Circuit layout evolution: An Evolvable Hardware Approach

Tatiana Kalganova[1], Julian F. Miller[2]

School of Computing, Napier University, 219 Colinton Road, Edinburgh, UK, EH14 1DJ, Tel.: (+44) 131 455 4304
(t.kalganova,[1] j.miller[2]) @dcs.napier.ac.uk

Abstract. The evolvable hardware technique is based on evolving the functionality and connectivity of a rectangular array of logic cells in addition to the layout of this array. The evolutionary process contains two main steps. Initially the genome fitness in given by the percentage of output bits, which are correct. Once 100% functional circuits have been evolved, the number of gates actually used in the circuit is taken into account in the fitness function. This allows us to evolve circuit with 100% functionality *and* minimise the number of active gates in circuit structure. We perform a number of experiments to investigate the behaviour of the second fitness function and the circuit layout during evolution. We find that the gate usage is linearly related to the total number of gates in the chromosome.

1 Introduction

Evolvable Hardware (EHW) approach is a recently developed technique to synthesise the electronic circuits using evolutionary algorithms. A central idea of this approach is to represent each possible electronic circuit as chromosome in an evolutionary process in which the standard genetic operators are carried out [1, 2, 3, 4, 5].

In this paper, we evolve combinational logic circuits. The EHW approach can be easily extended for the combinational multiple-valued logic circuits. This approach is an extension of EHW method proposed in [2, 6, 7] for binary circuits. A similar approach to the design multiple-valued circuit has been discussed in [8, 9]. Analysis of the EHW approach for both binary and multiple-valued functions shows us that the GA performance strongly depends on the number of rows and columns and the internal connectivity [9, 7]. In subsequent discussion we define the *circuit geometry* to mean the layout of the rectangular array of logic cells. It is characterised by just two numbers: the number of rows and columns in the cellular array. The degree of connectivity in the circuit called *levels-back* parameter defines how many columns of cells to the left of current column can have their outputs connected to the inputs of the current cell, this also applies to the final circuit outputs. The choice of suitable circuit geometry is very complicated task and is intimately linked the complexity of function implemented. In this paper we investigate the possibility of evolving the circuit geometry at the same time as trying to evolve 100% functional circuits. In this scheme, mutation is carried out in two ways. First, we can mutate genes associated with a circuit in a fixed geometry, and secondly, we can by mutation choose the circuit geometry. Often the objective in digital evolution behaviour is to merely produce a 100% functionally correct circuit. So, that the evolutionary process is terminated at this point. In this paper we continue to evolve the circuit beyond the point of 100% correctness by modifying the fitness function to include a measure of circuit's efficiency.

2 The Evolutionary Algorithm

In order to evolve combinational logic circuits, an elitist evolutionary algorithm using tournament selection with elitism and uniform crossover has been implemented, these details are given in the following subsections.

2.1 Encoding

There are two aspects required to define any combinational logic network. The first is the cell-level functionality and the second is the inter-connectivity of the cells between the circuit inputs and outputs. An encoding of chromosome was adopted that satisfies these two aspects.

A combinational logic circuit is represented as a rectangular array of logic gates (Fig. 1). Each logic cell in this array is uncommitted and can be removed from the network if they prove to be redundant. The inputs of combinational network such as logical constants, primary and inverted inputs, as well as the outputs of logic cells are labelled with an individual integer. In the work reported in this paper we define each logic function to be chosen from the set of functions AND, OR, NOT, EXOR with primary and inverted inputs or function defined behaviour of multiplexer. Each input of a logic gate may be connected to the output of a logic gate provided it is to the left of the cell, a logical constant, a primary or an inverted primary inputs.

The chromosome is represented by 3-level structure: 1) Geometry 2) Circuit 3) Gate (cell) structures. On the first level the global characteristics of the circuit are defined. There are levels-back parameter and the number of rows and columns. The circuit geometry is changed in this level. On the second level the array of cells are created and the circuit outputs are determined. Finally the third level represents the structure of each cell in the circuit. The data describing the cell contains the number of inputs, the array of inputs and the functional gene. The number of inputs in the cell depends on the type of cell and is defined when the value of functional gene is known.

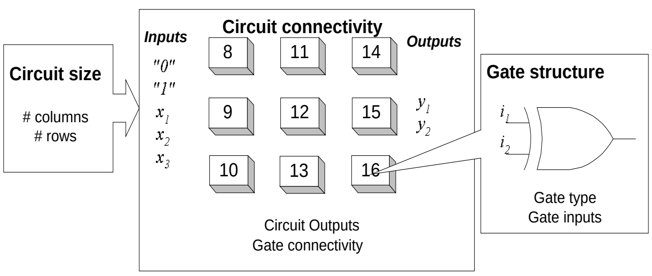

Fig. 1. Schematic of chromosome structure

2.2 Initialisation procedure

The initialisation procedure contains several steps: 1) Define circuit geometry of chromosomes in population; 2) Initialise the genotype of cells; 3) Generate the circuit outputs for each of the chromosome. The first step is constrained to observe the maximum number of rows and columns in the chromosome. During the second and third steps the initialisation of cell inputs and circuit outputs is performed according to the levels-back constraint and to the type of variables which are able to be present throughout all circuit. Thus if the logic constants are allowed as input connections throughout the circuit, then during initialisation procedure the inputs of gates can be chosen from the set of inputs constrained by levels-back or from the set of logical constants. The same procedure is true for the primary and inverted primary inputs.

2.3 Mutation

We use two types of mutation: parameter circuit mutation and geometry mutation. The parameter circuit mutation allows us to change the type of genes in chromosome excluding the number of columns and rows. The geometry mutation changes the numbers of rows or columns in the rectangular array. The maximum numbers of rows and columns are predefined. In both cases the mutation rate has to be chosen carefully, since it can dramatically affect on the GA performance.

PARAMETER CIRCUIT MUTATION: The parameter circuit mutation allows us to change the following three parameters of the circuit: 1) Cell input 2) Cell type and 3) Circuit output. Each of these parameters is considered as an elementary unit of the genotype. The parameter circuit mutation rate defines how many genes in the population are involved in mutation. The chromosome contains 3 different types of genes.

GEOMETRY MUTATION: Geometry mutation allows us to change the number of rows and columns in chromosome. Geometry mutation can be applied to each chromosome with the geometry mutation probability. The number of rows and columns are treated as an elementary unit of the genotype. Each such unit can be changed with a probability 0.5. The geometry mutation consists of the two main steps: 1) Gene mutation 2) Repair algorithm. On the first step the new number of columns or rows of the chromosome is randomly defined. This number cannot exceed the maximum number of columns or rows. On the second step the repair algorithm is applied to ensure that a chromosome with new geometry represents a valid genotype.

Let us consider geometry mutation process for chromosome with 3x3 circuit geometry. Let $N_{columns}$ and N_{rows} be the number of columns and rows of chromosome assigned to be mutated and *new_gene* is the new value of mutated genes, which is synthesised randomly. The gene mutation procedure is the following:

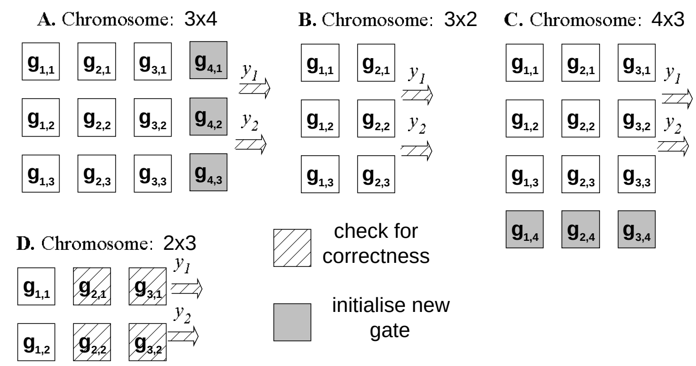

Fig. 2. The geometry mutation process (3x3 circuit geometry)

1. Define the circuit mutation rate P_{mg}.
2. Generate random number for each chromosome, $rand1 \in [0, 1]$.
3. If ($rand1 < P_{mg}$) the geometry mutation is applied to the current chromosome
4. Generate random number $rand2 \in [0, 1]$.
5. If ($rand2 < 0.5$) the number of columns in chromosome is chosen to be mutated and the new number of columns (*new_gene*) is generated from the range $[1, N^{\max}_{columns}]$.

Else the number of rows is considered as mutated gene and the new number of rows (*new_gene*) is generated from the range $[1, N^{\max}_{rows}]$.

When *new_gene* is defined, the geometry mutation is performed in the following manner. Let us consider the case when the mutated gene is the number of columns. In this case the new circuit structures, shown in Fig. 2 (A and B), can be synthesised. If ($new\_gene > N_{\text{columns}}$) we have to add new columns in the chromosome representation (Fig. 2 (A)). The gates in new columns are initialised using initialisation procedure. It is possible, however, that the circuit output disobeys the levels-back constraint. Thus, the chromosome may need to be repaired. The repair algorithm checks whether the circuit outputs obey the levels-back constraint, and whether all the cell inputs are valid. If the circuit output does not satisfy this condition a new circuit output is initialised. If ($new\_gene < N_{\text{columns}}$) we have to remove some columns in the circuit structure (Fig. 2 (B)). After the new structure is obtained, a repair algorithm is applied to the circuit output, because the circuit output can refer to a gate which no longer exists in the circuit. In the case when the mutated gene is the number of rows, the structures C and D given in Fig. 2 can be synthesised. If ($new\_gene > N_{\text{rows}}$) the new rows of gates are added to the circuit structure (Fig. 2 (C)). Again, these gates are initialised. There is no need to apply repair algorithm to the circuit outputs in this case because all connections are not changed and the circuit outputs will still refer to the correct logic cells in the circuit structure. If ($new\_gene < N_{\text{rows}}$) some rows are removed from the circuit structure (Fig. 2 (D)). In this case the inputs of the remaining gates as well as circuit outputs can refer to gates which are no longer present. The repair algorithm has to be applied to the each genotype of the gate and to the circuit outputs.

RECOMBINATION: Recombination is implemented with uniform crossover. The whole logic cell is considered as a gene. Two cells are chosen from two chromosomes and swapped. The number of chromosomes selected for breeding is defined by the crossover rate, which is carried out on a cellular level. In order to preserve the interconnection conditions, the repair algorithm checks the parameters of logic gates for correctness. When two chromosomes with different geometries undergo crossover it is very likely that merely swapping genes to produce the offspring, will generate invalid genomes. These would have to be repaired (randomly initialised), and this would introduce a considerable amount of randomness into the recombination process. Thus, the selection of correct crossover rate and its type is very important.

FITNESS FUNCTION: The evaluation process consists of the two main steps. First we are trying to find the circuits with 100% functionality and second we are trying to minimise the number of active gates in 100% functionality circuits. An *active gate* is a gate, which is proved to be not redundant.

Table 1. Experimental parameters

Population size	15
Number of generations	500
Number of GA runs	variable
Breeding rate	60
Crossover type	One-point
Selection pressure	0.75
Max number of rows	5 or 8
Max number of columns	5 or 8
Levels-back parameter	2
Target function	Add1c.pla

3 Experimental Results

In this section we will consider the some experimental results obtained for the one-bit adder with carry. We investigated how the number of 100% functional cases relates to the number of active gates used in circuit. The initial data for the GA is given in Table 1.

The distribution of the circuit geometry of the best chromosomes is shown in Fig. 3 (a) for a maximum circuit geometry 5x5 and in Fig. 4 (a) for 8x8 geometry. It is clear that in both cases the most number of 100% functionality cases is obtained when large numbers of logic gates are available. Thus, the GA performs better when there is large gate redundancy.

Fig. 3 (b) and Fig. 4 (b) show how the number of 100% functional varies with the number of active gates. The overall behaviour of this function in both cases (5x5 and 8x8 maximum circuit geometries) is very similar. Note that there is the definite number of active gates in the circuit when the highest number of 100% cases can be obtained. When very large number of gates is allowed to be used, many of them are not used in actual circuit structure.

In order to investigate how the redundancy of logic gates in chromosome influences the GA performance we calculate the percentage of gates actually used in final circuit structure.

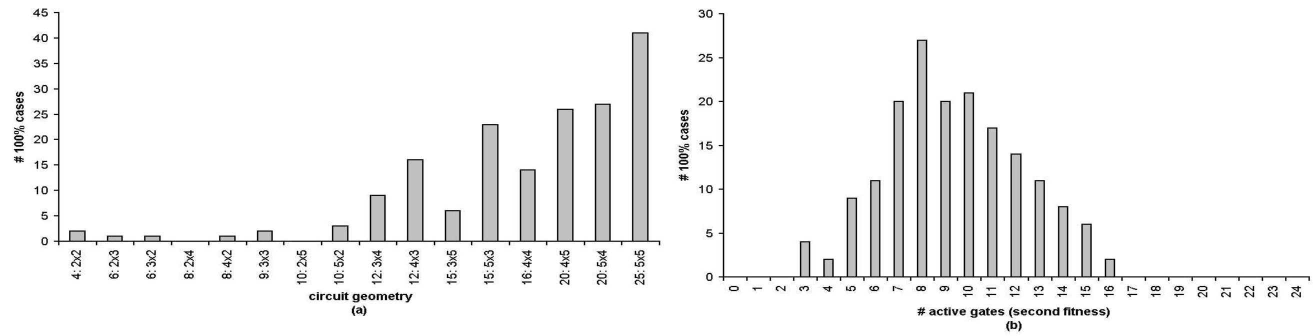

Fig. 3. The number of gates used in circuit geometry 5x5

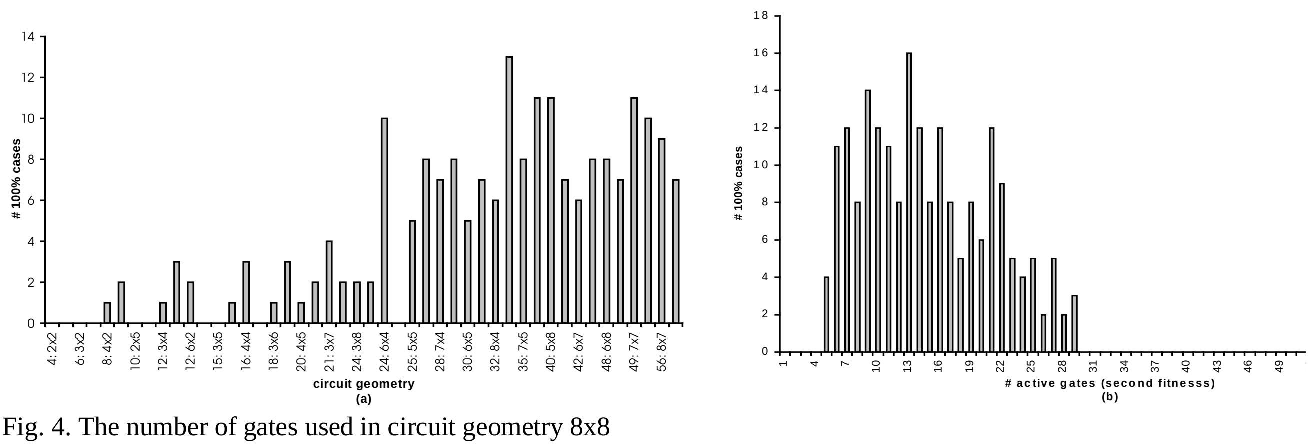

Fig. 4. The number of gates used in circuit geometry 8x8

In Fig. 6 the proportion of gates used in the circuit is plotted as a function of the maximum number of gates available. It is clear from this graph that the proportion of active and unused gates is constant with increasing the number of gates in circuit structure. This means that the number of active gates grows in linear way with increasing the number of possible gates in circuit. It implies that the most economical circuit will not occur often in larger circuits. The proportion varies erratically with smaller number of gates.

In order to compare the GA execution with fixed and flexible circuit geometry the following experiment has been carried out. The initial data for GA is given in Table 2. This experiment has been performed for the maximum circuit geometry of 5x5 and all possible circuit geometries from 2x2 to 5x5 for constant circuit geometries. It is interesting to note the behaviour of the second fitness is nearly the same for both cases (Fig. 7). This implies that allowing a flexible geometry does not reduce the number of 100% cases.

Table 2. Experimental parameters

Population size	15
Number of generations	500
Number of GA runs	25
Breeding rate	60
Crossover type	One-point
Selection pressure	75
Number of rows	Variable
Number of columns	Variable
Levels-back parameter	2
PLA file processing	Add1c.pla

A study of many histories of the GA runs revealed that the evolution proceeds in the three successive phases:

1. in the early generations, the population selects a particular geometry;
2. in the second phase, the structure of the circuit evolves in such a way, that the circuit implements the test function;
3. in the last phase, the evolution process is slower and tends to decrease the number of gates in the actual circuit.

Thus we can argue that since the geometry is chosen so early in the history of the GA evolution is proceeding with a largely fixed geometry. Consequently, the similar appearance of the graph shown in Fig. 7 is not so surprising. It is probable that the rather drastic way in which mutation can change the circuit geometry is causing this behaviour. A much smother geometrical mutation may lead to a more interesting interaction between the geometry, the number of 100% cases and the number of active gates.

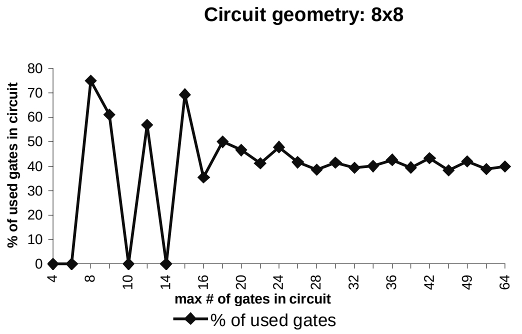

Fig. 5. Gate redundancy

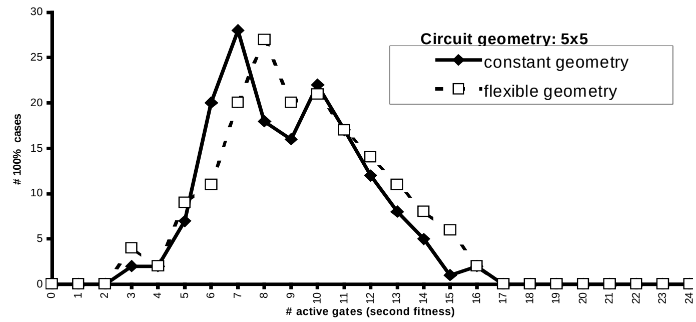

Fig. 6. Comparison GA execution with constant and flexible circuit geometries

4 Conclusion

This paper has described the evolutionary design of combinational logic circuits. The distinctive feature of proposed algorithm is that it allows us to evolve the circuit layout in addition to the circuit structure. We have defined a fitness function, which allows us estimate not only the functionality of circuit but to define how good the evolved 100% functional circuit is. The experimental results show that in terms of the 100% cases evolved the chromosome representation with flexible and permanent circuit geometry give similar results. Thus allowing the geometry to evolve is not especially harmful to the evolution process. Developing less drastic mutation operator may allow a greater amount of geometry change. Analysing the structure of circuit evolved allows us to make conclusion that proportion of the number of available gates in the circuit to the number of actual gates becomes fixed with increasing numbers of available gates in circuit. This is interesting finding and worthy of further investigation. It may be that increasing the number of generations will change this picture as the optimising phase of the GA takes place in the latter stages.

Bibliography:

1. Layzell P. (1998). A New Research Tool for Intrinsic Hardware Evolution. in *Proc. of The 2nd Int. Conf. on Evolvable Systems: From Biology to Hardware (ICES98), Lecture Notes in Computer Science*, Eds. Sipper M., et al, Vol. 1478, Springer-Verlag, Heidelberg, pp. 47-56.
2. Miller J. F., Thomson P., and Fogarty T. C. (1997). Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. in *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, Eds. D. Quagliarella, et al., Wiley.
3. Thompson A. (1997). An evolved circuit, intrinsic in silicon, entwined with physics, in *Proc. of the 1st Int. Conf. on Evolvable Systems: From Biology to Hardware (ICES96), Lecture Notes in Computer Science*, Eds. Higuchi T., et al. Vol. 1259, Springer-Verlag, pp.390–405.
4. Thompson A. (1998). On the Automatic Design of Robust Electronics Through Artificial Evolution in *Proc. of The 2nd Int. Conf. on Evolvable Systems: From Biology to Hardware (ICES98), Lecture Notes in Computer Science*, Eds. Sipper M., et al Vol. 1478, Springer-Verlag, Heidelberg, pp. 13-24.
5. Sipper M., Sanchez E., Mange D., Tomassini M., Perez-Uribe A., and Stauffer A. (1997). A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems, in *IEEE Trans. on Evolutionary Computation*, Vol. 1, No 1., pp. 83-97.
6. Miller J. F., Thomson P. (1998). Aspects of Digital Evolution: Geometry and Learning, in *Proc. of The 2nd Int. Conf. on Evolvable Systems: From Biology to Hardware (ICES98), Lecture Notes in Computer Science*, Eds. Sipper M. et al. Vol. 1478, Springer-Verlag, pp. 25-35.
7. Miller J. F., Thomson P. (1998). Aspects of Digital Evolution: Evolvability and Architecture, in *Proc. of The 5th Int. Conf. on Parallel Problem Solving from Nature (PPSNV), Lecture Notes in Computer Science*, Vol. 1498, Springer-Verlag, pp. 927-936.
8. Kalganova, T., J. Miller and N. Lipnitskaya (1998). Multiple-Valued Combinational Circuits Synthesized using Evolvable Hardware Approach in *Proc. of the 7th Workshop on Post-Binary Ultra Large Scale Integration Systems (ULSI'98) in association with ISMVL'98*, Japan. IEEE Press.
9. Kalganova, T., J. Miller and T. Fogarty (1998). Some Aspects of an Evolvable Hardware Approach for Multiple-Valued Combinational Circuit Design in *Proc. Of the 2nd Int. Conf. on Evolvable Systems (ICES'98).* Switzerland, Eds. M. Sipper, et al, Springer-Verlag, pp. 78-89.