Kalganova, T., J. Miller and T. Fogarty (1998) Some Aspects of an Evolvable Hardware Approach for Multiple-Valued Combinational Circuit Design. Proc. of Second International Conference on Evolvable System: From Biology to Hardware (ICES'98). Lausanne, Switzerland. pp. 78-89.

Some Aspects of an Evolvable Hardware Approach for Multiple-Valued Combinational Circuit Design

Tatiana Kalganova, Julian F. Miller and Terence C. Fogarty

Dept. of Computing, Napier University, 219 Colinton Road, Edinburgh, UK, EH14 1DJ. E-mail: t.kalganova, j.miller,t.fogarty@dcs.napier.ac.uk Ph: +44 (0)131 455 4304/5

Abstract. In this paper a gate-level evolvable hardware technique for designing multiple-valued (MV) combinational circuits is proposed for the first time. In comparison with the decomposition techniques used for synthesis of combinational circuits previously employed, this new approach is easily adapted for the different types of MV gates associated with operations corresponding to different algebra types and can include other more complex logical expressions (e.g. single-control MV multiplexer called T-gate). The technique is based on evolving the functionality and connectivity of a rectangular array of logic cells. The experimental results show how the success of genetic algorithm depends on the number of columns, the number of rows in circuit structure and levels-back parameter (the number of columns to the left of current cell to which cell input may be connected). We show that the choice of the set of MV gates used radically affects the chances of successful evolution (in terms of number of 100% functional solutions found).

1 Introduction

Evolvable Hardware extends the concepts of Genetic Algorithms to the evolution of electronic circuits. A central idea of this is that each possible electronic circuit can be represented as a chromosome in an evolutionary process in which the standard genetic operations over the circuits, such as initialization, recombination, elitism, selection are carried out. The evolving circuits may be evaluated using software simulation models [4, 7, 8, 12], or in some cases implemented directly in hardware [2, 11]. A number of investigations have been carried out for synthesis of binary logic circuits [4, 7].

Multiple-Valued (MV) Logic refers to the adoption of logic systems having more than two levels [3, 9]. It is generally felt that MV logic allows circuits to have increased functionality with a reduction in wiring density.

In this paper we present a method for the synthesis of combinational MV circuits. This approach is an extension of evolvable hardware method for binary logic circuits proposed in [1, 10]. The first attempts to evolve MV arithmetical combinational circuits have been discussed in [5, 6] and here we present more detailed results and further discussion about the most suitable parameters used in this approach. We examine the most effective geometry which allows the evolution of 100% functional circuits for a 3-valued one bit adder with carry logic function. We present the results of a number of experiments which show that the number of rows which allows most fully functional solutions depends on the number of outputs in logic function, and that the optimal number of columns depends strongly on the levels-back parameter. The levels-back parameter defines the number of columns to the left of current cell to which cell input may be connected. An analysis of different sets of MV gates shows that there is a particularly effective set of MV gates which allows us to evolve three times as many 100% functional solutions as any other set. A notable feature of this paper is that it shows how the chosen geometry of circuit design has a strong influence on the GA performance.

2 The Evolvable Hardware Method for Combinational MV Circuits

The method proposed here is based on evolving combinational MV networks employing a rectangular array of the logic cells. The logic cells in this array are uncommitted and can be removed from the network if they prove to be redundant. Let $X=\{x_1, x_2, \ldots, x_n\}$ and $Y=\{y_1, y_2, \ldots y_m\}$ be input and output variables of implemented *r*-valued function respectively. The number of variables in *X* is denoted by *n* and the number of functions in *Y* is defined by *m*. The inputs that are made available are logic constants “0”, “1”, …,“*r*-1”, where *r* is the radix, all primary inputs $x_1, x_2, \ldots, x_n$ and the unary operators acting on the primary inputs, for instance, complement of all primary inputs $\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}$. To illustrate this let us consider a 3-input 3-output 3-valued logic function which will be implemented on a 4 x 4 array of 3-valued logic cells with two inputs and one output (Figure 1). Input encoding is carried out as follows. The number of logic constants encoded as 0, 1, 2 respectively is 3. The set of primary inputs $X=\{x_1, x_2, x_3\}$ are labeled with 3, 4 and 5 output numbers respectively. The inverted inputs $\overline{x_1}, \overline{x_2}, \overline{x_3}$ are labeled as 6, 7 and 8 correspondingly.

Each cell in the array is labeled with an output number. The first cell having an output is labeled $(2n + r)$. These are numbered connection points, which are important for the network representation of the chromosome (discussed below). Each cell is described by $(k+1)$ integers, where *k* is the number of inputs in the logic MV cell. The first *k* numbers describe the cell inputs and the final integer defines the functional type of the cell. If this final integer is positive then the cell is assumed to be a MUX gate, otherwise it refers to one of the gate types listed in Table 1. The gate types listed in Table 1 have the following definitions (for an *r* valued 2-input function). Note that the over-bar in all the expressions in Table 1 refers to the complement (or inversion) operator defined below:

$MAX(x_1 , x_2)$ is the maximum of inputs x_1 and x_2 and in the binary case becomes identical to the inclusive-OR operation. $MIN(x_1 , x_2)$ is the minimum of inputs x_1 and x_2 and in the binary case becomes identical to the AND operation. $TSUM(x_1 , x_2)$ is referred to as the truncated sum operator and is defined as $MIN(x_1+x_2 , r-1)$, this again reduces to the inclusive-OR operation for the binary case. The truncated product operator $TPRODUCT(x_1 , x_2)$ is defined as $MAX(x_1+ x_2-(r-1) , 0)$, which in the binary case becomes the AND operation. *MODSUM*(x_1, x_2) and *MODPRODUCT*(x_1, x_2) are defined as (x_1+x_2) and $x_1 \cdot x_2$ in modulo r arithmetic. The former becoming the exclusive-OR operation in the binary case, while the latter reduces to the AND function. Finally the complement of a r-valued symbol x, indicated with an over-bar is defined as $(r-1) - x$ and represents the NOR operation in binary logic.

The user can choose any subset of this list of gates and later we examine the relative performances of various sub-sets. Note that the list of MV gates presented in Table 1 is just a small subset of possible gates proposed in the literature [5, 6].

In the discussion which follows we will use the following terminology. Associated with each rectangular array of logic cells are three geometrical constants: the number of rows, N_{rows}, the number of columns, $N_{columns}$, and the connectivity of the circuit which we refer to as the levels-back parameter, l.

The levels-back parameter l for a cell j ($j = 1, ..., N_{rows}$) in column i ($i = 1, \ldots, N_{columns}$) defines how many columns of cells to the left of column i can have their outputs connected to the inputs of the cell, this also applies to the final circuit outputs, and if $(i-l) <= l$, then any of the primary inputs can be connected to the cell in question. Also the primary outputs can be connected to the cell outputs according to the levels-back parameter. For example if the levels-back parameter is 2 for the circuit geometry shown in Figure 1, then the cells numbered from 17 to 21 can be additionally defined as outputs of this circuit.

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

Gene value	*Gate function*
-1	$MAX(x_1,x_2)=x_1 \vee x_2$
-2	$\overline{MAX(x_1,x_2)} = \overline{x_1 \vee x_2}$
-3	$TSUM\ (x_1, x_2)=x_1 \oplus x_2$
-4	$\overline{TSUM(x_1,x_2)} = \overline{x_1 \oplus x_2}$
-5	$TPRODUCT\ (x_1, x_2)=x_1 \otimes x_2$
-6	$\overline{TPRODUCT(x_1,x_2)} = \overline{x_1 \otimes x_2}$
-7	$MIN\ (x_1, x_2)=x_1 \wedge x_2$
-8	$\overline{MIN(x_1,x_2)} = \overline{x_1 \wedge x_2}$
-9	$MODSUM\ (x_1, x_2) = x_1 + x_2$
-10	$\overline{MODSUM(x_1,x_2)} = \overline{x_1 + x_2}$
-11	$MODPRODUCT\ (x_1, x_2) = x_1 \cdot x_2$
-12	$\overline{MODPRODUCT(x_1,x_2)} = \overline{x_1 \cdot x_2}$
-13	$\overline{x}$

A *chromosome* defines the connections in the network between the MV logic cells. In a circuit of 2-input, 3-valued logic gates each gate is represented by triple of integer numbers $<c^1 c^2 c^3>$ which define the connectivity between gates and the type of the examined gate.

The i-th cell is represented in the circuit by the integers $\mathbf{c_i}=<c_i^1 c_i^2 c_i^3>$. Gene c_i^3 defines the type of MV gate. If $c_i^3<0$, then the i-th cell is two-input one-output logic gate, else when $c_i^3 \geq 0$ the i-th cell a 3-1 MUX gate with single control input c_i^3. In the case of two-input one-output logic gate, the genes c_i^1 and c_i^2 represent the first and second input of logic gate respectively. If c_i^1, $c_i^2 < (2n+r)$, then the inputs of the i-th gate are connected to the primary inputs of circuit, otherwise these inputs are linked to the outputs of the c_i^1-th and c_i^2-th gates respectively. The gene $c_i^3<0$ defines the type of logic gates which are coded according to Table 1. For example the cells numbered 13 and 20 shown in Fig

1 are 2-input 3-valued gates. The cell 13 describes function $f_{13} = \overline{MODPRODUCT(2, x_1)}$ because gene c_{13}^{1}=2 defines logical constant, c_{13}^{2}=3 corresponds to the primary variable x_1 and the gene c_{13}^{3}=-12 describes *MODPRODUCT* gate with inverted output. The cell 20 describes logic function $f_{20} = TPRODUCT(\overline{x_3}, f_{13})$, where function f_{13} denotes behavior of cell 13. The gene c_{20}^{1}=8 determines the inverted variable $\overline{x_3}$. The gene c_{20}^{2}=13 shows the connection between 13 and 20 cells and is considered as second variable of *TPRODUCT* function. The third gene c_{20}^{3}=-5 encodes *TPRODUCT* type for the cell 20. So, the output of cell 20 can be described as $f_{20} = TPRODUCT(\overline{x_3}, \overline{MODPRODUCT(2, x_1)})$. In the case of MUX gate the gene c_i^{1} describes the first input of MUX gate, the next inputs of MUX gate are connected to the cells numbered (c_i^{1}+1), (c_i^{1}+2), …, (c_i^{1}+r-1). Note that the gene c_i^{2} is not used in this case. For example the cell 12 shown in Fig 1 is T-gate (MUX gate) with control input 3 (c_{12}^{3}=3=x_1) and with inputs 4, 5 and 6 respectively (the first input is c_{12}^{1}=4=x_2), the second input is 5=x_3 and the third input is 6= $\overline{x_1}$. Note that the gene c_{12}^{2}=7 is not used. The last genes of chromosome describe the *m* outputs of implemented function and define the outputs of cell, which should be considered as circuit outputs. For instance the output of cell 20 defines the logic function y_1 described by the expression: $y_1 = f_{20} = TPRODUCT(\overline{x_3}, \overline{MODPRODUCT(2, x_1)})$.

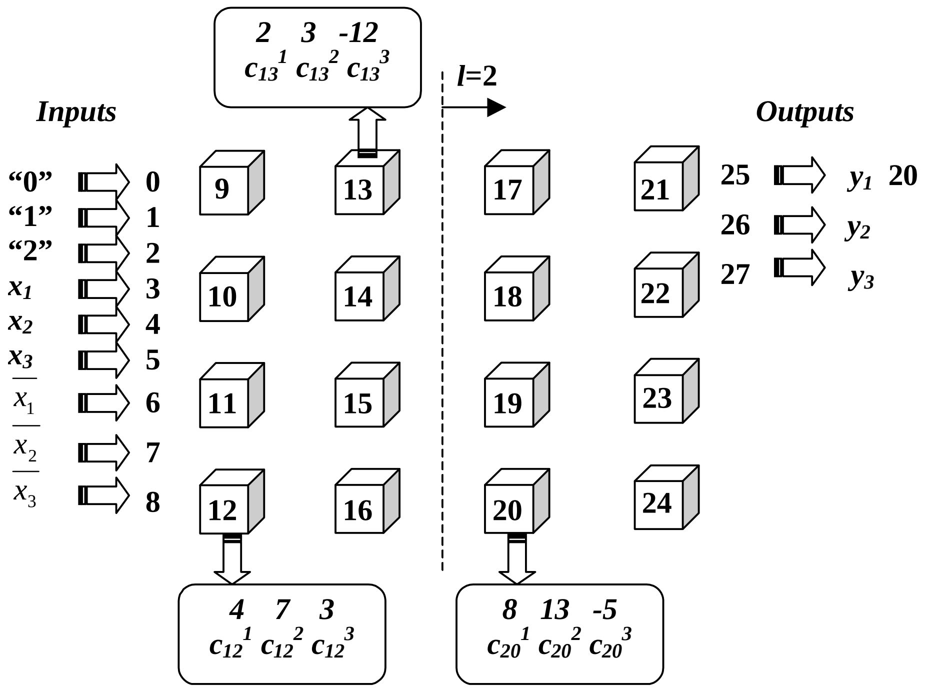


Fig 1. A 4 x 4 geometry of uncommitted MV logic cells with netlist numbering

The *fitness function* is defined as the percentage of the correct output digits for every input combination of the implemented MV functions. The *uniform crossover* and *mutation operators* are built in the traditional way for integer representation chromosome. The number of genes mutated in current population is defined by the *mutation rate*. The *crossover rate* shows how many chromosomes breed in current population. *Elitism* is used to promote the best chromosome obtained from one population to the next. The *selection operation* used is a variation on standard tournament selection (of size two) in which the winner of the tournament (the chromosome with the greater fitness) between two chromosomes chosen at random is accepted with certain probability (otherwise the loser of the tournament is chosen). This probability is called the *tournament discriminator*. If this probability is set to unity, then the tournament becomes the standard tournament sized two selection mechanisms. If this probability is less than unity, then the selection pressure on the population is reduced. Note that often the chromosome contains cells that are not actually connected to any of the outputs. The process of removing these redundant cells is carried out for chromosomes with 100% functionality after the GA has completed.

3 Relationship Between the Levels-back Parameter, Circuit Geometry and GA Performance

Table 2. Initial data

Population size	30
Crossover rate	100
Mutation rate	15
Number of generations	200
Number of GA runs	100
PLA file processing	Add32.pla
Radix of logic	3
Tournament Discriminator	70

The experiments in this paper were aimed at correctly evolving the functionality of a one-digit 3-valued adder with output carry. This is a circuit with 2 inputs and 2 outputs and requiring 9 input and output conditions for full specification (see Appendix 1.). The main purpose of these experiments was to investigate how the levels-back parameter and circuit geometry affect the performance of the GA. We investigate how the percentage of the 100% functionality circuit evolved by the GA and mean fitness values depend on the levels-back parameter and the number of rows and columns. In this series of experiments the functional basis contains all the MV gates shown in Table 1. The *functional basis* is the set of MV operators used to synthesize the circuit. The logical constants as well as the primary and inverted input variables can only be connected in accordance with the levels-back parameter. This prevents cells becoming wires in the circuit. For example cell MAX with one of the input 0 passes the signal of another input to next cell without change. The GA parameters used here are shown in Table 1.

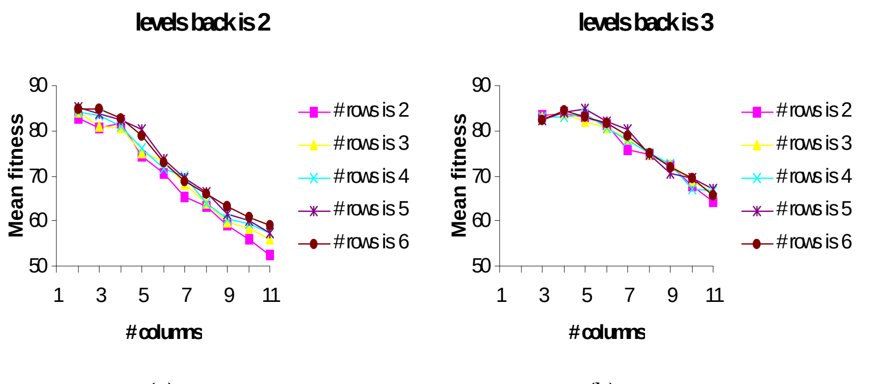


Fig. 2. Dependence the mean fitness on the number of columns for the different number of rows

Figures 2 and 3(a) show that the GA performance is not influenced strongly by the number of rows. However it is clear that the number of columns in the chosen geometry has a significant effect.

We can see that the number of rows used in the geometry has some effect since with *l=1* the number of rows must be at least as large as the number of outputs. Associated with every logic function having a minimum number of logic cells is a minimum depth (number of columns) which we denote, *d*. If the number of columns is less than *d* then clearly we cannot evolve 100% solutions.

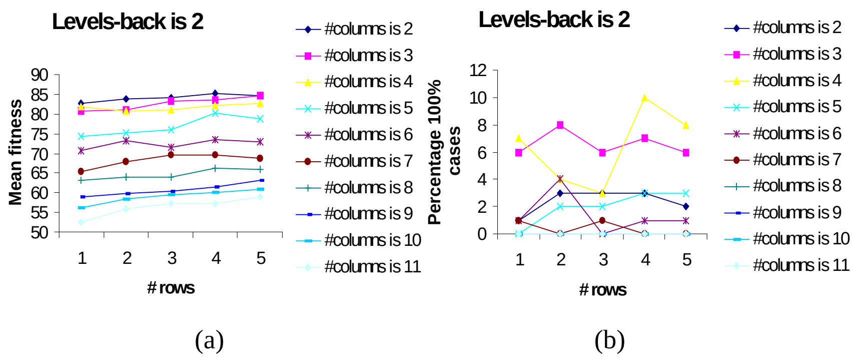
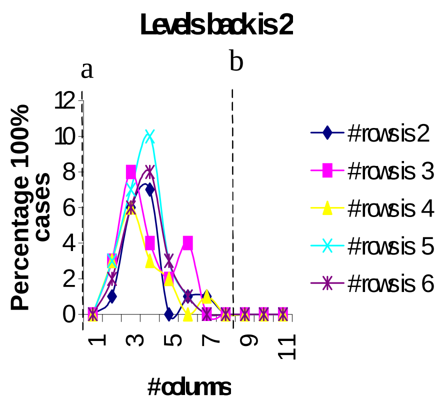


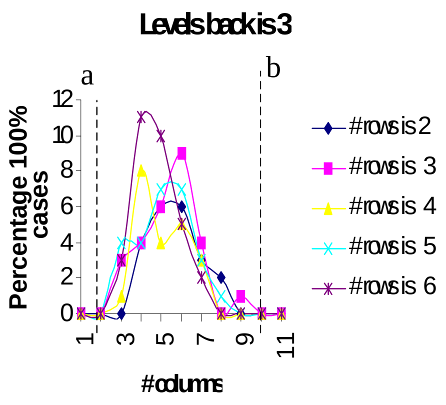
Fig. 3. Dependence of percentage 100% cases and the mean fitness on the number of rows for the different number of columns

Examining Figure 4 (a)-(d) (note that the graphs have been plotted as continuous functions for ease of viewing) we see that as *l* increases the minimum geometry at which we can obtain 100% solutions has to increase. This is understandable since the probability that we would connect the inputs to the first column of cells and the outputs to the last column of cells decreases quickly. Thus the lowest cut-off point for circuit geometry (which indicates the minimum number of columns at which the 100% functionality circuit are not evolved) must increase. The marked cut-off of the highest point, *b*, appears to indicate that only circuits within a certain size range are likely to be evolved so that when the geometry becomes too large (in terms of columns) it becomes no longer possible to connect up even the largest circuit.

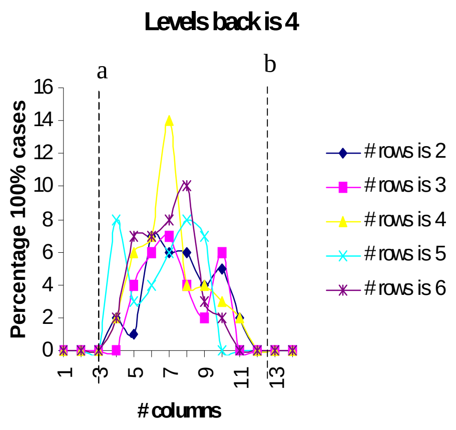
There are certainly a number of interesting features of these graphs (i.e. the marked modality) and it looks as if there are species of circuits each with a characteristic size which become more or less likely to be evolved as the number of columns of cells and the levels-back parameter are altered.



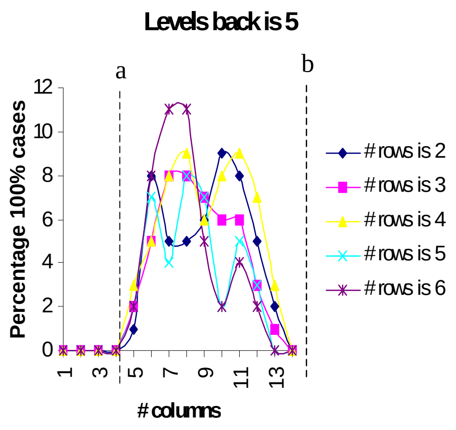
(a)



(b)



(c)



(d)

F i g . 4. Dependence percentage 100% cases on the number of columns for the different number of rows (levels-back = 2, 3, 4 and 5)

The dependence of the mean fitness with the number of columns is shown in Figure 5 for levels-back parameters equal to 2 and 3 respectively. Points *a′* and *a″* show the minimum fitness at which 100% functionality for the one-digit 3-valued adder with carry can be achieved. At these points the number of columns in the circuit geometry are 8 and 9 respectively. It is clear that if the mean fitness is less than 60% then the probability of obtaining a circuit with 100% functionality drops to zero. If the mean fitness is 85% or more for the optimal number of columns (points *b′* and *b″*) circuits with 100% functionality can be evolved with the highest probability.

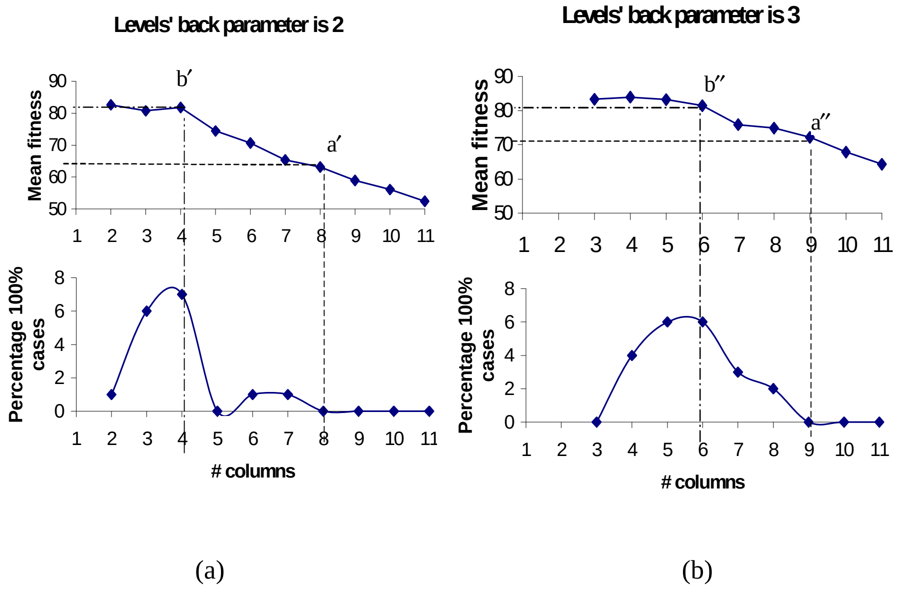


Fig. 5. Dependence of the mean fitness and the percentage of 100% cases on the number of columns.

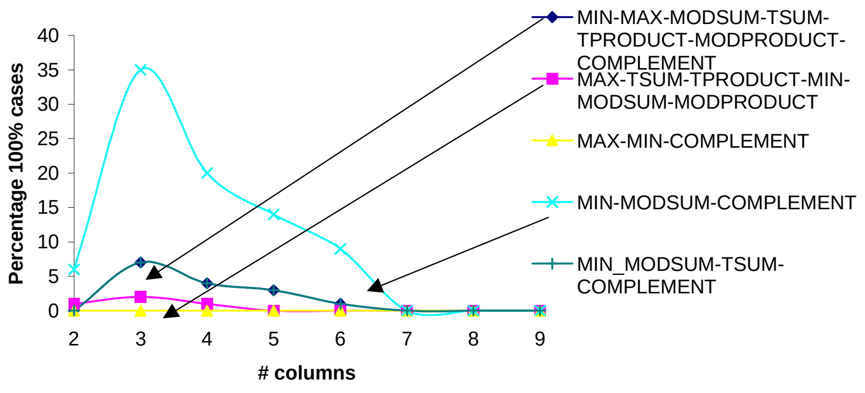


Fig.6. Dependence the GA performance on the set of MV gates

4. Influence of the MV gate set on the GA performance

In this section we will discuss how GA performance depends on the set of MV gates chosen for circuit design.

Figure 6 shows how GA performance depends on the set of MV gates which are used in the circuit design. It was found that of the gate sets used the best set was MIN-MODSUM-COMPLEMENT. This basis is startlingly better than some of the other bases. This set allows a five fold improvement in the GA performance in comparison with the MIN- MAX- TSUM- TPRODUCT- MODSUM- MODPRODUCT- COMPLEMENT set and a ten fold improvement over the MIN-MAX-TSUM-TPRODUCT-MODSUM-MODPRODUCT set. Note that the attempts to evolve circuits using only MIN-MODSUM-TSUM gates or MAX-MIN-COMPLEMENT gates didn't give any 100% circuits. These experiments were carried out using the same initial parameters as before but with a fixed number of generations equal to 500. For each MV basis the GA runs 100 times. The vertical axis of Fig.6 shows the percentage of MV logic circuits evolved with 100% functionality in 100 runs of the GA.

It is interesting to note that we expected to receive the highest percentage of 100% functionality for the case when we use *all* well-known MV gates because it gives a bigger choice of any MV gate for evolved circuit than any particular functionally complete basis with has a much more restricted number of MV gates. However experimental results show that the correct choice of MV logic gate set allows us significantly increase the GA performance without changing any of the GA parameters. This indicates the importance of choosing the architecture on which to conduct the evolutionary process, and indeed, such a choice has a much more significant impact on the success of the evolution than fine tuning the GA parameters.

5 An evolved one-digit 3-valued adder with output carry circuit

The arithmetic MV circuits synthesized using the proposed method have an unusual structure. It is for this reason that the obtained results are very interesting and allow us to consider arithmetic MV circuits from a new viewpoint.

Now we present some of the arithmetical circuits that we have so far been able to evolve using the above method. This circuit implements the 3-valued one-digit adder with output carry.

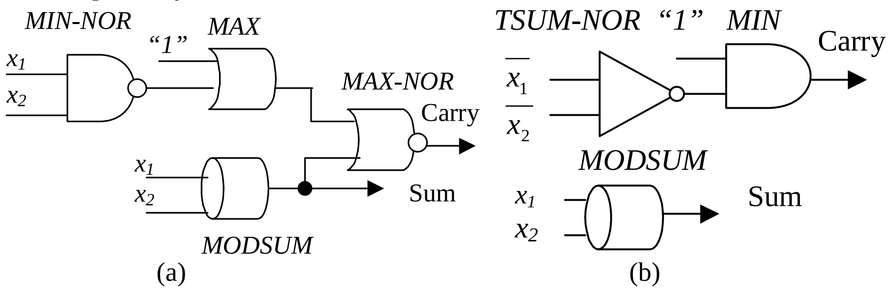


Fig. 7. Evolved solution for the one-digit 3-valued adder with output carry using (a) MIN-MAX-MODSUM (b) TSUM-MIN-MODSUM gates (note small circles represent inversion)

Using only MIN-MAX-MODSUM allowed the circuit shown in Figure 7(a) to be evolved (with a 3 column , 2 row geometry). The sum component in this design is implemented in the optimum way as it uses only one MODSUM gate. The analytical description of MV logic gates is given in Table 1. This circuit can be described analytically as follows:

$$y_{sum} = x_1 + x_2$$

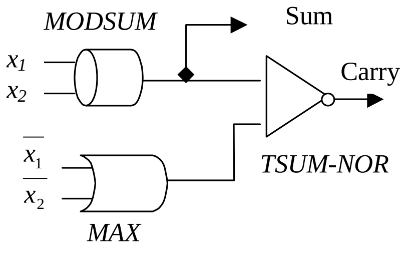
$$y_{carry} = \overline{(\overline{(x_1 \wedge x_2)} \vee 1) \vee (x_1 + x_2)}$$

When a set of basic gates had been changed to (with the 2x2 geometry) TSUM, MAX, MIN and MODSUM the design shown in Figure 7 (b) was evolved. The sum component in this circuit is the familiar sum-digit 3-valued circuit of conventional adders. Note that the sum and carry components in this design are implemented separately and can be expressed as follows:

$$y_{sum} = x_1 + x_2$$

$$y_{carry} = \overline{\overline{x_1}} \oplus \overline{\overline{x_2}} \wedge 1.$$

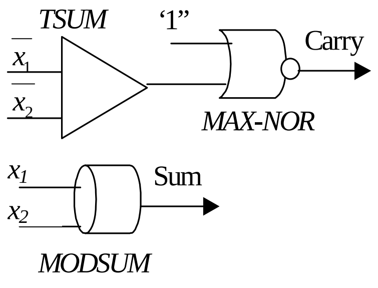
When the geometry was constrained to 2x2 and the set of basis gates chosen contained only MODSUM, TSUM and MAX gates with direct and inverted inputs and outputs the optimum designs shown in Figure 8 was obtained. This was a gratifying result to obtain as it is clear that these designs are an optimum solution. These circuits were previously unknown. The analytical representation of these designs are shown in Fig. 8 also.



$$y_{sum} = x_1 + x_2$$

$$y_{carry} = \overline{(x_1 + x_2) \oplus (\overline{x_1} \vee \overline{x_2})}$$

(a)



$$y_{sum} = x_1 + x_2$$

$$y_{carry} = \overline{(\overline{x_1} \oplus \overline{x_2}) \vee 1}$$

(b)

Fig. 8. Evolved optimum solution for the one-digit 3-valued full adder using MODSUM, TSUM and MAX

Evolving the one-digit 3-valued adder was easier to do with a larger geometry but resulted in a less efficient circuit. For instance, the circuits shown in Figure 7 have also been obtained with much larger geometries (3x4, 4x3, 4x4, 5x5, 4x6, 4x8, 4x10). Choosing too small a geometry ran the risk that no 100% solutions could be found because it was physically impossible to build the required functionality with few gates.

While using too large a geometry simply gave the GA too many possibilities to work with and it struggled to find the fully functional solutions.

6 SUMMARY

In this paper it has been shown that by evolving a linear chromosome of cell functionalities and connectivities based on a rectangular array of logic cells it is possible to evolve both traditional and novel designs for arithmetical MV circuits. The method uses a genetic algorithm to evolve both a netlist structure and functionality for MV logic cells. The fitness function tests the functionality of the circuit. This approach for the first time allows us to synthesize combinational MV circuits in different functionally complete basis or a combination of these. The method allows the synthesis of very novel circuit structures, which have never been seen before. We have evolved the 3-valued 2-digit adder with carry as an example. The results of some of the experiments showed that it is possible to improve the GA performance considerably by choosing carefully the number of columns of cells used and the levels-back parameter, but that the number of rows in the geometry was less important.

There are still many avenues for further work. Other ways of representing rectangular arrays of logic cells may be devised and also, the relationship between cell connectivity and the evolvability of designs has still to be explored, this would involve examining a suitable concept of cell-neighborhood. It is a feature of the current technique that one has to specify the functionality of the target circuit using a complete truth table, however this is impractical for circuits with large numbers of inputs. Further investigations are under way to see if these findings are carried over to other MV benchmarks.

REFERENCES

1. Fogarty T. C., Miller J. F., Thomson P.:Evolving Digital Logic Circuits on Xilinx 6000 Family FPGAs. The 2nd Online Conference on Soft Computing (1997), now pulished in Soft Computing in Engineering Design and Manufacturing, Roy R., and Pant R. K. (eds), Springer-Verlag, London, (1998) 299 - 305

2. Goeke M., Sipper M., Mange D., Stauffer S., Sanchez E., Tomassini M.: Online autonomous evolware, in Proceedings of The First International Conference on Evolvable Systems: From Biology to Hardware (ICES96), now published in Lecture Notes in Computer Science Vol. 1259, Springer-Verlag, Heidelberg, (1997) 96-106.

3. Hurst S.L.: MV Logic: Its Status and its Future, IEEE Transactions on Computers. Vol. C-33, (1984) 1160 - 1179

4. Iba I., Iwata M., Higuchi T: Machine Learning Approach to Gate-Level Evolvable Hardware. Proc. Of the 1st Int. Conference on evolvable Systems: From Biology to Hardware (ICES'96), now published in Lecture Notes in Computer Science, Springler-Verlag, Heidelberg, (1996) 118 - 135

5. Kalganova T., Miller J. F.: Evolutionary Approach to Design MV Combinational Circuits. Proc. of the 4th Int. conference on Applications of Computer Systems, ACS'97. Szczecin, Poland (1997) 333 – 339

6. Kalganova T., J.F. Miller and N. Lipnitskaya Multiple-Valued Combinational Circuits Sythesised using Evolvable Hardware Approach, Proc. of the 7th Workshop on Post-Binary Ultra Large Scale Integration Systems (ULSI'98) in association with ISMVL'98, Fukuoka, Japan, May 27-29, 1998.

7. Koza J. R.: Genetic Programming, The MIT Press, Cambridge, Massachusetts (1992)

8. Koza J.R., Andre D., Bennet III F.H., Keane M.A: Design of a High-Gain Operational Amplifier and Others Circuits by Means of Genetic Programming. Proc. Of the 6th Int. Conference on Evolutionary Programming, now published in Lecture Notes in Computer Science, Vol. 1213, (1997) 125 - 135

9. Mariani R., R. Roncella, R. Saletti, P. Terrini A Useful Application of CMOS Ternary Logic to the Realization of Asynchronous Circuits. Proc. of the 27th IEEE Int. Symposium on MV Logic (1997) 203 - 208

10. Miller J. F., Thomson P., Fogarty T. C.: Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study, chapter 6, in Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications. Editors: D. Quagliarella, J. Periaux, C. Poloni and G. Winter, published by Wiley, (1997)

11. Thompson A.: An evolved Circuit, Intrisic in Silicon, Entwined with Physics. Proc. Of the 1st Int. Conference on Evolvable Systems: From Biology to Hardware (ICES'96), now published in Lecture Notes in Computer Science, Springler-Verlag, Heidelberg, (1996) 390 - 405

12. Zebulum R., M. Vellasco, M. Pacheco: Evolvable Hardware Systems: Taxonomy, Survey and Applications. Proc. Of the 1st Int. Conference on Evolvable Systems: From Biology to Hardware (ICES'96), now published in Lecture Notes in Computer Science, Springler-Verlag, Heidelberg, (1996) 344 - 358

APPENDIX 1.

Truth table for 3-valued 2-input adder with output carry (Add32.pla)

x_1	x_2	**carry**	**sum**
0	0	0	0
0	1	0	1
0	2	0	2
1	0	0	1
1	1	0	2
1	2	1	0
2	0	0	2
2	1	1	0
2	2	1	1