Genetic Algorithm for FPGA Placement

Zoltan Baruch, Octavian Creț, and Horia Giurgiu

Computer Science Department, Technical University of Cluj-Napoca,
26, Barițiu St., 3400 Cluj-Napoca, Romania
{Zoltan.Baruch, Octavian.Cret, Horia.Giurgiu}@cs.utcluj.ro

Abstract – Field-Programmable Gate Arrays (FPGAs) are flexible circuits that can be (re)configured by the designer. The efficient use of these circuits requires complex CAD tools. One of the steps of the design process for FPGAs is represented by placement. In this paper we present a genetic algorithm for the FPGA placement problem, in particular for the *Atmel* FPGA circuits. Because of the limited routing resources of these circuits, the algorithm allocates a number of empty cells as part of the placement process, in order to use these cells for routing. The cost function used optimizes several different metrics, which include wirelength as well as measures of the routability of the placement. The experiments performed on a set of standard benchmark circuits are promising.

Keywords: VLSI Design, Placement, Genetic Algorithms

1. INTRODUCTION

Field-Programmable Gate Arrays (FPGAs) are flexible circuits that can be easily reconfigured by the designer. These circuits represent one of today's most popular digital logic implementation options. An important component of the FPGA-based design is the CAD software necessary to efficiently use the circuits. A typical FPGA design process consists of logic synthesis, technology mapping, placement, and routing. After the logic synthesis phase, the logic specification is divided in the technology mapping phase into logic functions that are directly implemented in the FPGA circuit. In the placement phase, these logic functions are assigned to specific cells of the circuit. Finally, in the routing phase, the logic signals are connected by programmable switches. In this paper, we focus on the placement problem.

The placement problem, with the objective to minimize the total wirelength, is NP-complete [8]. For real circuits, the solution space is too large to permit enumerative techniques. Therefore, heuristic algorithms are used, which require relatively short execution times (a polynomial function of the number of cells), and can find good solutions to the placement problem, not necessarily the best solutions. Exact algorithms, using techniques like dynamic programming, are computationally expensive [3].

A number of heuristic techniques have been developed for solving the placement problem. A widely used technique for hard combinatorial optimization problems, including placement, is simulated annealing (SA), introduced by Kirkpatrick, Gelatt and Vecchi [6]. A logarithmic temperature scheduling for the SA is proved to ensure global optimum solution. This requires a very slow cooling schedule, which can make SA prohibitively slow. Therefore, an efficient algorithm for the placement problem is desirable.

An efficient paradigm for solving hard optimization problems is the genetic algorithm (GA). Genetic algorithms work with a population of solutions. By the simulated evolution process of GA, over a number of generations, the candidate solutions retain the better characteristics of multiple solutions of earlier generations. This intrinsic parallelism provides the GA with an advantage over simulated annealing [6], which uses one single solution.

Because of such advantages, we are motivated to attempt a solution to the placement of FPGA

circuits by GA. To the best of our knowledge, there is no published GA targeted towards the placement problem for FPGA circuits. Hence, our first contribution is a GA for the placement of FPGA circuits. This algorithm was implemented for the *Atmel 6000* circuits.

A placement is acceptable if 100% routing can be achieved within a given area. This is not an easy task for FPGA architectures with very limited routing resources, like the *Atmel 6000* circuits [1]. A good placement will not only put connected blocks together, but will also ensure that logic elements are not placed too closely in order to ensure the routability of the circuit. This is not an easy task for FPGA architectures with very limited routing resources. Our second contribution is an algorithm which allocates a number of empty cells as part of the placement process, in order to use these cells for routing. The cost function used optimizes several different metrics. These metrics include wirelength as well as measures of the routability of the placement.

The rest of this paper is organized as follows. In the next section, we define the placement problem. In Section 3, we briefly describe some related work. In Section 4, we describe the genetic algorithm for the *Atmel* FPGA circuits. Experimental results are presented in Section 5. Finally, conclusions are made in Section 6.

2. THE PLACEMENT PROBLEM

Given a collection of cells or modules with ports on the boundaries and a collection of nets (which are sets of ports that are to be connected together), the process of *placement* consists of finding suitable physical locations for each cell on the entire layout. The locations are suitable if they minimize given objective functions, subject to certain constraints imposed by the designer, the implementation process, or layout style. The cells may be standard-cells, macro-cells, FPGA logic blocks, etc.

More formally, the placement problem can be defined as follows. Given a set of m modules, $M = \{M_1, M_2, \ldots, M_m\}$, a set of n nets $N = \{N_1, N_2, \ldots, N_n\}$, and a set of p primary input pins and primary output pins $R = \{R_1, R_2, \ldots, R_p\}$, we associate with each module $M_i \in M$ a set of nets N_{M_i}, where $N_{M_i} \subseteq N$. Similarly, we associate with each net $N_i \in N$ a set of modules M_{N_i}, where $M_{N_i} = \{M_j \mid N_i \in N_{M_j}\}$. We are also given a set of locations $L = \{L_1, L_2, \ldots, L_k\}$, where $k \geq n$. The placement problem is to assign each $M_i \in M$ to a unique location L_j such that some objective function is optimized. Usually each module is considered to be a point, and if M_i is assigned to location L_j then its position is defined by the coordinate values (x_j, y_j).

Depending on the technology used, different physical placement constraints exist. For gate-array technology, all modules have the same shape and size and are to be placed into predetermined locations on the placement area. For macro-cell technology, modules have different shapes and sizes, and the dimensions $w_i \times h_i$ of M_i for all the modules are given in the circuit specification. The placement area has dimensions $W \times H$ and is given in the circuit specification.

For performance driven placement, timing specifications are also given. Timing specifications of a circuit include signal arrival times at the primary inputs, the required signal arrival times at the primary outputs, internal delay d_i of a module M_i, and the maximum allowable signal skew C_i at module M_i for all the modules.

3. RELATED WORK

As mentioned before, to the best of our knowledge, there is no reported work which uses genetic algorithms for the placement of FPGA circuits. However, a number of placement tools for FPGA's have been described in the literature, which employ "traditional" methods.

Ebeling *et al.* [4] described automatic mapping tools for the Triptych FPGA architecture. These tools include a placement algorithm based on a simulated annealing approach. Beetem [2] introduced a penalty-driven iterative improvement algorithm for simultaneous placement and routing of FPGAs. Nag and Roy [7] presented an incremental placement algorithm for row-based FPGAs which analyzes post-layout timing and routability information to obtain better place-

ments. Togawa *et al.* [9] proposed a method for the simultaneous place and route of symmetrical FPGAs based on hierarchical bi-partitioning.

Gao [5] developed net-based and path-based performance driven placement algorithms for gate-arrays, macro-cells and Xilinx FPGAs, in order to minimize the signal arrival times at the primary output pins and the signal skews at the inputs of the modules. In the net-based placement algorithm, timing delay requirements are first translated into physical design constraints, such as net constraints. The placement algorithm then generates a placement under the guidance of the net constraints. In the path-based placement algorithm, path delays are considered explicitly during the placement process. This algorithm tries to minimize the total wire length and the latest arrival times at the primary output pins.

4. FPGA PLACEMENT USING GENETIC ALGORITHM

In this section we describe the genetic algorithm for the placement of FPGA circuits in general, and for the *Atmel 6000* FPGA in particular.

4.1 Overview of the Algorithm

The GA is a different approach to the placement problem. While other algorithms iteratively improve a placement by swapping two cells or moving a single cell, the GA works with a set of initial placements representing the initial *population*. Each individual in the population is a string of symbols. The symbols are known are *genes* and the string of genes is called *chromosome*. The chromosome represents a solution of the placement problem. A set of genes that form a partial solution is called a *schema*. Each placement of n cells (i.e., an individual) is an instance of 2^n - 1 schema corresponding to the non-empty subsets of the set of n cells in the placement. The algorithm tries to combine the partial solutions of two different placements to form a better placement.

The basic ideas of the GA were inspired by the process of biological evolution. The GA repeatedly performs the reproduce-evaluate cycle as follows. During each iteration (known as a *generation*) of this cycle the individuals in the current population are evaluated using some measure of *fitness*. The fitness is related to the cost function, and in the case of a minimization problem, is typically the inverse of the cost function. Based on the fitness value, two individuals at a time (called *parents*) are selected from the population, with the more fit individuals getting higher probability of being selected. Then, a number of genetic operators are applied to the selected parents to generate new individuals called *offsprings*, in a process known as *reproduction*. These genetic operators combine the features of both parents. Common operators are *crossover*, *mutation*, and *inversion*. After the genetic operators are applied, the new individuals are evaluated based on the cost function, and a new population is created for the next generation by probabilistically selecting individuals from the entire population, according to their relative fitness. This completes the reproduce-evaluate cycle for one generation or iteration of the GA.

4.2 Genetic Operators

Crossover is the main genetic operator. It operates on two parent individuals and generates an offspring. If Π_a and Π_b are two candidate individuals, representing valid placements, the crossover operation is defined as $\Pi_a \times \Pi_b \rightarrow \Pi_o$, where Π_o, the offspring, is another valid placement. The crossover combines schemata from both parents, and therefore the offspring inherits some of the characteristics of the parents. The simplest form of crossover consists of choosing a random cut point in the gene strings of the parents and generating the offspring by combining the segment of one parent to the left of the cut point with the segment of the other parent to the right of the cut.

This simple form of crossover cannot be applied here, as it can create strings that have no physical representation. For example, a simple crossover between two strings `ABCD` and `BDCA` could produce two new strings `ABCA` and `BDCD`. If each character corresponds to a cell in the placement problem, then a valid string should have one and only one instance of each character. Consequently, crossover operators that preserve

the correctness of the placement are needed. We use a modified crossover operator known as *partially mapped crossover* (PMX).

The PMX crossover is performed as follows. Two parents (1 and 2) are selected and a random cut point is chosen. The entire right substring of parent 2 is copied to the offspring. Next, the left substring of parent 1 is scanned from the left, gene by gene, to the point of the cut. If a gene does not exist in the offspring then it is copied here. However, if it already exists in the offspring, then its position in parent 2 is determined and the gene from parent 1 in the determined position is copied.

Consider the following two parents:

Parent 1: `BIDEF.GCHA`
Parent 2: `AGHCB.IDEF`

Let the cut point after position 4. The right substring in parent 2, which is `IDEF`, is copied into the offspring. Then the first parent is scanned from the left, and since gene `B` (position 0) is not in the offspring, it is copied to position 0. The next gene, `I` (position 1) exists in the offspring at position 5. The gene in position 5 in parent 1 is `G`, and this does not exist in the offspring, therefore gene `G` is copied to the offspring in position 1. The scanning of the first parent is continued in the same way, and the offspring generated is `BGCHAIDEF`.

It is relevant to mention which individuals would participate in crossover in every generation. It is essential that better fitting individuals participate in reproduction more often. If a certain individual reproduces for a large number of times in the early stages of the GA, it is likely that all solutions in the population would inherit some features of this individual and so tend to be the same. Hence, the parent selection strategy employed should avoid this premature convergence of the GA to a local optimum. We achieved this by not permitting any better fit parent to reproduce more than once in a single generation.

Mutation produces incremental random changes in the offspring generated by the crossover. The mutation operator generates new cell-coordinate triples. If the new triples perform well, then the configurations containing them are retained. The mutation is controlled by a parameter referred to as the mutation rate M_r. The mutation process would permit population diversity to be maintained in later stages of the GA. Mutation also helps the GA to avoid any local optimum.

The mutation is controlled by a parameter referred to as the mutation rate M_r. A low M_r means that the infusion of new genes is very low. Such low level mutation would disturb the solution structure by only a small amount, and yet it remains a favorable process for adaptation. A high M_r will cause the offsprings to lose the resemblance to their parents. The mutation process would permit population diversity to be maintained in later stages of the GA. Mutation also helps the GA to avoid any local optimum.

Inversion changes the effective length of a schema without altering the fitness of the individual, in order to increase the survival probability of longer schema. The inversion operator changes the positions of cell records in the string representing a placement but does not change the physical locations of the cell in the circuit. This operator takes a chromosome and two randomly chosen cut points along the length of the chromosome, and reverses the substring between the inversion points. For example, given the string `BID.EFGCH.A`, with the randomly chosen inversion points shown as dots, the result of the inversion operation is `BID.HCGFE.A`. The cell position in the records is not updated in this stage, so the new string still represents the same placement.

4.3 Next Generation Population Selection

The conventional GA employs the approach of generating two offsprings from two parents and replacing the parents by the two offsprings for the next generation process. However, we employed a different evolution strategy. In our approach, all offsprings replace an equivalent number of worst solutions in the current population. This strategy helps the survival of any better solution over many generations. The solutions generated by the genetic operators compete for at least one generation. This approach to next population selection provides an efficient performance for the GA.

4.4 Specific Issues for the Atmel FPGA Circuits

The cost function used optimizes several different metrics. These metrics include wirelength as well as measures of the routability of the placement. For wirelength calculation we use the semi-perimeter method. Minimizing the wirelength will generally have the effect of placing the cells tightly in the center of the array, which almost certainly results in unroutable nets. In order to ensure the routability of the placement, a number of empty cells in the array are used for routing. These cells must be allocated as part of the placement process.

While minimizing wirelength reduces the number of routing resources required globally, it cannot ensure that signals can be routed locally given the limited routing resources. We have added two components to the cost function which address this problem. The *local routability* component attaches a penalty to those situations where it can be determined that a net cannot be routed using local routing resources. Each function requires two or three inputs, only one of which can be supplied by a local or express bus [1]. Thus two-input functions must receive one of their signals from the neighbors, and three-input functions must receive two. There are four adjacent cells which can provide these inputs, and the local routability function checks to make sure that the required input signals are either present in these cells or that there are sufficient routing resources available so that they could be routed. Local routability finds only illegal placements that can be determined from the immediate context.

The *density balance* component is designed to prevent routing congestion due to a high concentration of functions in one part of the array. The metric consists of looking at small windows of cells and counting the number of used inputs in this region. To ensure that the empty cells are evenly spread, the penalty is the square of the number of used inputs above a threshold in a window, summed across all windows. The squaring is necessary to penalize the situations when the used inputs are highly concentrated. The threshold is required so that a small circuit mapped onto an array will not be spread throughout the array.

We examine every unique window in the FPGA, so that the windows overlap. Windows are also allowed to move beyond the chip boundaries, with the virtual cells beyond the chip edge assumed to have as many used inputs as the overall average. If we either did not allow windows to move beyond the chip boundaries, or assumed that the virtual cells had no used inputs, large number of cells would gather at the edge. Similarly, if we assumed that the virtual cells were completely filled with used inputs, these inputs would avoid the edge.

5. EXPERIMENTAL RESULTS

The experiments were performed on an IBM PC computer with a 200 MHz Pentium-MMX processor, under the Windows NT Version 4.0 operating system. The circuits used are from the MCNC (*Microelectronics Center of North Carolina*) benchmark suite.

The effect of population size and mutation rate on result quality was observed in order to determine optimum values for these parameters. Table 1 shows the final total wirelength as a function of population size (the mutation rate being constant, $M_r = 0.05$). The results were obtained for the *daio* circuit. Similar results were obtained for other circuits.

Table 1. Effect of population size on the total wirelength.

Population size	Total wirelength
20	436
40	321
60	349
80	269
100	311
120	346
140	293

Based on these experiments, the following values were used for the main parameters of the GA: population size $N_p = 80$, mutation rate $M_r = 0.05$, inversion rate $I_r = 0.15$, number of generations $N_g = 2000$.

The experiments were conducted as follows. The standard circuits were converted from the

EDIF format (*Electronic Design Interchange Format*) to the netlist format used by the technology mapping program. The circuits were then technology mapped for the *Atmel* 6002 FPGA circuit. The placement process was applied to the netlists obtained after the technology mapping. Some results are shown in Table 2, which indicates the initial and the final wirelengths using the Manhattan metric, and the running time in seconds.

Table 2. The total wirelength and run-time of the GA for a set of benchmark circuits.

Circuit	No. of cells	Total wirelength		Time (seconds)
		Initial	Final	
b1	28	726	245	42
c17	23	363	103	26
cm138a	43	1154	495	72
con1	33	889	320	50
daio	30	734	269	46
decod	75	2485	1265	168
majority	21	339	138	32
tcon	82	2264	1210	153
x2	58	2192	1126	129

6. CONCLUSIONS

We presented a genetic algorithm for the *Atmel* FPGA circuits. The algorithm allocates a number of empty cells as part of the placement process, in order to use these cells for routing. The cost function used optimizes several different metrics, which include wirelength and measures of the routability of the placement. The parameters of the GA were determined experimentally. The experiments performed on a set of standard benchmark circuits show the efficiency of the algorithm.

For comparison purposes, we are currently implementing a placement algorithm using a simulated annealing procedure. An adaptive GA for the same problem is also investigated.

REFERENCES

[1] Atmel Corporation, *Configurable Logic. PLD, FPGA, Gate Array Data Book*, San Jose, 1995.

[2] J. Beetem, "Simultaneous Placement and Routing of the LABYRINTH Reconfigurable Logic Array". *The International Workshop of Field-Programmable Logic and Applications*, Oxford, U.K., 1991, pp. 232-243.

[3] R. Chandrasekharam, S. Subhramanian, and S. Chaudhury, "Genetic Algorithm for Node Partitioning Problem and Applications in VLSI Design", *IEE Proceedings-E*, Vol. 140, No. 5, pp. 255-260, 1993.

[4] C. Ebeling, L. McMurchie, S. A. Hauck, and S. Burns, "Placement and Routing Tools for the Triptych FPGA", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 3, No. 4, pp. 473-482, 1995.

[5] T. Gao, *Performance Driven Placement and Routing Algorithms*, PhD Thesis, University of Illinois at Urbana-Champaign, 1994.

[6] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, No. 220, pp. 671-680, 1983.

[7] S. Nag, and K. Roy, "Iterative Wireability and Performance Improvement for FPGAs", *Proceedings of the 30th ACM/IEEE Design Automation Conference*, 1993, pp. 321-325.

[8] S. M. Sait, and H. Youssef, *VLSI Physical Design Automation*, McGraw-Hill Book Company, 1995.

[9] N. Togawa, M. Sato, and T. Ohtsuki, "A Simultaneous Placement and Global Routing Algorithm for Symmetric FPGAs", *The 2nd International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, Berkeley, CA, 1994.