Multi-FPGA Systems Synthesis by Means of Evolutionary Computation

J.I. Hidalgo[1], F. Fernández[2], J. Lanchares[1], J.M. Sánchez[2], R. Hermida[1], M. Tomassini[3], R. Baraglia[4], R. Perego[4], and O. Garnica[1]

[1] Computer Architecture Department (DACYA)
Universidad Complutense de Madrid, Spain
{hidalgo,julandan,rhermida,ogarnica}@dacya.ucm.es
[2] Departamento de Informática. Universidad de Extremadura, Spain
{fcofdez,jmsanchez}@unex.es
[3] Computer Science Institute. University of Laussane, Switzerland
Marco.Tomassini@iismail.unil.ch
[4] Istituto di Scienza e tecnologie dell´informazione "Alessandro Faedo", CNR, Italy
{Ranieri.Baraglia,Raffaele.Perego}@cnuce.cnr.it

Abstract. Multi-FPGA systems (MFS) are used for a great variety of applications, for instance, dynamically re-configurable hardware applications, digital circuit emulation, and numerical computation. There are a great variety of boards for MFS implementation. In this paper a methodology for MFS design is presented. The techniques used are evolutionary programs and they solve all of the design tasks (partitioning placement and routing). Firstly a hybrid compact genetic algorithm solves the partitioning problem and then genetic programming is used to obtain a solution for the two other tasks.

1 Introduction

An FPGA (Field Programmable Gate Array) is an integrated circuit capable of implementing any digital circuit by means of a configuration process. FPGAs are made up of three principal components: configurable logic blocks, input-output blocks and connection blocks. Configurable logic blocks (CLBs) are used to implement all the logic circuitry of a given design. They are distributed in a matrix way in the device. On the other hand, the input-output blocks (IOBs) are the responsible for connecting the circuit implemented in the FPGA with the external world. This "world" may be the application environment for which it has been designed, or a set of different FPGAs. Finally, the connection blocks (switch-boxes) and interconnection lines are the elements available to the designer for making the routing of the circuit. In most occasions we need to use some of the CLBs to accomplish the routing. The internal structure of an FPGA is shown in Figure 1. Logic blocks, IOBs and the interconnection resources are indicated on it [1].

Sometimes, the size of an FPGA is not enough to implement large circuits and it is necessary the use of Multi-FPGA system (MFS). There is a great number of MFSs, each of them suited for different applications [2]. These systems include not only integrated circuits but, also memories and connection circuitry. Nowadays MFS are used for a great variety of applications, for instance, dynamically re-configurable hardware applications [3], digital circuit emulation [4], and numerical computation [5]. There are a lot of different boards and topologies for MFS implementation. The two most common topologies are the mesh and crossbar types. In a mesh, the FPGAs

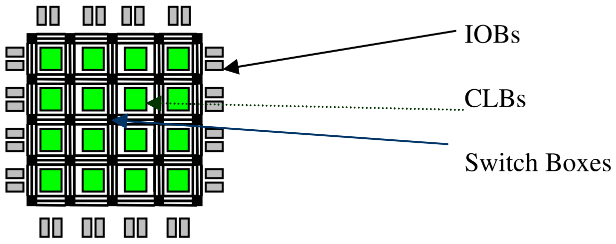

Fig. 1. General structure of an FPGA

are connected in the nearest-neighbor pattern. This kind of board has a simple routing methodology as well as an easy expandability and all programmable devices within the board have identical functionality. Figure 2a shows and FPGA with mesh-topology. On the other hand crossbar topologies, Figure 2b, separate the system components into logic and routing chips. This topology could be suitable for some specific problems, but crossbar topologies usually waste not only logic resources but also routing resources. For these reasons we have focused on mesh topologies

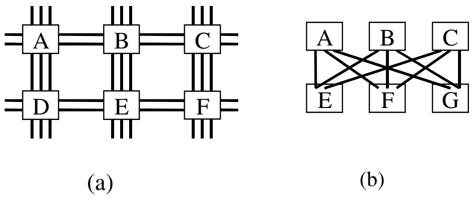

Fig. 2. Multi-FPGA Mesh (a) and crossbar (b) topologies

As in any integrated circuit device, MFSs design flow has three major tasks: partitioning, placement and routing. Frequently two of these tasks are tackled together, because when accomplishing the partitioning, the placement must be considered or vice versa. The solution that we are proposing on this paper covers the whole process. Firstly, we obtain the partitions of the circuit; each of them will be later implemented into a different FPGA. Second, we need to place and route the circuit using the FPGA resources. As we will see in the following sections, we use two different evolutionary algorithms: a hybrid compact genetic algorithm (cGA) for the partitioning step and a genetic programming technique for the routing and placement step. The rest of the paper is organized as follows. Section 2 describes the partitioning methodology. Section 3 shows how genetic programming can finish the design process within the FPGAs, while section 4 contains the experimental results and finally section 5 drafts our conclusions and the future work.

2 MFS Partitioning and Placement

Methodology: Partitioning deals with the problem of dividing a given circuit into several parts, in order to be implemented on a MFS. When using a specific board we must bear in mind several constraints related to the board topology. Some of these constraints are the number of available I/O pins and logic capacity. Although the logic capacity is usually a difficulty, the number of available pins is the hardest problem, because FPGA devices have a reduced number of them comparing with their logic capacity. In addition we must reserve some of the pins to interconnect the parts of the circuit placed on non-adjacent FPGAs. Most of the research related to the problem of partitioning on FPGAs has been adapted from other VLSI areas, and hence, they disregard the specific features of these systems. In this paper a new method for solving the partitioning and placement problem in MFSs is presented. We apply the graph theory to describe a given circuit, and then a compact genetic algorithm (cGA) with a local search improvement [17] is applied with a problem-specific encoding. This algorithm not only preserves the original structure of the circuit but also evaluates the I/O-pins consumption due to direct and indirect connections among FPGAs. It is done by means of a fuzzy technique. We have used the Partitioning93 benchmarks [6], described in the XNF hardware description language (Xilinx Netlist Format) [7].

Circuit Description: Some authors use hypergraphs for representing a circuit netlist, although there are some approximations, which use graphs. We have used an undirected graph representation to describe the circuit. This selection has been motivated because it can be adapted to the compact genetic algorithm code. We will describe later how the edges of his spanning tree can be used to represent a k-way partitioning. A spanning tree of a graph is a tree, which has been obtained selecting edges from this graph [8]. Then we use a hybrid compact genetic algorithm to search the optimal partitioning which works basically as follows. First we obtain a graph from the netlist description of the circuit, and then a spanning tree of that graph is obtained. From this tree, we select k-1 edges and they are eliminated in order to obtain a k-way partition. The partitions are represented by those deleted edges.

Genetic Representation: The compact genetic algorithm (cGA) uses the encoding presented in [9], which is directly connected with the solution of the problem. The code for our problem is based on the edges, which belong to the spanning tree. We have seen above how the partition is obtained by the elimination of some edges. We assign a number to every edge of the tree. Consequently a chromosome will have k-1 genes for a k-way partitioning, and the value of these genes can be any of the order values of the edges. For example, chromosome (3 4 6) for a 4-way partitioning, represents a solution obtained after the suppression of edge numbers 3, 4, and 6 from its spanning tree. So the alphabet of the algorithm is {0, 1… n-1} where n is the number of vertices of the target graph (circuit), because the spanning tree has n-1 edges.

Hybrid Compact Genetic Algorithm: The *cGA* does not manage a population of solutions but only mimics its existence [10]. It represents the population by means of a vector of values, $p_i \in [0,1]$, $\forall_i = 1,\ldots,l$, where l is the number of alleles needed to represent the solutions. In order to design a cGA for MFS partitioning we adopted the edge representation explained below and we consider the frequencies of the edges occurring in the simulated population. A vector V with the same dimension as the number of nodes minus one was used to store these frequencies. Each element v_i of V represents the proportion of individuals whose partition use the edge e_i. The vector elements v_i were initialised to 0.5 to represent a randomly generated population in which each edge has equal probability to belong to a solution [11]. Sometimes it is necessary to increase the selection pressure (Ps)rate to reach good results with a Compact Genetic Algorithm. A value for Ps near to 4 is usually a good value for MFS partitioning. It is not recommendable to increase this value very much because the computation time grows. Additionally, for some problems we need a complement for the cGA. We can combine heuristics techniques with local search algorithms to obtain this additional tool called hybrid algorithms. We have implemented a cGA with local search after a certain number of iterations in order to improve the solutions obtained by the only use of cGA. In [12] a compact genetic algorithm for MFSs partitioning was presented, and in [13] a Hybrid cGA was explained. Authors combine a cGA with the Lin-Kernighan (LK) local search algorithm, to solve TSP problems. The cGA part explores the most interesting areas of the search space and LK task is the fine-tuning of those solutions obtained by cGA. Following this structure we have implemented the hybrid cGA for MFS partitioning presented in [17]. In this paper we have used other heuristic different from LK, which is more feasible to the problem are solving.

Most of the local search algorithms try to perform search as exhaustive as possible. But, this can implies an unacceptable amount of computation time. In MFS problem, the ideal implementation of local search is to explore all the neighbour solutions to the current best solutions after a certain number of iterations. Unfortunately, the most computational expensive step of our cGA is the evaluation of the individuals. We have employed a local search heuristic every *n* iterations and as in parallel genetic algorithms we need to fix the value of *n* to keep the algorithm search in good working order. After an empirical study for different values the local search frequency, we obtain that n must be located between 20 and 60 with an optimal value (that depends on the benchmark) near to 50. So for our experiments we fixed the local search frequency *n* to 50 iterations, i.e. we develop a local search process every 50 iterations of the compact GA.

Remember that a chromosome has k-1 genes for a k-way partitioning, and the value of these genes are the edges eliminated to obtain a partitioning solution. In order to explain the algorithm we must define what a neighbour solution is. We say that solution A is a neighbour solution of B (and B is a neighbour solution of A) if the difference between their chromosomes is just one gene. For example solution represented by chromosome (1 43 56 78 120 **345** 789) is a neighbour solution of the partition represented by (1 43 56 78 120 **289** 789), in an 8-way partitioning problem. Our local search heuristic explores only one neighbour solution for each gene, that is k-1 neighbour solutions of the best solution every n iterations. For the sake of clarity we reproduce the explanation of the local search process presented in [17].

The local search process works as follows. Every n iterations, we obtain the best solution up to that time, which is denoted by BS. To obtain BS, first we explore the compact GA probability vector and select the k-1 most used genes (edges) to form MBS (vector best individual). We also have the best individual generated up to now (GBS) (similar to elitism). The best individual between MBS and GBS (i.e. which of them has the best fitness value) will be BS. After BS has been deduced at iteration n, the first random neighbour solution (TS1) to BS is generated substituting the first gene (edge) of the chromosome by a random one not used in BS. Then we evaluate the fitness value of BS (FV_{BS}) and the fitness value of TS1 (FV_{TS1}). If FV_{TS1} is better than FV_{BS}, TS1 is dropped to BS and the initial BS is eliminated, otherwise TS1 is eliminated. Then we repeat the same process using the new BS but with the second gene, to generate TS2. If the fitness value of TS2 (FV_{TS2}) is better than the present FV_{BS} then TS2 will be our new BS or, if FV_{TS2} is worst than FV_{BS}, there will be no change in BS. The process is repeated for all genes until the end of the chromosome, that is, k-1 times for a k-way partitioning. Although only a very small part of the solution neighbourhood space is explored, we improve the performance of the algorithm significantly (in terms of quality of our solutions) without drastically degrading its total computation time.

In order to clarify the explanation about the proposed local search method we can see an example. Let us suppose a graph with 12 nodes and its spanning tree, for a 5-way partitioning problem (i.e. is we want to divide the circuit into five parts). As we have explained, we will use individuals with 4 genes. Let us also suppose a local search frequency (n) of 50 and that after 50 iterations we have reached to a best solution represented by:

$$BS = 3, 4, 6, 7 \tag{2}$$

The circuit graph has 12 nodes, so its spanning tree is formed by 11 edges. The whole set of possible edges to obtain a partitioning solution is called E:

$$E = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \tag{3}$$

In order to generate TS1 we need to know the available edges A_{LS} for random selection, as we have said we eliminate the edges within BS from E to obtain A_{LS}:

$$A_{LS} = \{0, 1, 2, 5, 8, 9, 10\} \tag{4}$$

Now we randomly select an edge (suppose 0) to build TS1substituying it by the first gene (3) in BS:

$$TS1 = 0, 4, 6, 7 \tag{5}$$

The third step is the evaluation of TS1 (suppose FV_{TS1}=12) and comparing (suppose a minimization problem) with FV_{BS} (suppose FV_{BS} = 25). As FV_{TS1} is better than FV_{BS}, TS1 will be our new BS and the original BS is eliminated. Those changes also affect to A_{LS} because our new A_{LS} is:

$$A_{LS} = \{1, 2, 3, 5, 8, 9, 10\} \tag{6}$$

Table 1 represents the rest of the local search process for this example.

Table 1. Local search example

i	A_{LS}	*BS*	FV	Random gene	*TS*	FV	New BS
1	0,1,2,5,8,9,10	3,4,6,7	25	0	0,4,6,7	12	0,4,6,7
2	1,2,3,5,8,9,10	0,4,6,7	12	1	0,1,6,7	37	***0,1,6,7***
3	1,2,3,5,8,9,10	0,4,6,7	12	9	0,4,9,7	10	***0,4,9,7***
4	1,2,3,5,6,8,10	0,4,9,7	10	8	0,4,8,9	11	0,4,9,7
Pre-Local Search Best Solution: 3, 4, 6, 7							
Post-Local Search Best Solution: 0, 4, 9, 7							

3 Placement and Routing on FPGAs

Once different partitions have been obtained and assigned to different FPGAs, we must place components into FPGAs' CLBs and connect CLBs within each of the FPGAs. To do so, we use Genetic Programming (GP). A wide description of this technique can be found in [14].

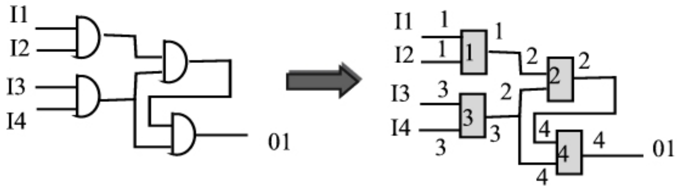

Fig. 3. Representing a circuit wit black boxes and labeling connections

Partitions Representation Using Trees: The main goal for this step is to implement a partition (circuit) -that has been obtained in the previous step- into an FPGA. We have thus to place each of the circuit component into a CLB and then to connect all the CLBs according to the circuit's topology. We have used Genetic Programming (GP) based in tree structures in this task. Therefore, circuits will be encoded here as trees. A given circuit is made up of components and connections. If we forget the name and function of each of the simple components (considering each of them as a black box), and instead we use only one symbol for representing any of them, a circuit could be represented in a similar way as the example depicted in figure 3. Given that components compute very easy logic function, any of them can be implemented by using any of the CLBs available within each FPGA. This means that we only have to connect CLBs from the FPGA according to the interconnection model that a given circuit implements, and then we can configure each of the CLB with the function that each component performs in the circuit. After this couple of simple steps we have got the circuit in the FPGA. Given that we employ Genetic Programming we have to encode the circuit in a tree. We can firstly number each component from the circuit, and then assign the number of those components to the ends of wires connected to them (see figure 3). Wires could now be disconnected without loosing any information. We could even rebuild the circuit by using labels as a guide.

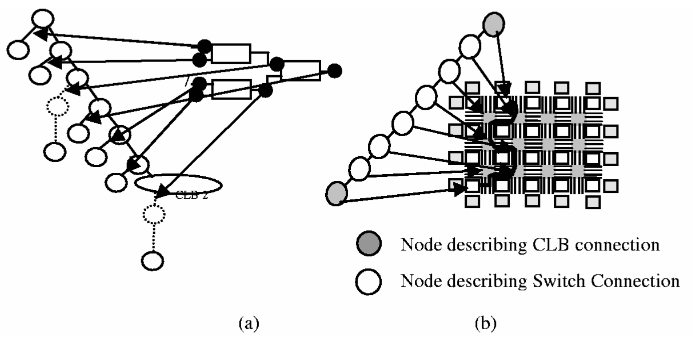
CLB 2
Node describing CLB connection
Node describing Switch Connection
(a)
(b)

Fig. 4. Encoding circuits by means of binary trees. a) Each branch of the tree describe a connection from the circuit. Dotted lines indicates a number internal nodes in the branch. b) Making connections in the FPGA according to nodes.

We may now describe all the wires by means of a tree by connecting each of the wires as a branch of the tree and keeping them all together in the same tree. By labeling both extremes of branches, we will have all the information required to reconstructing the circuits. This way of representing circuits allows us to go back and construct the real graph. Moreover, any given tree, randomly generated, will always correspond to a particular graph, regardless of the usefulness of the associated circuit. In this proposal, each node from the tree is representing a connection, and each branch is representing a wire. The next stage is to encode the path of wires into an FPGA. Each branch of the tree will encode a wire from the circuit. We have now to decide how each of the tree's branches can encode a set of connections. As seen in previous sections, mesh FPGAs contains CLBs, switch blocks and wire segments. Each wire segment can connect adjacent blocks, both CLBs and switch blocks. Several wire segments must be connected through switch blocks when joining two CLBs' pins according to a given circuit description. A connection in a circuit can be placed into an FPGA in many different ways. For example, there are as many choices in the selection of each CLB as the number of rows multiplied by the number of columns available in the FPGA (see figure 1, section 1). Moreover, which of the pins of the CLB will be used and how the wire traverses the FPGA has to be decided from among the incredibly high number of possibilities. Of course, the same connection can be made in many different ways, with more or fewer switch blocks being crossed by the wire.

Every wire in an FPGA is made up of two ends - these can connect to a CLB or to an IOB. On the other hand, as said above, a given number of switch connections may conform the path of the wire. In the representation we have used a branch from tree for codifying wires, CLB and IOB connections are described as each of the two end nodes which make up a branch. In order to describe switch connections, we add as many new internal nodes to the branch as switch blocks are traversed by wires (see figure 4b). Each internal node requires some extra information: if the node corresponds to a CLB we need to know information about the position of the CLB in the FPGA, the number of pin to which one of the ends of the wire is connected, and which of the wires of the wire block we are using; if the node represents a switch connection, we need information about that connection (Figure 4 graphically depicts how a tree describes a circuit).

It may well happen that when placing a wire into an FPGA, some of the required connections specified in the branch can not be made, because, for instance, a switch block connection has been previously used for routing another wire segment. In this case the circuit is not valid, in the sense that not all the connections can be placed into a physical circuit. In order for the whole circuit to be represented by means of a tree, we will use a binary tree, whose left most branch will correspond to one of its connections, and the left branch will consist of another subtree constructed recursively in the same way (left-branch is a connection and right-branch a subtree). The last and deepest right branch will be the last circuit connection. Given that all internal nodes are binary ones we can use only a kind of function with two descendants.

GP Sets: When solving a problem by means of GP one of the first things to do once the problem has been analyzed is to build both the function and terminal sets. The function set for our problem contains only one element: F={SW}, Similarly, the terminal set contains only one element T={CLB}. But SW and CLB may be interpreted differently depending on the position of the node within a tree. Sometimes a terminal node corresponds to an IOB connection, while sometimes it corresponds to a CLB connection in the FPGA (see figure 4,a). Similarly, a SW node sometimes corresponds to a CLB connection, while others affects switch connections in the FPGA. Each of the nodes in the tree will thus contain different information:

- If we are dealing with a terminal node, it will have information about the position of CLBs, the number of pins selected, the number of wires to which it is connected, and the direction we are taking when placing the wire.
- If we are instead in a function node, it will have information about the direction we are taking. This information enables us to establish the switch connection, or in the case of the first node of the branch, the number of the pin where the connection ends.

If we look at figure 4, we can observe that wires with IOBs at one of their ends are shorter –only needs a couple of nodes- than those that have CLBs at both ends –they require internal nodes for expressing switch connections-. Wires expressed in the latest position of trees have less space to grow, and so we decided to place IOB wires in that position, thus leaving the first parts of the trees for long wires joining CLBs.

Evaluating Individuals: In order for GP to work, individuals from the population have to be evaluated and reproduced employing the GP algorithm. For evaluating an individual we must convert the genotype (tree structure) to the phenotype (circuit in the FPGA), and then compare it to the circuit provided by the partitioning algorithm. We developed an FPGA simulator for this task. This software allows us to simulate any circuit and checks its resemblance to other circuit. Therefore, this software tool is in charge of taking an individual from the population and evaluating every branch from the tree, in a sequential way, establishing the connections that each branch specifies. Circuits are thus mapped by visiting each of the useful nodes of the trees and making connections on the virtual FPGA, thus obtaining phenotype. Each time a connection is made, the position into the FPGA must be brought up to date, in order to be capable of making new connections when evaluating the following nodes. If we evaluate each branch, beginning with the terminal node, thus establishing the first end of the wire, we could continue evaluating nodes of the branch from the bottom to the top. Nevertheless, we must be aware that there are several terminals related to each branch, because each function node has two different descendants. We must decide which of the terminals will be taken as the beginning of the wire, and then drive the evaluation to the top of the branch. We have decided to use the terminal that is reached when going down through the branch using always the left descendant (see figure 5).

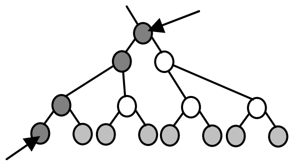

Fig. 5. Evaluating trees

In one sense there is a waste of resources when having so many unused nodes. Nevertheless they represent new possibilities that can show up after a crossover operation (in nature, there always exist recessive genes, which from time to time appear in descendants). These nodes are hidden, in the sense that they don't take part in the construction of the circuit and may appear in new individuals after some generations. If they are useful in solving the problem, they will remain in descendants in the form of nodes that express connections. The fitness function is computed as the difference between the circuit provided and the circuit described by the individual.

4 Experimental Results

- ***Partitioning and Placement onto the FPGAs***

The algorithm has been implemented in C and it has been run on a Pentium II 450 MHz. We have used the partitioning 93 benchmarks in XNF format. As the number and characteristics of CLBs depend on the device used for the implementation, we have supposed that each block of the circuits uses one CLB. We use the Xilinx's 4000 series. Table 2 contains some experimental results. It has seven columns which express: the name of the test circuit (Circuit), its number of CLBs (CLB), the number of connections between CLBs (Edges), the distribution of CLBs among the FPGAs (Distribution), the number of I/O pins lacking (p), the device proposed for the implementation (FPGA type), and the CPU time in seconds necessary to obtain a solution (T(sec)). From the results we can observe that there are some unbalanced distributions. This is a logic result because we need some circuits to pass the nets from one device to another. In addition our fitness function has been developed to achieve two objectives, so that the cGA works. In summary, the algorithm succeeds in solving the partitioning problem with board constraints. We have implemented a board and we have checked some basic circuits so we can conclude that the algorithms works.

Table 2. Partitioning and Placement Results for different benchmarks

Cir-cuit	CLBs	Edges	Distribution	p	FPGA type	T (sec)
S208	127	200	16,15,17,21,11,11,19,17	0	4003	20.70
S298	158	306	20,23,29,20,20,20,25,11	0	4003	5.92
S400	217	412	28,47,23,37,16,20,33,13	0	4005	52.96
S444	234	442	27,38,36,29,41,37,16,27	0	4005	52.99
S510	251	455	34,41,38,42,32,19,24,2	0	4005	96.74
S832	336	808	230,11,25,14,17,18,16,5	0	4008	96.74
S820	338	796	237,17,24,14,21,8,7,10	0	4008	138.93
S953	494	882	168,60,82,31,101,289,15	0	4008	194.65
S838	495	800	100,92,37,77,67,60,43,17	0	4008	293.45
S1238	574	1127	91,11,293,56,50,25,17,31	0	4008	320.14
C1423	829	1465	525,52,114,14,37,51,28,8	0	4020	273.65
C3540	1778	2115	614,135,88,89,56,26,28,2	0	4020	844.16

- ***Inter-FPGA Placement and Routing***

Several experiments with different sizes and complexities have been performed for testing the placement and routing process. [10]. One of them is shown in figure 6. We worked on a SUN workstation 167 Mhz. The main parameters employed were the following: Population size = 200, Number of generations = 500, Population size: 200, Maximum depth: 30, Steady State Tournament size: 10. Crossover probability=98%, Mutation probability=2%, Creation type: Ramp Half/Half. Add best to New Population. The GP tool we used is described in [16]. Figures 6 and 7 show one of the proposed circuits and one of the solutions found, respectively. More solutions found for this circuit are described in [15].

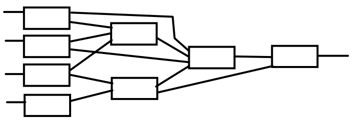

Fig. 6. Circuit to be tested.

5 Conclusions and Future Work

In this paper a methodology for circuit design using MFSs has been presented. We have used evolutionary computation for all steps in the design process. First, a compact genetic algorithm with a local search heuristic was used on achieving partitioning and placement for intra-FPGA systems and, for the Inter-FPGA tasks Genetic programming was used. This method can be applied for different boards and solves the whole design flow process. As future work, we are working now on the parallelization of all of the steps and studying Multi-Objective Genetic Algorithms (MOGA) techniques.

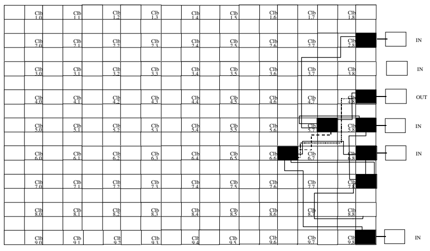

Fig. 7. A solution found for example on Figure 6

Acknowledgement. Part of this research has been possible thanks to the Spanish Government research projects number TIC2002-04498-C05-01 and TIC2002/750.

References

1. S. Trimberger. "Field Programmable Gate Array Technology". Kluwer 1994.
2. S .Hauck,: Multi-FPGA systems. Ph. D. dissertation. University of Washington. 1994
3. M. Baxter. "Icarus: A dinamically reconfigurable computer architecture" IEEE Symposium on FPGAs for Custom Computing machines, 1999, 278–279.
4. R. Macketanz, W. Karl. "JVX: a rapid prototyping system based on Java and FPGAs". In Field Programmable Logic: From FPGAs to Computing Paradigm, pages 99–108. Spinger Verlag, 1998
5. M.I. Heywood and A.N. Zincir-Heywood. "Register based genetic programming on FPGA computing platforms". Euro GP 2000, 44–59.
6. CAD Benmarching Laboratory, http://vlsicad.cs.ud.edu/
7. *XNF: Xilinx Netlist Format*", http://www.xilinx.com
8. F. Harary. "Graph Theory". Addison-Wesley 1968
9. J.I. Hidalgo, J. Lanchares, R. Hermida. "Graph Partitioning methods for Multi-FPGA systems and Reconfigurable Hardware based on Genetic algorithms", Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Program, Orlando (USA), 1999, 357–358.
10. G.R. Harik, F.G. Lobo, D. E. Goldberg "The Compact Genetic Algorithm". Illigal Report Nº 97006, August 1997. University of Illinois at Urbana-Champaign
11. G.R. Harik, F.G. Lobo, D. E. Goldberg "The Compact Genetic Algorithm". IEEE Transactions on Evolutionary Computation. Vol. 3, No. 4, pp. 287–297, 1999.
12. J.I. Hidalgo. R.Baraglia, R. Perego, J. Lanchares, F. Tirado. "A Parallel compact genetic algorithm for Multi-FPGA partitioning" Euromicro PDP 2001, 113–120. IEEE Press.
13. R, Baraglia, J.I.Hidalgo, and R. Perego. "A Hybrid Heuristic for the Travelling Salesman Problem ". IEEE Transactions on Evolutionary Computation. Vol. 5, No. 6, pp. 613–622, December 2001.
14. J.R. Koza: Genetic Programming. On the programming of computers by mens of natural selection. Cambridge MA: The MIT Press

15. F. Fernández, J.M. Sánchez, M. Tomassini, "Placing and routing circuits on FPGAs by means of Parallel and Distributed Genetic Programming ". Proceedings 4th international conference on Evolvable systems ICES 2001.
16. M. Tomassini, F. Fernández, L. Vannexhi, L. Bucher, "An MPI-Based Tool for Distributed Genetic Programming" In Proceedings of IEEE International Conference on Cluster Computing CLUSTER2000, IEEE Computer Society. Pp. 209–216
17. J.I. Hidalgo, J. Lanchares, A.ibarra,R. Hermida. A Hybrid Evolutionary Algorithm for Multi-FPGA Systems Design. Proceedings of Euromicro Symposium on Digital System Design, DSD 2002. Dortmund, Germany, September 2002. IEEE Press, pp. 60–68.