Combinational Digital Circuit Synthesis using Cartesian Genetic Programming from a NAND Gate Template

Muhammad Irfan, Qaiser Habib, Ghulam M. Hassan, Khawaja M. Yahya, Samira Hayat
University of Engineering & Technology
Peshawar, Pakistan
{Irfan697, qaiser.habib.khan}@gmail.com, {gmjally, yahyakm}@yahoo.com, samira_hayat@hotmail.com

***Abstract*—Evolutionary synthesis of combinational digital circuits is a promising research area and many a success has been achieved in this field. This paper presents a new technique for the synthesis of combinational circuits by using Cartesian Genetic Programming (CGP) and uniform NAND gate based templates. Using a uniform gate template implies an ease in the fabrication process but in some instances, the number of gates required may increase which can be optimized by CGP. The mutation operator has been used for achieving convergence. A 2-bit multiplier and 4-bit odd parity generator circuits have been evolved for experimentation and comparison to previous results. The results obtained are compared to earlier work done in the same field. Moreover, the relationship of evolution time (in terms of number of generations) to the population size has been established and analyzed.**

Keywords-NAND gate; Cartesian Genetic Programming; CGP; Combinational Digital Circuit Synthesis

I. INTRODUCTION

Many mechanical methods are available for combinational Circuit Synthesis e.g. Karnaugh Maps[1] and Quine-McCluskey[2][3] Methods. Karnaugh Maps (K-Maps) are based on representing Boolean functions in the form of a two dimensional map and are useful in minimizing functions with up to six variables while Quine-McCluskey procedure is a tabular method that can be extended to any number of inputs. These methods are sufficiently efficient in minimizing the circuit propagation delay but when it comes to minimizing the number of gates, better techniques are required [4].

Many approaches have been recorded in literature for the synthesis of combinational digital circuits e.g., Oltean and Grosan[5] described the application of Multi Expression Programming, which is a CGP variant, in the synthesis of combinational digital circuits; Miller, Job and Vassilev[6] proposed the evolution and representation of Combinational logic circuits using Cartesian Genetic Programming; Koza[7] used Gene Expression Programming and Ryan et al employed Grammatical Evolution for this purpose. However, none of these approaches focuses specifically on evolving synthesis-friendly combinational circuits using a single circuit element.

This paper uses Cartesian Genetic Programming (CGP) to synthesize combinational logic circuits, nevertheless, focus is shifted to the hardware implementation side and circuits which are more synthesis friendly are evolved. Any Boolean function can be completely built using only NAND or NOR functions [5].

In this research, NAND gates are used to facilitate the synthesis process and the difference in fitness is analyzed thoroughly. A fully optimized and reduced circuit which utilizes only a single type of gate may actually prove to be easier to synthesize as compared to circuits based on multiple types of gates [8]. The effect of change in the population size on convergence time is also analyzed in this research.

A fully optimized digital circuit can be evolved using CGP on a high-end computer, so that it is ready to be implemented on an FPGA or ASIC and hence, the heavy processing required for the evolution becomes a one-time cost.

To represent digital circuits by Cartesian graphs, an analogy between the CGP nodes and Gates and the Genotype and wire-connections of the circuit is established as described in [9].

Sections II, III and IV give an introduction to function synthesis using NAND gates, Cartesian Genetic Programming and the fitness function employed, respectively. In section V, the evolution of the logic circuit for a 2-bit multiplier and 4-bit odd parity generator, the relationship of the number of generations taken to evolve a circuit (convergence time) and the size of population is analyzed and depicted graphically.

II. FUNCTION GENERATION USING NAND GATES

The NAND function, which is a combination of NOT and AND functions, is a universal function. The graphical representation of NAND gate and its truth table for two inputs is shown in Figure 1. As a universal function, all other Boolean functions can be generated by using combinations of two or more NAND gates on the basis of certain logical theorems like the DeMorgan's theorem; which states that the complement of AND of two variables is equal to the OR of the complements of these variables. The generation of various Boolean functions from NAND gates is briefly outlined in Figure 2. It represents the examples of NOT, AND and OR gate implementation.

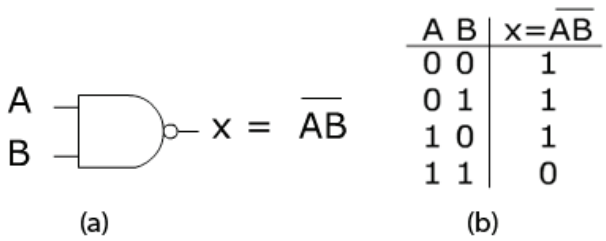

Figure 1. (a) NAND Gate (b) Truth Table

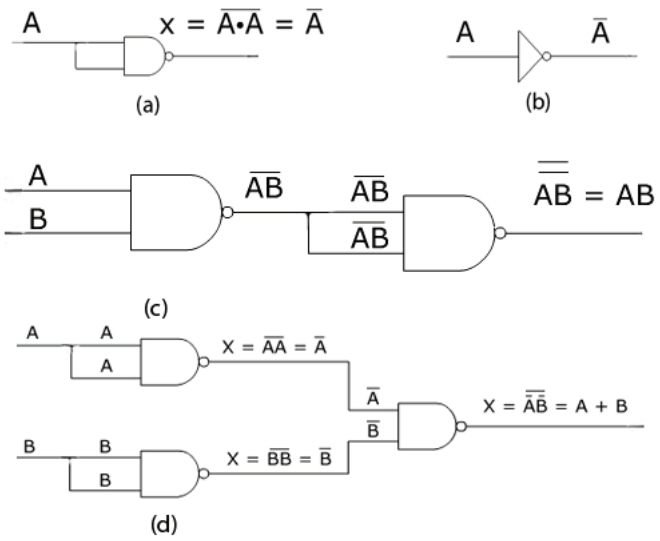

Figure 2. (a) & (b) NOT implementation (c) AND implementation (d) OR implementation

III. Cartesian Genetic Programming

Cartesian Genetic Programming, introduced by Miller [9], is a variant of Genetic Programming which interprets the individuals of a genetic computation in the form of CGP graphs instead of trees. The term "Cartesian" signifies the fact that the graphs have a two-dimensional numbering i.e. each node has an index in the form (x, y). Thus, we can represent such structures in the form of two dimensional arrays/matrices in a computer program. The general form of a Cartesian graph given by Miller and Harding [10] is shown in Figure 3. Each connection is numbered and a particular way of connecting the nodes can also be represented by a CGP string called Genotype. The general form of a Genotype is depicted in Figure 4.

Just like Genetic Programming, further generations are produced in CGP by breeding the parent generations and applying genetic operators: selection, crossover, and mutation.

A. Selection

Selection essentially means selecting individuals with the highest fitness and replacing with them, the individuals which are least fit in the parent generation. Hence a child generation, which has a higher overall fitness, is produced.

B. Crossover

Crossover entails the swapping of genes of two parent chromosomes to improve the diversity of the next generation.

C. Mutation

Mutation is the abrupt flipping of one or more genes of a Genotype. At times, a single character might be the only cause of error which may be retained in every subsequent generation, thus degrading its fitness; such an error can only be eliminated by using mutation.

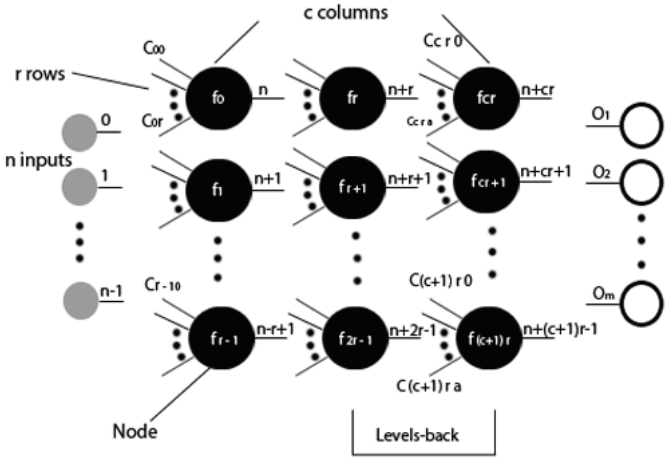

Figure 3. General form of Miller's CGP graph[10]

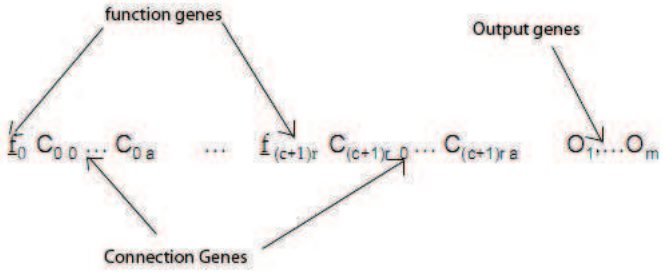

Figure 4. General form of a Genotype[10]

IV. Fitness

It is essential to define a fitness measure and a fitness criterion in any CGP program. Fitness measure gives us an information about the fitness of a Genotype i.e. the level of closeness of the Genotype to the one which is considered fit enough to be the solution of a given problem. Fitness criterion signifies the level of fitness which, when achieved, will solve the problem and consequently the genetic computation will no longer have to iterate further. Thus, the goal of Cartesian Genetic Programming (CGP) is to bring the fitness measure as close to the fitness criterion as possible.

For a truth table with four rows as mentioned in Table 1, the equality between each row's current output and required output increases the fitness while the inequality between them decreases the fitness. The maximum fitness for truth-table

based problems is generally taken to be equal to the number of rows. Thus, for a two input circuit whose truth table consists of four rows, the maximum fitness i.e. 100% fitness is four, as depicted in Table 1.

TABLE 1: FITNESS AND ERROR (MAXIMUM ERROR AND FITNESS IS 4, Z IS THE EXPECTED OUTPUT, ZOUT IS THE OUTPUT OF THE CURRENT GENOTYPE)

X	Y	Z	Zout	Error	Fitness
-	-	-	-	0	4
0	0	1	1	0	4
0	1	0	1	1	3
1	0	0	1	2	2
1	1	0	0	2	2

In this paper, fitness is represented in terms of error and error is the inverse of fitness, as mentioned in Table 1. Thus, maximum fitness implies least error. So, the goal is to minimize the error down to zero which will automatically increase the fitness to maximum.

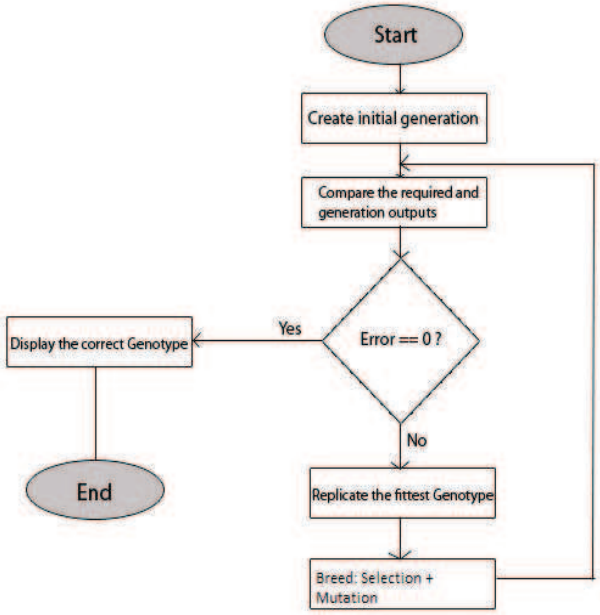

Figure 5. Generic flowchart of a CGP program

V. ANALYSIS AND EXPERIMENTATION

For the experimental cases, two problem sets are taken: 2-bit multiplier and 4-bit odd parity generator.

A. 2-bit multiplier

A 2-bit multiplier is evolved using Cartesian Genetic Programming and a record of the number of gates as well as the number of generations used to evolve the required circuit is stored. A 2-bit multiplier multiplies two 2-bit binary numbers and obtains a possible 4-bit output. The CGP parameters used are given in Table 2. The results obtained after the experiments are tabulated in Table 3.

TABLE 2. CGP PARAMETERS

Parameter	Value
Number of Rows	1
Number of columns	(2*num_gates) +1
Levels back	Maximum
Mutation	1-point
Random function	Seed 1, Dev C++ compiler

The average number of gates per output is rounded up. Miller, Job and Vassilev [6] synthesized the same circuit using 10-gates while the proposed method synthesizes using a maximum of 7-gates only. This provides an economy of up to 3-4 gates per output.

The total number of generations is the iterations taken to evolve the circuits for all four outputs and this is a measure of the convergence time of the circuit. Thus, a larger value of the number of generations implies longer convergence time and vice versa. The population size is the number of Genotypes taken at a time for the evolution process.

TABLE 3. 2-BIT MULTIPLIER RESULTS

Population size	Total generations	Average number of gates per output
3	5421771	7
4	2914309	6
5	2863723	6
6	2159391	6
7	2170431	6
8	480822	7
9	712480	6
10	176761	7
12	106415	6
14	43676	6
16	66062	6
18	35300	6
20	26196	6
25	12295	6
30	26424	6
40	24871	6
50	4033	6

Figure 6 represents the relationship between the number of generations and the population size. It can be observed that with an increase in the population size, the number of generations taken to evolve completely decreases. Hence, a choice can be made between processing complexity and convergence time. It is also worth noting that the circuit complexity (number of gates used) remains essentially constant as the population size is varied. The final evolved circuit for a 2-bit multiplier is given in Figure 7.

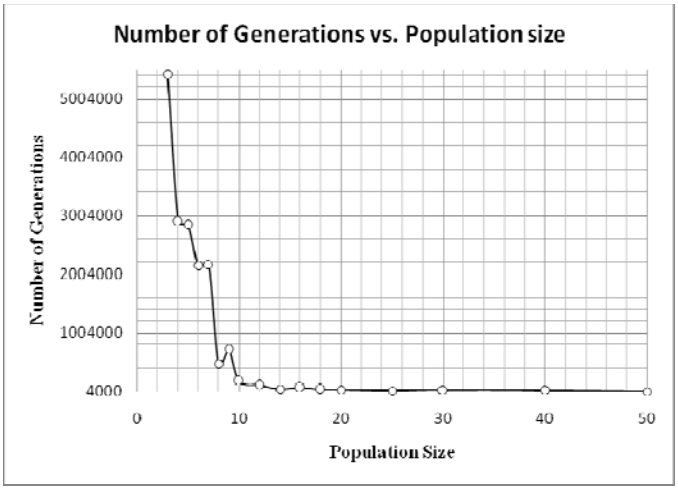

Figure 6. Number of generations vs. Population size

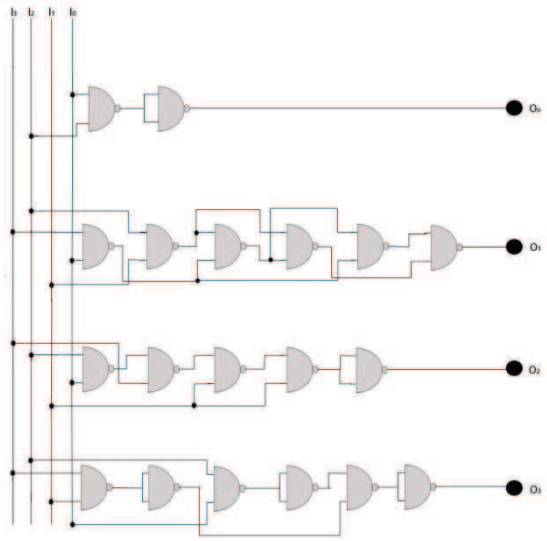

Figure 7. 2-bit multiplier circuit

B. *4-bit odd parity generator*

A 4-bit odd parity generator circuit is also evolved using the same technique which generates a logic-1 at the output for even number of 1's in the input and a logic-0 for odd number of 1's in the input. Considering the complexity of the circuit, experiments are conducted on various population sizes and number of gates. The CGP parameters are kept the same as mentioned in Table 2. The experimental results are shown in Table 4.

It can be observed again that as the population size increases, the number of generations taken to evolve completely is decreased. The number of gates used range from 16 to 20. An evolved circuit for population size 27 and 18 gates in 10104 generations is given in Figure 8.

VI. Conclusion

This paper describes the synthesis of combinational logic circuits using Cartesian Genetic Programming by using a

TABLE 4. 4-BIT ODD PARITY CIRCUIT EVOLUTION RESULTS

Population size	Number of generations	Gates used
30	8115	17
32	11840	18
34	9395	19
36	9406	18
38	17656	16
40	14345	19
42	15336	18
44	36276	19
46	13429	18
48	10906	20
50	1300	20
52	17659	21
54	29412	19
56	73638	18
58	14496	20
60	29958	17

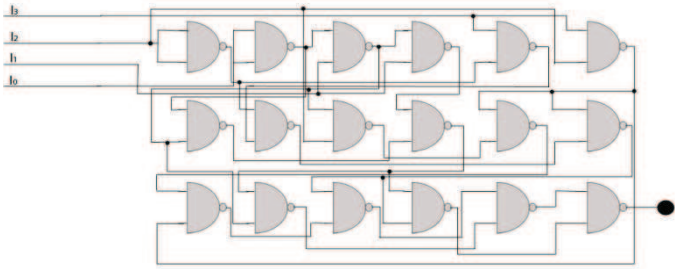

Figure 8. Resulting evolved circuit for a 4-bit odd parity generator circuit

NAND based template which eases the process of fabrication. Analysis shows that the convergence time (number of generations needed) is inversely proportional to the population size and hence selection can be made depending on the application by selecting an optimal trade-off between the population size and evolution time.

VII. Future Work

Future work may include the exploiting of other evolutionary techniques using a NAND gate template. Similarly, convergence speed can be further optimized by using hybrid search operators. This hybridization is also expected to synthesize such complex digital circuits which cannot be efficiently synthesized using mutation only. It is also planned to experiment with evolutionary techniques for the synthesis of sequential digital circuits which rely not only on the current input values but also on the previous output and input values.

References

[1] M. Karnaugh, "The Map Method for Synthesis of Combinational Logic Circuits", Transaction of the AIEE, Communications and Electronic, November, 1953, 72(I): pp. 593-599.

[2] W. V. Quine, “A Way to Simplify Truth Function”, American Mathematical Monthly, 1955, 62(9): pp. 627-631.
[3] E. J. McCluskey, ”Minimization of Boolean Function”, Bell Systems Technical Journal, November 1956, 35(5): pp. 1417-1444.
[4] Coelho, A.C., Christiansen, A.D. and Aguirre, A.H. (2001), “Towards Automated Evolutionary Design of Combinational Circuits”, Comput. Electr. Eng., 27, 1-28.
[5] Oltean, M. and Grosan, C., “Evolving Digital Circuits using Multi Expression Programming”, NASA/DoD Conference on Evolvable Hardware, (24-26 June, Seattle), Edited by R. Zebulum (et. al), IEEE Press, NJ, 2004, 87-90
[6] J. F. Miller, D. Job and V.K. Vassilev, “Principles in the Evolutionary Design of Digital Circuits - Part I, Genetic Programming and Evolvable Machines”, Vol. 1(1), pp. 7 – 35, Kluwer Academic Publishers, 2000.
[7] J. R. Koza, “Genetic Programming: On the Programming of Computers by Means of Natural Selection”, MIT Press, Cambridge, MA, 1992.
[8] A. Hernandez-Aguirre, C. A. Coello-Coello, and B. P. Buckles. “A Genetic Programming Approach to Logic Function Synthesis by means of Multiplexers”, D. K. A. Stoicaand J. Lohn, editors, The First NASA/DoD Workshop on Evolvable Hardware, pages 46–53. IEEE Computer Society, 1999.
[9] J. F. Miller and P. Thomson, “Cartesian Genetic Programming”, Proceedings of the 3rd International Conference on Genetic Programming (EuroGP2000), R. Poli, J.F. Miller, W. Banzhaf, W.B. Langdon, J.F. Miller, P. Nordin, T.C. Fogarty (Eds.), LNCS 1802, Springer-Verlag, Berlin, pp. 15-17, 2000.
[10] Miller, J., Harding, S.: Cartesian Genetic Programming. GECCO 2008 Tutorials, ACM, pp. 2701-2725.