Towards the *Automatic Design* of More Efficient Digital Circuits

Vesselin K. Vassilev
School of Computing
Napier University
Edinburgh, EH14 1DJ, UK
v.vassilev@dcs.napier.ac.uk

Dominic Job
School of Computing
Napier University
Edinburgh, EH14 1DJ, UK
d.job@dcs.napier.ac.uk

Julian F. Miller
School of Computer Science
University of Birmingham
Birmingham, B15 2TT, UK
j.miller@cs.bham.ac.uk

Abstract

This paper introduces a new methodology of evolving electronic circuits by which the process of evolutionary design is **guaranteed** *to produce a functionally correct solution. The method employs a mapping to represent an electronic circuit on an array of logic cells that is further encoded within a genotype. The mapping is many-to-one and thus there are many genotypes that have equal fitness values. Genotypes with equal fitness values define subgraphs in the resulting fitness landscapes referred to as* **neutral networks**. *This is further used in the design of a neutral network that connects the conventional with other more efficient designs. To explore such a network a navigation strategy is defined by which the space of all functionally correct circuits can be explored. The paper shows that very efficient digital circuits can be obtained by evolving from the conventional designs. Results for several binary multiplier circuits such as the three and four-bit multipliers are reported. The evolved solution for the three-bit multiplier consists of* 23 *two-input logic gates that in terms of number of two-input gates used is* 23.3% *more efficient than the most efficient known conventional design. The logic operators required to implement this circuit are* 14 AND*s,* 9 XOR*s, and* 2 *inversions (*NOT*). The evolved four-bit multiplier consists of* 57 *two-input logic gates that is* 10.9% *more efficient (in terms of number of two-input gates used) than the most efficient known conventional design. The optimal size of the target circuits is also studied by measuring the length of the neutral walks from the obtained designs.*

1. Introduction

The evolutionary design of electronic circuits refers to an autonomous process in which a highly efficient circuit *may* occur in a population of interacting instances of a logic function. The possibilities for automatic design of electronic circuits using evolutionary algorithms have been explored in the analogue [10, 23, 18, 2, 30, 22, 24] and digital [5, 4, 15, 6, 8, 13] domains. In general the methodology of evolving circuitry and machinery used is most easily described in the framework of a simple evolutionary algorithm. A disadvantage of such an approach however is that the evolution may end up with a functionally incorrect evolved circuit. This may cause a problem if circuit evolution is implemented on hardware and functionally correct circuit is required in real time.

This paper introduces a method of evolving electronic circuits by which the process of evolutionary design is *guaranteed* to produce a functionally correct solution. The method is inspired by the concept to study the evolutionary design of electronic circuits as a search on a fitness landscape. The fitness landscape is simply a search space derived from the combination of three components: a set of genotypes, a fitness function by which a fitness value is assigned to each genotype, and a neighbourhood relationship within the set of genotypes specified by the evolutionary operator [7]. These components define the landscape structure that affects the evolutionary search [9, 16, 11]. The fitness landscapes associated with the evolution of various arithmetic functions, mainly the two-bit multiplier circuit, were studied in [27, 29]. It was shown that the landscape underlying graph is the *generalised* Hamming hypercube that results in subspaces with different characteristics. The circuit evolution landscapes are characterised with *neutrality* [28, 13] that appeared to be beneficial for the evolutionary design of circuitry [3], and particularly the three-bit multiplier [25]. The neutrality is a landscape characteristic that refers to genotypes with equal fitness values [19, 21]. The set of genotypes with equal fitness values is called a *neutral network*, if the genotypes define a connected subgraph in the landscape.

Electronic circuits have been evolved and it has been reported that the obtained solutions significantly differ in construction from the conventional designs [15, 24, 13]. This implies a *gap* between the human design space and the efficient evolved designs. However, the fitness landscape asso-

ciated with the evolutionary design of an electronic circuit represents the space of all designs in such a way that allows one to *construct* a neutral network of functionally correct circuits, and thus, to connect the human design space with the other more efficient circuits. This is referred to as the *neutral bridge*. To explore the neutral bridge, a navigation strategy is defined by which one can evolve from the conventional to more efficient solutions. The method is applied to digital circuit evolution, particularly the evolution of several binary multiplier circuits. Solutions for the three and four-bit multiplier are proposed that are 23.3% and respectively 10.9% more efficient in terms of gates usage than the conventional designs. The evolved three-bit multiplier consists of 23 two-input gates that in terms of logic operations is 14 *AND*s, 9 *XOR*s, and 2 inversions (*NOT*). The evolved four-bit multiplier consists 57 two-input gates. Evidence that firstly, the three-bit multiplier cannot be further optimised, and secondly, the four-bit multiplier can be implemented with a less number of two-input gates is also proposed.

2. Digital Circuit Evolution

2.1. The Evolutionary Algorithm

The evolutionary algorithm used in the design of digital circuits in this paper is that adopted in the framework of Cartesian Genetic Programming in which the genotypes are rectangular arrays rather than trees [12, 14]. The algorithm deals with a population of digital feed-forward electronic circuits that are instances of a particular program. The population consists of $1+\lambda$ genotypes where λ is usually about 4. Initially the elements of the population are chosen at random. The fitness value of each genotype is evaluated, by calculating the number of total correct outputs of the encoded electronic circuit in response to all appropriate input combinations. The mechanism of population update is implemented by truncation selection and mutation that implies similarities with other evolutionary techniques such as $(1+\lambda)$ Evolution Strategy [20, 1] and the Breeder Genetic Algorithm [17]. To update the population, the mutation operator is applied to the fittest genotype, and hence, an offspring is generated. The offspring together with the parent constitute the new population. The mutation operator is defined as the percentage of genes in a single genotype that are to be randomly mutated. In this paper the percentage chosen results in 3 mutated genes per genotype.

2.2. The Genotype-Phenotype Mapping

To encode a digital electronic circuit into a genotype, a genotype-phenotype mapping is defined. This is done via rectangular array of cells each of which is an atomic two-input logic gate or a multiplexer. In this paper the allowed logic gates are *AND*, *AND* with one input inverted, and *XOR*.

In general the array consists of $n \times m$ three-input cells, n_I inputs, and n_O outputs. The inputs of the array are the inputs of the represented phenotype that in digital circuit evolution is a combinational circuit. The internal connectivity of the array is defined by the connections between the array cells. The inputs of each cell are only allowed to be inputs of the array or outputs of the cells with lower column numbers. The internal connectivity is also dependent on a *levels-back* parameter that defines the array inputs and cells to which a cell or an array output can be connected. This is done in the following manner. Consider that the levels-back parameter is equal to L. Then cells can be connected to cells from L preceding columns. If the number of preceding columns of a cell is less than L then the cell can also be connected to the inputs of the array. In this paper the array cells and outputs are maximally connectable since the number of rows is set to one and the levels-back is equal to the number of columns. The gate array output connectivity is defined in a similar way. The output connections of the array are allowed to be outputs of cells or array inputs. Again, this is dependent on the neighbourhood defined by the levels-back parameter. The number of outputs of the array is equal to the number of outputs of the represented phenotype. An illustration of the array of logic cells is given in Figure 1.

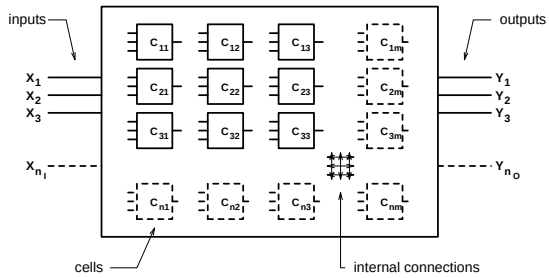

Figure 1. The phenotype that is a digital circuit is encoded within a genotype by an array of logic cells.

The genotype is a linear string of integers and it consists of two different types of genes that are responsible for the functionality and the routing of the evolved array of cells. Hence, the genotype is defined by four parameters of the array: the number of allowed logic functions, the number of rows, the number of columns, and the levels-back. The first parameter defines the functionality of logic cells, while the latter three parameters determine the layout and routing of the array. Note that the number of inputs and outputs of the array are specified by the objective

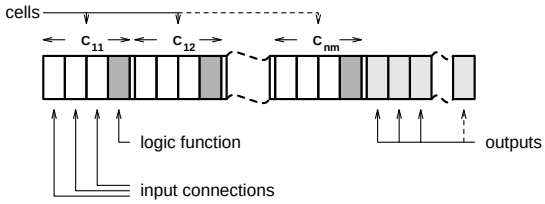

Figure 2. The genotype with respect the rectangular array of logic cells shown in Figure 1.

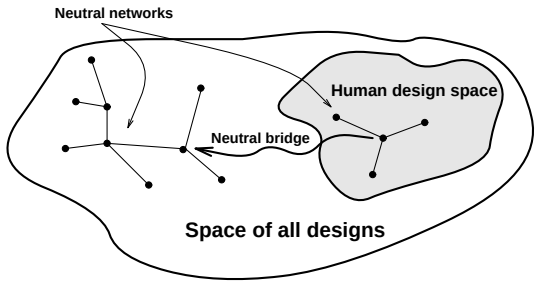

Figure 3. The "neutral bridge" between the human design space and the evolved more efficient circuits in the space of all designs.

function. The logic functions are represented by letters associated with the allowed cell functionality. The connections are defined by indexes that are assigned to all inputs and cells of the array. Each array input X_k is labelled with $k-1$ for $1 \leq k \leq n_I$, and each cell c_{ij} is labelled with an integer given by $n_I + n(j-1) + i - 1$ for $1 \leq i \leq n$ and $1 \leq j \leq m$. Thus the genotype consists of groups of four integers that encode the cells of the array, followed by a sequence of integers that represent the indexes of the cells connected to the outputs of the array. The first three values of each group are the indexes of the cells to which the inputs of the encoded cell are connected. In this case the third connection is redundant since only two-input gates are used. The last integer of the group represents the logic function of the cell. The genotype representation is illustrated in Figure 2.

3. Exploring the Space of Solutions

The space of all circuit designs can be explored by Evolutionary Algorithms. In [13] this was seen as a process of assembling circuit solutions from a number of component parts and testing them in their environment. Thus an efficient solution *may* occur far beyond the limits of the human design space. An efficient solution can also be attained starting from the conventional design by evolving more efficient modules that can replace parts (sub-circuits) of the design. Thus the space of all designs can be explored by moving on a neutral network composed of functionally correct solutions (Figure 3). The question that may arise here is how can one be sure that such a network exists?

The fitness landscapes associated with the evolution of digital circuits are characterised with neutrality [28] that appeared to be beneficial for circuit evolution [3, 25]. The neutrality was revealed by studying the information characteristics of the landscapes for the two and three-bit multiplier circuits as well as the four-bit parity function [13]. The sources of landscape neutrality in digital circuit evolution originate mainly from the genotype-phenotype mapping, and they are

Input redundancy Inputs of cells that are not used in the operating circuits. This refers to the case where the function of a cell does not use all inputs of the cell.

Cell redundancy Cells whose outputs are not connected in the operating circuit.

Functional redundancy The case in which the number of cells of a digital circuit is higher than the optimal number needed to implement this circuit.

Logic equivalency A characteristic of designs in which a (sub-)circuit can be substituted with another logically equivalent (sub-)circuit that has the same number of gates.

Phenotype equivalency The possibility to encode a digital circuit in different ways.

Here since only two-input gates are used, the sources of landscape neutrality are cell redundancy, functional redundancy, logic equivalency and phenotype equivalency. The functional redundancy, logic equivalency, and phenotype equivalency are hardly controllable characteristics of the genotype-phenotype mapping, since they are much more related to the nature of the problem and its genotype representation. Hence the only way to design a neutral network of functionally correct circuits is to allow cell redundancy. Indeed [26] showed that landscape neutrality embedded by cell redundancy is beneficial for the evolutionary design of digital circuits (agreed with findings in [3]), that answers the question from the previous paragraph in the affirmative.

To evolve in a network of functionally correct circuits, the evolutionary algorithm (section 2) was modified in the following way: firstly, the population is initialised with mutated copies of the conventional design and the design itself, and secondly, the best genotype in the population always represent the most *efficient functionally correct* circuit. A

functionally correct is the circuit with highest possible fitness. The efficiency of the functionally correct circuit is defined by the number of used gates. The lower the number of used gates is, the higher is the efficiency of the circuit. In itself, this is a method for automatic design because the evolution is seeded with designs that are generated by conventional techniques used in the logic synthesis of combinational circuits.

4. Evolving Binary Multiplier Circuits

Solutions of evolved multiplier circuits are given. These are the most efficient known designs, and they are obtained by evolving on the neutral bridge from the conventional designs. The evolved designs are obtained on an array of $1 \times m$ cells with levels-back set to m. The number of columns m is chosen with respect the evolved arithmetic function and the cell redundancy needed to construct the neutral network. The allowed logic gates is *AND*, *XOR*, and *AND* with one input inverted. For each solution, a study of the possibility for further reduction of the number of gates is also proposed. This is done by measuring the length of the neutral walks from each genotype recorded in every improvement of the efficiency of the circuit. The algorithm of neutral walks as given in [19] is defined as follows: start from a configuration, generate all neighbours, select a neutral one at random that results in an increase in the distance from the starting point and continue moving until the distance cannot be further increased. Here two genotypes were considered as neutral if they represent functionally correct circuits with the same number of gates.

Solutions for the two-bit multiplier circuits are not discussed in this paper since this was exhaustively studied in [13] where it was shown that the number of two-input gates of the most efficient evolved and conventional designs is 7 (in fact the conventional design is implemented by 8 two-input gates one of which can be easily removed with a simple manipulation).

4.1. Three×Two-Bit Multiplier

The three×two-bit multiplier is a combinational circuit in which the logic operation carried out is three-bit by two-bit multiplication. The conventional design of this arithmetic circuit requires 17 two-input logic gates. The most efficient design achieved by evolving from the conventional solution requires 13 two-input logic gates This is a 23.5% more efficient design. The logic operators required to implement the circuit are 9 *AND*s, 4 *XOR*s, and 1 inversion (*NOT*). To evolve a 13 two-input gates solution is not a very difficult task. Indeed 55 out of 100 evolutionary runs of $200,000$ generations gave 13 gates solutions. These were evolved on an array of 1×17 cells, allowing the cell redundancy to increase during the run.

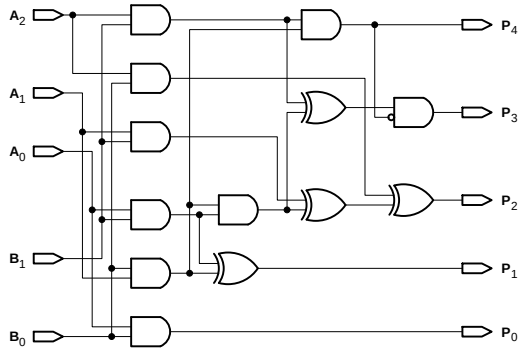

Figure 4. The three×two-bit multiplier circuit evolved from the conventional design. The circuit consists of 13 two-input gates: *AND*, *XOR*, and *AND* with one input inverted.

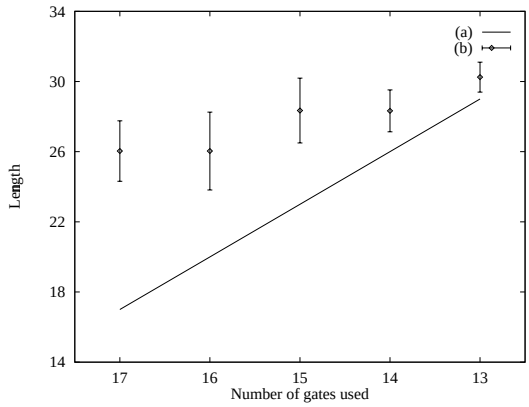

Figure 5. The length of neutral walks on the "neutral bridge" for the three×two-bit multiplier: (a) the number of redundant genes, and (b) the average length of $1,000$ neutral walks per recorded genotype.

A simple observation of the evolved designs revealed that some of the circuits are similar in construction, especially those achieved in longer evolutionary runs for which the increase of the efficiency was implemented gate by gate. An example of such a design is shown in Figure 4. The circuit seems to be the most efficient since the length of the neutral walks measured for each recorded genotype tends towards the number of redundant genes, shown in Figure 5. The figure gives the length of the neutral walks averaged on $1,000$ walks (indicated with diamonds) together with stan-

dard deviations. The line represents the number of redundant genes resulting only from input and cell redundancy. These characteristics are obtained for an array of 1×17 cells. For an array of $n \times m$ cells, the number of redundant genes defined by the two types of redundancy is equal to $nm + 3r_c$ where r_c is the number of redundant cells. The difference between the length of the neutral walks and the number of redundant genes obtained for the most efficient design is to be expected since the circuit is logically equivalent to many different designs with the same number of gates.

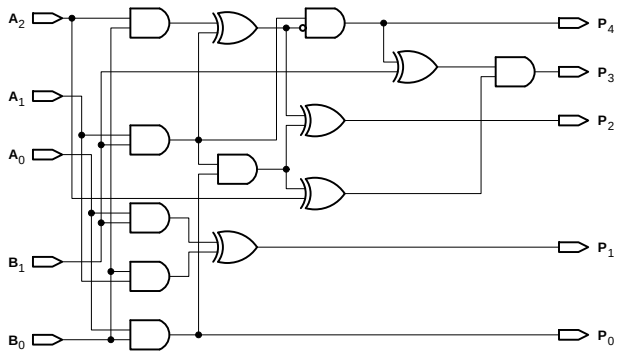

Figure 6. The three×two-bit multiplier circuit evolved from the conventional design - a strange solution. The circuit consists of 13 two-input gates: *AND*, *XOR*, and *AND* with one input inverted.

The circuits that differ in construction from the aforementioned designs can be classified as strange solutions. Indeed the example in Figure 6 is strange in that how the multiplication is carried out in the circuit. For instance, one of the *AND* gates in the most left column of gates has been transferred to *XOR* and moved to the right. Further observation showed that such solutions usually appear in very short runs characterised with sharp increase in the efficiency of the best circuit. It can be assumed that these designs are not typical evolutionary designs, since they seem to occur accidentally.

4.2. Three×Three-Bit Multiplier

It is relatively easy to evolve a 24 two-input gates solution for the three×three-bit multiplier (three-bit multiplier) circuit [1]. Indeed the 24 gates solutions found are 21 in 100 evolutionary runs of 20 million generations on an array of 1×30 cells. However the 24 gates solution is not the optimal one. More efficient circuits were achieved by increasing the cell redundancy. Thus 4 of 100 evolutionary runs of 20 million generations gave 23 two-input gates solutions. These were evolved on an array of 1×35 cells, by allowing 5 redundant cells. The schematic of an evolved three-bit multiplier is shown in Figure 7. The circuit consists of 23 two-input gates and it is implemented by 14 *AND*, 9 *XOR*, and 2 *NOT* logic operators. The logic operators should not be confused with two-input logic gates! The two-input logic gates are basic units used in the modern FPGA[2] technology, while the logic operators indicate basic logic functions in the Boolean algebra. The design in itself is very similar in construction to other 24 gates solutions for the three-bit multiplier evolved from *scratch*. For a comparison, one may refer to [13] (see also the appendix in [27]).

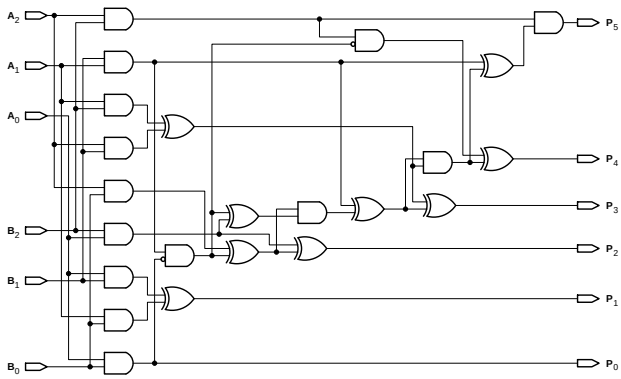

Figure 7. The three-bit multiplier circuit evolved from the conventional design. The circuit consists of 23 two-input gates: *AND*, *XOR*, and *AND* with one input inverted.

The 23 two-input gates design seems an optimal solution. Indeed the length of the neutral walks from the genotypes recorded in every improvement of the efficiency of the circuit tends to the number of redundant genes resulting from the input and cell redundancy. The length of the neutral walks averaged on $1,000$ walks per genotype is given in Figure 8. The line in the figure shows the number of redundant genes resulting from the input and cell redundancy. These characteristics are calculated for an array of 1×35 cells. The results imply that the functional redundancy is maximally reduced.

4.3. Four×Three-Bit Multiplier

The logic operation carried out in the four×three-bit multiplier is four-bit by three-bit multiplication. The conventional design of the circuit requires 47 two-input gates.

[1] The conventional design of the three-bit multiplier circuit consists of 30 two-input gates.

[2] Field-Programmable Gate Array.

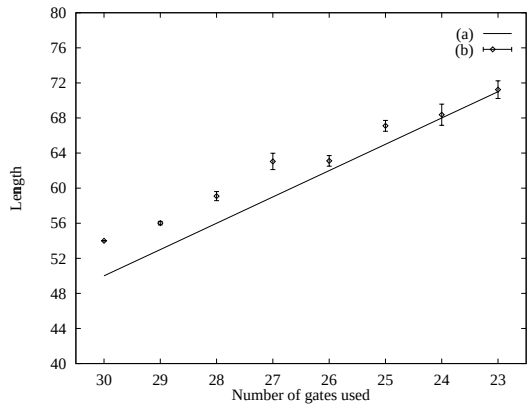

Figure 8. The length of neutral walks on the "neutral bridge" for the three-bit multiplier: (a) the number of redundant genes, and (b) the average length of $1,000$ neutral walks per recorded genotype.

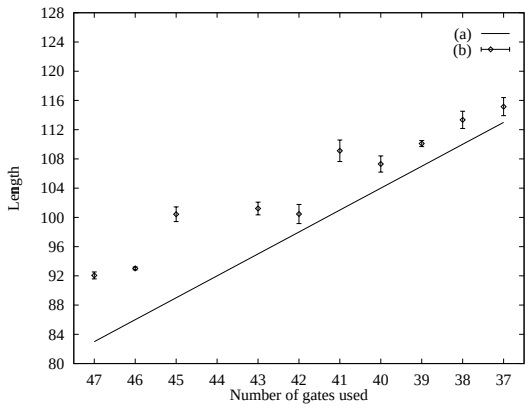

Figure 9. The length of neutral walks on the "neutral bridge" for the four×three-bit multiplier: (a) the number of redundant genes, and (b) the average length of $1,000$ neutral walks per recorded genotype.

Initially the evolution from the conventional design was set to proceed on an array with five redundant cells. Many experiments were carried out and many efficient solutions were found, the most efficient of which required 38 gates. The length of the neutral walks from the 38 two-input gate solution were close enough to the number of redundant genes to conclude that this might be most efficient design. However, having the experience of evolving the 23 gates solution for the three-bit multiplier, the experiment was repeated with an increased number of redundant cells. Thus another evolutionary run was performed on an array of 1×56 cells (9 redundant cells). This gave at generation $100,984,378$ a solution of 37 two-input gates. This is an improvement of approximately 21.28%. The evolved four×three-bit multiplier is shown in Figure 10. It is implemented by 22 *AND*, 15 *XOR*, and 7 *NOT* logic operators.

Figure 9 shows the length of the neutral walks averaged on $1,000$ walks. Again, these are represented in the figure with diamonds, while the line is the number of redundant genes resulting from the cell redundancy and input redundancy. The starting points for the walks were the recorded genotypes during the evolutionary run. Although the length of the walks imply that the functional redundancy is optimised, there is a reason to believe that the circuit can be implemented with 36 two-input gates. Indeed the length of the walks from the genotype of the most efficient design differs from the number of redundant genes in a small range, and the reason for this might be the existence of either logic equivalency or functional redundancy. The figure also shows that the number of redundant cells is high enough to attain further optimisation.

4.4. Four×Four-Bit Multiplier

The evolutionary design of the four×four-bit multiplier (four-bit multiplier) circuit is a very difficult task. When comparing to the previous cases, the difficulty is rapidly increased, a consequence of the problem with scale in the circuit design. The conventional design consists of 64 two-input gates. One evolutionary run of $700,000,000$ generations was performed. The most efficient four-bit multiplier circuit attained in this evolutionary run occurred in generation $643,274,721$, and it consists of 57 two-input gates. The schematic of the circuit is shown in Figure 11. It can be seen that it requires 35 *AND*, 22 *XOR*, and 10 *NOT* logic operations. The circuit was evolved from the conventional on an array of 67 cells. It is 10.93% more efficient from the conventional design in terms of number of gates used.

The solution shown in Figure 12 is not an optimal one. This was revealed by measuring the length of the neutral walks from the genotypes recorded during the run. The length of the neutral walks together with the number of redundant genes are depicted in Figure 12. These characteristics were estimated in the same manner as before. The figure shows that the functional redundancy can be further optimised.

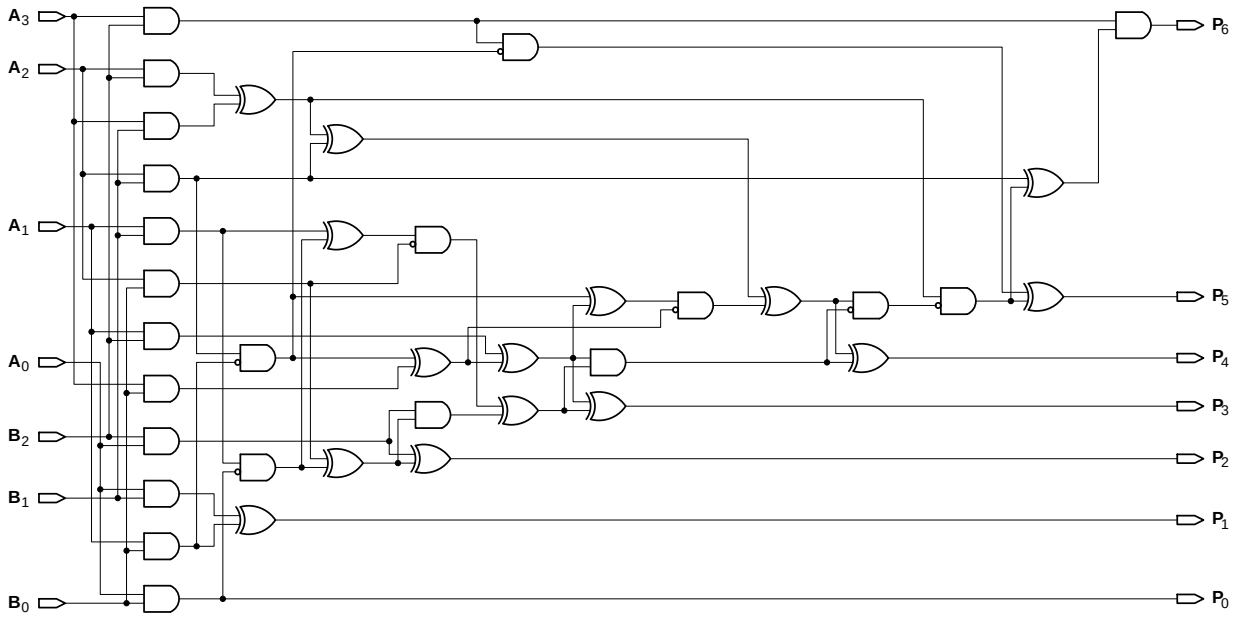

Figure 10. The four×three-bit multiplier circuit evolved from the conventional design. The circuit consists of 37 two-input gates: *AND*, *XOR*, and *AND* with one input inverted.

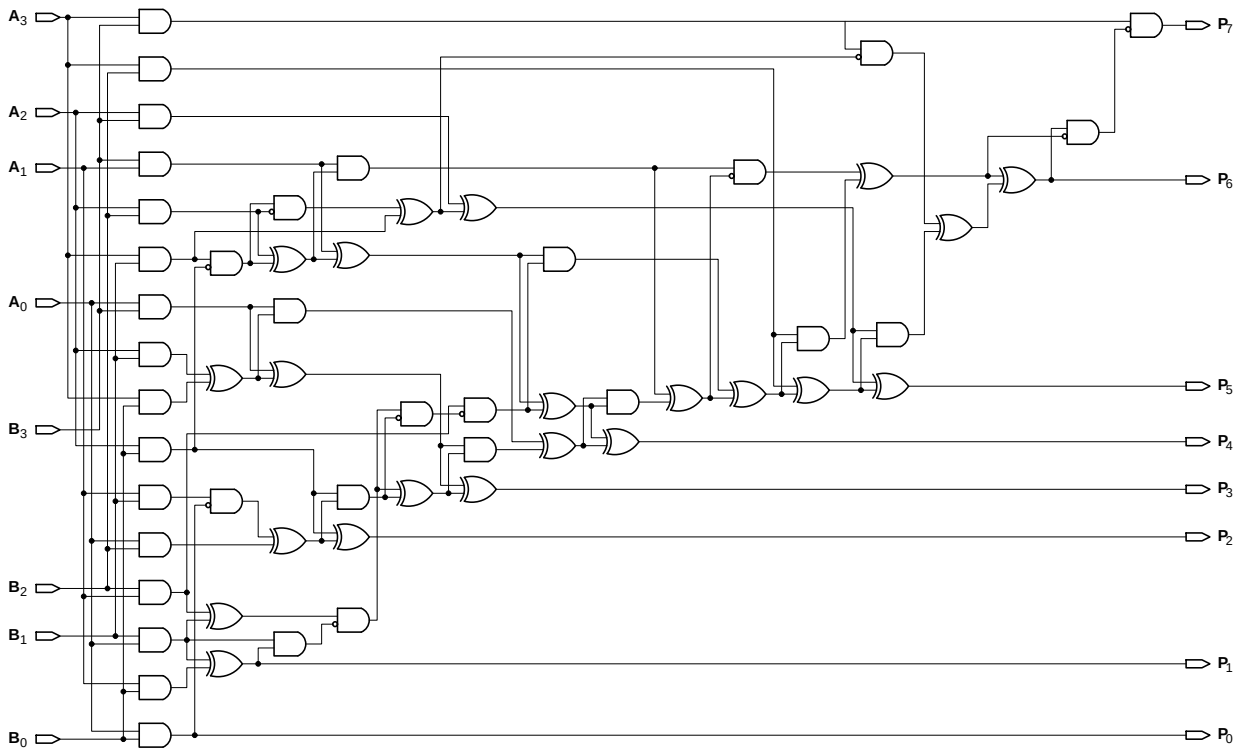

Figure 11. The four-bit multiplier circuit evolved from the conventional design. The circuit consists of 57 two-input gates: *AND*, *XOR*, and *AND* with one input inverted.

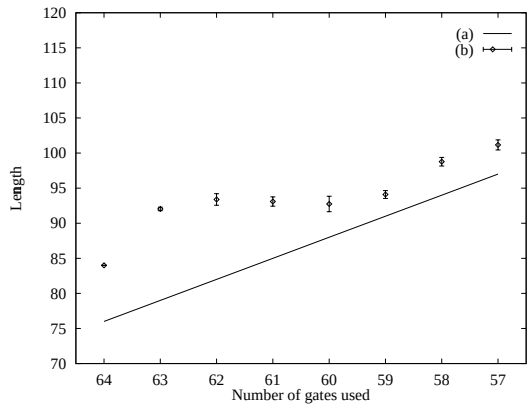

Figure 12. The length of neutral walks on the "neutral bridge" for the four-bit multiplier: (a) the number of redundant genes, and (b) the average length of $1,000$ neutral walks per recorded genotype.

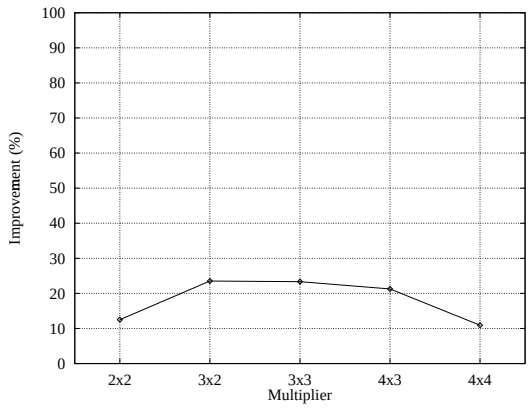

Figure 13. The percentage of improvements in terms of number of two-input gates attained for the binary multiplier circuits.

5. Discussion

A method for evolving electronic circuits has been introduced. The method uses an evolutionary algorithm in which the initial population is seeded with the conventional design of the target circuit. In fact the population can be initialised with any functionally correct solution. What is more important here is how to ensure the existence of a network that will connect all functionally correct solutions? This is possible by allowing redundancy in the genotype. The genotype in Cartesian Genetic Programming represents a rectangular array of cells (nodes) that in digital circuit evolution are merely logic gates. The layout of the array specifies the number of cells used in the evolutionary design. When the number of cells is higher than the number of gates required to implement the evolved electronic circuit, it is said that the array has redundant cells. Cell redundancy is vitally important for the evolutionary design of circuits [25]. It allows sub-circuits to occur that later may be connected in the operating circuit and thus to cause an increase of the efficiency of the circuit.

The method provides an alternative way of evolving electronic circuits by which a functionally correct solution that may be more efficient from the conventional design (the starting point) is guaranteed. This allows the automatic design of circuits with a number of *desirable* characteristics such as fault tolerance, an optimal number of components, etc. Here the method was applied to digital circuit evolution and particularly the design of binary multiplier circuits. The reported solutions are significantly more efficient than the conventional designs in terms of two-input gates usage.

The percentage of improvements attained is shown in Figure 13 (the conventional design of the two-bit multiplier is considered in the figure with 8 two-input gates).

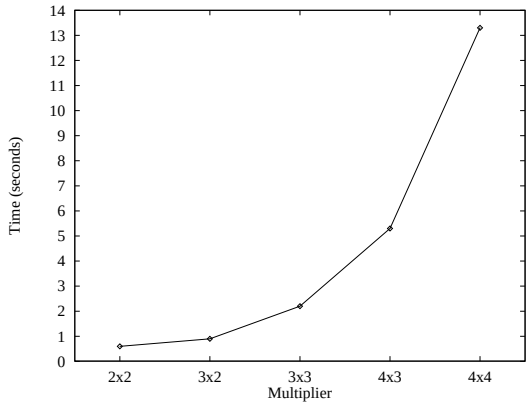

Figure 14. The time consumed by Pentium 200MHz computer to perform $10,000$ generations with a population of 5 elements for various multiplier circuits.

Some will probably argue that the experiments reported in this paper are time consuming. Indeed to attain an efficient design for the four-bit multiplier one needs to evolve millions of generations. This however does not represent any difficulty, since $10,000$ generations for the four-bit multiplier with a population of 5 elements take on an ordinary computer (Pentium 200MHz, Win95, RAM 64MB) exactly 13.3 seconds. The major impediment in circuit design is the problem of *scale* and it comes from the very fast grow of the

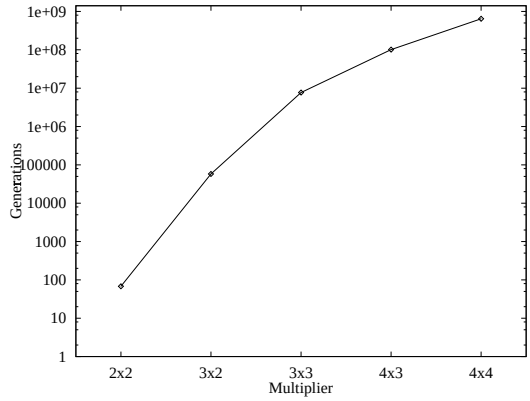

Figure 15. The number of generations required to evolve the circuits reported in this paper.

size of the truth table of the evolved logic function. This is illustrated in Figure 14. The figure represents the time consumed by Pentium 200MHz computer to perform $10,000$ generations with a population of 5 elements for various multiplier circuits. The number of generations also grows with the size of the circuit. The generations performed to obtain the reported circuits are shown in Figure 15 (the 7 gates two-bit multiplier requires 68 generations from the 8 gates conventional design). The figure reveals that the grow is nearly exponential.

In this paper efficient evolved designs for the binary multiplier circuits have been proposed. These circuits differ from the conventional designs. When comparing an evolved with a conventionally designed circuit one may probably notice that the logic operation carried out is built in a completely different manner. The question here is what is the mechanism of computation in the evolved designs? Is it possible to infer a principle and thus to define a methodology for building more efficient designs? All these are questions for future research.

References

[1] T. Bäck, F. Hoffmeister, and H. P. Schwefel. A survey of evolutionary strategies. In R. Belew and L. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, 1991. Morgan Kaufmann.

[2] S. J. Flockton and K. Sheehan. Intrinsic circuit evolution using programmable analogue arrays. In M. Sipper, D. Mange, and A. Pérez-Uribe, editors, *Proceedings of the 2nd International Conference on Evolvable Systems: From Biology to Hardware*, volume 1478 of *Lecture Notes in Computer Science*, pages 144–153, Heidelberg, 1998. Springer-Verlag.

[3] I. Harvey and A. Thompson. Through the labyrinth evolution finds a way: A silicon ridge. In T. Higuchi, M. Iwata, and W. Liu, editors, *Proceedings of the 1st International Conference on Evolvable Systems*, volume 1259 of *Lecture Notes in Computer Science*, pages 406–422, Berlin, 1996. Springer-Verlag.

[4] H. Hemmi, T. Hikage, and K. Shimohara. Adam: A hardware evolutionary system. In *Proceedings of the 1st International Conference on Evolutionary Computation*, volume 1, pages 193–196, Piscataway, NJ, 1994. IEEE press.

[5] T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, and T. Furuya. Evolving hardware with genetic learning: A first step towards building a darwin machine. In J.-A. Meyer, H. L. Roitblat, and W. Stewart, editors, *From Animals to Animats II: Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour*, pages 417–424, Cambridge, MA, 1993. MIT Press.

[6] H. Iba, M. Iwata, and T. Higuchi. Machine learning approach to gate-level evolvable hardware. In T. Higuchi and M. Iwata, editors, *Proceedings of the 1st International Conference on Evolvable Systems: From Biology to Hardware*, volume 1259 of *Lecture Notes in Computer Science*, pages 327–343, Heidelberg, 1997. Springer-Verlag.

[7] T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Albuquergue, NM, 1995.

[8] I. Kajitani, T. Hushino, D. Nishikawa, H. Yokoi, S. Nakaya, T. Yamauchi, T. Inuo, N. Kajihara, M. Iwata, D. Keymeulen, and T. Higuchi. A gate-level ehw chip: Implementing ga operations and reconfigurable hardware on a single lsi. In M. Sipper, D. Mange, and A. Pérez-Uribe, editors, *Proceedings of the 2nd International Conference on Evolvable Systems: From Biology to Hardware*, volume 1478 of *Lecture Notes in Computer Science*, pages 1–12, Heidelberg, 1998. Springer-Verlag.

[9] S. A. Kauffman. Adaptation on rugged fitness landscapes. In D. Stein, editor, *Lectures in the Sciences of Complexity*, SFI Studies in the Sciences of Complexity, pages 527–618. Addison-Wesley, Reading, MA, 1989.

[10] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane. Automated design of both the topology and sizing of analog electrical circuits using genetic programming. In J. S. Gero and F. Sudweeks, editors, *Artificial Intelligence in Design*, pages 151–170. Kluwer Academic Publishers, MA, 1996.

[11] B. Manderick, M. de Weger, and P. Spiessens. The genetic algorithm and the structure of the fitness landscape. In R. K. Belew and L. B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 143–150, San Mateo, CA, 1991. Morgan Kaufmann.

[12] J. F. Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the 1st Genetic and Evolutionary Computation Conference*, volume 2, pages 1135–1142, San Francisco, CA, 1999. Morgan Kaufmann.

[13] J. F. Miller, D. Job, and V. K. Vassilev. Principles in the evolutionary design of digital circuits. *Journal of Genetic*

Programming and Evolvable Machines, 1(1/2,3), 2000. In press.

[14] J. F. Miller and P. Thomson. Cartesian genetic programming. In *Proceedings of the 3rd European Conference on Genetic Programming*, Lecture Notes in Computer Science, Berlin, 2000. Springer-Verlag. In press (available via http://www.cs.bham.ac.uk/∼jfm/).

[15] J. F. Miller, P. Thomson, and T. Fogarty. Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study. In D. Quagliarella, J. Periaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, pages 105–131. Wiley, Chechester, UK, 1997.

[16] M. Mitchell, S. Forrest, and J. Holland. The royal road for genetic algorithms: Fitness landscapes and ga performance. In J. Varela and P. Bourgine, editors, *Proceedings of the 1st European Conference on Artificial Life*, pages 245–254, Cambridge, MA, 1991. MIT Press.

[17] H. Mühlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm (bga). *Evolutionary Computation*, 1(4):335–360, 1993.

[18] M. Murakawa, S. Yoshizawa, T. Adachi, S. Suzuki, K. Takasuka, M. Iwata, and T. Higuchi. Analogue ehw chip for intermediate frequency filters. In M. Sipper, D. Mange, and A. Pérez-Uribe, editors, *Proceedings of the 2nd International Conference on Evolvable Systems: From Biology to Hardware*, volume 1478 of *Lecture Notes in Computer Science*, pages 134–143, Heidelberg, 1998. Springer-Verlag.

[19] C. M. Reidys and P. F. Stadler. Neutrality in fitness landscapes. Technical Report 98-10-089, Santa Fe Institute, 1998. Submitted to *Appl. Math. & Comput.*

[20] H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Chichester, UK, 1981.

[21] P. F. Stadler. Spectral landscape theory. In J. P. Crutchfield and P. Schuster, editors, *Evolutionary Dynamics — Exploring the Interplay of Selection, Neutrality, Accident and Function*. Oxford University Press, New York, 1999. To appear.

[22] A. Stoica, D. Keymeulen, R. Tawel, C. Salazar-Lazaro, and W. Li. Evolutionary experiments with a fine-grained reconfigurable architecture for analog and digital cmos circuits. In A. Stoica, D. Keymeulen, and J. Lohn, editors, *Proceedings of the 1st Nasa/DoD Workshop on Evolvable Hardware*, pages 76–84, Los Alamitos, CA, 1999. IEEE Computer Society.

[23] A. Thompson. *Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution*. Springer-Verlag, London, 1998.

[24] A. Thompson, P. Layzell, and R. S. Zebulum. Explorations in design space: Unconventional electronics design through artificial evolution. *IEEE Transactions on Evolutionary Computation*, 3(3):167–196, 1999.

[25] V. K. Vassilev and J. F. Miller. The advantages of landscape neutrality in digital circuit evolution. In *Proceedings of the 3rd International Conference on Evolvable Systems: From Biology to Hardware*, Lecture Notes in Computer Science, Berlin, 2000. Springer-Verlag. In press (available via http://www.dcs.napier.ac.uk/∼vesselin/).

[26] V. K. Vassilev and J. F. Miller. Embedding landscape neutrality to build a *bridge* from the conventional to a more efficient three-bit multiplier circuit. In *Proceedings of the 2nd Genetic and Evolutionary Computation Conference*, San Francisco, CA, 2000. Morgan Kaufmann. Submitted (available via http://www.dcs.napier.ac.uk/∼vesselin/).

[27] V. K. Vassilev, J. F. Miller, and T. C. Fogarty. Digital circuit evolution and fitness landscapes. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1299–1306, Piscataway, NJ, 1999. IEEE Press.

[28] V. K. Vassilev, J. F. Miller, and T. C. Fogarty. Digital circuit evolution: The ruggedness and neutrality of two-bit multiplier landscapes. In D. M. Harvey, editor, *Evolutionary Hardware Systems*, pages 6/1–6/4, London, 1999. IEE Press.

[29] V. K. Vassilev, J. F. Miller, and T. C. Fogarty. On the nature of two-bit multiplier landscapes. In A. Stoica, D. Keymeulen, and J. Lohn, editors, *Proceedings of the 1st NASA/DoD Workshop on Evolvable Hardware*, pages 36–45, Los Alamitos, CA, 1999. IEEE Computer Society.

[30] R. S. Zebulum, M. A. Pacheco, and M. Vellasco. Analog circuits evolution in extrinsic and intrinsic modes. In M. Sipper, D. Mange, and A. Pérez-Uribe, editors, *Proceedings of the 2nd International Conference on Evolvable Systems: From Biology to Hardware*, volume 1478 of *Lecture Notes in Computer Science*, pages 154–165, Heidelberg, 1998. Springer-Verlag.