Automated Design of Combinatorial Logic Circuits

Iva Brajer and Domagoj Jakobović
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, Zagreb, Croatia
Email: iva.brajer@gmail.com, domagoj.jakobovic@fer.hr

***Abstract*—This paper deals with automated design of combinatorial circuits with the use of *Cartesian Genetic Programming* (CGP). The synthesis is based on user specifications of network functionality, while the network structure may be predefined. The results show that CGP approach is able to match the desired functionality while preserving other performance criteria, such as latency and number of gates. Additionally, the evolution process may use Verilog network descriptions as input files, which facilitates the design for larger number of inputs and test patterns.**

I. Introduction

The design of combinatorial electronic circuits is a complex task requiring a significant effort and time cost of the involved experts. This process may include the identification of appropriate target gate type and technology, minimization and optimization subject to various domain specific constraints (such as timing, number of gates, etc.) and mapping the design onto the target device [1]. Lately, this problem has been addressed with a black-box approach where evolutionary algorithms, such as genetic algorithms and genetic programming, have been used to evolve the desired functionality and other aspects of the target circuitry. This approach has come to be known as *Evolvable Hardware*, which is a generic term used to denote various methods of designing electronic circuits with evolutionary metaheuristics [2][3].

In this paper we present an application of *Cartesian Genetic Programming* (CGP) to design combinatorial circuits and evaluate its efficiency. The procedure is fully automated, without the need for human expert intervention in the process. The target circuitry is evolved based on the desired logical functionality. The definition of the circuit functionality is based on two components: first a behavioral description of the network is defined in Verilog. The system then internally builds the executable simulation module and the internal truth table which is used in the evolution process. The results show that the presented approach may produce solutions with full compliance with desired functionality.

The rest of this paper is organized as follows: Section II briefly revises this area, while in Section III we describe the CGP and the structure of our evolutionary system. Section IV describes the use of Verilog for the specification of circuit functionality, and Section V gives examples of the application and the obtained results.

II. Automatic Synthesis of Combinatorial Circuits

As the production of electronic circuits increases, the demand in both financial and time resources calls for different design methods. The traditional approach includes assigning human experts for the development of functional components, but this may quickly become inefficient for a large number of networks with even slightly different requirements [2]. Partial or complete automation of this task has become an active area of research which includes various algorithms and technologies.

One possible approach to this problem is the use of evolutionary algorithms (EA), in which the desired functionality is automatically constructed using the principles of evolution. In an EA, a set (population) of possible solutions (individuals) are created and modified in a sequence of iterations (generations), to yield improved properties over time (in the evolution process). In each generation, all the individuals are evaluated and their quality is usually represented with a single *fitness value* (e.g., the number of correctly mapped outputs for a set of input vectors). Individuals undergo random changes using a mutation operator and they combine their structure creating new solutions with a crossover operator. Finally, based on their fitness, the selection operator imitates the natural evolution by selecting better individuals for the next generation.

The evolution process is stopped when a predefined criterion is met, e.g. when a 'satisfactory' solution is evolved or after a predefined time has elapsed. The final solution is usually the best individual from the last generation.

This approach is stochastic, and does not guarantee finding an optimal solution in every run (e.g. a circuit with full functionality). On the other hand, the evolution can be repeated many times with no additional effort, which can produce acceptable solutions with variable success rate. Although this method may be used to synthesize various types of electronic circuits, in this work we concentrate on combinatorial circuits only.

A common method of evaluation of candidate combinatorial circuits includes a set of input vectors and a corresponding set of output values. The number of successfully mapped outputs may be directly used as a fitness measure.

However, there are other network properties that may be taken into account, such as the total number of gates, number of levels, propagation delay etc. These properties may be explicitly included in the fitness function, which can lead

to multi-objective evolutionary algorithms. In this work we consider a single fitness value based only on functionality, as is the case in most of similar applications [4][5][6], but additional properties are also examined and reported. One advantage of evolutionary synthesis is that it offers a set of potential solutions (instead of just one); if there are more individuals with the same level of functionality, we may choose the one which exhibits better performance in the desired secondary objective.

III. Network Design with Cartesian Genetic Programming

A. Cartesian Genetic Programming

The *Cartesian Genetic Programming* (CGP) was actually inspired by the structure of FPGA components [1][7][8]. In CGP, a representation (also known as *genotype*) of a potential solution is a vector of constant length consisting of natural numbers. The vector is used to construct a directed graph, which in turn can be interpreted as an FPGA structure, where cells are represented with graph nodes and connections with arcs in the graph. An example CGP individual and a corresponding directed graph are shown in Fig. 1.

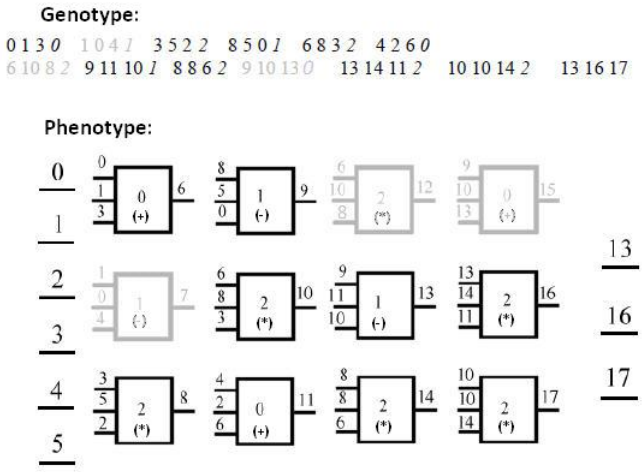

Fig. 1: CGP example; mapping genotype (vector) to phenotype (directed graph) with 6 inputs, 3 outputs and 3 functions in squares (0 as summation, 1 as subtraction and 2 as multiplication). Grey nodes represent unconnected nodes to the outputs.

In this example, each node has an equal number of inputs (3) and a single output. Each node performs one function from a set of predefined functions (descriptive symbols are in brackets), which is denoted with a single index value. In this way, each node is represented with four values in the vector: the first three are the inputs, and the fourth one is the function index. The position of those values in the vector corresponds to the node placement in the graph (top to bottom, from left to right). Node outputs are connected to input(s) of nodes in the next column (or further than the next one); nodes in the same column cannot be connected. The node interconnectivity is defined by the *levels back* parameter, which determines how many previous columns of nodes may have their outputs connected to a node in the current column (the primary inputs are treated in the same way as node outputs). In this example, levels back parameter is two columns.

The CGP allows for redundancy in its representation, which may be caused with the following [9]:

- *node redundancy* occurs if a node is not connected (directly or through other nodes) to any of the outputs;
- *functional redundancy* is caused with a number of nodes comprising a function that could be represented with a smaller number of nodes;
- *input redundancy* occurs if an input is not connected with any of the nodes.

Node and input redundancy are considered beneficial, since they allow *neutrality*. Neutrality represents presence of multiple different vectors with the same fitness value during evolution process. Functional redundancy, on the other hand, is undesirable since it increases the solution size and the problem search space.

An important component in using CGP is the definition of parameters, since they may have a great influence on the quality of solutions. The parameters that are inherent to the CGP include the number of rows and the number of columns in the representation, as well as the number of input connections for nodes in the graph. The choice for these parameters should be guided with the expected complexity of the combinatorial circuit we are trying to synthesize; this is a crucial decision and the one which needs the greatest amount of domain knowledge.

Additional parameters must be defined that are a part of the initial problem, such as the number of inputs and outputs of the network, the set of functions that can be used as graph nodes and levels back parameter.

B. Evolution of Combinatorial Circuits

In the process of evolution, first an initial population of solutions is created randomly, following predefined rules for generating valid individuals [9]. The population size must be defined beforehand, and it remains constant during the entire run. In each generation, all the individuals are evaluated (more detailed in Sec. IV) and being subjected to genetic operators - mutation, crossover and selection.

The CGP defines common crossover and mutation operators, which allows this genotype structure to be used in any evolutionary algorithm. There are many variations of these, but here we only describe the variant used in this work. The mutation operator chooses a random point in the vector that represents an individual, and randomly changes the value at the chosen position. Following predefined rules, the change in the chosen value will depend on the position in the vector (i.e. whether it represents an input of a node or its function), so the operator always yields a valid individual [9].

The crossover operator creates a single new individual (child) from two existing individuals (parents). The operator itself is relatively simple, due to the CGP structure: a single crossover point is selected as a random position in

both parental vectors. The first part of the child, before the crossover point, is taken from one of the parents and the other part is taken from the other parent. No additional conditions are verified, because this operation always produces a valid individual.

The selection operator is a part of the evolutionary algorithm and is not inherent to the CGP itself. In our experiments a simple roulette-wheel selection was used for parent selection [10].

IV. Circuit Specification using Verilog

As mentioned in Section II, the circuit functionality may be defined with the truth table which defines the desired output for every possible input. However, if the number of inputs and outputs is larger, the truth table becomes cumbersome to handle and construct, especially when devising multiple circuits. An alternative way of defining the circuit function - and not its internal structure - is using a hardware description language. The approach and implementation presented in this work allows circuit description in Verilog [11].

The choice of Verilog as a description language was guided with the availability of tools for conversion of Verilog specification into program code that could be used within the CGP evolutionary algorithm. CGP was implemented with the help of *Evolutionary Computation Framework* (ECF), which is a C++ framework implementation of various evolutionary techniques [10]. The conversion from Verilog to a C++ code was made possible with the Verilator tool [12][13].

Verilator takes as the input a Verilog description of the circuit and generates C++ code that is able to simulate the desired network behaviour. Additional functionality is added, in the form of wrapper files, that allow the simulator to be called from the evolutionary system [9]. The simulator can be called every time a potential solution needs to be evaluated, and the simulator output is compared to the output of candidate circuit from the population.

To avoid repeated simulation for the same input vector (repeating the calculation to obtain the same output), a separate *input generator* module is devised that uses the simulator to generate an internal truth table of the predefined size [9]. This allows faster evaluation of different individuals and is a complete automatic process (invisible to the user).

The input generator accepts an additional parameter that defines the percentage of the truth table that will be used (and generated) for fitness evaluation, which may be denoted as *domain coverage* [9]. For instance, if the coverage is 50%, only one half (randomly chosen) of possible inputs and corresponding outputs will be used for candidate circuits functionality evaluation. This feature is added since for larger number of inputs the truth table may be of considerable size and can greatly slow down the evolution process. It is important to note that, if the domain coverage is less than 100%, in every generation a new partial truth table is generated randomly which covers the denoted percentage of test vectors.

The final output of the evaluation is an integer fitness value in the range of $[0, n]$, where n is the actual size of the internal truth table.

A. Limitations

It is obvious that the Verilog language allows the definition of different types of networks and with different modes of description. Verilog description that is of interest here should be behavioral, where only the functionality is defined and not the internal structure (as opposed to a synthesizable description) [11]. Furthermore, only combinatorial circuits must be defined, with no memory elements (e.g. a description containing a `reg` keyword). This condition is not checked automatically, and we rely on the designer to be able to specify the behaviour of a combinatorial circuit only.

V. Results and Analysis

The experiments described in this section were designed to answer the following questions:

- Is the CGP able to provide a functionally correct network?
- What is the influence of CGP parameters (such as number of rows and columns) to the quality of results?
- What is the quality of the results considering internal structure, in comparison with human-made circuits?

All the experiments are made with a 100% domain coverage, i.e. all the possible input combinations are included in the truth table. The following parameter settings are used in the experiments, unless stated otherwise:

- population size of 50 individuals;
- termination condition is 200 generations without improvement of the best individual;
- number of repetitions is at least 10 for each parameter set;
- the initial function set contains functions AND, OR and NOT.

A. Test Case: binaryToESeg_Behavioral

This test case covers the conversion of a binary coded digit to 8-segmented display, with only the signal for the "E" segment as shown in Fig. 2 [11]. The number of inputs is 4, thus covering 16 possible input vectors, and a single bit is the output. An example of a human-made solution is shown in Fig. 3.

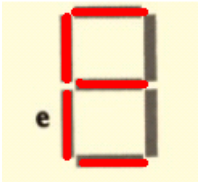

Fig. 2: 8-segmented display with only the "E" segment functional signals.

The CGP system was executed with different genotype sizes (row and column number) and different levels back parameter values. Population size was set to 100 individuals. The convergence of the CGP system is described using the

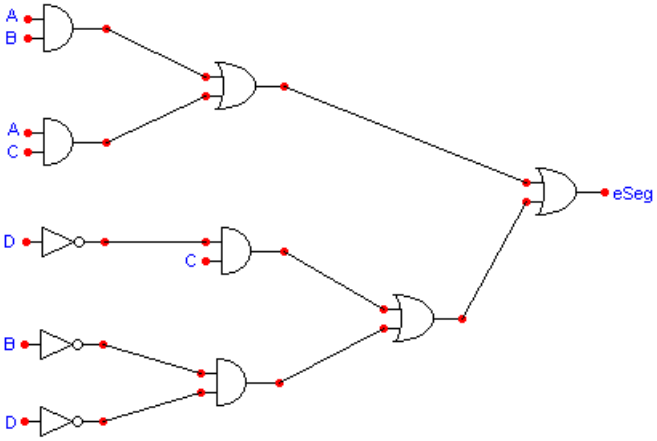

Fig. 3: Human-made solution to the "E" segment problem (binaryToESeg_Behavioral).

maximum and average results of best individuals out of all repetitions for each parameter setting. The results for this experiment are shown in Table I.

TABLE I: Results for test case: binaryToESeg

CGP parameters			Fitness value	
Rows	Columns	Levels back	Mean	Maximum
4	4	1	14.7	15
4	4	2	14.6	15
4	4	4	13.9	15
4	8	1	14.3	15
4	8	4	14.7	16
4	8	8	14	15
8	8	4	14.9	16
8	4	1	14.7	16
8	4	2	14.8	16
8	4	4	14.2	15
8	8	1	14.8	16
8	8	8	14.2	15

CGP was able to produce five correct solutions (functionally correct), with three solutions better than human-made solution; an example of a correctly produced circuit is given in Fig. 4. It can be seen that the best obtained CGP solution incorporates a smaller number of nodes and the same number of levels as the human-made solution.

B. Test Case: synCaseWithDefault

This example from [11] takes 3 single-bit inputs and a single output, comprising the truth table of size 8. The human-made solution to this problem is shown in Fig. 5.

The first experiment obtained results with average quality shown in Table II.

Total number of correct solutions (with fitness 8) was five, with one solution as good as human-made solution and two solution better than human-made solution (one of them is shown in Fig. 6).

In this case an additional experiment was made with extended function set, comprising AND, OR, NOT, XOR and

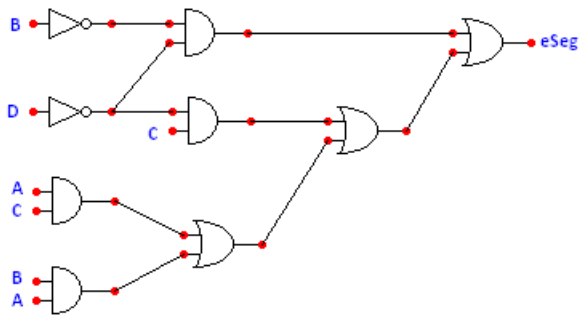

Fig. 4: Solution produced by CGP for the "E" segment problem (binaryToESeg_Behavioral).

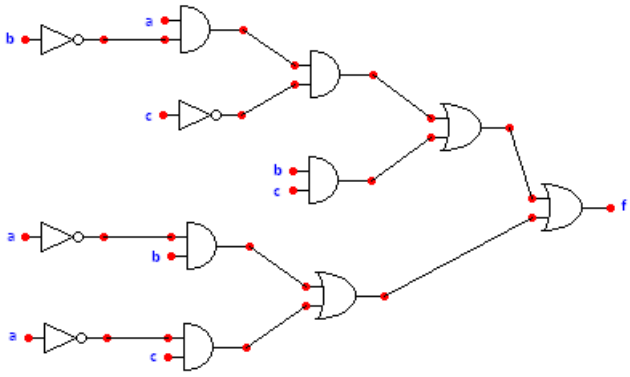

Fig. 5: Human-made solution to the 3 single-bit input and single bit output problem (synCaseWithDefault).

XNOR functions, which produced 29 correct solutions with 25 better than human-made solution. The purpose was to investigate whether a better solution, in terms of the number of nodes and levels, could be evolved. This proved to be true, as CGP produced three solutions with only 4 nodes; the convergence results are shown in Table III, and one of the best obtained circuits is given in Fig. 7.

C. Test Case: oneBitFullAdder

This is an example of a full adder which takes 3 bit inputs and 2 outputs [11], with the maximum fitness value of 16 (2 outputs for every input combination). A human-made solution is shown in Fig. 8.

With the default set of parameters, the CGP was again able to produce a correct solution, for combinations of row and column numbers shown in Table IV.

The structure of three best CGP solutions was equal to the human-made solution in this case, which is the best known solution with this function set [11] (total number of correct solutions was 17).

D. Test Case: decoder2To4

The final example is a 2 to 4 decoder with 3 inputs, including the `enable` signal [15].

TABLE II: Results for test case: synCaseWithDefault (basic function set)

CGP parameters			Fitness value	
Rows	Columns	Levels back	Mean	Maximum
3	3	1	6	6
3	3	3	6	6
3	5	1	6.6	7
3	5	2	6.2	7
3	5	5	6.1	7
5	3	1	6.2	7
5	3	3	6.1	7
5	5	1	6.7	8
5	5	2	6.4	8
5	5	5	6	6

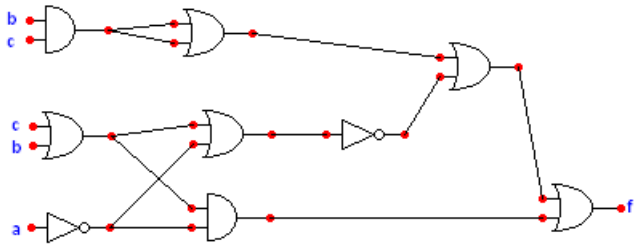

Fig. 6: Solution produced by CGP with basic function set for the 3 single-bit input and single bit output problem (synCaseWithDefault).

The maximum fitness value is 32, which corresponds to 4 correct outputs for every combination of 3 inputs. The default set of parameters was used, along with the population size of 100 individuals. In this experiment the CGP was also able to produce 8 correct solutions, as shown in Table V. An example of a CGP solution is given in Fig. 10.

In this case the CGP solution was not of the same quality as the human made circuit, as shown in Fig. 9, since it includes one additional level and a single additional node. However, the functionality is still preserved with only a small difference in network structure.

E. Discussion

The answer to the first question is clear: CGP is able to provide multiple different functionally correct networks for all test cases.

Considering the second question, the influence of CGP parameters to the quality of results, it shows that parameters close to human-made solutions produce the best results. However, if a better solution exists, human-made solutions can be used as a starting point where parameters (such as number of rows and columns) are gradually reduced until fitness value shows to be increasing.

The third question about the quality of the results, considering internal structure in comparison with human-made circuits, provides interesting results. The first test case (binaryToESeg_Behavioral) and the second test case (synCaseWithDefault) produced better solutions than human-made ones. The third test case (oneBitFullAdder) produced solutions equal to the human-made solution. However, the fourth test case (decoder2To4) did not produce any better solutions. Considering the complexity of certain test cases, we can conclude that the results reflect that complexity.

TABLE III: Results for test case: synCaseWithDefault (extended function set)

CGP parameters			Fitness value	
Rows	Columns	Levels back	Mean	Maximum
3	3	1	7.1	8
3	3	3	7.1	8
3	5	1	7.1	8
3	5	2	7.3	8
3	5	5	7.1	8
5	3	1	7.3	8
5	3	3	7	7
5	5	1	7.8	8
5	5	2	7.7	8
5	5	5	7.1	8

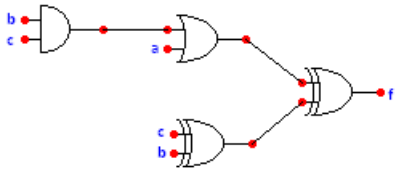

Fig. 7: Solution produced by CGP with extended function set for the 3 single-bit input and single bit output problem (synCaseWithDefault).

Since our primary goal was to find functionally correct solutions (in which we succeeded), the fact that CGP can also implicitly performs the optimization and produces better solutions shows how neutrality is an important feature of CGP.

The only heavy downside was a time-consuming evolution process, that depends on the size of the solutions (individuals in population).

VI. Conclusions

This paper describes the use of *Cartesian Genetic Programming* (CGP) for automatized synthesis of combinatorial circuits. The main purpose of this approach is the simplification of the design process usually requiring human expertise. The results show that the CGP-based evolutionary system is able to produce solutions that satisfy the functional requirements. Additionally, the results show good properties regarding the total number of gates and network levels. The presented application allows network behaviour specification with Verilog, which can facilitate the design process for larger number of input combinations.

References

[1] J.F Miller, P Thomson, and T Fogarty, "Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study," Deptartment of Computer Studies, Napier University, Edinburgh, Case Study 1997.

[2] Timothy G.W Gordon and Peter J Bentley, "On Evolable Hardware," in Soft Computing in Industrial Electronics.: Physica-Verlag, 2002, pp. 279-323.

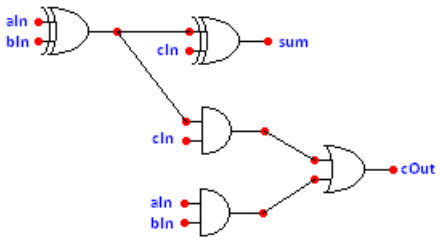

Fig. 8: Human-made solution for the full adder problem (equivalent to the solution produced by CGP).

TABLE IV: Results for test case: oneBitFullAdder

CGP parameters			Fitness value	
Rows	Columns	Levels back	Mean	Maximum
2	3	1	13.9	14
2	3	3	14.8	16
3	2	1	14.4	15
3	2	2	14.5	15
3	3	1	14.6	16
3	3	3	14.6	15
3	4	1	13.9	15
3	4	2	14.8	16
3	4	4	14.6	16
4	3	1	14.8	16
4	3	3	14.5	16
4	4	1	15	16
4	4	2	15.2	16
4	4	4	14.4	15

[3] Jim Torresen, "An Evolvable Hardware Tutorial," in Proceedings of the 14th International Conference on Field Programmable Logic and Applications (FPL'2004), Leuven, Belgium, 2004, pp. 821-830.

[4] Carlos A. Coello Coello, Alan D. Christiansen, and Arturo Hernández Aguirre, "Automated Design of Combinational Logic Circuits Using Genetic Algorithms," in Proc. of the Int. Conf. on Artificial Neural Nets and Genetic Algorithms, ICANNGA'97, University of East Anglia, Norwich, England, 1997, pp. 335-338.

[5] Cecília Reis and J. A. Tenreiro Machado, "An Evolutionary Approach to the Synthesis of Combinational Circuits," in Proc. ICCC 2003 IEEE Int. Conference on Computational Cybernetics, Siofok, Hungary, 2003.

[6] John R Koza, Genetic Programming. On the Programming of Computers by Means of Natural Selection. Cambridge, Massachusetts, USA: The MIT Press, 1992.

[7] J.F. Miller, "An Empirical Study of the Efficiency of Learning Boolean Functions Using a Cartesian Genetic Programming Approach," in Proceedings of the Genetic and Evolutionary Conference (GECCO'99), San Francisco, 1999, pp. 1135-1142.

[8] J.F. Miller and P. Thomson, "Cartesian Genetic Programming," in Proceedings of the Third European Conference on Genetic Programming (EuroGP2000), Berlin, 2000, pp. 121-132.

[9] Iva Brajer, "Oblikovanje kombinacijskih mreža uporabom evolucijskih algoritama," Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Master Thesis 951, 2009.

[10] Domagoj Jakobović. (2011, april) ECF - Evolutionary Computation Framework. (online). http://gp.zemris.fer.hr/ecf/

[11] Donald E Thomas and Philip R Moorby, The Verilog Hardware Description Language, 5th Ed. Kluwer Academic Publishers, 2002.

[12] Wilson Snyder, Duane Galbi, and Paul Wasson. (2010, november) Introduction to Verilator. (online). http://www.veripool.org/wiki/verilator

[13] Wilson Snyder. (2011, april) Verilator - Documentation. (online). http://www.veripool.org/ftp/verilator_doc.pdf

[14] Uroš Peruško and Vlado Glavinić, Digitalni sustavi. Zagreb: Školska knjiga, 2005.

[15] Allison and Robert W. (2011, may) Verilog Examples. (online). http://www.cecs.csulb.edu/~rallison/pdf/Decoder_2_to_4.pdf

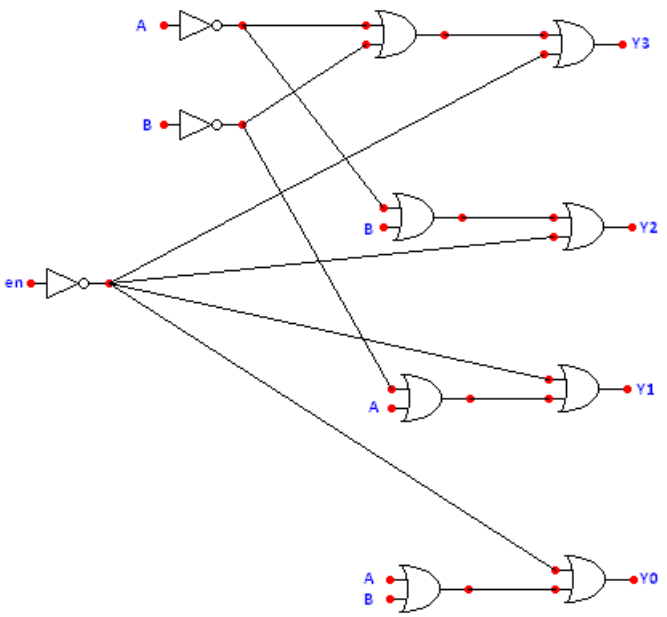

Fig. 9: Human-made solution to the decoder 2 to 4 problem (decoder2To4).

TABLE V: Results for test case: decoder2To4

CGP parameters			Fitness value	
Rows	Columns	Levels back	Mean	Maximum
4	6	3	30.4	32
6	4	1	30	32
6	6	6	30.5	32
6	8	4	30.9	32
6	8	8	30.7	32

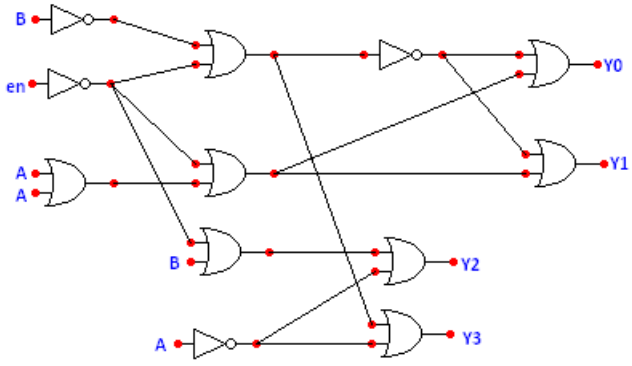

Fig. 10: Produced solution by CGP for the decoder 2 to 4 problem (decoder2To4).