Complexity of a Phenotype with Greater Plasticity for Digital Evolvable Hardware

PABLO A. SALVADEO[1], ANGEL VECA[2], and RAFAEL CASTRO-LÓPEZ[3]
[1]Lab. de Computación Reconfigurable
Universidad Tecnológica Nacional Regional Mendoza
Rodriguez 273, Ciudad de Mendoza
ARGENTINA
[2]Instituto de Automática
Universidad Nacional de San Juan
Av. San Martín (Oeste) 1109, San Juan
ARGENTINA
[3]Instituto de Microelectrónica de Sevilla
Universidad de Sevilla
Américo Vespucio s/n, Sevilla
ESPAÑA
pablo.salvadeo@frm.utn.edu.ar, aveca@inaut.unsj.edu.ar, and castro@imse-cnm.csic.es

Abstract: – This paper analyzes the complexity of a new phenotype for evolvable hardware with more plasticity than the traditional phenotype. The plasticity is due to that fact that this phenotype is unstructured and that property allows it to perform combinational and sequential tasks. The complexity is evaluated in relation to Shannon's limit (lower) and Lupanov's limit (upper). These limits define a bounded and sufficient search space for the combinational circuits. However, in the studied case (multipliers with 2-bits word width operands) the lowest number of iterations is achieved when the number of cells available is around four-times the Shannon's limit. Moreover, at calculate the mutation rate of digital organisms using the Lynch's equation, which relates the mutation rate to the genome length of the species, the circuit size stays around Lupanov's limit. Finally, it important to note that compact circuits are obtained directly, with an evolutionary algorithm that only follows a wished-for functionality, where the economy of the phenotypes is an emergent property of the process.

Key-Words: – Circuit Size, Complexity Limits, Evolvable Hardware, Iterations, Lupanov's limit, Phenotype, Shannon's limit.

1 Introduction

Evolvable hardware is born at the intersection of three disciplines: electronics, computation, and biology [1]. That intersection is defined as the area where evolutionary algorithms are utilized to design electronic systems. Thus, the hardware produced using this technique is bio-inspired [2]. Evolvable hardware is also divided into three classes: extrinsic, intrinsic, and complete [3]. The difference between these classes is based on the software and hardware used in the evolutionary process. In the extrinsic class, the entire process is run on software and the candidate evaluations are realized via simulations. The intrinsic class is differentiated from the extrinsic class in that the evaluations are realized directly in the hardware. For this purpose, the Field-Programmable Gate Array (FPGA) is the most habitually used digital reconfigurable hardware. Finally, in the complete class, the entire process happens on the FPGA.

On the other hand, evolvable hardware deserves a conceptual redefining. It is just not right to treat this technique as a simple application of the evolutionary algorithms that are used to design electronic systems. It is much better to consider the task that is to be performed (to be automated) as being part of the digital organism's behavior, where that behavior is more complex and encompasses the evolution. In turn, the digital organism can be analyzed from three points of view or dimensions: behavioral, phenotypic, and genotypic. This definition of evolvable hardware is more global and compact. Thus, the technique consists of defining these three dimensions in terms of the function of the task that the system must perform. Certainly, the term, “digital organism”, has been used previously, but in a more specific form that was associated with replicants that compete by resources [4], long before the term, “artificial creature”, was used in the field of artificial neural networks [5].

This paper analyzes the phenotypic complexity of a new digital organism species. This complexity is evaluated by assessing the results of an extrinsic class of evolvable hardware when multipliers with 2-bits word width operands are sought.

The rest of paper is organized in the following manner: Section 2 examines the classic phenotype and its genotype; Section 3 presents the phenotypic complexity concept and the definitions of base and size; Section 4 analyzes the theoretical limits of the complexity; Section 5 presents the new phenotype; Section 6 discusses the results of the experiment; and Section 7 presents the conclusions.

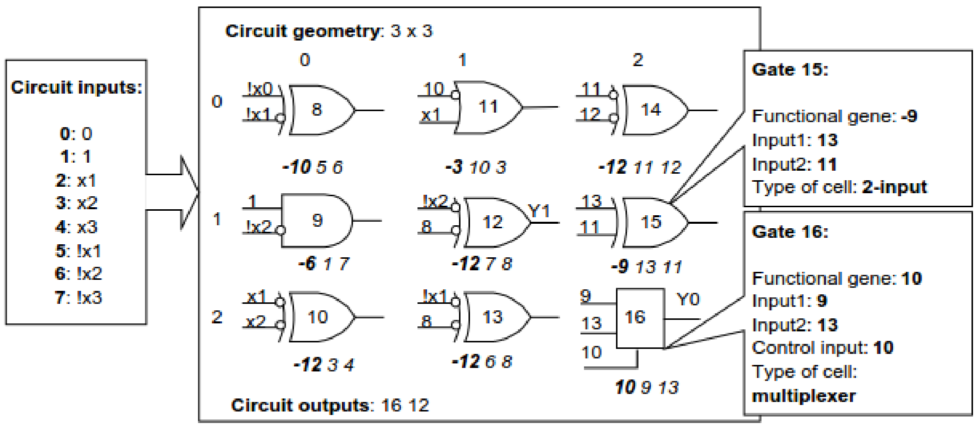
2 Classic Phenotype

The traditional or classic phenotype consists of a rectangular array where each cell is occupied by a logic gate of two inputs; the system inputs are on the left and the system outputs are on the right. This phenotype is known as gate-level Cartesian Genetic Programming (CGP) [6]. The types of available logic gates are restricted to a small group of basic logical functions. The interconnections are constrained from left to right (forward), and it is typical that the inputs of a column can only connect to the outputs of the previous column. However, there are cases with more restrictions, where the rows are also limited, as well as other cases that are more lax, where the cells can connect to the right-most columns with a depth greater than one. These constraints are due to the fact that the task selected to test the technique in the current state-of-the-art is the combinational circuit design, and as is known, the combinational circuit definition implies an acyclic graph since the outputs only depend on the inputs. Fig. 1 shows a diagram of a typical phenotype with its genotype extracted from [7].

The inputs of the function to implement, as well as its inverses and the logic constants 0 and 1, are usually found between the inputs of the phenotype. The inclusion of constants allows a logic gate of two inputs to be simplified as a NOT (inverter) or a WIRE (no-inverter). Moreover, in general, this inclusion implies a decrease in the total number of logic gates and gives greater homogeneity to the structure, since one input gate is not necessary. For example, in [8] the number of WIREs is maximized to obtain a more compact circuit size.

The genotype is a chain formed by groups of three values. The first value represents the logic gate function. The second and third values identify the position of two other cells associated with each gate input, thus {function, cell#, cell#}. The number of groups coincides with the available nodes of the array. The identifiers of the cells from which the system outputs are taken are found at the genotype end; thus {cell#}. Obviously, the number of genes matches the number of system outputs. As detailed in Fig. 1, when the function value is positive, the cell controls the selection signal of MUX. When the function value is negative, this represents the logic function of the gate. Both binary and decimal codification is used equally in state-of-the-art

(a)



(b)

-10 5 6	-6 1 7	-12 3 4	-3 10 3	-12 7 8	-12 6 8	-12 11 12	-9 13 11	10 9 13	16	12

Fig. 1a. The classic phenotype of evolvable hardware; **b.** its genotype [*7*].

systems. This present work uses binary codification because it simplifies the mutation; due to the binary alphabet, this genetic operation is simply a bit-flip, a NOT. It is important to consider the order in which the cells are numbered. If the cells are numbered by rows instead of by columns, the problem becomes disruptive. If this happens, two nearby circuits in the phenotypic plane can be far in the genotypic plane, making the problem more difficult for the algorithm.

Ultimately, to complete the image of digital organisms found in current state-of-the-art, it is important to define the behavior, which is already known partially because the tasks to be performed are combinational, typically multipliers. Then, to complete the behavior it is necessary to specify the algorithm associated with the evolutionary process. Genetic algorithms [9] and evolution strategies [10] are used indistinctly. The difference between them is that in genetic algorithms the population is larger, while in evolution strategies the iteration number is larger; thus, the evaluation number (the iteration-population product) is equal in both. Moreover, the obtained results are similar for both algorithms, and one is not more advantageous than the other. The fitness is equal to the number of coincidences between the outputs of the truth table of the wished-for Boolean function and the circuit output values, in which the maximum fitness is equal to M 2^N, where N is the number of inputs and M is the number of outputs.

This function is used exclusivity in all studies to calculate the fitness, except in [11] where the 1s are heavier than the 0s but without benefits. One approach that is used habitually is to divide the search process into two parts. First, the fitness is associated with the task, and when a 100% functional phenotype is achieved, the small circuits (those that use fewer cells) are weighted [7]. Then, it can be said that the first stage is the design and the second stage is the optimization (minimization). Finally, it is important to note that this digital organism species of evolvable hardware with a direct evolution search suffers from a lack of scalability, with a limit well defined as a 4-bits multiplier (eight inputs and outputs). In [12], the current state-of-the-art is reviewed briefly, and it is concluded that improvements are needed in this area.

3 Phenotypic Complexity

The central theme of the theory of computing is to identify a task's complexity in terms of the resources that are consumed. Then, if the complexity of a Boolean function needs to be calculated, it can be measured as the minimum number of operations required to make such a function. Of course, this measure depends on the available types of cells. If a combinational circuit is used for the implementation, the complexity is the minimum number of gates based on the types of gates that are available. This available operations set defines the base. Thus, the complexity of a function with respect to a specific base is the minimum number of operations required to conduct the computing.

A base is complete when it allows for implementing any Boolean function. This is very important; a complete base is necessary if an adaptive system is required. Obviously, when the base is changed, the circuit properties, the size, and the depth (or level) are modified. The first property is the number of gates and the second property is the direct longer path between the input and the output. When using evolutionary algorithms in the synthesis, the depth is not limited; however, when employing classic methods, such as Karnaugh's maps [13] or its computational equivalent Quine–McCluskey [14,15], the depth is constrained to two levels. In addition, if the base is complete, e.g., {AND, OR, NOT}, the search space is bounded accelerating the adaptation. Table 1 shows that the bases used in the current state-of-the-art are mainly super-complete. This property is due to the introduction of elements that belong to the problem that is being addressed; e.g., the use of XOR in arithmetic circuits (in this work we include this type of cell in the base because the task is to multiply). The WIRE (that copies the input value into the output) allows obtaining smaller circuits if the number of these gates (WIREs) is maximized during the evolution. It is also normal find the base {AND, OR, NOT} together with its complement (the same base but denied) {NAND, NOR, WIRE}. In addition, the logic constants, 0 and 1, are usual, and the gates with one or both inputs denied. This fact allows it generate equivalent functions, which is evidenced if the laws of Boole's algebra, e.g., De Morgan's laws, are applied. The equivalent functions expand the polymorphism (morphologic diversity) of the phenotypes, and include MUX, which produces the same effect. Note that an MUX allows the system to implement the logic functions, but it can also be considered to be a flow control expression (IF select THEN z ELSE y).

Table 1 The digital evolvable hardware bases.

Function	Base	Ref.
multiply	AND, XOR, NOT, NAND, NOT-XOR, WIRE, MUX	[15]
multiply and add	AND, OR, NOT, XOR, MUX	[16]
multiply	AND, OR, NOT, XOR, NAND, NOR, WIRE	[17]
multiply and add	AND, OR, NOT, XOR	[18]
multiply and others	AND, OR, NOT, XOR, WIRE	[7]

4 Theoretical Limits

For the complete base {AND, OR, NOT}, the limits for the circuit size have been found. Shannon shows, for most of the circuits with N inputs, that compute a Boolean function, that the circuit size should be greater than $2^N/N$ (asymptotically) [20]. This fact allows one to fix a lower limit for the genotype length, confining the search space. Meanwhile, Lupanov proves that if the circuit size is greater that $2^N/N$, all functions of the N inputs can be achieved by a fraction of this value [21]. This second fact allows one to fix an upper potential limit that is twice Shannon's limit. Then, these two limits are very important for the size of the phenotypes and their genotypes, obviously only for the same base. Then, the task of implementing a Boolean function through a combinational circuit has a minimum complexity defined by the Shannon's limit and a maximum complexity equal to twice that limit.

Table 2 shows the lower limit, the number of used gates, and the amount of cells available for the multipliers and adders with an increasing number of inputs. For the multipliers, in order to increase the inputs, the phenotype must deviate more and more from the lower limits (the distances are: 3 gates, 15 gates, and 28 gates); however, these distances are greater than the fraction of the lower limit. Therefore, it can be said that, according to Lupanov, the circuits could be smaller. It can also be said that the technique seems to lose its efficiency to increase the number of inputs. However, the number of available cells, and therefore the search space, is too great, i.e., it is oversized. For example, for a multiplier of 4-bits the lower limit is 32 and the amount of available cells is 269, but this can be fixed on 64 (twice the Shannon's limit). Thus, the search space can be reduced while success can be ensured. On the other hand, the size of the phenotype is always approximately equal to the lower limit. That demonstrates a measure of the efficiency of the evolvable hardware as a technique of synthesis.

Table 2 The sizes and limits of the multipliers and adders.

Function	Inputs	Shannon's limit	Available cells	Used gates	Ref.
multiply 2-bits	4	4	12	7	[16]
			48	-	[15]
			10	7	[18]
			25	7	[7]
			7	7	[17]
add 2-bits	5	7	9	6	[16]
			15	10	[18]
multiply 3-bits	6	11	35	28	[16]
			30	26	[18]
			57	23	[17]
add 3-bits	7	19	20	12	[16]
multiply 4-bits	8	32	269	60	[17]

As discussed in the previous two paragraphs, to ensure that the evolutionary process can build a 100% functional system, it should work with a complete base and a size equal to or greater than the minimum complexity. In the current state-of-the-art these topics are never addressed, although as previously mentioned, the used bases are super-complete and of a specific kind {AND, OR, NOT}.

In the adders (see Table 2), the used number of gates is below the Shannon's limit, i.e., the addition seems to be less complex than the multiplication. Indeed, this fact is clear if the multiplication is thought of as a progressive addition or if one compares the fitness of both the adders and the multipliers for the same number of inputs. Table 3 shows a comparison of the fitness between the adders (without carry) and the multipliers. From a functional viewpoint, it is evident that the multipliers are more complex than the adders. For the same width of operand, the wished-for fitness is doubled in the multipliers with respect to the adders.

Table 3 Fitness of the multipliers and the adders.

Function	Bits by operand	Inputs	Outputs	Wished-for fitness
Adder	2	4	2	32
	3	6	3	192
	4	8	4	1024
Multiplier	2	4	4	64
	3	6	6	384
	4	8	8	2048

5 New Phenotype

The new phenotype has two fundamental properties that differentiate it from the traditional phenotype. First, it is unstructured, i.e., there is no a restriction on the interconnection mode of the nodes. At first glance, this plasticity seems to be a disadvantage, but it is not, even though the search space increases. This flexibility allows the new phenotype to perform both combinational and sequential tasks, covering the two digital system behaviors. Second, the phenotype is a hardware description in Very-high-speed-integrated-circuit Hardware Description Language (VHDL) [22] that can be synthesized in FPGA or in an Application-Specific Integrated Circuit (ASIC). Obviously, it can also be seen as a graphic. Furthermore, a hardware description in a standard language facilitates the modification and evaluation (simulation), as well as the union, between the design tools.

An unstructured phenotype allows loops, giving rise to cyclic graphs. However, these structures can also achieve a behavior that is purely combinational. An analysis method to investigate whether or not a cyclic graph is combinational is presented in [23]. Loops that allow obtaining compact combinational circuits are reported in [24]. A method to synthesize combinational circuits with loops that obtain small circuits is reported in [25]. Fig. 2 shows a trivial circuit of [26] to explain combinational circuits with loops. This trivial circuit is combinational, although, at first glance, it does not seem to be. However, the output value is always 0 when the input is 0, and the same thing happens with 1. Thus, this trivial circuit respects the behavioral definition of a combinational circuit: the output values only depend on the input values. However, this trivial circuit does not meet the classic structural definition, because it is a cyclic graph (it has a loop) instead of an acyclic graph. Evidently, the restriction over the phenotype found in the current state-of-the-art originates from the classic structural definition of a combinational circuit. However, a broader, new definition allows one to create a more flexible phenotype. The availability of the logic constants 1 and 0 as system inputs permits the construction of NOT and WIRE from two inputs gates, and it can decrease the total gate number, e.g., an AND with an input in 0 is a constant. Moreover, a transformation in WIRE implies a reduction in the effective number of cells. A typical configuration in which a WIRE appears is a cell in which the inputs are connected to the same signal. In turn, with the new phenotype, a set of new possible simplifications, which can reduce the circuit size, appears due to the presence of loops, e.g., the trivial circuit shown in Fig. 2 is a WIRE that removes two gates. These facts, and the foundation presented in the previous paragraph, indicate that the new phenotype can lead to a compact circuit.

The new genotype has a minor modification in the structure of the groups of three values. In this case, the position of two cells associated to each gate input first appears, and then the logic gate function appears; thus {cell#, cell#, function}. This order facilitates the process of constructing the phenotype from the genotype, with the phenotype being a description in VHDL that follows a style, named dataflow, where all the sentences are concurrent and simple [27]. Therefore, each set of three values of the genotype is translated into a concurrent sentence of the phenotype (description). At the same time, VHDL demands that a signal have only one driver, i.e., multiple drivers for the same signal are prohibited. In order to save that restriction, each sentence is applied to its own signal, whose number coincides with the cell position in the array. Taking Fig. 1 as an example, the translation of node 15 would be: `s15 <= s13 XOR s11`, where the letter "s", followed by a number, identifies each signal. Fig. 3 shows the description of the new phenotype in VHDL; it includes a multiplier of 2-bits (four inputs and outputs) with 20nodes. Fig. 3 also shows its pseudo-genotype (remember that the used codification is binary). It is important to note that the signals 0 and 1 are associated, respectively, with the logic constants 0 and 1. The inputs start from signal number two. In the example shown in Fig. 3, the inputs go from `s2` to `s5` because there are four inputs, and the nodes go from `s6` to `s25` because there are 20nodes. Meanwhile, the kind of data used, `std_ulogic`, allows one to decrease the simulation time of the candidates' evaluation. This type is "unresolved", i.e., it does not have a resolution function, and neither is it a subtype. Thus,

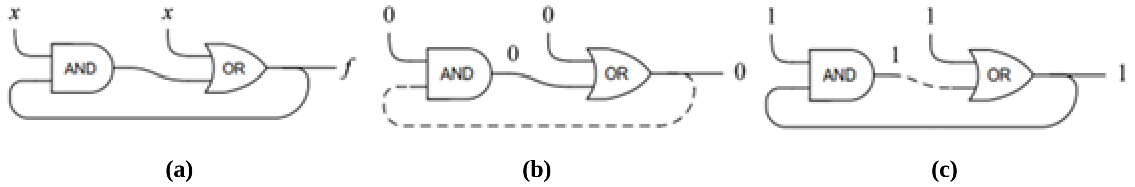


Fig. 2a. Trivial combinational circuit with a loop; equivalent circuit with **b.** x=0 and **c.** x=1.

(a)

```
library ieee;
use ieee.std_logic_1164.all;
entity phenotype is port (
i0, i1, i2, i3 : in std_ulogic;
o0, o1, o2, o3 : out std_ulogic);
end entity;
architecture dataflow of phenotype is
    signal s0,s1,s2,s3,…,s25 : std_ulogic;
begin
    s0 <= '0';
    s1 <= '1';
    s2 <= i0;
    s3 <= i1;
    s4 <= i2;
    s5 <= i3;
    s6 <= s20 AND s14;
    s7 <= s14 XOR s13;
    s8 <= s20 XOR s22;
    s9 <= s29 NAND s4;
    …
    s23 <= s5 NAND s2;
    s24 <= s12 XOR s14;
    s25 <= s12 XOR s8;
    o0 <= s17;
    o1 <= s8;
    o2 <= s7;
    o3 <= s20;
end dataflow;
```

(b)

20 14 AND	14 13 XOR	…	5 2 NAND	12 14 XOR	12 8 XOR	17	8	7	20

Fig. 3a. New phenotype; **b.** pseudo-genotype.

to evaluate each expression it is not necessary use the resolution function to make a conversion, as would happen with `std_logic` since this is a "resolved" subtype of `std_ulogic` [28]. Finally, it should be mentioned that the used base is {AND, NAND, XOR}, which is small and homogeneous, and it gives good results.

6 Results

The complexity of the new phenotype is evaluated by building the multipliers with 2-bits word width operands. To finish completing the behavior of the digital organism, an algorithm that is similar to an evolution strategy (1+λ) is used, where λ is equal to 4. The difference is that if the mutations do not produce improvement, a crossover of one point is used, as in a genetic algorithm. At the same time, the search start point is a null genome, with all its bits in zero. Then, at the beginning, the mutations to 1 are more frequent; thus, the phenotype size increases.

Fig. 4 shows the iterations (generations) and fitness while the number of available nodes in the phenotype increases. In this experiment, the maximum iteration number is 320,000, the maximum fitness is 64 and, the number of genes mutant is 10 regardless of the genotype length. Shannon's limit for the wished-for combinational circuit is four. For this quantity of available nodes, the fitness value is 59 (92% of the maximum fitness), i.e., the phenotypic functionality is 92%. Later, when the goal of six available cells is achieved 98% functionality is reached, and when twice the lower limit (eight) is reached the functionality is 100%. With respect to the iterations, these decrease until the 17 available cells, and then the iterations begin to increase. This fact shows that it is possible to increase the quantity of cells beyond the upper limit and increase the speed with which the results are obtained even though the search space increases. In this case, the number of available cells that minimizes the iterations is approximately four-times greater than the Shannon's limit.

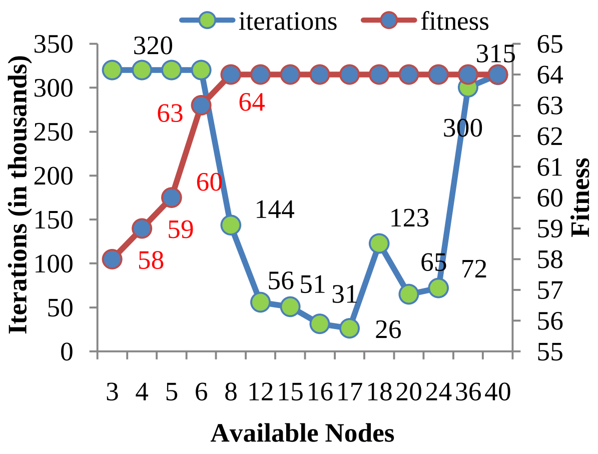


Fig. 4 Iterations and fitness vs. available nodes for multipliers with 2-bits word width operands where the number of genes mutant is fixed at 10.

Fig. 5 presents the iterations and size versus the available nodes for multipliers with 2-bits word width operands. The first thing that can be observed is that although in this case does not used a two-step strategy to obtain the compact circuits (design follow by optimization), the minimum value of the current state-of-the-art (seven gates) is achieved many times. It is also noted that this happens after the upper limit is reached. It is thought that this fact is related to the greater flexibility of the new phenotype. The phenotype loops allow for more compact combinational circuits; they also increase the quantity of the cases where it is possible to simplify the logic gates (to appear more WIREs). Note that even when there are 40cells (10-times the lower limit), the size does not increase too much (it is eight). Therefore, it is evident the evolutionary process maintains the phenotype economy.

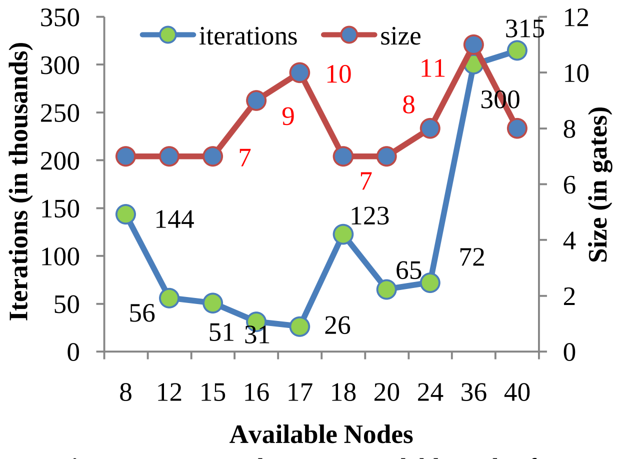


Fig. 5 Iterations and size vs. available nodes for multipliers with 2-bits word width operands where the number of genes mutant is fixed at 10.

The question then arises: what initial mutation rate (the relation between the number of mutant genes and the genotype length) will improve the economy of phenotypes? For example, Fig. 6 exhibits the variations needed to obtain an initial mutation rate of 4.46%. While it is possible to keep trying with many rates, it is already evident that one rate is good for a specific number of available cells while for others it produces large phenotypes. Therefore, it may be adequate to take a look at the bio-inspiration.

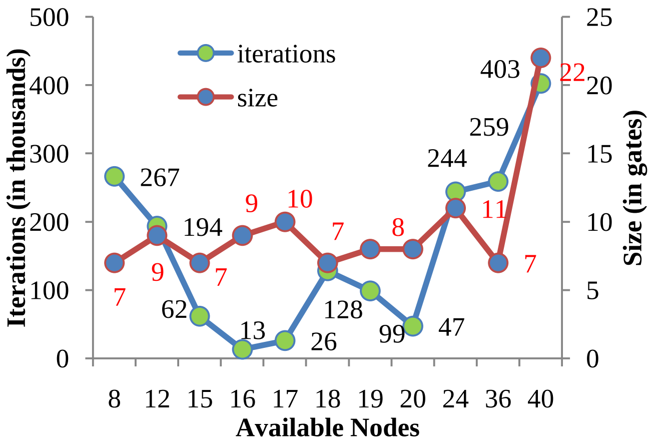


Fig. 6 Iterations and size vs. available nodes for multipliers with 2-bits word width operands with an initial mutation rate of 4.46%.

In [29], Lynch reports the relationship between the genotype length and the mutation rate of the species. This expression has a fitting of 80%. This is shown in equation (1), where: T is the mutation rate and L is the genotype length. However, this expression is not useful because the genotypes of the digital organisms are much shorter than the natural genotypes. The genotypes of natural organisms consist of 10^6 base pairs while the genotypes of digital organisms are in the hundreds of bits. Accordingly, equation (1) is multiplied by a scale constant equal to one-tenth to achieve the appropriate rates for the length of the digital genomes.

$$T = -0.81 + 0.68 \log_{10}(L) \qquad (1)$$

Fig. 7 shows the iterations and phenotype sizes in the function of available nodes when Lynch's equation (multiplied by one-tenth) is used to calculate the initial mutation rate. Observe that the size dispersion is very small, with the phenotypes ranging between seven and eight gates, except for 33 available nodes where the value is sixteen. However, as can be seen in Fig. 5 and Fig. 6, a peak always appears. Therefore, it can be said that Lynch's equation maintains the economy of the phenotypes even though the number of available nodes increases.

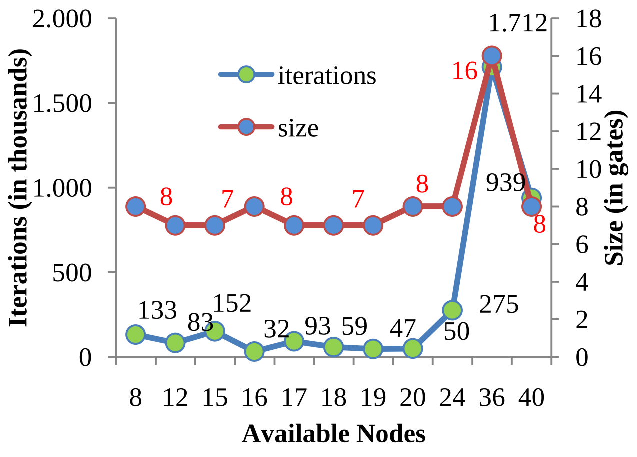


Fig. 7 Iterations and size vs. available nodes for multipliers with 2-bits word width operands with an initial mutation rate that follows Lynch's equation.

7 Conclusions

The new phenotype of the evolvable hardware, which is more flexible than the classic phenotype, allows for performing both combinational and sequential tasks because it is unstructured. At the same time, the presence of loops makes it possible to find combinational circuits that are as small as those in the current state-of-the-art, but without following a two-step evolution process that includes design and optimization (minimization).

Regarding the phenotype complexity, Shannon's limit (lower) and Lupanov's limit (upper) define a bounded and sufficient search space for the combinational circuits. However, in the studied case (multipliers with 2-bits word width operands) the lowest number of iterations is achieved when the number of cells available is around four-times the Shannon's limit. In addition, when calculating the mutation rate of digital organisms using Lynch's equation, which relates the mutation rate to the genome length of the species, the circuit size stays around Lupanov's limit.

References:

[1] T. G. W. Gordon and P. J. Bentley, "On Evolvable Hardware," in in Soft Computing in Industrial Electronics, S. Ovaska and L. Sztandera, 2002, pp. 279-323.

[2] Y. Bar-Cohen, "Nature as a Model for Mimicking and Inspiration of New Technologies," Int'l J. of Aeronautical & Space Sci., vol. 13, no. 1, pp. 1-13, 2012.

[3] A. Upegui and E. Sanchez, "Evolvable FPGAs," in Reconfigurable Computing: The Theory and Practice of FPGA-based Computation, S. Hauck and A. DeHon, Eds.: Morgan Kaufmann: Elsevier, 2008, ch. 33, pp. 725-752.

[4] C. O. Wilke and C. Adami, "The biology of digital organisms," Trends in Ecology & Evolution, vol. 17, pp. 528-532, 2002.

[5] H. de Garis, "Circuits of Production Rule GenNets," in Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Innsbruck, Austria, 1993, R. F. Albrecht, C. R. Reeves, and N. C. Steele, Eds. Vienna: Springer Vienna, 1993, pp. 699-705. [Online]. http://dx.doi.org/10.1007/978-3-7091-7533-0_101

[6] J.F. Miller, "Cartesian Genetic Programming," in Natural Computing Series, J.F. Miller, Ed.: Springer-Verlag Berlin Heidelberg, 2011, ch. 2.

[7] T. Kalganova and J. Miller, "Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness," in Evolvable Hardware, 1999. Proceedings of the First NASA/DoD Workshop on, 1999, pp. 54-63.

[8] C. A. Coello-Coello, A. D. Christiansen, and A. Hernández, "Use of Evolutionary Techniques to Automate the Design of Combinational Circuits," International Journal of Smart Engineering System Design, vol. 2, no. 4, pp. 299-314, 2000.

[9] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[10] N. Hansen, D. V. Arnold, and A. Auger, "Evolution Strategies," in Handbook of Computational Intelligence, J. Kacprzyk and W. Pedrycz, Eds.: Springer, 2015, ch. 44, pp. 871-898.

[11] J. Torresen, "Evolving Multiplier Circuits by

Training Set and Training Vector Partitioning," in Evolvable Systems: From Biology to Hardware, A. M. Tyrrell, P. C. Haddow, and J. Torresen, Eds.: Springer Berlin Heidelberg, 2003, vol. 2606, pp. 228-237. [Online]. http://dx.doi.org/10.1007/3-540-36553-2_21

[12] F. Cancare, S. Bhandari, D. B. Bartolini, M. Carminati, and M. D. Santambrogio, "A bird's eye view of FPGA-based Evolvable Hardware," in Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on, June 2011, pp. 169-175.

[13] M. Karnaugh, "A map method for synthesis of combinational logic circuits," Transactions of the AIEE, Communications and Electronics, vol. 72, no. 1, pp. 593-599, 1953.

[14] E. J. McCluskey, "Minimization of boolean functions," Bell Systems Technical Journal, vol. 35, no. 5, pp. 1417-1444, 1956.

[15] W. V. Quine, "A way to simplify truth functions," American Mathematical Monthly, vol. 62, no. 9, pp. 627-631, 1955.

[16] K. Glette and J. Torresen, "A Flexible On-Chip Evolution System Implemented on a Xilinx Virtex-II Pro Device," in Evolvable Systems: From Biology to Hardware, J. M. Moreno, J. Madrenas, and J. Cosp, Eds.: Springer Berlin Heidelberg, 2005, vol. 3637, pp. 66-75. [Online]. http://dx.doi.org/10.1007/11549703_7

[17] J. F. Miller, P. Thomson, and T. Fogarty, "Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study," in Genetic Algorithms and Evolution Strategy in Engineering and Computer Science, D. Quagliarella et al., Eds. Chichester, England: Morgan Kaufmann, 1998, pp. 105-131.

[18] Z. Gajda and L. Sekanina, "An Efficient Selection Strategy for Digital Circuit Evolution," in Evolvable Systems: From Biology to Hardware, G. Tempesti, A. M. Tyrrell, and J. F. Miller, Eds.: Springer Berlin Heidelberg, 2010, vol. 6274, pp. 13-24. [Online]. http://dx.doi.org/10.1007/978-3-642-15323-5_2

[19] T. Kalganova, "An Extrinsic Function-Level Evolvable Hardware Approach," in 3° European Conference on Genetic Programming, EuroGP2000, Edinburgh, UK, 2000.

[20] C. E. Shannon, "The synthesis of two-terminal switching circuits," Bell System Technical Journal, vol. 28, pp. 59-98, 1949.

[21] O. B. Lupanov, "On a method of circuit synthesis," Izvestia VUZ (Radiofizika), vol. 1, pp. 120-140, 1958.

[22] IEEE Standard Association, 1076-2008 VHDL Language Reference Manual, 2008.

[23] S. Malik, "Analysis of cyclic combinational circuits," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 13, no. 7, pp. 950-956, Jul 1994.

[24] R. L. Rivest, "The Necessity of Feedback in Minimal Monotone Combinational Circuits," IEEE Transactions on Computers, vol. C-26, no. 6, pp. 606-607, June 1977.

[25] M. D. Riedel and J. Bruck, "The synthesis of cyclic combinational circuits," in Design Automation Conference, 2003. Proceedings, June 2003, pp. 163-168.

[26] M. D. Riedel, "Combinational Circuits with Feedback," Caltech, Ph.D. dissertation 2004.

[27] S. Brown and Z. Vranesic, Fundamentals of digital logic with VHDL design, 3rd ed.: McGraw-Hill, 2008.

[28] J. Morris, Reconfigurable Computing, Resolution Functions, 2009, University of Auckland.

[29] M. Lynch, "Evolution of the mutation rate," Trends in Genetics, vol. 26, no. 8, pp. 345-352, 2010. [Online]. http://dx.doi.org/10.1016/j.tig.2010.05.003