Cartesian genetic algorithm for boolean synthesis with power consumption restriction

Jaime Vitola*, César Pedraza**
Universidad Santo Tomás
*Ingeniería Electrónica, **Ingeniería Telecomunicaciones
Colombia
jaimevitola, cesarpedraza@usantotomas.edu.co

Jose I. Martínez
Universidad Rey Juan Carlos
DATCCCIA
Madrid, Spain
joseignacio.martinez@urjc.es

Johanna Sepúlveda
University of São Paulo
Microelectronics Laboratory LME
São Paulo, Brazil
jsepulveda@lme.usp.br

***Abstract*—The use of evolutionary algorithms in the boolean synthesis is an interesting technique to generate hardware structures with multiple restrictions. However, one characteristic of these algorithms is their high computational load. This paper presents the implementation of a parallel cartesian genetic programming (CGP) for boolean synthesis on a FPGA-CPU based platform. Power consumption and critical path restrictions were included into the algorithm in order to generate structures to solve any problem. As results a 2-bit comparator is presented, as well as response time and data transitions probability.**

I. Introduction

There are different strategies to reduce the number of gates necessary to implement a boolean function. Karnaugh diagrams [1] and Quine-McCluskey algorithm [2] are some of the most popular techniques. Other strategies have been created to reduce other parameters [3] such as size of the circuit and critical path [4] and power consumption [5], [6].

Cartesian Genetic Programming (CGP) has been presented as an alternative to optimize boolean functions with any number of restrictions. Since presented by Miller [7], CGP has been used as whole investigation field applied to digital signal processing [8]–[10], medicine applications [11], arts [7] and boolean synthesis [12], [13].

CGP can be implemented in software and hardware-based platforms. CPUs, Graphic Processing Units (GPUs), High Performance Computers (HPCs) and Field Programmable Gate Arrays (FPGAs) are some of them. Despite all this possibilities, FPGAs are the best option for evolving and synthesizing hardware [14] because the fitness calculation for each individual are carried out faster than software solutions. However, designing a hardware solution for this kind of problems is harder than other solutions.

This article proposes a solution to the digital hardware design with CGP by including the power consumption into the restrictions. The section 2, shows the main aspects of CGP applied to the boolean synthesis. The third section presents a description of the hardware coprocessor designed to calculate the CGP fitness function. Section 4 shows the experiments and results of the algorithm and finally conclusions and future works are presented.

978-1-4799-6839-8/14/$31.00 © 2014 IEEE

II. Genetic programming and boolean synthesis.

Cartesian Genetic Programming was born as an optimization strategy that imitates the Darwin's evolution theory [15]. In the same way genetic algorithms evolve a population, implementing selection, mutation, crossing and other processes [16]–[18], CGP evolves individuals, but they are represented in a cartesian or matrix structure. This representation allows to evolve multiple solutions or perform a multi-objective solution [16], [19], [20]. Figure 1 shows the cartesian representation for a boolean circuit. It can be observed that cells are arranged in columns and rows, allowing to connect the output of any cell to the input of any cell of the next column. Each cell are implemented with any logical function (and, or, nand, nor, xor, xnor, not, yes), so a set of connected gates in a cartesian way defines an individual or candidate solution.

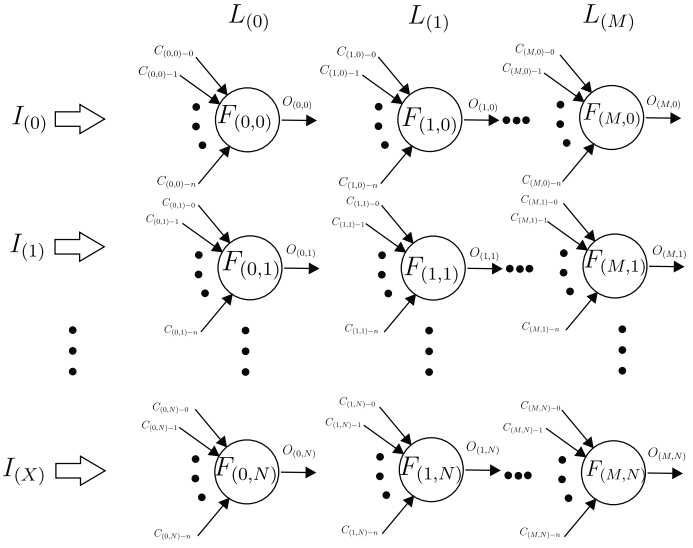


Fig. 1: System architecture.

In figure 1 can be observed that nodes $F(0,0)\ldots..F(M,N)$ represents the logical functions for the circuit, $C(0,0)$–$n\ldots.C(M,N)$–n are the inputs for each cell column and they comes from the output from the previous column. The output of the circuit can be included as a restriction of the algorithm called *level* and determines the

critical path of the solution, therefore any column can be used as output. Finally $I(0)\ldots I(X)$ are the inputs of the circuit.

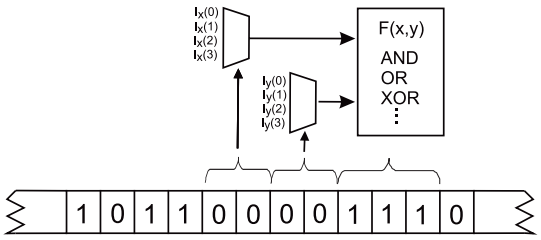


Fig. 2: Chromosome.

This representation will be coded by a stream of bits usually are called *chromosome* and will be part of a candidates called population. The genotype size (chromosome size) can be calculated by adding the total number of bits required to represent a boolean function, plus the number of bits to select the inputs of that function, multiplied by the number of desired gates of an individual (figure 2). It can be inferred that the search space is as large as 2^{L_g}, being 2^{L_g} the number of bit of the genotype, which is usually a large number even for a small candidate circuit. This situation represents a problem in the CGP implemented by software, resulting in evolution times of days or even months.

In order to reduce the evolution times, different platforms can be used: multiple cores CPUs, Graphic Processing Units and High Performance Computers are some of them. In order to take advantage of this platforms, it is required parallel versions of a CGP-based algorithm that involves the island evolution. In this approximation a set of individuals are evolved separately in different cores or threads and then, after a certain number of generations, the best individuals of each thread are migrated to improve the algorithm performance.

In order to evaluate the adaptability of an individual (whether or not a circuit accomplish with the restrictions), a fitness function must be designed. A fitness function for evolving hardware usually requires to meet the boolean behaviour an individual, the number of gates (G), the number of levels or critical path (L) and if implemented, the power consumption of it (P).

In equation 1 the fitness function for the proposed GP is shown. Constants ω_1, ω_2 ω_3 and ω_4 are used for establishing the weights for each one of the parameters. The double-summation term calculates the number of matches of the individual X for all the possible combinations at the output with the target functions Y; the $G(X)$ function is used for calculating the number of logic gates of a chromosome. The m constant is the number of outputs in the circuit and n the number of possible combinations of inputs in the circuit.

It is well known that power consumption of CMOS circuits is proportional to the operation frequency f, parasite capacitances C, the squared of the source voltage VDD^2 and the probability of the data transitions inside the circuit α [21]. Therefore, it is relevant to develop a CGP that selects individuals with a reduced probability of data transitions.

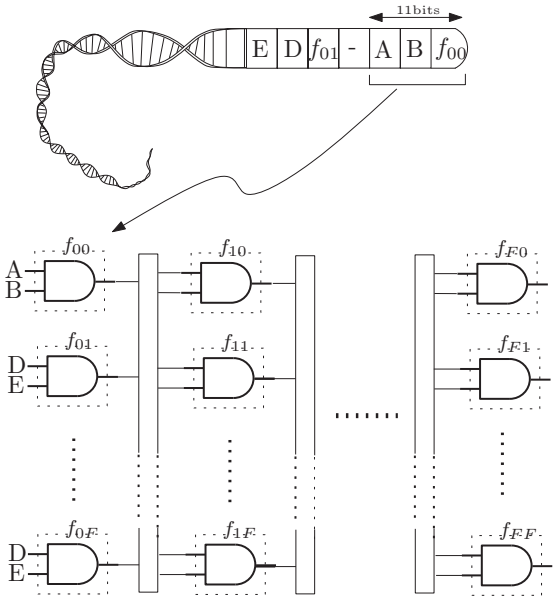


Fig. 3: Individual representation

This parameter can be measured by performing a montecarlo method, that infers the probability of data transitions inside a boolean circuit [21]. In this technique random numbers are placed at the inputs of a circuit, emulating different situations of transitions of the system.

III. Hardware and software implementation.

One of the main characteristics of evolutionary algorithms are their intrinsic parallelism. Populations can be divided in smaller groups in order to be evolved in different cores in a multi-core platform (island approach). This approximation allows to reduce the evolution times without interfere in its results. For this reason, CGP for hardware are good candidates to be implemented in hardware platforms such as FPGAs. On the other hand, it has been demonstrated that boolean synthesis with evolutionary algorithms are better performed in hardware because testing the boolean response of a circuit is executed faster than software [22].

Figure 3 shows the implemented representation. It can be observed that each cell are coded by a 11-bit stream, therefore the genotype size can be calculated. After the first cell of the cartesian individual is coded, the second cell of the same column is coded into the chromosome. The others cells are coded until the a M=16 and N=16 individual is completed.

According to this representation, and taking into account that fitness function has a high computational load [22], a hardware coprocessor with a fitness calculation unit (FCU) was designed (see figure 4). This coprocessor calculates all the possible outputs for an individual and compare it against an objective function, as shown in equation 1. Aditionally, the $P(x)$ and $G(x)$ terms are calculated by this FCU. Finally, a mersenne-twister unit was inserted in order to accelerate the random generation process and the montecarlo method to calculate the $P(x)$ or data probability transitions. All data are

$$fitness = \omega_1.[\sum_{j=1}^{m}\sum_{i=1}^{n} Y(j,i) - X(j,i)] + \omega_2.G(x) + \omega_3.L(x) + \omega_4.P(x) \quad (1)$$

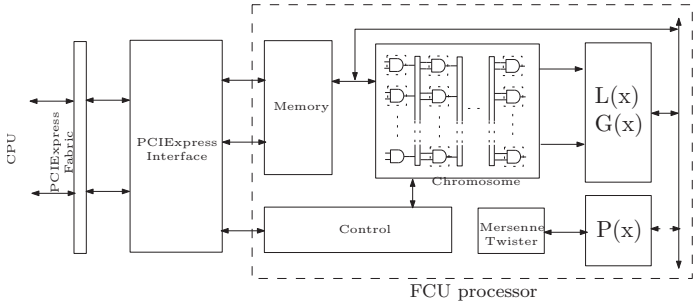


Fig. 4: Fitness Calculation Unit

transferred from a CPU software to a register bank and a 48kB memory, via PCIe x1 bus inside the FCU.

IV. Performance evaluation.

A. Experiment setup.

The CGP was implemented in a 2.8GHz Intel Core i7, 8GB and Virtex 5 ML507 heterogeneous platform. The FPGA board provides a PCIe 1 lane communication interface. Different scenarios with different parameters was tested:

- 4, 8 and 12 variables problems. This parameter determines the complexity of the problem.
- 512, 1024 and 2048 individuals in population. This parameter offers an idea of the scalability of the system
- 1 and 10000 generations.

Results were compared against previous works with other representations [22].

B. Response time.

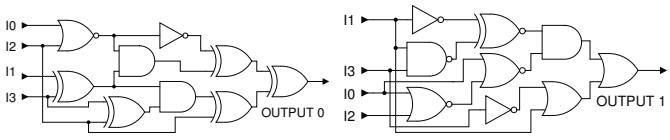
Table I shows the performance of the system with the implemented CGP algorithm. It can be observed a better performance when compared with other platforms and other representation when 8-bit problems where evolved. However it is important to highlight that CGP allows to evolve multiple output circuits, which means it was evolved more than one objective function.

On the other hand, it can be observed that different number of variables do not affect response time substantially, even when the problem complexity grows exponentially. This is because fitness is calculated by hardware, and the response time of the coprocessor is quite less than the software response time executed by the CPU, that calculates the other evolutionary functions.

C. Resulting circuits and power consumption.

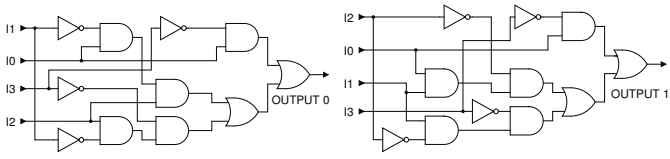
Different objective functions were tested. Figures 5a and 5b shows the results for output0 $(I_1I_0 > I_3I_2)$ and output1 $((I_1I_0 < I_3I_2))$ when a 2-bit comparator were evolved. It can be observed that in each case, 9 gates were used. This configuration were the most optimal in terms of data probability transitions. In order to compare against an optimal synthesis, a Xilinx XST tool were used to generate circuits for the same functions. Figures 6a and 6b show the resulting circuits where 11 gates were used. It is important to highlight that AND, OR and NOT gates are simpler than other gates to produce. However, at the beginning of the evolution process, a 4 level restriction was introduced, which means a critical path of 4. Hence, the resulting circuits could have a better time response.

Finally, figure 7 shows the behavior of transitions inside the circuit when the circuits were evolved. It is observed that it is important to run the process along more than 5000 generations.



(a) Output 0 (b) Output 1

Fig. 5: Resulting circuits for the CGP.



(a) Output 0 (b) Output 1

Fig. 6: Resulting circuits for the XST synthesis tool.

V. Conclusions and future work.

The CGP can be used as an interesting strategy in boolean synthesis problems with more restrictions different to the number of gates. This work demonstrated that power consumption and critical path can be added to the algorithm to generate interesting circuits.

TABLE I: Response time of the algorithm.

Variables	Population Size	RT CGP	RT CPU	RT GPU
4	512	0.427470	0.02	0.9
4	1024	0.856721	0.04	
4	2048	1.708830	0.05	
8	512	0.595945	0.22	3.0
8	1024	1.181023	0.44	4.5
8	2048	2.363067	0.71	
12	512	0.755783	2.1	
12	1024	1.512320	4.2	75.0
12	2048	3.016414	12.5	
16	512	0.922569	–	–
16	1024	1.869172	–	–
16	2048	3.681106	–	–