliers, used in [13], [44], [49], [70], [88], [94], and [95], and parity circuits, used in [30], [85]–[87], and [108].

Then, the MCNC benchmark [81], usually used by the logic design community, was taken into account. Finally, simple logic circuits with randomly generated truth tables were considered. The rationale for choosing several benchmarks is that it is beneficial to be able to compare the method proposed here with other existing methods in order to establish its contribution.

IV. Extrinsic EHW

In this section, an explanation of the system used to evolve combinational logic circuits is given. The EA that has been used and the chromosome representations, fitness function, genetic operators, and BIE are presented.

A. Description of the Algorithm

The EA used for the simulation is the well-known $(1 + \lambda)$ ES already tested for its in performance in [24], [25], [33], and [83], where λ represents the population size (see Fig. 4). First, all the chromosomes are randomly initialized. Second, the fitness value of each individual is computed. Third, the fittest individual is selected. Fourth, the previously selected individual is used to test if the conditions to stop the process have been met. These conditions are: the fitness value of the chromosome is 100% or the number of generations has reached the maximum value set by the user for that particular experiment. If the conditions are not met, a new population will be generated by mutating the best chromosome (selected at the third step) λ times in order to obtain other λ individuals. In the next cycle, all λ newly created chromosomes are evaluated, the fitness value of each of them is compared with the fitness value of the best chromosome of the previous generation, and the best of these $(1 + \lambda)$ individuals is selected.

B. Chromosome Encoding

The chromosome defines the structure of the logic circuit and the connectivity between logic gates. In our approach, the logic circuit has been presented as a rectangular array of logic cells. The type of each logic cell is randomly chosen from the set of

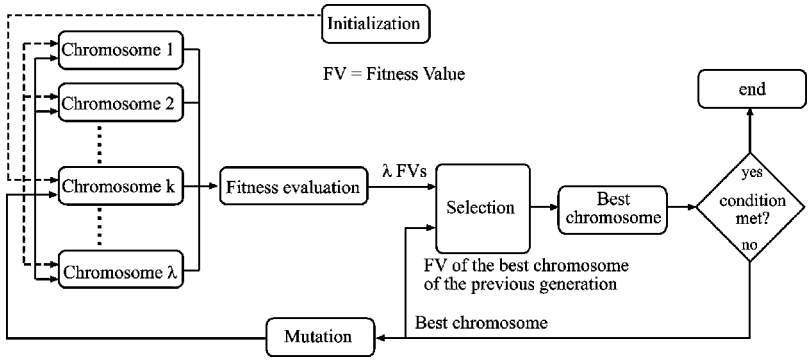

Fig. 4. Schema of $(1 + \lambda)$ ES.

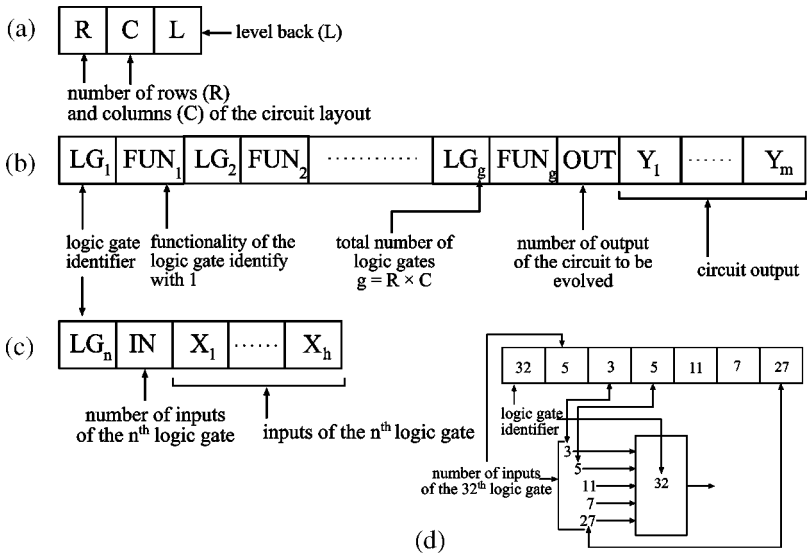

Fig. 5. Chromosome's structure. (a) Geometry level. (b) Functional level. (c) Connection level. (d) Example of a chromosome at the connection level. In this example, the logic gate identified by the number 32 is considered. This logic gate has five inputs, which are taken from the outputs of the logic gates identified by the numbers 3, 5, 11, 7, and 27.

AND, OR, XOR, NOT, and MUX, where MUX is a multiplexer with two inputs and one control signal. The chromosome has been represented by a three-level structure.

- Geometry level [see Fig. 5(a)] contains information about the number of rows, the number of columns of the rectangular array, and the degree of internal connectivity, also referred to as level-back parameter [34]. The level-back parameter defines how many columns of cells to the left of the current column may have their outputs connected to the inputs of the current cell.
- Functional level [see Fig. 5(b)] describes the array of cells and determines the circuit's outputs.
- Connection level [see Fig. 5(c)] represents the structure of each cell in the circuit and the connections between them.

In Fig. 6, an example of chromosome encoding for a circuit layout with three inputs and two outputs is given. The circuit layout is a circuit with two rows and three columns, and the level back is set to three; therefore, the chromosome at geometry level is "2," "3," and "3." The chromosome at functional level encodes the array of cells. For example, the logic gate identified by the output "3" is an XOR gate. XOR is encoded with

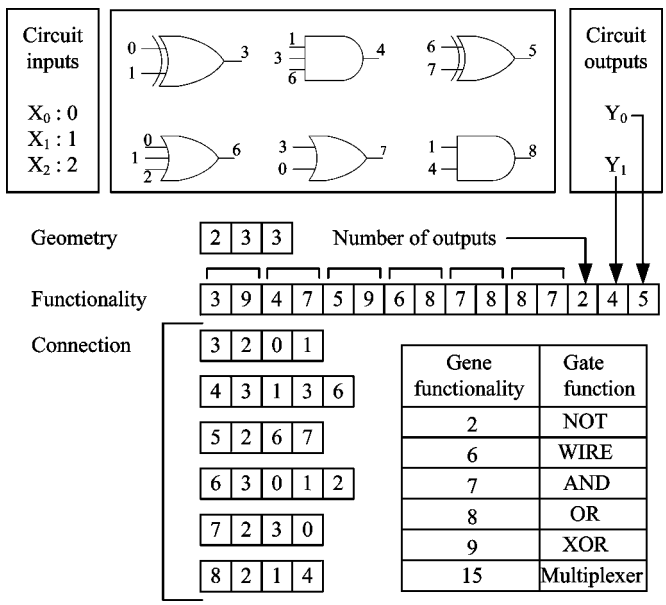

Fig. 6. Example of a chromosome encoding.

the number "9." Thus, the first two cells of this chromosome are "3" and "9." The logic gate with output "4" is an AND gate (encoded with the number "7"); thus, the next two cells of the functional level's chromosome are "4" and "7." All the other logic gates are encoded in the same way. The last two cells of that chromosome contain the outputs of the circuit (in this case, the outputs of the circuit are taken from the output of the logic gates "4" and "5").

The chromosome at connection level identifies the logic gates, its number of inputs, and from which gate's output those inputs are taken. The chromosome of the first logic gate (first row, first column) contains "3" (which is the logic gate identifier), then "2" (which means that that circuit has two inputs), and then "0" and "1," which means that one input is taken from X_0 and the other from X_1. All the other five logic gates are encoded in the same way.

C. Fitness Function

The fitness function evaluates the evolved circuits in terms of their functionality. The fitness function selected for the experiments has two main criteria, namely; 1) design, and, once the circuit is fully functionally evolved, 2) optimization, which leads to reduced numbers of active logic gates used in the circuit configuration.

The fitness function f_{tot} is calculated as

$$f_{\text{tot}} = \begin{cases} f_1, & \text{if } f_{\text{tot}} \leq 100 \text{ design} \\ f_1 + f_2, & \text{if } f_{\text{tot}} > 100 \text{ optimization} \end{cases} \tag{2}$$

where f_1 is a design criterion that defines the percentage of correct output bits produced by the evolved circuit after the application of all possible input combinations. f_2 is the optimization criterion for the optimization stage.

The fitness function for the functionality of the evolved circuit f_1, or the so-called design criterion, has been calculated as

$$f_1 = 100 - \frac{100}{mp} \sum_{f_c=0}^{2^n-1} \sum_{i=0}^{m-1} |y_i - d_i| \tag{3}$$

where m and n are the number of outputs and inputs of the given logic function, respectively, p is the number of input–output combinations, y_i is the ith digit of the output combination produced by the evaluation of the circuit, d_i is the desired output for the fitness case f_c, and $|y_i - d_i|$ is the absolute difference between the actual and the required outputs.

The fitness function for the optimization stage has been calculated below, where N_{LG} is the number of total logic gates present in the chromosome and is equal to the number of rows multiplied by the number of columns of the chromosome. $N_{\text{PLG}}^{\max}$ is the number of primitive logic gates necessary for building the logic gate with the highest number of inputs present inside the chromosome. Fig. 7 shows how to decompose a logic gate with four inputs. Therefore, if a logic gate has four inputs, the number of primitive logic gates necessary to build it is 3. N_{row} and N_{col} are the number of rows and columns of the chromosome, respectively. $N_{\text{PLG}_{(i,j)}}$ is the number of primitive

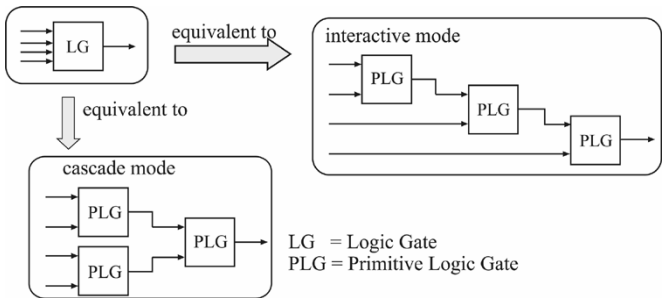

Fig. 7. Decomposition of a logic gate with four inputs into primitive logic gates.

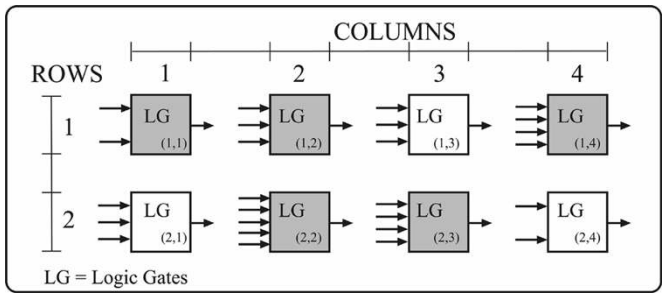

Fig. 8. Example of a possible chromosome configuration.

logic gates necessary to build the (i, j)th logic gates. $N_{\text{PLG}_{(i,j)}}$ is 0 if the (i, j)th logic gate is unconnected, i.e.,

$$f_2 = N_{\text{LG}} N_{\text{PLG}}^{\max} - \sum_{1=1}^{N_{row}} \sum_{j=1}^{N_{\text{col}}} N_{\text{PLG}_{(i,j)}}. \tag{4}$$

An example of how the fitness function for a chromosome during the optimization stage is calculated is given below. Suppose that the chromosome in examination is a rectangular array of two rows and four columns (see Fig. 8) and for a particular configuration the highlighted logic gates are connected (see Fig. 8). Therefore, the total number of logic gates N_{LG} is two rows multiplied by four columns, so $N_{\text{LG}} = 8$.

The logic gate with the highest number of inputs is the cell marked with index (2, 2), which contains five inputs; consequently, $N_{\text{PLG}}^{\max} = 4$. The fitness function for the optimization stage can be calculated as

$$\begin{aligned} f_2 &= N_{\text{LG}} \times N_{\text{PLG}}^{\max} - \sum_{i=1}^{N_{\text{row}}} \sum_{j=1}^{N_{\text{col}}} N_{\text{PLG}_{(i,j)}} \\ &= 8 \times 4 - (1 + 2 + 0 + 3 + 0 + 4 + 3 + 0) = 19. \end{aligned} \tag{5}$$

Fig. 9 shows the behavior of the fitness function during the evolution of functionality for the function

$$f = \lceil sqrt(x) \rceil \tag{6}$$

with four inputs and three outputs. In Fig. 9, two different stages are noticeable. The first shows the design of the function, so with each generation the fitness function value increases until it reaches 100%; therefore, the functionality of the circuit is completely evolved. During the first stage, the fitness function has been calculated using (3). The second stage starts just after the circuit is evolved. This stage performs the optimization of evolved circuits by reducing the number of active logic gates. Furthermore, during this stage, the fitness function, as

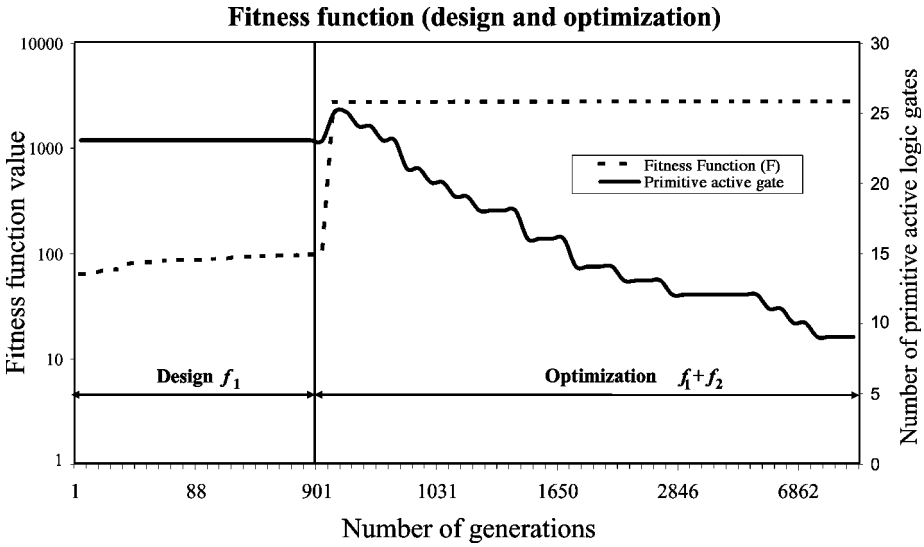

Fig. 9. Fitness function behavior. This graph shows the effect of the use of the selected fitness function during the evolution of a logic circuit. When the fitness value reaches 100% (functionality evolved), the optimization process begins.

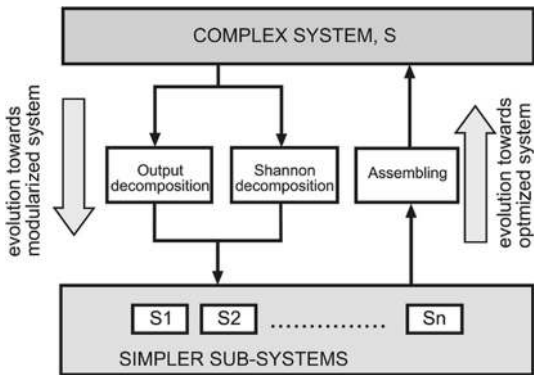

Fig. 10. BIE approach.

calculated using (4), also increases its value because the circuit is better optimized.

D. BIE

BIE [62], [84] operates by gradually decomposing a complex system into a series of simpler ones when the evolution does not bring any improvement in terms of fitness function value (see Fig. 10). These simpler blocks are evolved separately and then merged together once completely developed. If, during the evolution of each single subsystem, the stalling effect in the fitness function (see Fig. 3) occurs again, the single subcircuits will be decomposed again and again until all the subcircuits are simple enough to be evolved. The systems are decomposed by using Shannon and output decomposition (see Fig. 2). Evolution occurs on both sides. First toward modularization (having simpler and smaller logic circuits) and second toward an optimized system by assembling the simpler subcircuits together. This system is completely automatic and does not require any human intervention.

E. Genetic Operators

Gene mutation and elitism [21] have been used in our extrinsic EHW system. Elitism ensures that the best individual of one generation is transferred to the next one. The mutation operator is involved in changing the value of some genes inside the chromosome. The aim of this operation is to bring more change (diversity) into the population. By increasing the mutation rate, the genetic search will be transformed into a random search but will help to reintroduce lost genetic material [22].

TABLE II
INITIAL DATA FOR THE EXPERIMENTS CARRIED OUT WITH $(1+\lambda)$ ES

Number of generations		800.000
Population size, λ		5
Number of runs per each experiment		For circuits with up to 4 inputs 100 runs. For circuits with 5 and 6 inputs 50 runs.
Elitism is applied		
Gene mutation rate		0.05
Circuit layout	Rows	10
	Columns	10
	Level Back [34]	10

V. LIMITATIONS OF EHW EVOLUTION

In order to identify the limitations of the previous systems, a number of experiments have been carried out. The purpose of these experiments was to quantify how the performance of the evolutionary process is dependent on the complexity of the tasks used. Logic circuits have been evolved using the extrinsic EHW approach with $(1+\lambda)$ ES described in detail in the previous section, with $\lambda = 5$. Each logic circuit, randomly generated, has been evolved either 50 or 100 times, and the average number of generations needed for each run has been reported. The initial configuration used for evolving the logic circuits has been set out in Table II, where all the parameters for the ES are outlined. The obtained results were classified according to the number of inputs, number of outputs of logic circuits, and number of generations required to evolve such circuits. In Fig. 1, the relationship between the dimension of the circuits and the required number of generations in order to evolve them has been considered. The experimental results have shown that the number of generations required in order to evolve a

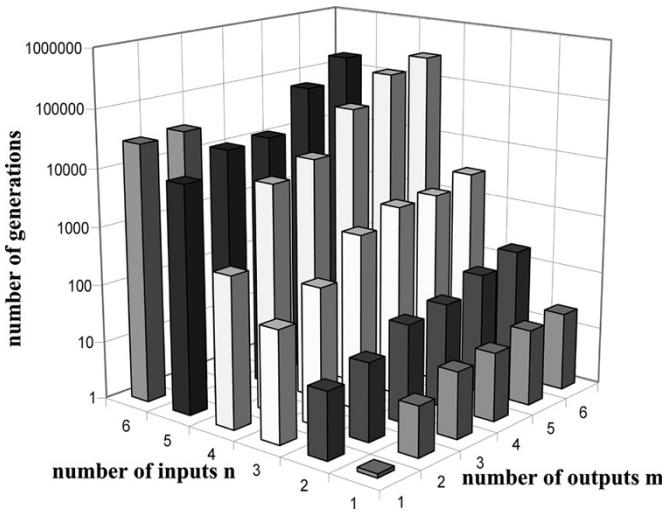

Fig. 11. Average number of generations required to evolve logic circuits with n inputs and m outputs using $(1 + \lambda)$ ES.

logic circuit is mainly dependent on the number of inputs. The system set-up together with the EHW algorithm used was able to evolve only the circuit for which the results are given in Fig. 11. The considered system has not been able to evolve more complex logic circuits. Based on the obtained results, one may conclude that there is a need for the development of a method that would concentrate on the input decomposition for EHW systems. That method is proposed in the next section. The experimental results prove that such a method will tackle the scalability problem better than the methods focused on output decomposition. This paper is devoted to proposing that method.

Therefore, a new system that is capable of reducing the number of required generations and at the same time improving the fitness values and evolving larger logic circuits has been introduced.

VI. Generalized Disjunction Decomposition

In this section, the proposed method, GDD, which speeds up the evolutionary process and optimizes the logic circuit, is explained. This method improves the scalability for evolving logic circuits. This method has been introduced as a result of the limitation of EHW evolution demonstrated in the previous section: the number of generations required to evolve a circuit is mainly dependent on the number of inputs rather than on the number of outputs.

A. Proposed Method

This method is based on rewriting the truth table in such a way that the inputs needed to describe the system are decomposed in two parts. Supposing that a system with n inputs and m outputs, see Fig. 12(a), should be evolved using either the extrinsic EHW approach previously described or other EAs. The functionality of this system can be described by the truth table given in Fig. 12(b), where $p = 2^n$ is the number of products (or the so-called number of input–output combinations). The system depicted in Fig. 12(a) can be decomposed into two subsystems as shown in Fig. 13(a).

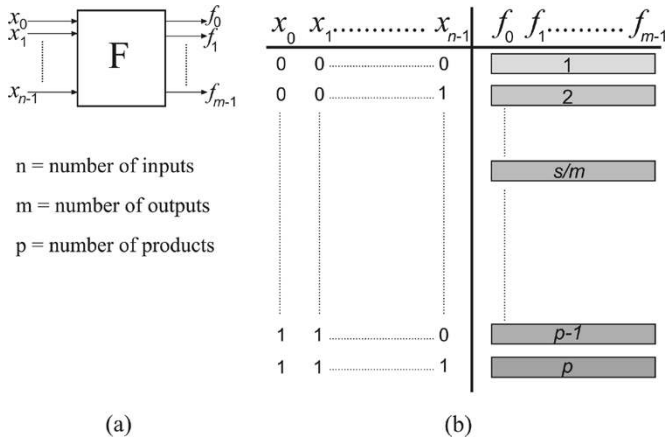

Fig. 12. General description of logic circuits.

Subsystem G with r inputs and s outputs represents the evolvable part of the newly created system, where

$$s = m \times 2^{n-r} \tag{7}$$

and subsystem H with $(s + n - r)$ inputs and m outputs represents the fixed part of the circuit that is generated using multiplexers. This part does not participate in the evolutionary process. The structure of this subcircuit depends on the number of used inputs and outputs. The next section gives a description of the system H. The decomposition of the system F into the two subsystems (G and H) is done automatically. At the moment, the user decides how many inputs G should have. Then the initial system F, defined by its truth table, is decomposed. Software in C++ has been written to accomplish this decomposition. Subsystem G can be evolved using either the traditional EHW approach or any other scalable approach such as divide-and-conquer, BIE, etc. The complexity of the evolutionary process will depend on the type of method used. Let us consider the process of generating the truth table for subsystem G. Let us assume that r should be always less than $n (r < n)$, where r is the number of inputs in the G subsystem and n is the number of inputs in the initial system. The new truth table, shown in Fig. 13(b), has been calculated by applying the following procedure.

- Generation of all the input combinations of the truth table G.
- Identification, per each input combinations, of the truth table G, s/m output combinations on truth table F, wherever the input combinations of the truth table G match in sequence the input combinations of the initial system (truth table F).
- The outputs, of the initial system (truth table F) previously identified, become the outputs of the reduced truth table (output of G).

B. Multiplexer Part

In this section, the complexity of subsystem H, the multiplexer part, is shown. A multiplexer with two inputs and one control signal is composed of four logic gates: one NOT, two AND, and one OR, as shown in Fig. 14.

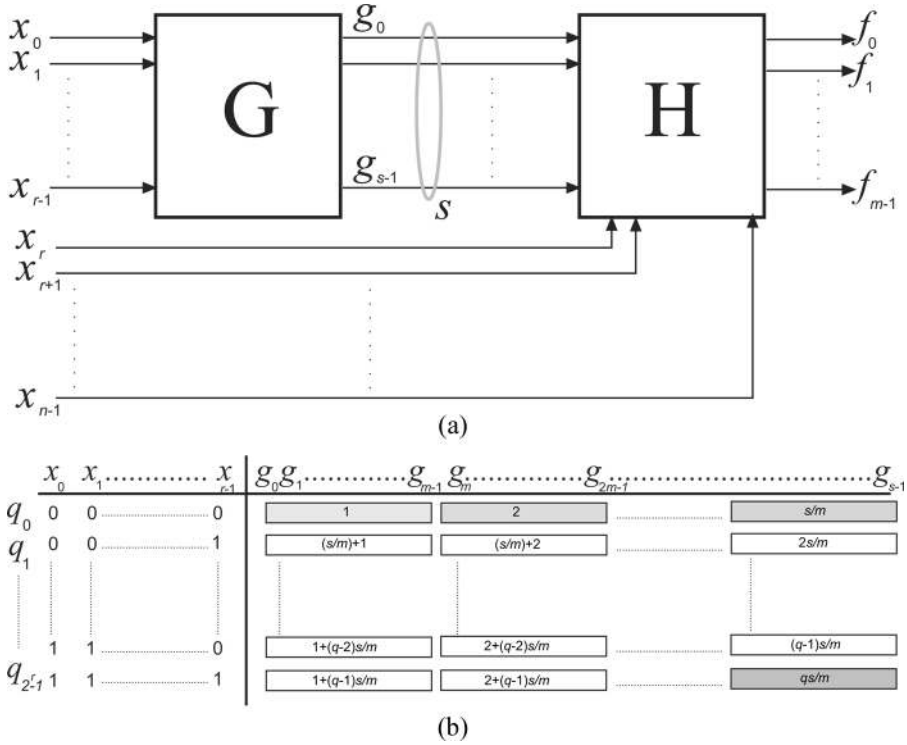

Fig. 13. (a) Proposed decomposition of the initial logic circuit. r and g refer to the number of inputs and outputs of the reduced subsystem, respectively. (b) Truth table of the evolved part of the proposed subsystem.

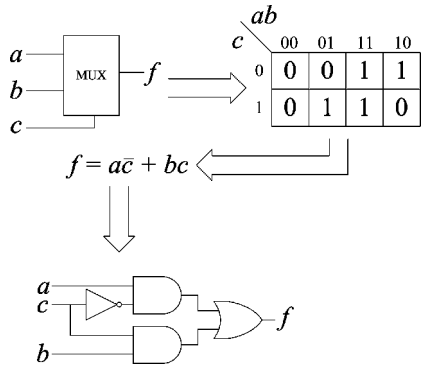

Fig. 14. Multiplexer with one control signal and how it is built with logic gates.

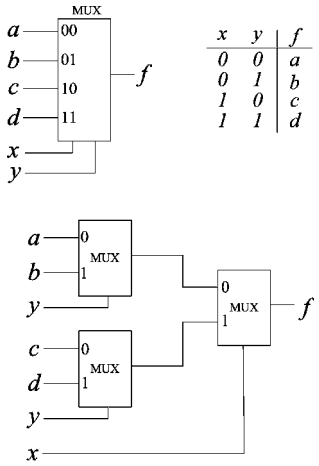

Fig. 15. Multiplexer with two control signals.

In Fig. 15, a multiplexer with two control signals is shown, as well as the process for constructing it. The number of logic gates required for a generic multiplexer is given by

$$N_{\lg} = 4(2^c - 1) \tag{8}$$

where $N_{\lg}$ is the number of logic gates required to build the multiplexer with c control signals.

C. Case Study

In this section, a case study regarding the generation of a truth table for subsystem G based on a simple example is considered. The truth table corresponding to the function given in (6) with four inputs and three outputs has been taken into account as an initial system to be decomposed. Therefore, the system F has $n = 4$ [number of inputs (x_0, x_1, x_2, x_3)], $m = 3$ [number of outputs (f_0, f_1, f_2)], and $p = 16$ (number of products or input combinations).

The truth table of this function is shown in Fig. 16(a). Supposing that a subsystem G with only two inputs is to be generated. Therefore, the new system will have $r = 2$ [number of inputs of the subsystem G (x_0, x_1)] and $s = 12$ [number of outputs of G, calculated based on (7)].

The new system is shown in Fig. 17. In order to generate the truth table of G, the procedure described in the previous section should be followed. First, all of the input combinations have been generated. Therefore, in this case, the input combinations of subsystem G are 4: $(x_0, x_1) = (00), (01), (10), (11)$. Second, for each input combination generated, the s/m output combinations of the initial system, wherever the input

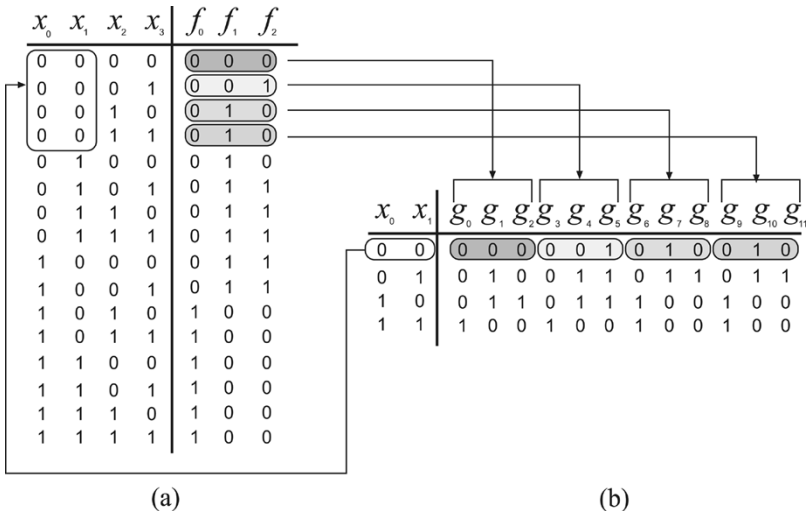

Fig. 16. (a) Truth table of the initial function F. (b) Newly generated truth table for subsystem G.

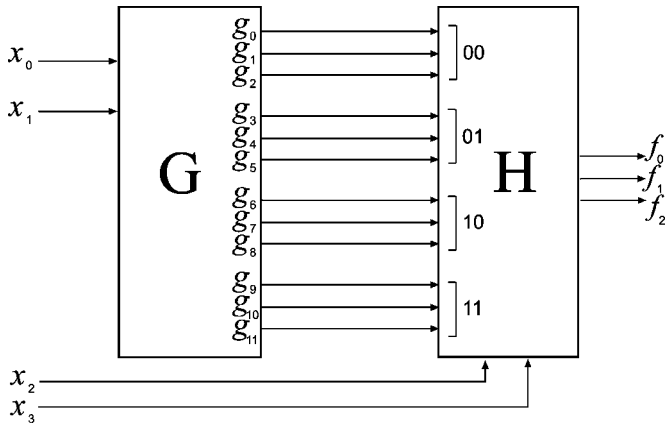

Fig. 17. Decomposed subsystems obtained using generalized decomposition strategy. G is the evolvable part, and H is the multiplexer part.

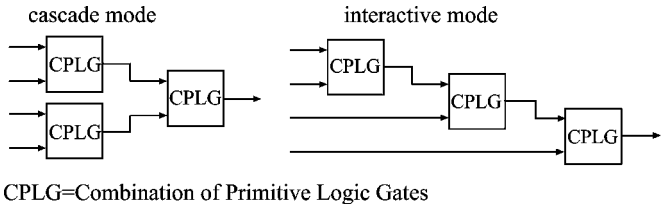

Fig. 18. Connections between building blocks in cascade and interactive mode. This figure explains how building blocks could be connected inside the circuit layout. The mode in which they are connected is automatically selected during the evolution.

combinations of the truth table G match in sequence the input combinations of F, should be identified. Let us identify the $s/m = 4$ output combinations for the input combination $(x_0, x_1) = (0, 0)$. The truth table F has been examined and the output of all the s/m input combinations that include (0, 0) for (x_0, x_1) have been considered as outputs for subcircuit G when $(x_0, x_1) = (0, 0)$. Once the outputs for the input combination $(x_0, x_1) = (0, 0)$ have been generated, the input combination (0, 1) is considered. The truth table generated for subsystem G is given in Fig. 16(b).

The number of logic gates required to implement the fixed part (H-system) of the new system, based on (8), is 12.

VII. Setting Parameters

In this section, the system setup used to simulate the GDD is provided. The building blocks (a combination of primitive logic gates) that participate in evolutionary processes are AND, OR, XOR, NOT, and a multiplexer with two inputs and one control. Each logic gate has up to four inputs. The connections between building blocks inside the chromosome are interactive and in cascade mode (see Fig. 18). The mode in which they are connected is automatically selected. Furthermore, the output of a primitive logic gate of the nth columns could be connected to the input of a primitive logic gate of $(n - l)$th columns with $l < n$, where l represents the level back [34], and of $(n + k)$th columns, where $k \geq 0$ and $(n + k)$ is less than or equal to the number of columns of the circuit layout.

The truth tables used to describe the logic circuits are compatible with the Berkeley standard format for the physical description of a PLA [119].

The system used for evolving circuits with BIE is shown in Fig. 10, while for the GDD the schema is shown in Fig. 19.

In Table III, EA's parameters are given, where the number of generations refers to the number of cycles that each experiment has been evolved, population size refers to the number of different chromosomes, and gene mutation rate and elitism are the genetic operators used.

Each logic circuit has been evolved 100 times. For the given logic functions, the results are considered only if the logic circuit has a success rate of 100%. In Table IV, the features of the circuit layout used during evolution are given. The size of the chromosome (i.e., circuit layout) is chosen according with the complexity of the task being evolved. The more complex the task being evolved, the larger the circuit layout selected. Those parameters are not chosen according to previously published results; rather, they are tuned in order to obtain the best results. Definitions of the number of rows, columns, and level back have been provided in Section IV-B. The experiments are run with a

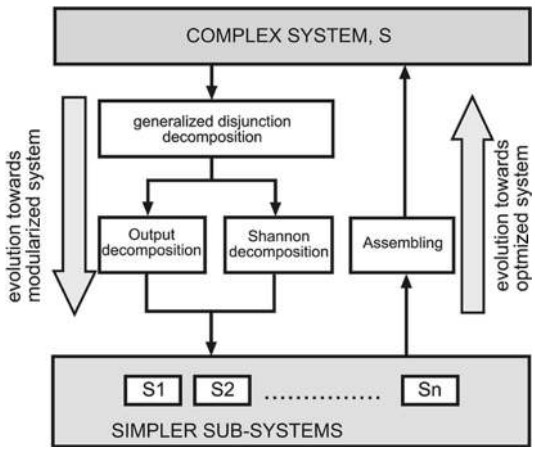

Fig. 19. System used for evolving logic circuits. The GDD is implemented into BIE.

TABLE III
INITIAL PARAMETER FOR THE EXPERIMENT CARRIED OUT WITH BIE AND WITH THE PROPOSED APPROACH

Termination criteria	1,000,000 generations or 2000 generations without improvements of the fitness value
Population size	5
Number of runs for each logic circuit	100
Gene mutation rate	0.05
Elitism	

desktop PC with a Pentium IV at 3.00 GHz and 768 MB of RAM. The software is written in C++.

VIII. EXPERIMENTAL RESULTS

In this section, the results of the logic circuits evolved by using the GDD together with BIE are shown.

The aim of the experiments is to prove that the proposed method requires less generations and improves scalability for designing logic circuits in EHW in comparison with existing evolutionary computation methods. As the standard EHW approach, which does not use any decomposition technique, has high limitations in the evolution of relatively large logic circuits, it was decided not to compare the results of this method with the results obtained with GDD. The logic circuits analyzed in this paper have been taken from different sources: some of them were randomly generated, others were taken from MCNC benchmark [81], others describe the behavior of multipliers of different complexity, and others are even n-bit parity circuits traditionally used within the EHW community. In this section, another benefit of the use of GDD is also shown: GDD is capable of reducing the time required for a single iteration, therefore reducing the time required for the entire evolutionary process.

A. Evolving Randomly Generated Logic Circuits

The circuits analyzed here are randomly generated. The complexity of these circuits is quite low; therefore, it is easy to evolve them. However, it has been decided to evolve those circuits in order to show how the use of GDD could bring some benefits to the evolution of small logic circuits over BIE.

TABLE IV
INITIAL DATA. DIMENSION SIZE AND CONNECTIVITY OF THE CIRCUIT LAYOUT USED DURING SIMULATIONS

Circuit name	Circuit layout used		
	Number of Rows	Number of Columns	Level Back [34]
Logic circuits randomly generated			
6-5	3	80	80
6-4	3	80	80
6-4	10	10	10
6-3	10	10	10
6-2	10	10	10
6-1	10	10	10
6-5	10	10	10
Multiplier circuits			
Mult3	10	10	10
Mult4	10	10	10
Mult5	3	80	80
Mult6	3	80	80
Even parity circuits			
Party13e	3	80	80
Parity14e	3	80	80
Parity15e	3	80	80
Parity16e	3	80	80
Parity17e	3	80	80
Logic circuits taken from MCNC benchmark library			
Majority	10	10	10
9sym	3	80	80
add2_7	10	10	10
5xp1	3	80	80
addm4	3	80	80
co14	3	80	80
con1	3	80	80
rd84	3	80	80
t841	4	100	100
cm162	3	80	80
add6	3	80	80
alu4	3	80	80

In Table V, the experimental results are shown. The circuits evolved with the GDD are better optimized and the number of generations is significantly reduced. In that table, all the characteristics of the circuits have been given: name, number of inputs, outputs, and products. For example, by looking at the logic circuit 6–4, it has six inputs, four outputs, and 64 input–output combinations. Then, the numbers of generations (average out of 100 runs and the best run) needed for evolving the logic circuits have been reported. The next two columns give the average and best times (values are expressed in seconds) for each experiment. The next two columns provide the value of the fitness function for the final optimized solutions. For circuits 6–4, it can be observed that it is evolved using BIE (first row, six inputs and four outputs) and a different configuration (four inputs and 16 outputs) is obtained using the GDD.

B. Evolving Multipliers

The evolution of multipliers is a quite difficult task. Several researchers have recognized the importance of the evolution of these circuits, and in the past few years this benchmark has become widely used within the EHW community. In Table VI, experimental results regarding the evolution of multipliers are given. Like BIE, GDD is not able to evolve a six-bit multiplier within the maximum number of generations set for this experiment. Therefore, the maximum number of generations for this

TABLE V
EXPERIMENTAL RESULTS FROM BIE AND GDD, WHERE IN, OUT, AND P ARE THE NUMBER OF INPUTS, OUTPUTS, AND PRODUCTS IN THE GIVEN LOGIC FUNCTION. EACH LOGIC CIRCUIT HAS BEEN EVOLVED 100 TIMES WITH A SUCCESS RATE OF 100%

Info circuit					**Experimental results – Logic circuits randomly generated**					
					Number of generations performed		**Total time spent per each experiment in seconds**		**Final fitness value**	
name	**method**	**in**	**out**	**p**	**average**	**best**	**average**	**best**	**average**	**best**
6-5	BIE	6	5	64	40,425	26,381	1,194	624	13,815	25,088
	GDD	4	20	16	16,121	9,446	410	257	15,658	17,931
6-4	BIE	6	4	64	30,095	14,587	1,099	442	11,866	20,456
	GDD	4	16	16	10,507	6,744	321	159	23,937	34,030
6-3	BIE	6	3	64	30,754	16,099	366	229	3,498	6,404
	GDD	4	12	16	8,251	4,489	129	64	7,458	10,972
6-2	BIE	6	2	64	12,886	4,160	289	94	1,598	3,486
	GDD	4	8	16	3,500	564	54	12	2,152	5,614
6-1	BIE	6	1	64	10,784	4,406	136	59	1,483	3,887
	GDD	3	8	8	3,684	1,575	62	25	3,049	5,102

TABLE VI
EXPERIMENTAL RESULTS FROM BIE AND GDD. EACH LOGIC CIRCUIT HAS BEEN EVOLVED 100 TIMES WITH A SUCCESS RATE OF 100%. FOR THE EVOLUTION OF THE SIX-BIT MULTIPLIER, THE MAXIMUM NUMBER OF GENERATIONS WAS SET TO 3 000 000 BECAUSE OF THE DIFFICULTIES TO EVOLVE IT. ONLY THREE RUNS HAVE BEEN CARRIED OUT FOR THIS MULTIPLIER AS THE AVERAGE REQUIRED EVOLUTION TIME IS 48.48 h

Info circuit					**Experimental results – Multipliers**					
					Number of generations performed		**Total time spent per each experiment in seconds**		**Final fitness value**	
name	**method**	**in**	**out**	**p**	**average**	**best**	**average**	**best**	**average**	**Best**
Mult3	BIE	6	6	64	21,948	9,030	288	126	4,279	2,373
	GDD	4	24	16	9,156	4,434	123	67	8,820	14,219
Mult4	BIE	8	8	256	146,663	117,495	1,718	1,468	13,019	20,592
	GDD	6	32	64	87,411	70,999	1,040	594	23,554	30,926
Mult5	BIE	10	10	1024	740,164	685,372	24,088	17,458	48,786	52,860
	GDD	8	40	256	506,347	482,789	16,033	15,338	152,372	171,368
Mult6	BIE	12	12	4096	Not evolved					
	GDD	10	48	1024	2,665,547	2,582,678	174,528	156,924	517,667	578,296

circuit was increased to 3 000 000. In [117], it has already been proven for one run that GDD is able to solve this multiplier, but only by using a higher number of generations. The six-bit multiplier is also evolved using another configuration of GDD, with nine inputs and 96 outputs. Three experiments have been carried out with this configuration, and the average number of generations required is 2 536 135 while the average time per experiment is 32.00 h; a bit less than the configuration with ten inputs.

C. Evolving Even n-Bit Parity Function

Parity functions are often used to check the accuracy of stored or transmitted binary data in computers because a change in the value of any one of its arguments toggles the value of the function. As a result of this sensitivity to its inputs, the parity function is difficult to learn [29]. This benchmark, as with the multipliers, is popular within the evolutionary computation community. The even n-bit parity circuit produces a response equal to the sum modulo 2 of its inputs. Table VII shows the experimental results for the evolution of those circuits. From that table, it can be observed that with the use of GDD the evolution of a large circuit is feasible. GDD is able to evolve large even parity bit circuits never evolved before. The best parity bit circuit successfully evolved previously was presented in [86], and it was a 12-bit parity circuit. It was evolved using a developmental model based on a biological map between genotype and phenotype.

D. Evolving Logic Circuits From MCNC Benchmark

The experimental results reported in this section are those in relation to the evolution of circuits taken from the MCNC benchmark [81], [110]. This benchmark was first presented at the 1990 International Workshop on Layout Synthesis. Since then, several researchers within the GA community have used it as a test bench for floor planning [111], cell placement [112], [113], power consumption [114], etc. It is still not popular within the EHW community because of the complexities of those circuits. However, here, it has been decided to evolve those circuits in order to show that GDD has the capability to cope with these circuits. Figs. 20–23 show the relationships between the values of the fitness function, the number of generations, and the time spent for each experiment. "Reduced circuit" refers to a circuit that has been obtained by applying the proposed method to the original circuit. Each

TABLE VII
EXPERIMENTAL RESULTS FROM BIE AND GDD. THE LOGIC CIRCUITS PARITY16E AND PARITY17E HAVE BEEN EVOLVED TEN TIMES, THE OTHERS 100 TIMES. ALL THE CIRCUITS HAVE BEEN EVOLVED WITH A SUCCESS RATE OF 100%

Info circuit					**Experimental results – Parity Functions**					
					Number of generations performed		**Total time spent per each experiment in hours**		**Final fitness value**	
name	**method**	**in**	**out**	**p**	**average**	**best**	**average**	**best**	**average**	**best**
Parity11e	BIE	11	1	2,048	4,679	304	0.530	0.089	2,538	9,954
	GDD	7	16	128	17,509	2,313	0.223	0.039	12,613	36,701
Parity12e	BIE	12	1	4,096	Not evolved					
	GDD	8	16	256	17,571	2,210	0.374	0.066	13,323	34,727
Parity13e	BIE	13	1	8,192	Not evolved					
	GDD	9	16	512	7,631	2,562	0.270	0.102	9,601	18,021
Parity14e	BIE	14	1	16384	Not evolved					
	GDD	10	16	1,024	15,526	3,753	0.451	0.139	6,375	22,450
Parity15e	BIE	15	1	32,768	Not evolved					
	GDD	10	32	1,024	27,422	7,147	1.878	0.659	34,143	76,996
Parity16e	BIE	16	1	65,536	Not evolved					
	GDD	10	64	1,024	70,100	40,912	4.667	2.668	50,412	87,020
Parity17e	BIE	17	1	131,072	Not evolved					
	GDD	11	64	2,048	123,417	80,125	15.394	10.6128	88,568	117,127

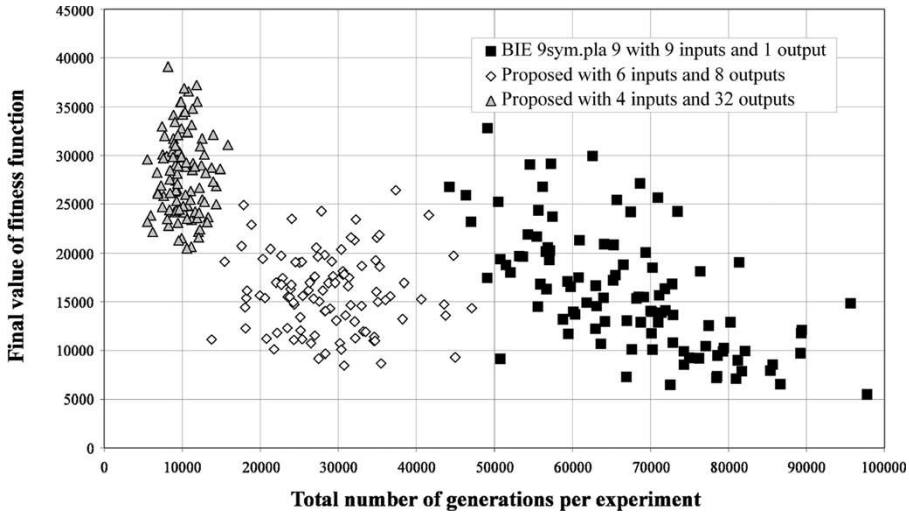

Fig. 20. Relationship between the final value of the fitness function reached at the end of each experiment and the number of generations required for evolving the circuit 9sym. The circuit has been evolved 100 times, and all the results are shown.

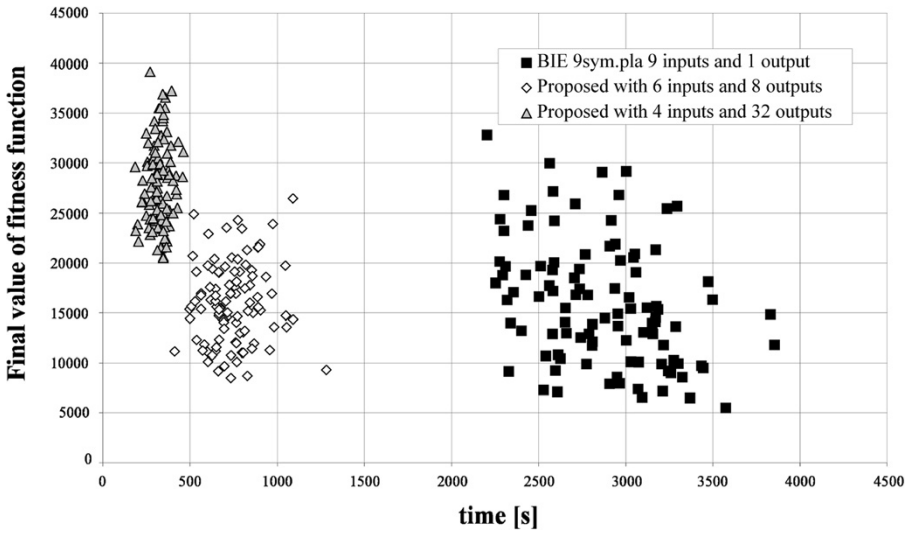

Fig. 21. Relationship between the final value of the fitness function reached at the end of each experiment and the CPU time required for evolving the circuit 9sym. The circuit has been evolved 100 times.

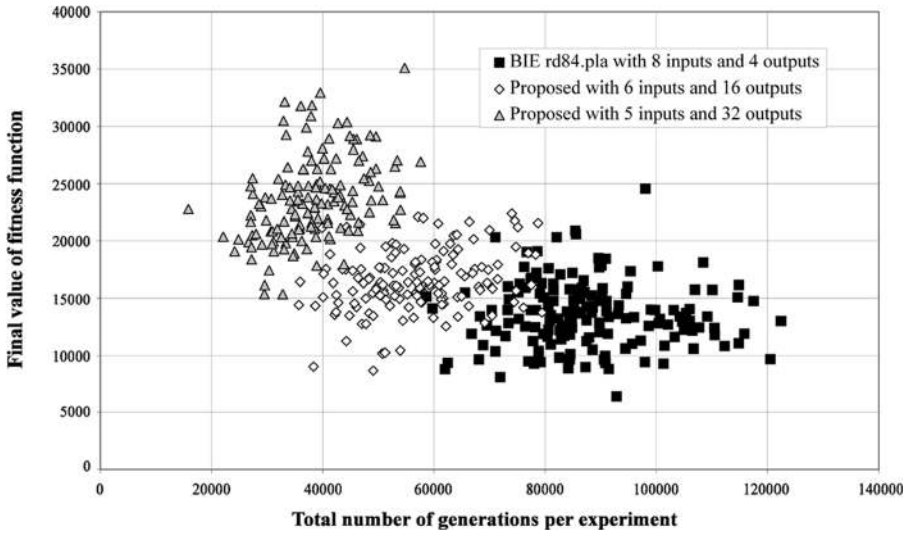

Fig. 22. Relationship between the final value of the fitness function reached at the end of each experiment and the number of generations required for evolving the circuit rd84. The circuit has been evolved 100 times.

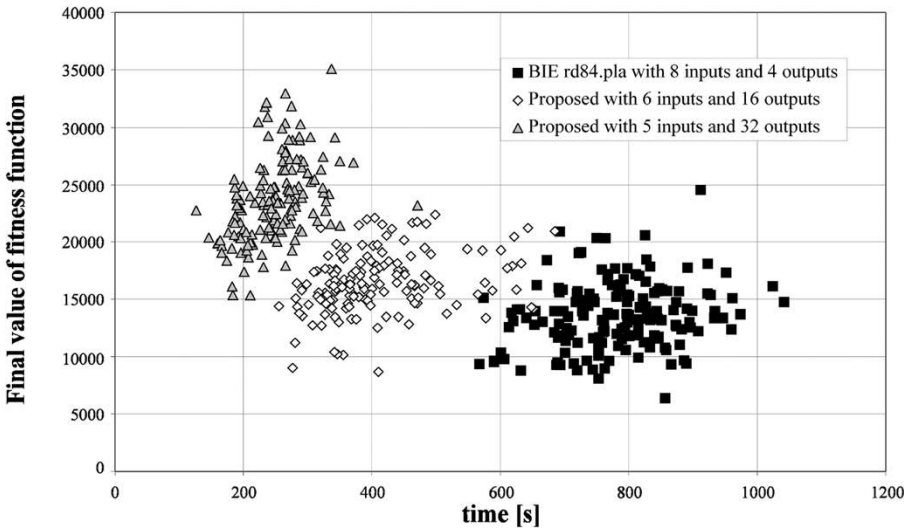

Fig. 23. Relationship between the final value of the fitness function reached at the end of each experiment and the CPU time required for evolving the circuit rd84. The circuit has been evolved 100 times.

graph compares the evolution results between the original logic circuits and two different configurations of the "reduced circuit." For all the evolved logic circuits, it has been observed that a smaller number of generations is required for the evolution of the "reduced circuits" and at the same time better values of fitness are achieved. The same results for the time required for each experiment have been found. For any of the given graphs, it can be seen that when the number of inputs is reduced the fitness value for each experiment is increased and the time and number of generations are reduced. In Table VIII, all the experimental results obtained by using BIE and GDD are shown. It should be noted that for the evolution of the circuit alu4, only five runs are analyzed and the maximum number of generations for those experiments was set at 2 000 000; this was done because of the high computational time required for the simulations. Even with this initial set up, BIE was not able to completely solve the task.

E. Reduction of Time Required for a Single Iteration

Another benefit of GDD is that it is able to reduce the time required for a single iteration μ [see (1)]. This leads to a reduction of the total time required for the entire evolutionary process.

To illustrate this advantage, the time required for a single iteration for circuits with different numbers of inputs has been calculated. The system set up for those simulations is a circuit layout with three rows and 80 columns and level back equal to the number of columns.

The logic circuit in examination was the 17-bit even parity. Using GDD, circuits with 16 inputs and two outputs, with 15 inputs and four outputs, and so on, down to a circuit with two inputs, were created, and those circuits were used as a test bench for this experiment. As can be observed from the results shown in Fig. 24, the required time for a single iteration increases with the number of inputs: going from 16.172 ms for a circuit with two inputs to 35.79 s for a circuit with 17 inputs.

TABLE VIII
EXPERIMENTAL RESULTS FROM BIE AND GDD. EACH LOGIC CIRCUIT HAS BEEN EVOLVED 100 TIMES, WITH THE EXCEPTION OF alu4, WHICH HAS BEEN EVOLVED FIVE TIMES. ALL EXPERIMENTS HAVE REACHED A SUCCESS RATE OF 100%

Info circuit					Experimental results – Logic circuits from MCNC benchmark					
					Number of generations performed		Total time spent per each experiment in seconds		Final fitness value	
name	method	in	out	p	average	best	average	best	average	best
9sym	BIE	9	1	512	67,041	44,261	2,852	2,204	15,976	32,790
	GDD	6	8	64	28,741	13,771	745	412	15,971	26,448
		4	32	16	10,142	5,540	323	185	28,034	39,128
add2_7	BIE	7	4	128	28,121	10,535	269	112	3,036	5,268
	GDD	5	16	32	11,665	4,358	90	29	7,465	14,190
		4	32	16	7,448	4,541	52	32	13,248	20,455
5xp1	BIE	7	10	128	43,643	22,623	1,878	1,003	16,994	30,008
	GDD	5	40	32	24,560	13116	884	518	51,659	77,001
addm4	BIE	9	8	512	168,053	127206	10,713	8,039	40,847	68,204
	GDD	7	32	128	132,414	103563	4,908	3,753	73,027	94,339
co14	BIE	14	1	16,384	184,476	150,838	70,877	64,222	5,024	7,531
	GDD	10	16	1,024	50,733	14,139	6,240	3479	13,179	33,075
con1	BIE	7	2	128	6,584	2,177	286	126	3,015	8,881
	GDD	5	8	32	7,092	2,307	212	79	10,036	17,760
		3	32	8	4,893	2,553	136	71	22,358	30,441
rd84	BIE	8	4	256	87,752	58,640	781	568	13,571	24,545
	GDD	6	16	64	56,764	35,698	410	256	16,473	22,388
		5	32	32	38,533	15,808	250	126	23,701	35,104
t841	BIE	16	1	65,536	Not evolved					
	GDD	9	128	512	59,7469	463,396	20,250	13,482	396,399	445,554
cm162	BIE	14	5	16,384	Not evolved					
	GDD	10	80	1,024	136,421	103,305	12,167	9,060	54,913	71,887
add6	BIE	12	7	4,096	Not evolved					
	GDD	9	56	512	352,361	270,375	21,652	17,278	76,360	85,417
alu4	BIE	14	8	16,384	Not evolved					
	GDD	10	128	1,024	1,369,394	1,309,923	128,386	115,142	198,294	200,169

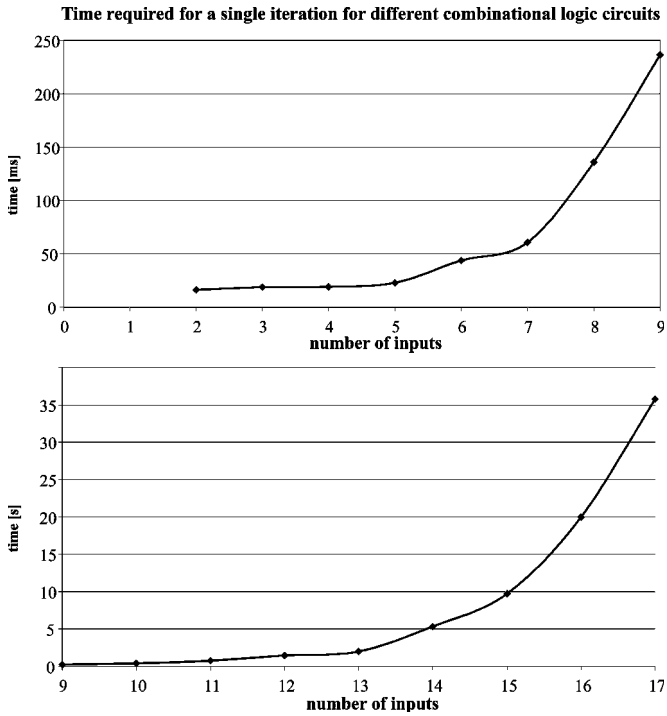

Fig. 24. Time per iteration for circuits with different numbers of inputs.

TABLE IX
REDUCTIONS, EXPRESSED IN PERCENTAGE, OF THE NUMBER OF GENERATIONS, THE TIME SPENT, AND THE IMPROVEMENTS OF THE FITNESS VALUE BY USING THE PROPOSED METHOD COMPARED WITH BIE SOLUTIONS

Name	in	out	Improvements in percentages of GDD over BIE		
			Number of generations	Time	Fitness function's value
Logic circuits randomly generated					
6-5	6	5	60.12%	65.66%	13.34%
6-4	6	4	65.09%	70.79%	101.73%
6-3	6	3	73.17%	64.75%	113.21%
6-2	6	2	72.84%	81.31%	34.67%
6-1	6	1	65.84%	54.41%	105.60%
Logic circuits taken from MCNC benchmark library					
Majority	5	1	82.09%	52.94%	87.34%
9sym	9	1	84.87%	88.67%	75.48%
add2_7	7	4	73.51%	80.67%	336.36%
5xp1	7	10	43.73%	52.93%	203.98%
addm4	9	8	21.21%	54.19%	78.78%
co14	14	1	72.50%	91.20%	162.32%
con1	7	2	25.68%	52.45%	641.56%
rd84	8	4	56.09%	67.99%	74.64%
Multiplier circuits					
Mult3	6	6	58.28%	57.29%	106.12%
Mult4	8	8	40.40%	39.46%	80.92%
Mult5	10	10	31.59%	33.44%	212.33%

These experiments were performed with a personal computer with 768 MB of RAM and a Pentium IV processor at 3.00 GHz.

F. Analysis of Results and Comparison to Techniques

In this section, an analysis of the obtained results is outlined. The experimental results have shown a reduction of the number of generations required to fully evolve combinational circuits for all the benchmarks used following a reduction in time spent performing the evolution.

Furthermore, increased fitness function values have been noticed, which leads on to better optimized logic circuits, although the presented method aims to improve scalability rather than optimization of the evolved logic circuits.

TABLE X
MOST COMPLEX COMBINATIONAL CIRCUITS PREVIOUSLY EVOLVED KNOWN TO THE AUTHORS AND COMPARISON OF THOSE WITH GDD. "IN" REFERS TO THE NUMBER OF INPUTS

Benchmark	Most complex combinational circuits evolved.			
	With other techniques			GDD
	year	IN	Approach	IN
Parity problem	2005	12	It has been evolved by Gordon et al. [86] by modeling the map between genotype and phenotype based on biological development.	17
	2001	9	Evolved by Bleuler et al. using an improved version of Strength Pareto Evolutionary Algorithm [109]. The success rate for that problem was 17 good solutions out 31runs.	
	2004	8	Evolve by Oltean using Traceless Genetic Programming [108]. The author had performed 10 runs, and only at the 9th was able to evolve it. The method used is very fast, since only 10,000 generations have been used, but also a large search space of possible solutions, 5000 individuals have been used for the simulations.	
	2004		It has been evolved by Miller et al. [87]. To solve that circuit an average of 135,056 generations is required. The method used to solve the task is Embedded Cartesian Genetic Programming ECGP [87].	
	2003	5	It has been evolved by Miller et al. in [85] using Developmental Cartesian Genetic Programming DCGP [85] and 100,000 iterations.	
Multiplier circuits	2005	12	Stomeo et al. have presented in [117] the evolution of a 6-bit multiplier based on *programmable logic array* (PLA) structure using GDD and ES with dynamic mutation rate. In [117] only the result of one run was presented without given a statistical analysis.	12
	2003	10	Torresen has evolved this multiplier with increased complexity evolution approach [80] together with partitioned training vector and partitioned training set [70]. The exact number of generations required to evolve that circuit is not given, but it is more than one million.	
	2000	8	Vassilev et al. [44] have evolved this circuit using CGP together with (1+λ) ES. This multiplier was fully evolved after 643,274,721 generations.	
	2000	6	Arslan et al. in [94] have shown the ability to directly evolve this multiplier in a *hardware description language* (HDL) using a genetic algorithm that utilizes a range of macro cells together with primitive logic gates. The average of required generations, as reported in [94] is bigger than 10,000.	
	2000		Evolved by Vassilev et al. in [49]. The simulations have been carried out using CGP together with (1+λ) ES. Small logic circuits were used as building blocks.	
	2003		In [82] Kalganova et al. have evolve this circuit using BIE [62][84] together with an assembling strategy proposed in [82].	
	2005	4	In [88] Hartmann et al. have evolved this multiplier using CGP and investigated its complexity using Lempel-Ziv compression for the chromosome.	
	2004		Hartmann et al. [13] have evolved this multiplier using tournament selection genetic algorithm. Half million was the average number of generations for the evolution of that circuit.	
	2004		Oltean has evolved this circuit using multi expression programming, which is a variant of genetic programming [118].	
	2002		Hartmann et al. [95] have extrinsically evolved this multiplier in a noisy environment. The algorithm used was (1+λ) ES with neutral genetic drift [95]. The aim of the work presented in [95] was not to evolve large combinational circuits, but to investigate noise robustness and fault tolerance.	
MCNC	2000	7	In 0 Kalganova has evolved the z5xp1 circuit, taken from MCNC benchmark. The approach used was the BIE without the assembling strategy proposed in [82]. The circuit was fully evolved after 5,000,000 generations.	14
	2004	4	In [115] Ali et al. have proposed an evolutionary algorithm to design sequential logic circuits. The most complex circuit evolved with the algorithm presented in [115] was the Tav, taken from MCNC benchmark. The succeed rate for the design of that algorithm was 9%.	
	2005	2	In [116] Shanthi at al. have shown the evolution of a sequential circuit with 3 state variables. A two level evolution based on DCGP [85] has been adopted as EA. The total number of generations required to fully evolve that circuit is 48,639.	

The best results for GDD are scalability. Several combinational circuits, taken from different benchmarks, never previously evolved, have been successfully evolved. Table IX shows improvements in terms of the number of generations, evolution time, and fitness values that GDD brought over the BIE evolutionary method. The best result is a reduction of 84% of the number of generations required to fully evolve a circuit. This result was obtained for the circuit 9sym taken from the MCNC benchmark.

In Table X, the most complex parity bit circuits, multipliers, and circuits from the MCNC benchmark before evolving are considered and compared with the results obtained by GDD. A comparison of GDD against the conventional method is not given because the aim of the experiments set in this paper is to show that the proposed method is able to automatically design large combinational logic circuits. The method proposed here does not deal with the optimization of the evolved circuits. It means that hand design logic circuits are better optimized in terms of number of logic gates, but GDD can design the same circuits avoiding the high cost of human design.

In conclusion, the main advantages of this method, as proven from the simulations, are as follows.

- Fewer numbers of generations required to evolve the system in comparison with BIE.
- Less computation time, for two reasons. First, the computation time required is smaller because the number of generations is less [see (1)]. Second, because the truth table treated with GDD is much smaller and can be processed quickly; therefore, the required time for a single iteration is reduced, as exposed in Section VIII-E. For instance, an even parity circuit with 17 inputs (as evolved in Section VIII-C) described with the Berkley format is 2689 KB. The same truth table rewritten with GDD is only 157 KB. Therefore, the necessary time to run a single generation for the EA is reduced.
- Improved optimization of the evolved circuit. The circuits evolved here using GDD have better fitness values when compared with those evolved by BIE, although the aim of the experiments was to show that bigger logic circuits could be evolved and not specifically to show that better optimization could be achieved. To show that the proposed method actually tackles the optimization problem, the number of generations for the experiments would not be set at 1 000 000, rather a much larger number would have been chosen.
- The possibility of evolving larger circuits, as the 17-bit even parity, the six-bit multiplier, and the alu4 (14 inputs).
- GDD is independent of the strategies used; therefore, it could be implemented in different evolvable system environments. Here, it has been implemented into BIE; therefore, the system is completely automatic and does not require any information from the user.

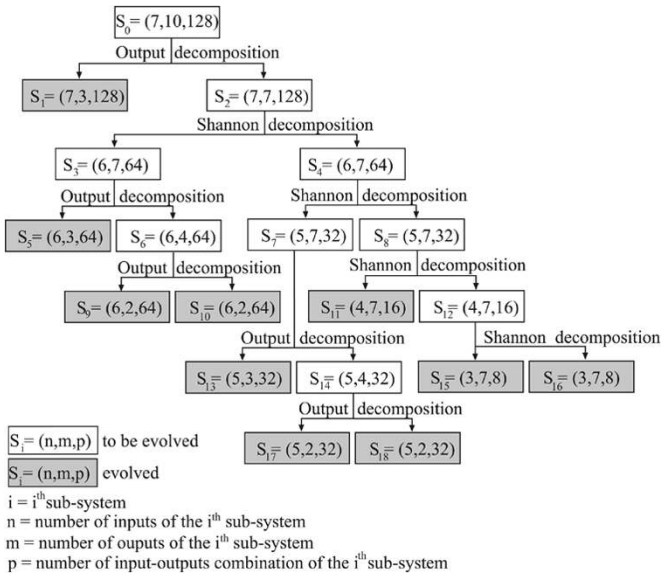

Fig. 25. Example of output and Shannon decompositions of a system (with seven inputs and ten outputs) into several subsystems. The EA is applied untill a subsystem is small enough to be completely evolved. If it will not become evolved within a certain number of generations, it will be decomposed into subsystems.

G. Evolving Complex Logic Circuits

The necessary time to evolve a circuit, as reported in (1), is dependent on the number of required generations and the time necessary for a single iteration. GDD is able to reduce the number of generations and also to reduce the time required for a single iteration as shown in Section VIII-E.

The time required for a single iteration is highly dependent on the complexity of the fitness function and on the hardware used. Further reductions of the evolution time will be achieved by using faster hardware. The experimental results presented here have shown a success rate of 100% for all the analyzed circuits. This success rate is as a result of the nature of the core of the system used, which is BIE [62], [84]. BIE works by progressively reducing the complexity of the logic circuits, which should be evolved. Therefore, a complex task, which is difficult to evolve, will be decomposed until the EA is able to evolve it. In Fig. 25, an example of how the decomposition of the initial system would be done with the use of BIE is reported. The method of decomposing a system into other subsystems (using output and Shannon decomposition see Fig. 2) is completely automatic. The initial system is decomposed into smaller subsystems until the EA is able to evolve them.

IX. Conclusion

In this paper, the GDD for EHW, which is used for the evolution of logic circuits, has been presented and compared with other EHW techniques.

The proposed algorithm has been tested based on the evolution of not only multipliers and even n-bit parity circuits, traditionally used by the EHW community, but also with logic circuits taken from MCNC benchmark library, traditionally used in logic design, and also randomly generated circuits.

The experimental results have confirmed that GDD requires significantly fewer generations to evolve fully functional solutions, reduces the time for a single iteration during the evolutionary process, and allows the evolution of large circuits. The 17-bit parity circuit, the six-bit multiplier, and the alu4, which is a circuit with 14 inputs and eight outputs never evolved before with any other techniques, were evolved using the method proposed here.

GDD demonstrates that is beneficial for solving more complex logic circuits. Furthermore, the evolved circuits have reached higher values of fitness during the optimization stages when compared with BIE, although the work presented here is based on the evolution of large circuits rather than into the optimization.

Acknowledgment

The first author would like to thank A. M. Walsh for help and support, and the BIIS Research Group at Brunel University. The authors acknowledge the anonymous referees.

References

[1] X. Yao and T. Higuchi, "Promises and challenges of evolvable hardware," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 29, no. 1, pp. 87–97, Feb. 1999.

[2] H. de Garis, "Evolvable hardware: Principles and practice," *Commun. ACM*, vol. 40, no. 8, Aug. 1997. [Online]. Available: http://www.cs.usu.edu/~degaris/papers/CACM-Hard.html

[3] M. Sipper and E. M. A. Ronald, "A new species of hardware," *IEEE Spectr.*, vol. 37, no. 3, pp. 59–64, Mar. 2000.

[4] A. Stoica, R. Zebulum, and D. Keymeulen, "Mixtrinsic evolution," in *Proc. 3rd Int. Conf. Evolvable Syst.: From Biol. Hardw.*, Edinburgh, U.K., Apr. 17–19, 2000, pp. 208–217.

[5] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, N. Salami, N. Kajihara, and N. Otsu, "Real-world applications of analog and digital evolvable hardware," *IEEE Trans. Evol. Comput.*, vol. 3, no. 3, pp. 220–235, Sep. 1999.

[6] D. Keymeulen, R. S. Zebulum, Y. Jin, and A. Stoica, "Fault-tolerant evolvable hardware using field-programmable transistor arrays," *IEEE Trans. Rel.*, vol. 49, no. 3, pp. 305–316, Sep. 2000.

[7] A. Stoica, R. S. Zebulum, M. I. Ferguson, D. Keymeulen, V. Duong, and T. Daud, "Evolutionary configuration of field programmable analog devices," in *Proc. IEEE Aerosp. Conf.*, Mar. 8–15, 2003, vol. 6, pp. 2565–2572.

[8] J. Torresen, "An evolvable hardware tutorial," in *Proc. 14th Int. Conf. FPL*, Antwerp, Belgium, Aug. 2004, pp. 821–830.

[9] J. F. Miller, D. Job, and K. Vassilev, "Principles in the evolutionary design of digital circuits—Part I," *Genet. Program. Evolvable Mach.*, vol. 1, no. 1/2, pp. 7–35, Apr. 2000.

[10] A. Thompson, "On the automatic design of robust electronics through artificial evolution," in *Proc. 2nd Int. Conf. ICES*, 1998, pp. 13–24.

[11] P. Haddow and G. Tufte, "Bridging the genotype–phenotype mapping for digital FPGAs," in *Proc. 3rd NASA/DoD Workshop Evolvable Hardware*, Long Beach, CA, 2001, pp. 109–115.

[12] K. C. Tan, L. F. Wang, T. H. Lee, and P. Vadakkepat, "Evolvable hardware in evolutionary robotics," *Auton. Robots*, vol. 16, no. 1, pp. 5–21, Jan. 2004.

[13] M. Hartmann and P. C. Haddow, "Evolution of fault-tolerant and noise-robust digital designs," *Proc. Inst. Elect Eng.—Comput. Digit. Tech.*, vol. 151, no. 4, pp. 287–294, Jul. 18, 2004.

[14] J. D. Lohn, "Experiments on evolving software models of analog circuits," *Commun. ACM*, vol. 42, no. 4, pp. 67–69, Apr. 1999.

[15] J. D. Lohn and S. P. Colombano, "Automated analog circuit synthesis using a linear representation," in *Proc. 2nd Int. Conf Evolvable Syst.: From Biol. Hardware*, 1998, pp. 125–133.

[16] S. V. Hum, M. Okoniewski, and R. J. Davies, "An evolvable antenna platform based on reconfigurable reflectarrays," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Washington, DC, Jun. 29–1, 2005, pp. 139–146.

[17] J. D. Lohn, D. S. Linden, G. S. Hornby, W. F. Kraus, A. Rodriguez, and S. Seufert, "Evolutionary design of an x-band antenna for NASA's space technology 5 mission," in *Proc. IEEE Antenna Propag. Soc. Int. Symp. USNC/URSI Natl. Radio Sci. Meeting*, 2004, vol. 3, pp. 2313–2316.

[18] J. D. Lohn, G. S. Hornby, and D. S. Linden, "An evolved antenna for deployment on NASA's space technology 5 mission," in *Genetic Programming Theory and Practice II*. Boston, MA: Kluwer, ch. 18.

[19] A. M. Tyrrell, R. A. Krohling, and Y. Zhou, "Evolutionary algorithm for the promotion of evolvable hardware," *Proc. Inst. Elect. Eng.—Comput. Digit. Tech.*, vol. 151, no. 4, pp. 267–275, Jul. 18, 2004.

[20] R. J. Terrile, H. Aghazarian, M. I. Ferguson, W. Fink, T. L. Huntsberger, D. Keymeulen, G. Klimeck, M. A. Kordon, L. Seungwon, and P. von Allmen, "Evolutionary computation technologies for the automated design of space systems," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Washington, DC, Jun. 29–1, 2005, pp. 131–138.

[21] D. E. Goldberg, *Genetic Algorithm in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[22] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE JNL Comput.*, vol. 27, no. 6, pp. 17–26, Jun. 1994.

[23] D. B. Fogel, "What is evolutionary computation? "*IEEE Spectr.*, vol. 37, no. 2, pp. 26–32, Feb. 2000.

[24] T. Bäck, F. Hoffmeister, and H. P. Schwefel, "A survey of evolutionary strategies," in *Proc. 4th Int. Conf. Genetic Algorithms*, R. Belew and L. Booker, Eds, San Francisco, CA, 1991, pp. 2–9.

[25] H.-P. Schwefel, *Numerical Optimization of Computer Models*. Chichester, U.K.: Wiley, 1981.

[26] J. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. of Michigan Press, 1975.

[27] M. D. Vose, *The Simple Genetic Algorithm*. Cambridge, MA: MIT Press, 1999.

[28] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[29] ——, *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, MA: MIT Press, 1994.

[30] R. Riolo and B. Worzel, *Genetic Programming Theory and Practice (Genetic Programming)*, vol. 6. Norwell, MA: Kluwer, 2004, p. 336.

[31] C. Ryan, J. J. Collins, and M. O. Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *Proc. 1st Eur. Workshop Genetic Program.*. New York: Springer-Verlag, 1998, vol. 1391, pp. 83–95.

[32] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*. New York: Wiley, 1966.

[33] J. Miller, "An empirical study of the efficiency of learning Boolean functions using a Cartesian genetic programming approach," in *Proc. Genetic Evol. Comput. Conf.*, Orlando, FL, Jul. 1999, vol. 1, pp. 1135–1142.

[34] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Proc. EuroGP*, vol. 1802, R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Forgaty, Eds. Edinburgh, U.K., Apr. 16, 2000, pp. 121–132.

[35] M.-S. Ko, T.-W. Kang, and C.-S. Hwang, "Function optimisation using an adaptive crossover operator based on locality," *Eng. Appl. Artif. Intell.*, vol. 10, no. 6, pp. 519–524, Dec. 1997.

[36] E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs Paralleles, Reseaux et Systems Repartis*, vol. 10, no. 2, pp. 141–171, 1998.

[37] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 6, no. 5, pp. 443–462, Oct. 2002.

[38] A. D. Bethke, "Comparison of genetic algorithms and gradient-based optimizers on parallel processors: Efficiency of use of processing capacity," Logic Comput. Group, Univ. Michigan, Ann Arbor, MI, Tech. Rep. 197, 1976.

[39] G. Harik, F. Lobo, and D. Goldberg, "The compact genetic algorithm," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 287–297, Nov. 1999.

[40] C. A. Coello, A. D. Christiansen, and A. A. Hernández, "Use of evolutionary techniques to automate the design of combinational circuits," *Int. J. Smart Eng. Syst. Des.*, vol. 2, no. 4, pp. 299–314, Jun. 2000.

[41] J. Torresen, "Possibilities and limitations of applying evolvable hardware to real-world applications," in *Proc. 10th Int. Conf. Field Program. Logic Appl.*, Villach, Austria, 2000, pp. 230–239.

[42] T. Higuchi, M. Iwata, I. Kaijitani, M. Murakawa, S. Yoshizawa, and T. Furuya, "Hardware evolution at gate and function level," in *Proc. Int. Conf. Biologically Inspired Auton. Syst.: Comput. Cogn. Action*, Durham, NC, Mar. 1996, pp. 4–5.

[43] T. Higuchi, M. Iwata, I. Kajitani, H. Yamada, B. Manderick, Y. Hirao, M. Murakawa, S. Yoshizawa, and T. Furuya, "Evolvable hardware with genetic learning," in *Proc. IEEE ISCAS*, May 12–15, 1996, vol. 4, pp. 29–32.

[44] D. Job, V. K. Vassilev, and J. Miller, "Towards the automatic design of more efficient digital circuits," in *Proc. 2nd NASA/DoD Workshop Evolvable Hardware*, Jul. 13–15, 2000, pp. 151–160.

[45] E. Stomeo and T. Kalganova, "Improving EHW performance introducing a new decomposition strategy," in *Proc. IEEE Conf. Cybern. Intell. Syst.*, Singapore, Dec. 1–3, 2004, pp. 439–444.

[46] C. C. Santini, J. F. M. Amaral, M. A. C. Pacheco, and R. Tanscheit, "Evolvability and reconfigurability," in *Proc. IEEE Int. Conf. Field-Program. Technol.*, 2004, pp. 105–112.

[47] S. Verel, P. Collard, and M. Clergue, "Scuba search: When selection meets innovation," in *Proc. CEC*, Jun. 19–23, 2004, vol. 1, pp. 924–931.

[48] J. F. Miller and P. Thomson, "Aspects of digital evolution: Evolvability and architecture," in *Proc. Parallel Probl. Solving Nature V*, A. E. Eiben, T. Bäck, M. Schoenauer, and H. P. Schwefel, Eds., Lecture Notes in Computer Science, vol. 1498, Berlin, Germany, Springer-Verlag 1998, pp. 927–936.

[49] V. K. Vassilev and J. F. Miller, "Scalability problems of digital circuit evolution evolvability and efficient designs," in *Proc. 2nd NASA/DoD Workshop Evolvable Hardware*, Jul. 13–15, 2000, pp. 55–64.

[50] H. Hemmi, J. Mizoguchi, and K. Shimohara, "Development and evolution of hardware behaviours," in *Toward Evolvable Hardware: The Evolutionary Engineering Approach*, vol. 1062, E. Sanchez and M. Tomassini, Eds. Berlin, Germany: Springer-Verlag, 1996, pp. 250–265.

[51] G. W. Greenwood, "On the practicality of using intrinsic reconfiguration for fault recovery," *IEEE Trans. Evol. Comput.*, vol. 9, no. 4, pp. 398–405, Aug. 2005.

[52] L. Altenberg, "The evolution of evolvability in genetic programming," in *Advances in Genetic Programming*, K. Kinnear, Ed. Cambridge, MA: MIT Press, 1994, ch. 3, pp. 47–74.

[53] A. Thompson, "An evolved circuit, intrinsic in silicon, entwined with physics," in *Proc. 1st ICES*, 1996, pp. 390–405.

[54] I.-L. Yen and R. Paul, "Key applications for high-assurance systems," *IEEE J. Comput.*, vol. 31, no. 4, pp. 35–36, Apr. 1998.

[55] H. de Garis, "An artificial brain: ATR's CAM-brain project aims to build/evolve an artificial brain with a million neural net modules inside a trillion cell cellular automata machine," *New Gener. Comput. J.*, vol. 12, no. 2, pp. 215–221, Jul. 1994.

[56] P. Andersen, "Evolvable hardware: Artificial evolution of hardware circuits in simulation and reality," M.S. thesis, Dept. Comput. Sci., Univ. Aarhus, Aarhus, Denmark, 1998.

[57] T. G. W. Gordon and P. J. Bentley, "On evolvable hardware," in *Soft Computing in Industrial Electronics*, S. Ovaska and L. Sztandera, Eds. Heidelberg, Germany: Physica-Verlag, 2002, pp. 279–323.

[58] A. J. Hirst, "Notes on the evolution of adaptive hardware," in *Proc. Adapt. Comput. Eng. Des. Control*, Plymouth, U.K., 1996, pp. 212–219.

[59] G. Tufte and P. Haddow, "Prototyping a GA pipeline for complete hardware evolution," in *Proc. 1st NASA/DoD Workshop Evolvable Hardware*, Jul. 19–21, 1999, pp. 18–25.

[60] C. A. Coello, A. D. Christiansen, and A. A. Hernández, "Towards automated evolutionary design of combinational circuits," *Comput. Electr. Eng.*, vol. 27, no. 1, pp. 1–28, Jan. 2001.

[61] A. Thompson, I. Harvey, and P. Husbands, "Unconstrained evolution and hard consequences," in *Toward Evolvable Hardware: The Evolutionary Engineering Approach*, vol. 1062, E. Sanchez and M. Tomassini, Eds. Berlin, Germany: Springer-Verlag, 1996, pp. 136–165.

[62] T. Kalganova, "Bidirectional incremental evolution in evolvable hardware," in *Proc. 2nd NASA/DoD Workshop Evolvable Hardware*, Jul. 13–15, 2000, pp. 65–74.

[63] D. J. Xu and M. L. Daley, "Design of optimal digital filter using a parallel genetic algorithm," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 42, no. 10, pp. 673–675, Oct. 1995.

[64] T. Higuchi, M. Murakawa, M. Iwata, I. Kajitani, W. Liu, and M. Salami, "Evolvable hardware at function-level," in *Proc. IEEE Int. Conf. Evol. Comput.*, Apr. 1997, pp. 187–192.

[65] M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, and T. Higuchi, "Hardware evolution at function-level," in *Proc. Parallel Probl. Solving Nature IV*, M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwefel, Eds., Berlin, Germany, 1996, vol. 1141, pp. 62–71.

[66] A. Thompson, P. Layzell, and R. S. Zebulum, "Explorations in design space: Unconventional electronics design through artificial evolution," *IEEE Trans. Evol. Comput.*, vol. 3, no. 3, pp. 167–196, Sep. 1999.

[67] T. Kalganova, "An extrinsic function-level evolvable hardware approach," in *Proc. 3rd EuroGP*, R. Poli and W. Banzhaf, Eds, Edinburgh, U.K., Apr. 2000, pp. 60–75.

[68] H.-S. Seok, K.-J. Lee, B.-T. Zhang, D.-W. Lee, and K.-B. Sim, "Genetic programming of process decomposition strategies for evolvable hardware," in *Proc. 2nd NASA/DoD Workshop Evolvable Hardware*, Jul. 13–15, 2000, pp. 25–34.

[69] K. A. Vinger and J. Torresen, "Implementing evolution of FIR-filters efficiently in an FPGA," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Chicago, IL, Jul. 2003, pp. 26–29.

[70] J. Torresen, "Evolving multiplier circuits by training set and training vector partitioning," in *Proc. 5th ICES*. New York: Springer-Verlag, Mar. 2003, vol. 2606, pp. 228–237.

[71] ——, "Exploring knowledge schemes for efficient evolution of hardware," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Seattle, WA, Jun. 24–26, 2004, pp. 209–216.

[72] J. Torresen, J. W. Bakke, and L. Sekanina, "Recognizing speed limit sign numbers by evolvable hardware," in *Proc. 8th Int. Conf. PPSN VIII*, Birmingham, U.K., Sep. 2004, vol. 3242, pp. 682–691.

[73] Y. Zhang, S. L. Smith, and A. M. Tyrrell, "Digital circuit design using intrinsic evolvable hardware," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Seattle, WA, Jun. 24–26, 2004, pp. 55–62.

[74] J. C. Gallagher, S. Vigraham, and G. Kramer, "A family of compact genetic algorithms for intrinsic evolvable hardware," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 111–126, Apr. 2004.

[75] L. Sekanina and R. Ruzicka, "Easily testable image operators: The class of circuits where evolution beats engineers," in *Proc. Conf. Evolvable Hardware NASA/DoD*, Jul. 9–11, 2003, pp. 135–144.

[76] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, Dept. Comput. Commun. Sci., Univ. Michigan, Ann Arbor, MI, 1975.

[77] L. Sekanina, "Design methods for polymorphic digital circuits," in *Proc. 8th IEEE Des. Diagn. Electron. Circuits Syst. Workshop*, Sopron, Hungary, 2005, pp. 145–150.

[78] ——, "Evolutionary design of gate-level polymorphic digital circuits," in *Proc. Appl. Evol. Comput.*, 2005, pp. 185–194.

[79] J. Torresen, "A divide-and-conquer approach to evolvable hardware," in *Proc. 2nd Int. Conf Evolvable Syst.: Biol. Hardware*, 1998, pp. 57–65.

[80] ——, "Increased complexity evolution applied to evolvable hardware," in *Proc. ANNIE*, St. Louis, MO, Nov. 1999, pp. 429–436.

[81] S. Yang, *Logic Synthesis and Optimisation Benchmark User Guide Version 3.0, Microelectronics Center of North Carolina*, Research Triangle Park, NC, 1991.

[82] I. Baradavka and T. Kalganova, "Assembling strategies in extrinsic evolvable hardware with bi-directional incremental evolution," in *Proc. 6th EuroGP*, Essex, U.K.: Springer-Verlag, vol. 2610, pp. 276–285.

[83] T. Kalganova and J. Miller, "Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness," in *Proc. 1st NASA/DoD Workshop Evolvable Hardware*, Jul. 1999, pp. 54–63.

[84] T. Kalganova and M. Wilson, "Bi-directional incremental evolution: An approach for evolving relatively large combinational logic circuits," *J. Evol. Comput.,* submitted for publication.

[85] J. F. Miller and P. Thomson, "A developmental method for growing graphs and circuits," *Proc. 5th Int. Conf. Evolvable Syst.: Biol. Hardware*, Trondheim, Norway, Mar. 17–20, 2003, pp. 93–104.

[86] T. G. W. Gordon and P. J. Bentley, "Development brings scalability to hardware evolution," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Washington, DC, Jun. 29–01, 2005, pp. 272–279.

[87] A. J. Walker and J. F. Miller, "Evolution and acquisition of modules in Cartesian genetic programming," in *Proc. EuroGp*, 2004, vol. 3003, pp. 187–197.

[88] M. Hartmann, P. K. Lehre, and P. C. Haddow, "Evolved digital circuits and genome complexity," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Washington, DC, Jun. 29–01, 2005, pp. 79–86.

[89] J. Koza, F. H. Bennett, D. Andre, and M. A. Keane, *Genetic Programming III. Darwinian Invention and Problem Solving*. San Mateo, CA: Morgan Kaufmann, 1999.

[90] D. Levi, "HereBoy: A fast evolutionary algorithm," in *Proc. 2nd NASA/DoD Workshop Evolvable Hardware*, Jul. 13–15, 2000, pp. 17–24.

[91] T. G. W. Gordon and P. J. Bentley, "Bias and scalability in evolutionary development," in *Proc. Genetic Evol. Comput. Conf.*, Jun. 29–Jul. 1, 2005, pp. 83–90.

[92] P. van Remortel, B. Manderick, and T. Lenaerts, "Gene interaction and modularisation in a model for gene-regulated development," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Seattle, WA, Jun. 24–26, 2004, pp. 253–260.

[93] S. Xian-He and D. T. Rover, "Scalability of parallel algorithm–machine combinations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 6, pp. 599–613, Jun. 1994.

[94] T. Arslan and B. I. Hounsell, "A novel genetic algorithm for the automated design of performance driven digital circuits," in *Proc. Congr. Evol. Comput.*, Jul. 6–19, 2000, vol. 1, pp. 601–608.

[95] M. Hartmann, P. Haddow, and F. Eskelund, "Evolving robust digital designs," in *Proc. Conf. Evolvable Hardware NASA/DoD*, Jul. 15–18, 2002, pp. 36–45.

[96] D. Keymeulen, A. Stoica, R. Zebulum, Y. Jin, and V. Duong, "Fault-tolerant approaches based on evolvable hardware and using a reconfigurable electronic devices," in *Proc. IEEE Int. Integr. Rel. Workshop Final Rep.*, Oct. 23–26, 2000, pp. 32–39.

[97] J. Langeheine, K. Meier, J. Schemmel, and M. Trefzer, "Intrinsic evolution of digital-to-analog converters using a CMOS FPTA chip," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Seattle, WA, pp. 18–25.

[98] A. Stoica, "Toward evolvable hardware chips: Experiments with a programmable transistor array," in *Proc. 7th Int. Conf. Microelectron. Neural Fuzzy Bio-Inspired Syst.*, Granada, Spain, Apr. 7–9, 1999, pp. 156–162.

[99] A. Stoica, R. Zebulum, and D. Keymeulen, "Progress and challenges in building evolvable devices," in *Proc. 3rd NASA/DoD Workshop Evolvable Hardware*, Jul. 12–14, 2001, pp. 33–35.

[100] Xilinx, Inc., *XC6200 Field Programmable Gate Arrays*, Apr. 1997. Data Sheet, Version 1.10.

[101] ——, *Virtex 2.5V Field Programmable Gate Arrays*, 2000, Xilinx.

[102] ——, *XC400XLA/XV Field Programmable Gate Arrays*, Oct. 1999, Xilinx.

[103] ——, *Virtex-II Platform FPGA User Guide*, Mar. 2005, Xilinx UG002 (v2.0).

[104] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Syst.*, vol. 9, no. 3, pp. 193–212, Jun. 1995.

[105] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed., 1991, pp. 69–93.

[106] J. R. Koza, S. H. Al-Sakran, and L. W. Jones, "Cross-domain features of runs of genetic programming used to evolve designs for analog circuits, optical lens systems, controllers, antennas, mechanical systems, and quantum computing circuits," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Washington, DC, Jun. 29–01, 2005, pp. 205–214.

[107] J. R. Koza, M. A. Keane, and M. J. Streeter, "Routine high-return human-competitive evolvable hardware," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Seattle, WA, Jun. 24–26, 2004, pp. 3–17.

[108] M. Oltean, "Solving even-parity problems using traceless genetic programming," in *Proc. CEC*, vol. 2, Jun. 19–23, 2004, pp. 1813–1819.

[109] S. Bleuler, M. Brack, L. Thiele, and E. Zitzler, "Multiobjective genetic programming: Reducing bloat using SPEA2," in *Proc. Congr. Evol. Comput.*, May 27–30, 2001, vol. 1, pp. 536–543.

[110] K. Kozminski, "Benchmarks for layout synthesis—Evolution and current status," in *Proc. 28th ACM/IEEE Des. Autom. Conf.*, Jun. 17–21, 1991, pp. 265–270.

[111] L. Chang-Tzu, C. De-Sheng, and W. Yi-Wen, "An efficient genetic algorithm for slicing floorplan area optimization," in *Proc. IEEE ISCAS*, May 26–29, 2002, vol. 2, pp. 879–882.

[112] H. Esbensen, "A genetic algorithm for macro cell placement," in *Proc. EURO-VHDL/EURO-DAC*, Sep. 7–10, 1992, pp. 52–57.

[113] G. Holt and A. Tyagi, "GEEP: A low power genetic algorithm layout system," in *Proc. 39th IEEE Midwest Symp. Circuits Syst.*, Aug. 18–21, 1996, vol. 3, pp. 1337–1340.

[114] S. Roy and B. K. Sikdar, "Power conscious BIST design for sequential circuits using ghost-FSM," in *Proc. 12th ATS*, Nov. 16–19, 2003, pp. 190–195.

[115] B. Ali, A. E. A. Almaini, and T. Kalganova, "Evolutionary algorithms and their use in the design of sequential logic circuits," in *Genetic Program. Evolvable Mach.*, Mar. 2004, vol. 5, pp. 11–29.

[116] A. P. Shanthi, L. K. Singaram, and R. Parthasarathi, "Evolution of asynchronous sequential circuits," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Washington, DC, Jun. 29–Jul.1 2005, pp. 93–96.

[117] E. Stomeo, T. Kalganova, C. Lambert, N. Lipnitsakya, and Y. Yatskevich, "On evolution of relatively large combinational logic circuits," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Washington, DC, Jun. 29–Jul. 1, 2005, pp. 59–66.

[118] M. Oltean and C. Grosan, "Evolving digital circuits using multi expression programming," in *Proc. NASA/DoD Conf. Evolvable Hardware*, Seattle, WA, pp. 87–94.

[119] E. Stomeo, T. Kalganova, and C. Lambert, "Analysis of genotype size for an evolvable hardware system," in *Proc. ICCI*, Prague, Czech Republic, Aug. 26–28, 2005, pp. 74–79.

Emanuele Stomeo (S'06) received the Laurea degree in electronic engineering from Politecnico di Torino, Turin, Italy, in 2003. From 2000 to 2003, he studied image processing and digital design at RWTH Aachen University, Aachen, Germany. He is currently working toward the Ph.D. degree in computer science and engineering at Brunel University, West London, U.K. He carried out his Master's thesis work at Philips Research Laboratories, Aachen, in 2002–2003.

He is currently a member of the Bio-Inspired Intelligent Systems Research Group at Brunel University. His research interests are in evolvable hardware, evolutionary computation, design of digital circuits, and bioengineering applications.

Tatiana Kalganova received M.Sc. degree from the Belarusian State University of Informatics and Radioelectronics, Minsk, Belarus, in 1994 and the Ph.D. degree from Napier University, Edinburgh, U.K., in 2000.

In August 2000, she joined the Electronic and Computer Engineering Department, Brunel University, Middlesex, U.K. Her research interests are evolvable hardware, ant colony algorithms, and scalability in AI systems.

Dr. Kalganova was awarded a grant from the International Soros Science Education Program (ISSEP) for distinctive achievements in the field of exact sciences in 1996, and a personal grant by the Education Ministry of the Republic of Belarus for distinctive achievements in the field of exact sciences in 1997.

Cyrille Lambert received the diplôme d'éducation supérieure spécialisée degree in microelectronic engineering from Pierre et Marie Currie University, Paris, France, in 2000, and is currently working toward the Ph.D. degree at Brunel University, Middlesex, U.K. He carried out his thesis work at the Swiss Centre for Electronics and Microtechnology, Inc., Neuchâtel, Switzerland in 1999–2000.

After spending three years in the industry as a Digital Design Engineer, he joined, in 2003, the Computer Science and Engineering Department, Brunel University, and is currently a member of Bio-Inspired Intelligent Systems at Brunel University.