Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness

Tatiana Kalganova
School of Computing
Napier University
219 Colinton Road
Edinburgh, UK, EH14 1DJ
t.kalganova@dcs.napier.ac.uk

Julian Miller
School of Computing
Napier University
219 Colinton Road
Edinburgh, UK, EH14 1DJ
j.miller@dcs.napier.ac.uk

Abstract

We use evolutionary search to design combinational logic circuits. The technique is based on evolving the functionality and connectivity of a rectangular array of logic cells whose dimension is defined by the circuit layout. The main idea of this approach is to improve quality of the circuits evolved by the genetic algorithm (GA) by reducing the number of active gates used. We accomplish this by combining two ideas: 1) using multi-objective fitness function; 2) evolving circuit layout. It will be shown that using these two approaches allows us to increase the quality of evolved circuits. The circuits are evolved in two phases. Initially the genome fitness in given by the percentage of output bits that 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 circuits with 100% functionality ***and*** *minimise the number of active gates in circuit structure. The population is initialised with heterogeneous circuit layouts and the circuit layout is allowed to vary during the evolutionary process. Evolving the circuit layout together with the function is one of the distinctive features of proposed approach. The experimental results show that allowing the circuit layout to be flexible is useful when we want to evolve circuits with the smallest number of gates used. We find that it is better to use a fixed circuit layout when the objective is to achieve the highest number of 100% functional circuits. The two-fitness strategy is most effective when we allow a large number of generations.*

Evolvable Hardware 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 such as initialisation, recombination, selection are carried out. The circuits may be evaluated using software simulation models [1], [2], [3], [4] or alternatively evolved entirely in hardware [5], [6}, [7], [8].

In this paper, we limit our focus to combinational logic circuits, which contain no memory elements. Such circuits contain no feedback paths. Note that this approach can be easily extended for the combinational multiple-valued logic circuits. The approach is an extension of evolvable hardware method proposed in [3], [9], [10], [11] for binary combinational circuits. A similar approach to the design multiple-valued combinational circuit has been discussed in [11], [12], [13]. A discussion concerning a suitable set of logic gates was given in [13]. It has been shown that the GA performance strongly depends on the set of logic gates used to produce the 100% functionally circuits. In [12] experiments were reported which revealed the dependence the GA performance with gate array dimensions and the degree of internal connectivity. Analysis of the evolvable hardware 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], [12]. 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* 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.

This paper presents an extension of the methods discussed above. Here we will discuss two possible ways to improve the quality of evolved circuits. In a previous work the sole objective was to evolve 100% functional circuits. The purpose of our work is to consider this aspect together with attempting to improve the evolved circuits in terms of the number of active gates used. One of the obvious ways to improve it is to use a multi-objective fitness function. Thus in previous works the objective in digital evolution behaviour was to merely produce a 100%

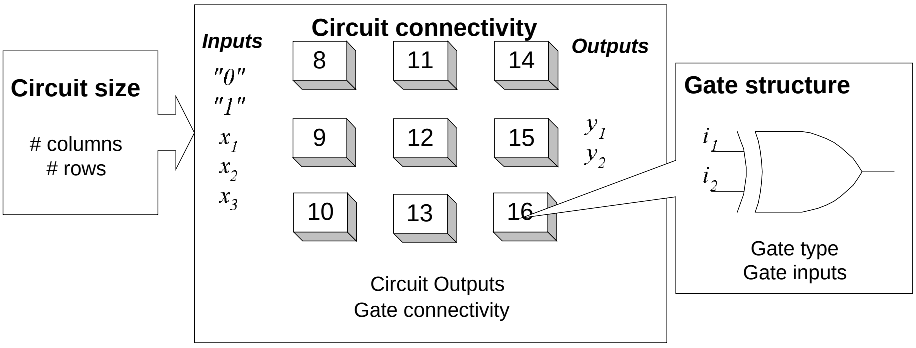

Fig. 1. Schematic of chromosome structure

functionally correct circuit (F_1 fitness). So, the evolutionary process is terminated at this point. Here 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 (F_2 fitness). As we mention above the choice of suitable circuit geometry is a very complicated task and is intimately linked with the complexity of the function implemented. So, in order to avoid this we investigate the possibility of evolving the circuit geometry at the same time as trying to evolve 100% functional circuits. The circuit geometry defines the length of the chromosome, thus we work with chromosomes of variable length. 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. The main purpose of circuit layout evolution was to try to evolve the best circuit layout together with evolving circuit functionality. However during the GA execution we find the interesting result that actually using a flexible circuit geometry allows us to reduce the number of active gates in circuit [14], [15]. This was unexpected. In our further research we define several strategies for the GA. We investigate cases where we use homogeneous, heterogeneous or partially heterogeneous (heterogeneous only at the initialisation stage of GA) circuit layouts during GA execution and determine the GA performance as a function of both fitness measures.

1 The Evolutionary Algorithm

In order to evolve combinational logic circuits, an evolutionary algorithm using tournament selection with elitism and uniform crossover has been implemented, these details are given in the following subsections. During the evolution process we only allow the circuit layout to be changed by mutation by altering the number of rows or columns. In this case we will refer to this as *heterogeneous circuit layout* during *evolution*. When the circuit geometry is not changed during evolution process, we refer to it as the *homogeneous circuit layout*.

1.1 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 to any cell in the combinational network may be logical constants, primary and inverted inputs, as well as the outputs of logic cells which are in columns to the left of the cell in question. 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 a multiplexer.

The chromosome is represented by a 3-level structure: 1) Geometry structure; 2) Circuit structure; 3) Gate (cell) structure. At the first level the global characteristics of the circuit are defined: These are levels-back and the number of rows and columns. The circuit geometry can be changed at this level. At 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. This data consists of the number of inputs, the input connections 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 (i.e. a multiplexer has three inputs while all others have only two inputs). Note that the number of inputs as well as the number of outputs are allowed to be variable, but in this paper we consider only 2 or 3 input 1 output gates.

An example of the chromosome representation with the actual circuit structure is given in **Fig. 2**. Let us examine a possible circuit representing a 1-bit adder with carry. This

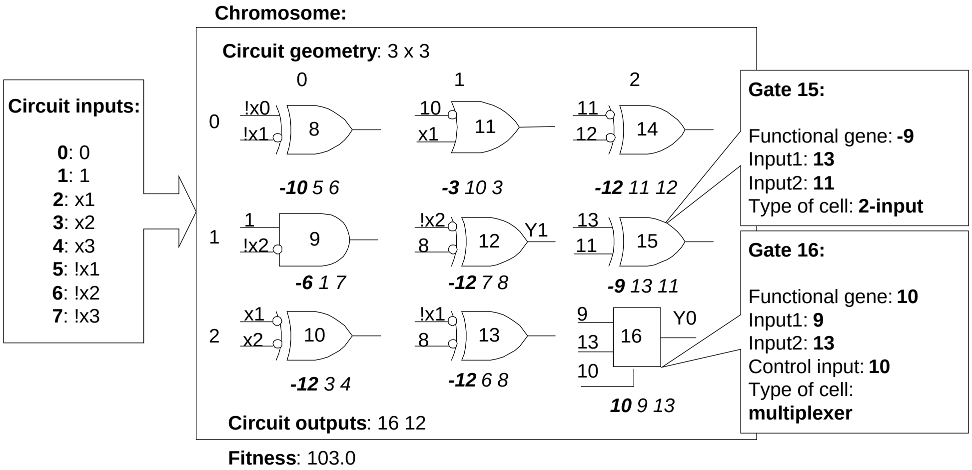

Fig. 2. An example of the phenotype and corresponding genotype of a chromosome with 3x3 circuit geometry

function has 3-inputs and 2-outputs and is implemented here on a combinational network with 3x3 circuit geometry (N_{column} x N_{row}). The circuit inputs are labelled as follows: 0 and 1, which represent the logical constants 0 and 1 respectively, labels 2, 3 and 4 correspond to the input variables x_0, x_1 and x_2 respectively. The inverted inputs !x_0, !x_1, !x_2 are represented as 5, 6 and 7. In this example the functional gene (shown in bold) represents one of the 13 possible gates (AND, OR, EXOR with primary and inverted inputs or multiplexer). The functional gene may be a positive or negative integer. If positive then the function is a multiplexer and the integer represents the control connection. If functional gene is negative, we use an encoding table to define the type of gate (**Table 1**).

The output of each cell is assigned an individual address. Thus the output of the cell located in the 0th column and in the 0th row is labelled as 8. The output of logic cell in the 2nd column and 2nd row is labelled as 16. The number of circuit outputs is defined by the number of outputs in the logic function implemented. The logic cell label determines each of these outputs. Let us examine the encoding of the 12th logic cell in genotype <-12 7 8>. We refer to this representation of gate as *gate genotype*. The functional gene defining the type of this gate is -12. This value corresponds to the EXOR gate with inverted inputs in the gate-encoding table (Table 1).

The examined cell has two inputs. The first input is connected to the input !x_2 and second to the output of the 8th cell. The cell 8 depends on two variables: !x_0 and !x_1. So, in this case the logic function that describes the 12th cell depends on three variables: !x_0, !x_1 and !x_2. Let us consider the 16th gate, with genotype <10 9 13>. Since the functional gene is positive, the gate is multiplexer and the functional gene with value 10 corresponds to the control input. The inputs of this multiplexer are connected to the outputs of gates 9 and 13. The outputs of circuit are connected to the outputs of the 16th and 12th logic gates. The fitness F of a chromosome is defined as follows:

$$F = \begin{cases} F_1 = c, & if \quad c < 100.0 \\ F_2 = c + \gamma, & if \quad c = 100.0 \end{cases}$$

where c is the percentage of the circuit output bits that are correct, γ is the number of gates that are *not* involved in the circuit.

Table 1. Cell gate functionality according to the negative gene value in chromosome

Functional gene	Gate function
-1	x_1 AND x_2
-2	x_1 AND !x_2
-3	!x_1 AND x_2
-4	!x_1 AND !x_2
-5	x_1 OR x_2
-6	x_1 OR !x_2
-7	!x_1 OR x_2
-8	!x_1 OR !x_2
-9	x_1 EXOR x_2
-10	x_1 EXOR !x_2
-11	!x_1 EXOR x_2
-12	!x_1 EXOR !x_2

The maximum F_2 is equal to (100.0+ $N_{rows}^{\max} * N_{columns}^{\max}$) in this case no gates are used. The fitness function for one-bit adder with carry the fitness is 103.0 for the circuit shown in Fig. 2. This means that this circuit represents a 100% functional one-bit adder with carry and there are 3 logic gates that are not involved in the combinational implementation of this circuit. In other words, there are 6 gates, which are actually used to synthesise the one-bit adder with carry, because $F_2^{\max}$ =100.0+3*3=109.0.

1.2 Objective Function and Fitness

One of the objectives of combinational circuit design is to construct a circuit utilising the minimum number of gates from the behavioural specification of the circuit given by the truth table. The evaluation process consists of the two main steps. First we are trying to find the circuits with 100% functionality (F_1 fitness) and second we are trying to minimise the number of active gates in 100% functional circuits (F_2 fitness). An *active gate* is a gate, which is proved to be not redundant. We use two strategies in our GA: 1) F_1; 2) F_1+F_2. In the first strategy, the chromosome is evaluated using F_1 fitness only and once the 100% functional circuit evolved, the evolution process is terminated. In the case of F_1+F_2 strategy, F_2 fitness is activated as soon as F_1=100.0 and the number of inactive gates in circuit is estimated. When flexible circuit geometry is employed, F_2 is calculated based on the maximum available circuit layout.

1.3 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 chromosome.

The first step defines the circuit geometry for the chromosomes. In flexible circuit layout, any circuit geometry may be used up to the maximum number of rows and columns. In fixed circuit layout all chromosomes have the same circuit geometry. We say that we have *homogeneous circuit layout* during *initialisation* process when the circuit layout for all chromosomes is the same. The *heterogeneous circuit layout* occurs when the chromosomes are initialised with different circuit layouts. During the second and third step the initialisation of cell inputs and circuit outputs is performed in accordance with the levels-back constraint and 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.

1.4 Mutation

We use two types of mutation: circuit mutation and geometry mutation. The *circuit mutation* allows us to change the type of genes in a chromosome but excludes 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 the GA performance.

Circuit Mutation: The circuit mutation allows us to change the following three features 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 circuit mutation rate defines how many genes in the population are involved in mutation. The chromosome contains 3 different types of genes, whose number is :

$$N_{genes} = \sum_{i=1}^{pop\_size} (3 \cdot N^{i}_{gates} + N_{outputs})$$

where $N_{outputs}$ is the number of outputs in the circuit, N^{i}_{gates} is the number of gates in the *i*-th chromosome.

Geometry Mutation: Geometry mutation allows us to change the number of rows and columns in a chromosome. Geometry mutation is applied to each chromosome with a given probability. In this case the numbers of rows and columns are treated as an elementary unit of the genotype. Either the number of rows or the number of columns is changed with equal probability. The geometry mutation consists of the two main steps: 1) Gene mutation 2) Repair algorithm. In the first step the new number of columns or rows of the chromosome is randomly defined. At the second step the repair algorithm is applied to ensure that a chromosome with a 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_value* is the new value of mutated genes chosen randomly. The gene mutation procedure is the following:

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_value*) 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_value*) is generated from the range [1, $N^{\max}_{rows}$].

Let $N^{current}$ be the number of rows or columns in chromosome in which the gene is assigned to be mutated and N^{max} is the maximum number of rows or columns which is allowed to be in circuit structure. Then the new value of the mutated genes can be defined using one of the following three strategies:

1. Global geometry mutation (GGM), *new_value* $\in$ [1, N^{max}];
2. Bounded geometry mutation (BGM), *new_value* $\in$ [1, $N^{current}$];
3. Local geometry mutation (LGM), *new_values* = $N^{current}\pm1$.

The first strategy allows us to generate new circuit geometry. The number of columns and rows is randomly defined in the ranges [1, $N^{\max}_{columns}$] and [1, $N^{\max}_{rows}$] respectively. The new number of columns and rows is not related to the current circuit geometry. The second strategy is used to reduce the circuit geometry used in chromosome. The idea of this strategy came from observing that using the global geometry mutation tended to produce circuits with larger circuit geometry. The third strategy is assigned based on the idea of local search of circuit geometry. This strategy guarantees to produce comparatively small numbers of new cells in the chromosome in comparison with the first one.

After *new_value* is defined, the geometry mutation is performed in the following manner. First, consider the case when the mutated gene is the number of columns. In this case the new circuit structures, shown in **Fig. 3** (structures A and B), can be synthesised. If (*new_value* > $N_{columns}$), we have to add new columns in the chromosome representation (**Fig. 3** (structure A)). The gates in new columns are initialised using the 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_value* < $N_{columns}$) we have to remove some columns in the circuit structure (**Fig. 3** (structure 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. 3** can be synthesised. If (*new_value* > N_{rows}) the new rows of gates are added to the circuit structure (**Fig. 3** (structure 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_value* < N_{rows}) the last (N_{rows} **-** *new_value*) rows are removed from the circuit structure (**Fig. 3** (structure D)). In this case the inputs of the remaining gates as well as circuit outputs can refer to gates which are no longer present. Therefore each gate genotype and the circuit outputs have to be repaired.

1.5 Recombination

Recombination is implemented with uniform crossover. For two chromosomes, the uniform crossover generates two new chromosomes by swapping two genes in chromosomes. Because our chromosome structure contains three levels, on each level the components of chromosome can be examined like a “gene” or “swapping block”. Thus we have three different crossover operators: 1. Gene uniform crossover; 2. Cell uniform crossover; 3. Geometry uniform crossover. 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 inputs of the 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. Therefore, the selection of the correct crossover rate and its type is very important.

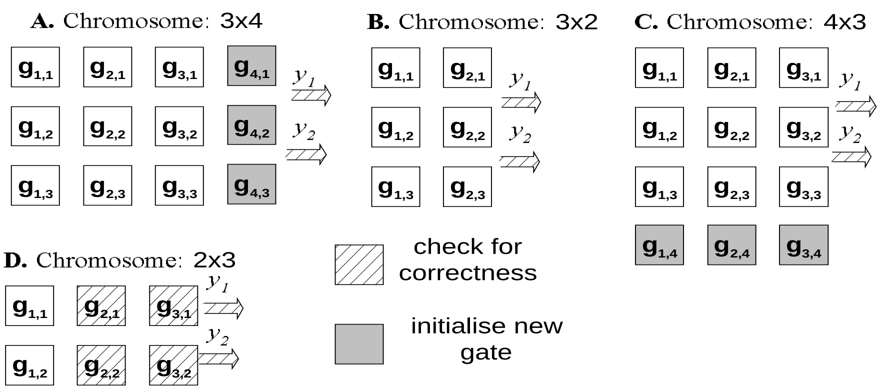

Fig. 3. The geometry mutation process for a chromosome with geometry 3x3

When we refer to the *gene uniform crossover*, we mean that any gene of logic cell as well as the circuit outputs can be exchanged. In case of *cell uniform crossover* the data, describing the behaviour of a logic gate such as functional gene, inputs, control input, are swapped. In *geometry uniform crossover*, the columns or rows of logic cells in addition to circuit outputs are involved in the crossover process. In this case whole column or row of logic gates is swapped and connections are restored if necessary.

Let us consider the "restoring process" in the case when the cells to be swapped belong to chromosomes with different circuit layouts. In this case the cell will refer to different cells in the circuit because of the specific features of encoding. In order to avoid it we correct the cell data in such a way that they refer to the cells positionally located in the same place as with the parents' chromosome. In the case when the cell contains a connection to a non-existent cell, a new connection is randomly generated such that it is valid. Let us consider the case mentioned above with an example of cell uniform crossover with parent chromosomes with 3x2 and 3x3 circuit geometry and assume the cell to be swapped is located in 2nd row and 3nd column (**Fig. 4**). Let us consider the case where the cell from parent 2 is exchanged with cell in parent 1. This cell has connection to the 10th and 11th cells in circuit. Positionally it corresponds to the cells located in 2nd column and 1st and 2nd (10 and 11) rows. When we exchange this cell in the chromosome with 3x3 circuit geometry, this cell now represents the connections with cells located in 1st column and 3rd row and in 2nd column and 1st row. Thus the positional connection is broken. In order to restore it we have to reassign the inputs for this cell according to the labelling process in chromosome. Thus this cell now will be described to <-4 11 12>. The same process is applied to the cell in parent 1. But in this case the input refers to the cell located in 2nd column and 3rd row. Because the chromosome where this cell is going to be allocated has only 2 rows (parent 2 has only 2 rows), this input has to be initialised. Thus the "restoring process" allows us to preserve the positional connections of cells and provides a less destructive process.

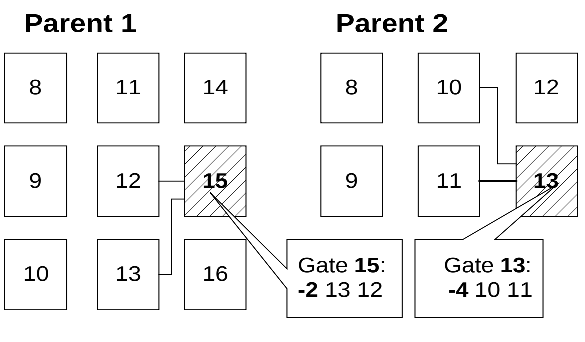

Fig. 4. Parents for cell uniform crossover

2 Experimental Results

In this section we will consider some experimental results obtained for the one-bit adder with carry and two-bit multiplier. We perform two main types of experiments: 1) Fitness function; 2) Crossover and Mutation strategies.

The main idea of these experiments is to define which of the GA strategies allows us to determine whether circuit geometry evolution brings some advantages or not. The initial data for the experiments is given in **Table 2**.

2.1 Experiment 1: Two fitness function strategies

The following experiment shows us how using different fitness evaluation strategies affects the GA performance and the quality of circuits evolved. For this purpose the same experiments were performed for fixed and flexible circuit geometry with and without the F_2 fitness function. The experimental results obtained are summarised in the Table 3. Comparing the best average F_1 finesses and the number of 100% functional cases for 3 mutation strategies, we find that the global geometry mutation is the most effective. But in terms of the number of active gates in the circuit evolved the best results were obtained in Table 3. It is interesting to note that when we evolve functions during 1000 (add1c.pla) or 3000 (mult2.pla) generations, we do not achieve significant improvements in terms of the number of active gates in circuit. When we increase the number of generations to 50000 it is clear that the average best F_2 fitness is improved. Thus, in the case of add1c.pla function we can notice improvements for 2-3 gates, but in case of mult2.pla it is improved only slightly. One of the reasons why we can see only small improvements for the mult2.pla function is that the first GA with F_1 only achieves a sufficient number of 100% functional circuits when the number of generations is this large. Therefore the optimising fitness function F_1+F_2 does not have long enough to make a significant difference. It is interesting to note that when we use a fixed geometry, the average best F_1 fitness is higher in comparison with the same experiments for flexible circuit layout. However the average best F_2 fitness for this case is the lowest one and this does not provide good solutions in terms of the number of active gates. If we consider the GA performance in terms of the number of 100% functional circuits evolved, it is best to use the fixed circuit geometry but if we use the flexible one we should employ global geometry mutation. Thus, we can conclude that fixed geometry is useful only in terms of 100% functional circuits evolved but that using a flexible circuit geometry provides better quality circuits. Note that using (F_1+F_2) fitness strategy allows us to improve the quality of circuits evolved as well.

Table 2. Initial data

Target function	Experiment 1		Experiment 2	
	Add1c.pla	**Mult2.pla**	**Add1c.pla**	**Mult2.pla**
Population size	15	15	15	15
# of generations	1000 / 50000	3000 / 50000	1000	4000
# of GA runs	100	100	100	100
Crossover type	Cell Uniform	Cell Uniform	Gene Uniform Cell Uniform Geometry Uniform	Gene Uniform Cell Uniform Geometry Uniform
Crossover rate	0.6	0.6	0.6	0.6
Selection type	Tournament	Tournament	Tournament	Tournament
Selection pressure	1.0	1.0	1.0	1.0
Mutation type	Cell Geometry	Cell Geometry	Cell Geometry	Cell Geometry
Circuit mutation rate	1.2	1.2	1.2	1.2
Geometry mutation rate	5.0	5.0	5.0	5.0
Fitness type[1]	F_1 / F_1+F_2	F_1 / F_1+F_2	F_1+F_2	F_1+F_2
Type of geometry mutation strategy[2]	*GGM* *BGM* *LGM*	*GGM* *BGM* *LGM*	*GGM* *BGM* *LGM*	*GGM* *BGM* *LGM*
Gate distribution	Proportional	Proportional	Proportional	Proportional
Geometry	Flexible Fixed	Flexible Fixed	Flexible Fixed	Flexible Fixed
Max # of rows	5	5	5	5
Max # of columns	5	5	5	5
Levels-back	2	2	2	2

Table 3. Experiment 1: Strategy of fitness function

Geometry Type	Geometry Mutation Type	# generations	Fitness Type	Add1c.pla			Mult2.pla		
				Average best fitness1 (F_1)	**Average best fitness2 (F_2)**	**# 100% cases**	**Average best fitness1 (F_1)**	**Average best fitness2 (F_2)**	**# 100% cases**
Fixed	-	*1000 / 3000*	F_1	98.6875	114.696	79	94.0938	110.750	4
		1000 / 3000	F_1+F_2	97.6875	114.712	66	94.2500	111.000	7
		50000	F_1+F_2	99.8750	117.418	98	98.3594	111.152	46
Flexible	Global	*1000 / 3000*	F_1	94.7500	117.206	34	92.6719	111.400	5
		1000 / 3000	F_1+F_2	94.0625	117.483	29	91.0300	114.000	2
		50000	F_1+F_2	99.4375	118.923	91	96.0781	115.115	26
	Boundary	*1000 / 3000*	F_1	92.1875	116.909	22	90.0312	100.000	0
		1000 / 3000	F_1+F_2	91.0000	116.793	29	90.5000	115.000	1
		50000	F_1+F_2	96.9375	119.369	65	94.6562	115.273	22
	Local	*1000 / 3000*	F_1	94.5625	117.816	38	91.4219	104.000	1
		1000 / 3000	F_1+F_2	93.4375	117.345	29	88.5300	116.000	2
		50000	F_1+F_2	98.8125	119.244	82	95.5312	115.227	22

[1] F_1 and (F_1+F_2) are the fitness without and with estimation of the number of active gates in circuit
[2] GGM, BGM and LGM are the global, boundary and local geometry mutations

2.2 Experiment 2: Crossover and Mutation strategies

In this series of experiments we have tried to define the best mutation and crossover strategy as well as the best overall GA strategy. We investigate the GA performance for three types of crossover (gene, cell and geometry uniform crossovers), two types of mutation (circuit and geometry mutations) and three types of geometry mutation (global, boundary and local). We define two main strategies for evolution and initialisation processes. Each of these strategies is defined by the homogeneous or heterogeneous circuit layout. The results obtained are shown in Table 4 – Table 7. The tables are organised according to the GA strategies used.

The first results were obtained for the case when during initialisation and evolution processes the heterogeneous circuit layout is produced (Table 4). So, chromosomes with different circuit layouts are produced at each stage of the GA. The circuit and geometry mutations have been used together. Observing the results according to geometry mutation type we find that global geometry mutation is better in terms of the best average F_1 fitness and the number of 100% functional circuits evolved. It is interesting to note that the local geometry mutation produces the second best results after global mutation. However in terms of the quality of circuits evolved (estimated by the average best F_2 fitness) we find that boundary or local geometry mutation is more advantageous. We can not make any conclusions as to which is the best crossover.

In the next series of experiments (Table 5) the condition were the same as before except that only geometry mutation is used. Firstly, it is interesting to note that we obtain very poor results in terms of the number of 100% cases evolved and no 100% functional two-bit multiplier was obtained. Furthermore, we only evolved a few one-bit adders. We see that the worst crossover operator is the cell uniform type. The highest best average F_1 fitness was obtained for geometry uniform crossover. We conclude that using circuit mutation and geometry mutation together is more effective.

Table 4. Experiment 2: Using circuit mutation (p_m=1.2) and geometry mutation (p_m=5.0)

Crossover Type	Geometry Mutation Type	Add1c.pla			Mult2.pla		
		Average best fitness1	**Average best fitness2**	**# 100% cases**	**Average best fitness1**	**Average best fitness2**	**# 100% cases**
Initialisation		**Heterogeneous circuit layout**					
Evolution		**Heterogeneous circuit layout**					
Gene Uniform	Global	94.875	117.263	38	92.7188	115.000	4
	Boundary	90.8750	116.542	24	90.1875	115.000	3
	Local	93.6875	117.964	28	91.5156	114.667	3
Cell Uniform	Global	95.0625	116.950	40	91.9062	115.500	2
	Boundary	92.5625	117.500	28	92.2656	111.667	3
	Local	93.6875	117.733	30	91.9375	115.000	1
Geometry Uniform	Global	94.8125	116.944	36	93.3750	111.833	6
	Boundary	91.4375	117.500	20	91.6406	114.000	5
	Local	94.9375	117.303	33	92.5781	113.000	2

Table 5. Experiment 2: Using geometry mutation (p_m=5.0), flexible geometry

Crossover Type	Geometry Mutation Type	Add1c.pla			Mult2.pla		
		Average best fitness1	**Average best fitness2**	**# 100% cases**	**Average best fitness1**	**Average best fitness2**	**# 100% cases**
Initialisation		**Heterogeneous circuit layout**					
Evolution		**Heterogeneous circuit layout**					
Gene Uniform	Global	79.0000	100.0	0	82.0625	100.0	0
	Boundary	76.1250	114.5	2	79.6406	100.0	0
	Local	81.0000	119.0	2	81.4531	100.0	0
Cell Uniform	Global	80.7500	100.0	0	82.6562	100.0	0
	Boundary	73.3125	100.0	0	80.1406	100.0	0
	Local	76.2500	100.0	0	82.0156	100.0	0
Geometry Uniform	Global	83.0000	118.0	4	84.4688	100.0	0
	Boundary	76.9375	115.0	1	81.8750	100.0	0
	Local	81.7500	116.0	4	83.3594	100.0	0

Table 6. Experiment 2: Using circuit mutation (p_m=1.2), fixed geometry 4x4 (add1c.pla) and 5x5 (mult2.pla)

Crossover Type	**Add1c.pla**			**Mult2.pla**		
	Average best fitness1	**Average best fitness2**	**# 100% cases**	**Average best fitness1**	**Average best fitness2**	**# 100% cases**
Initialisation	**Homogeneous circuit layout**					
Evolution	**Homogeneous circuit layout**					
Gene Uniform	95.3125	113.976[3]	42	94.2500	110.333	3
Cell Uniform	93.0625	115.636[1]	22	95.1094	110.100	10
Geometry Uniform	94.8750	114.000[1]	36	95.7969	110.857	14

Table 7. Experiment 2: Using circuit mutation (p_m=1.2), flexible geometry at the initialisation stage

Crossover Type	**Add1c.pla**			**Mult2.pla**		
	Average best fitness1	**Average best fitness2**	**# 100% cases**	**Average best fitness1**	**Average best fitness2**	**# 100% cases**
Initialisation	**Heterogeneous circuit layout**					
Evolution	**Homogeneous circuit layout**					
Gene Uniform	92.3125	116.708	24	92.1250	117.000	1
Cell Uniform	94.3750	116.366	41	91.3438	113.667	3
Geometry Uniform	91.5625	116.786	28	92.2031	113.000	1

The next series of experiments (Table 6) show us the GA performance with circuit mutation only. In this case the homogeneous circuit layout is retained for both stages of initialisation and evolution. In this case we are using an entirely fixed geometry. We find that geometry uniform crossover works best for mult2.pla and gene uniform crossover for add1c.pla. In terms of the best average F_2 fitness the best performance is obtained with cell uniform crossover for add1c.pla and with geometry uniform crossover for mult2.pla. This could be explained because of the different complexity associated with the landscapes of these two functions: mult2.pla is considerably more difficult to evolve than add1c.pla. We always find that the geometry uniform crossover is the best for mult2.pla. It is interesting to note that using gene uniform crossover produces the better results in terms of the best average F_1 fitness and the number of 100% cases. Comparing the obtained results with the best results discussed above we can conclude that the pure fixed geometry works perfectly well in comparison with flexible one when we need to obtain the maximum number of 100% functional circuits. But in terms of the best average F_2 fitness we find that we evolve poorer circuits. Thus we find one of the disadvantages of using the fixed geometry: evolving a large number of 100% cases does not provide us with the best solution in terms of the number of active gates used in circuit.

Table 7 shows results of some interesting experiments. We have tried to combine using fixed and flexible circuit geometry at the different stages of GA performance. Thus we have a heterogeneous circuit layout at the initialisation stage and homogeneous circuit layout during evolution. This means that the different circuit layout can be defined only at the initialisation. It is interesting to note that in this case cell uniform crossover delivers the best performance. In both cases of add1c.pla and mult2.pla functions we obtain better results in terms of the number of 100% cases. The best average F_2 fitness has been significantly improved and increased by more then 2 gates in comparison with results obtained for the previous experiment (Table 5). This improvement was probably caused by the fact that a variety of circuit geometries were available so that by selection we can arrive at a more optimal solution. However we should note that the best level F_2 fitness was obtained using heterogeneous circuit layout at both stages of initialisation and evolution with geometry mutation.

3 Conclusions and Further Work

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. These two aspects allow us to improve the quality of evolved circuits in terms of the number of active gates in circuit. We investigated several strategies of GA with and without flexible circuit layout. Analysis of experimental results allows us to make the following conclusions:

[3] * F_2 is normalised for 5x5 circuit geometry, thus this value is comparable with other.

1. The fixed circuit geometry is very good when we need to evolve as many 100% functional circuits as possible.
2. Using flexible geometry permits us to decrease the number of gates in a circuit in comparison with using a fixed circuit layout.
3. Utilizing geometry mutation without circuit mutation leads to poor results in terms of the number of 100% functional circuits evolved.
4. Applying the (F_1+F_2) strategy with a large number of generations allows us to obtain better results in terms of the number of active gates in circuit.
5. Suitable crossover operators depend on the choice of GA strategy.
6. The global geometry mutation produces the best results in terms of the average best F_1 fitness and the number of 100% functionality circuits evolved.
7. The boundary and local geometry mutations are very good in terms of obtaining the circuits with smaller numbers of active gates in circuit.

So, we can conclude that using flexible circuit geometry is more beneficial to the quality of the evolved circuits.

A great deal of further work could be done in the area. Other multi-objective approaches such as the Pareto technique, with weight aggregation could be experimented with, and further investigation of using multi-input multi-output gates could be carried out.

References

1. Coello Coello, C. A., A. D. Christiansen and A. H. Aguirre (1997). Automated Design of Combinational Logic Circuits using Genetic Algorithms in *Proc. of the Int. Conf. on Artificial Neural Nets and Genetic Algorithm (ICANNGA'97)* Eds.: D.G. Smith, et al. Publisher: Springer-Verlag, pp. 335-338.

2. Iba H., Iwata M., and Higuchi T. (1997). Machine Learning Approach to Gate-Level Evolvable Hardware 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, Publisher: Springer-Verlag, Heidelberg, pp. 327 – 343.

3. 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, Publisher: Wiley.

4. Miller J. F., and Thomson P. (1998). Evolving Digital Electronic Circuits for Real-Valued Function Generation using a Genetic Algorithm in *Genetic Programming: Proc. of the 3rd Annual Conf.*, Eds.: Koza, J. R. et al. San Francisco, Publisher: CA: Morgan Kaufmann , pp. 863-868.

5. 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, Publisher: Springer-Verlag, Heidelberg, pp.390 – 405.

6. Thompson A. (1998). On the Automatic Design of Robust Electronics Through Artificial Evolution in *Proceedings of The 1st Int. Conf. on Evolvable Systems: From Biology to Hardware (ICES98), Lecture Notes in Computer Science*, Eds.: Sipper M., et al Vol. 1478, Publisher: Springer-Verlag, Heidelberg, pp. 13-24.

7. Layzell P. (1998). A New Research Tool for Intrinsic Hardware Evolution. in *Proceedings of The 1st 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.

8. 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.

9. Miller J. F., Thomson P. (1998). Aspects of Digital Evolution: Geometry and Learning, in *Proceedings of The 1st Int. Conf. on Evolvable Systems: From Biology to Hardware (ICES98), Lecture Notes in Computer Science*, Eds.: Sipper M., et al Vol. 1478, Publisher: Springer-Verlag, Heidelberg, pp. 25-35.

10. Miller J. F., Thomson P. (1998). Aspects of Digital Evolution: Evolvability and Architecture, in *Proceedings of The 5th Int. Conf. on Parallel Problem Solving from Nature (PPSNV), Lecture Notes in Computer Science*, Vol. 1498, Publisher: Springer-Verlag, Heidelberg, pp.927-936.

11. Miller J., T. Kalganova, N. Lipnitskaya and D. Job (1999). The Genetic Algorithm as a Discovery Engine: Strange Circuits and New Principles in *Proc. Of AISB Symposium on Creative Evolutionary Systems (CES'99)*, Edinburgh, UK.

12. 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).* Lausanne, Switzerland, Eds.: M. Sipper, et al. Publisher: Springer-Verlag, pp. 78-89.

13. 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*, Fukuoka, Japan. Publisher: IEEE Press.

14. Kalganova T., J. Miller and T. Fogarty (1999). Evolution of the digital circuits with variable layouts in *Proc. of the Genetic and Evolutionary Computation Conference (GECCO'99)*, Orlando, Florida, USA.

15. Kalganova T. (1999). A New Evolutionary Hardware Approach for Logic Design in *Proc. of GECCO Student Workshop in association with GECCO'99*, Orlando, USA.