CAD SYSTEM FOR THE ATMEL FPGA CIRCUITS

Zoltan Baruch
E-mail: Zoltan.Baruch@cs.utcluj.ro
Octavian Creț
E-mail: Octavian.Cret@cs.utcluj.ro
Kalman Pusztai
E-mail: Kalman.Pusztai@cs.utcluj.ro

Computer Science Department, Technical University of Cluj-Napoca,
26-28 Bariţiu St., 3400 Cluj-Napoca, Romania

ABSTRACT

In this paper we present a CAD system for logic design using the *Atmel 6000* series FPGA circuits. The design input is a textual description in the *ABEL* hardware description language. This description is compiled into a set of equations. From this set of equations, an internal representation of the digital circuit is generated. Then, the CAD system performs the technology mapping, placement and routing steps, and generates a file for configuring the FPGA circuit. The technology mapping algorithm also tries to reduce the complexity of the placement and routing steps. We describe a bipartitioning algorithm, that not only balances the size of the two partitions, but also evenly distributes the connections among them. The routing algorithm implemented simultaneously treats the global and local routing. According to the sorting of the connection list, we can have two kinds of optimizations: area optimization and speed optimization.

1. INTRODUCTION

Field-Programmable Gate Arrays (FPGA's) are flexible circuits that can be easily reconfigured by the designer, reducing considerably the design cycle [10]. There are many commercial FPGA's. The RAM-based FPGA's, such as *Xilinx*'s *XC3000* and *XC4000*, or *Atmel*'s *6000* series [1], are a widely used class of them. The architecture of a RAM-based FPGA consists of an array of user configurable logic blocks, and a set of programmable interconnection resources used for routing [11]. Each logic block implements a part of the design logic, and the routing resources are used to interconnect the logic blocks.

In computer-aided logic design for FPGA circuits, the main operations performed by the software are the following:

- *Generation of the internal representation*. The design specification, as a schematic or a description in a hardware description language (HDL), must be compiled into an internal representation, which will be used in all the subsequent design steps.

- *Technology mapping.* The intermediate form is adapted for the logic blocks inside the FPGA, considering the restrictions introduced by their architecture (number of inputs, number and type of the functions inside each block).
- *Placement.* Placement consists in assigning each vertex of the logical network to a specific logic block of the FPGA circuit.
- *Routing.* The routing achieves the interconnection of the logic blocks. This operation is usually performed in two steps. In the global routing step, interconnection paths at the global level are chosen, according to certain restrictions which are effective at the global level. In the detailed routing, the starting point is the result of the global routing, the goal being to establish the routing paths at the detailed level. The result of this step is a list of interconnection segments for each group of terminals.
- *Circuit configuration*. The design representation (mapped, placed and routed) is transformed into the bitstream file used for the configuration of the FPGA circuit.

In this paper we present a CAD system for logic design using the *Atmel 6000* series FPGA circuits. The design input is a textual description in the *ABEL* hardware description language. This description is compiled into a set of equations using the compiler of the *Easy-ABEL* development system. From the set of equations, an internal representation of the digital circuit is generated. Then, the CAD system performs the technology mapping, placement and routing steps, and generates a file for configuring the FPGA circuit. Finally, the structure of the configured circuit can be displayed graphically.

The remainder of this paper is organized as follows. In Section 2, we present background information for the technology mapping, placement, and routing problems. In Section 3, we briefly describe the generation of the internal representation for the design. In Section 4, we present the technology mapping algorithm for the *Atmel 6002* FPGA circuit. Section 5 describes the congestion-balanced partitioning algorithm. Section 6 presents the routing algorithm. Finally, conclusions are presented in Section 7.

2. BACKGROUND

2.1 Technology Mapping

Technology mapping is the task of transforming an arbitrary multiple-level logic representation into an interconnection of logic elements from a given library of elements. Technology mapping is a crucial step in the synthesis of logic circuits for different technologies, such as sea-of-gates, gate arrays, or standard cells. The quality of the synthesized circuits depends heavily on this step.

The technology mapping transformation implies two distinct operations [7]: recognizing logic equivalence between two logic functions, and finding the best set of logically equivalent gates whose interconnection represents the original circuit. The fist operation, called *matching*, involves equivalence checking and input assignment. Checking for logic equivalence has been proved to be NP-complete. Input assignment is also computationally complex. The second operation, called *covering*, involves finding an alternate representation of a Boolean network using logic elements that have been selected from a restricted set.

The two operations intrinsic to technology mapping, matching and covering, are computationally difficult. For this reason, several approaches to technology mapping have been pursued and implemented in research and commercial mapping tools (rule-based technology mappers and heuristic algorithms). In this paper, we consider an algorithmic approach to the technology mapping problem.

Most algorithmic approaches to technology mapping attack the problem by dividing it into sub-tasks. First, Boolean networks are partitioned into an interconnection of single-output sub-networks, with the property that each internal vertex has unit outdegree (i.e., fan-out). Then, each sub-network is decomposed into an interconnection of two-input functions (e.g., AND, OR, NAND, or NOR). Each sub-network is modeled by a directed acyclic graph (DAG) called a *subject graph*. Finally, each subject graph is covered by an interconnection of library cells, to produce the final circuit.

2.2 Placement

Given a collection of cells or modules with ports on the boundaries, and a collection of nets (sets of ports that are to be wired together), the process of placement consists of finding suitable physical locations for each cell on the entire layout [8]. The quality of placement is measured by several objective functions, such as the total wirelength, the maximum cut, or the maximum density. Because the wirelength is only known after the routing phase, estimates are used. Some commonly used techniques for estimation of wirelength required by a given placement are the following [8]: semi-perimeter method, complete graph, minimum chain, source to sink connection, Steiner tree approximation, minimum spanning tree.

The placement of cells in order to minimize the total wirelength is an NP-complete problem. Therefore, heuristic techniques are used. Heuristic algorithms for placement can be classified into two categories: constructive and iterative. One of the most used placement techniques is the partitioning-based method.

In computer-aided design, partitioning is the task of clustering objects into groups so that a given objective function is optimized with respect to a set of design constraints. The partitioning techniques are usually based on a graph model of the design. Each node in the graph represents a physical component, and each edge represents a physical connection between two components. The main objective of partitioning is to decompose a graph into a set of subgraphs to satisfy the given constraints, such as the size of the subgraph, while minimizing an objective function, such as the number of edges connecting two subgraphs.

The graphs can be partitioned for performance or for physical size. When partitioning for performance, we cluster graph nodes on critical paths while minimizing communication defined by the number of times control or data is passed between clusters. When partitioning for physical cost, we cluster graph nodes by the type of operations they perform while minimizing the number of wires between different clusters [9]. The difference between partitioning for performance and for physical cost can be explained by its efficiency in time and space. Partitioning for performance optimizes time utilization, while partitioning for physical cost optimizes component utilization.

In general, there are two basic partitioning techniques: constructive methods and iterative improvement methods. The constructive methods partition the graph by starting with one or more seed nodes and adding nodes to the seeds one at a time. The iterative improvement

methods start with an initial partition, and then successively improve the results by moving objects between partitions.

An example of iterative improvement method is the min-cut partitioning. The min-cut partitioning algorithm (also known as the *Kernighan-Lin* algorithm) partitions a given graph $G = (V, E)$ of $2n$ nodes into two equal subgraphs of n nodes minimizing the connections between the two subgraphs. The algorithm starts with an arbitrary partition of V into two subsets V_1 and V_2. On each iteration the algorithm interchanges k pairs ($k \leq n$) of vertices between two sets. It stops when no further improvement is possible.

2.3 Routing

A typical RAM-based FPGA consists of configurable I/O blocks (IOB), an array of configurable logic blocks (CLB), and the interconnect resources (wiring segments and programmable switches). Each programmable switch is a pass-transistor controlled by a static RAM cell. The content of the RAM cell determines whether the pass-transistor is on or off.

The routing resources can be modeled as a graph. Each node in the graph represents either a wiring segment or a CLB pin. Each edge represents a programmable switch in a connection box or a switch matrix. A connection box allows CLB pins to be connected to the routing channel, while a switch matrix provides routing paths from one channel to another. Based on the graph model, the routing of a net consists in finding a tree on the graph that spans over all nodes corresponding to all terminals of the net. In a feasible solution, all trees must be disjoint.

There are only a few published routers for RAM-based FPGA's: *CGE* [4], *SEGA* [6] and *TRACER-fpga* [5]. *CGE* (*Coarse Graph Expansion*) [4] first uses a global routing to decompose each net into a number of two-terminal connections. Its primary goal is to distribute the connections among the channels in order to balance the channel densities. It then chooses for each two-terminal connection exact wiring segments to implement the path assigned during the global routing. A cost function is used to iteratively select among all exact paths the best one. This iteration halts when no more uncompleted exact paths are left.

SEGA (*SEGment Allocator*) [6] is intended for FPGA's with variable-length wiring segments. It addresses the allocation of wiring segments to connections with the goal to match the length of the wiring segments to the length of the connections. *SEGA* uses the same strategy as *CGE*; the main difference comes from the cost function.

TRACER-fpga [5] consists of two stages: initial router, and rip-up and rerouter. During the first stage, nets are routed sequentially and independently of one another, ignoring the existence of any previously routed nets. Inevitably, there will be conflicts over the usage of routing resources among nets. During the second stage, conflicts are resolved iteratively. Within an iteration, some nets are ripped-up and rerouted. The selection of nets for ripping-up is guided by a simulated evolution-based optimization technique. The rerouting is done with the expansion router, except that the presence of other already routed nets is no longer ignored.

3. GENERATION OF THE INTERNAL REPRESENTATION

The starting point for the implementation of a digital circuit using the proposed CAD system is a description of the circuit in a high-level hardware description language. The language

used is called *ABEL-HDL* (*ABEL Hardware Description Language*). From the source file containing the description of the digital circuit in this language an intermediate description is generated, using the compiler of the *Easy-ABEL* development system. This description is written to a .PDS file, containing the equations of the circuit.

The .PDS file contains two main sections:

- declarations (project title, chip, list of pins, list of nodes, etc.);
- equations, describing the functionality of the circuit.

Each equation in the .PDS file is parsed and a graph is generated, where the nodes are basic logic components (AND/OR gates with two-inputs, inverters, D flip-flops, tri-state buffers). For example, for the equation:

$$OUT1 = (IN1 * IN2 + /IN3) \quad (1)$$

the graph shown in Figure 1 is generated.

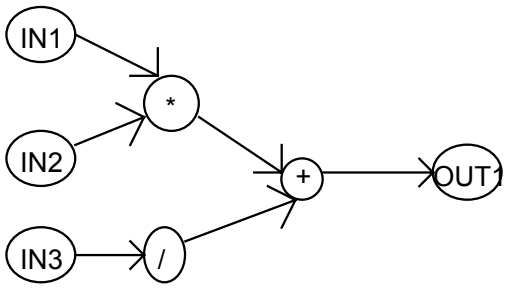

Figure 1. Example of graph generated from an equation.

A registered assignment is implemented by a D flip-flop. Depending on the functionality of the circuit, the registered signals may have one of the following extensions:

.CLKF This extension indicates which signal is routed to the CLK input of the D flip-flop. This extension corresponds to the *.clk* extension in the *ABEL-HDL* language.

.TRST This extension indicates which signal is controlling the output (in this case, we have 3-state output pins). This extension corresponds to the *.oe* extension in the *ABEL-HDL* language.

.SETF, .RSTF These extensions indicate which signal performs the *preset* of the D flip-flop. These extensions corresponds to the *.pr* extension in the *ABEL-HDL* language.

After building each equation's graph, the new graph must be added to the general graph of the circuit. An important step is *redundancy elimination*. Instead of having multiple separate graphs, we combine them (if possible), trying to minimize the number of nodes, and consequently, the complexity of the circuit. This operation is called *elimination of duplicate nodes* and it is performed at every equation parsing.

Finally, according to the list of pins, the input and output pins are identified. The program makes the distinction between output pins and internal nodes. An internal node is a signal used only inside the circuit. The program generates a matrix, where $a_{ij} = 1$ if there is a connection between the node i and j (an edge from i to j or from j to i in the circuit's graph).

4. TECHNOLOGY MAPPING

Our algorithm uses the logical network – obtained after the internal circuit representation step – as a starting point. This logical network has some useful properties which will be used by the algorithm.

The first steps in technology mapping are partitioning and decomposition. These steps have been implemented by the internal representation generation step, a logical network being generated with the required properties. The logical network is already decomposed (i.e., it contains only the basic logic components – the basic functions).

Therefore, the only step that must be implemented is the network covering, which includes also the Boolean matching step. For the network covering, we have to take into consideration the library of available logic cell configurations of the *Atmel 6002* FPGA circuit.

An important aspect that we have taken into account is that the technology mapping step will be followed by the placement and the routing steps. If the number of logic cells generated by the technology mapping step is too large, that will considerably increase the complexity of the placement step.

At the routing step, we have to take care of the *maximum capacity of the routing channels* inside the FPGA circuit. We establish in the program a constant which expresses that capacity, and we *duplicate* the cells whose outdegree is greater than this constant, appropriately updating the connections in the design structure – otherwise the design would not be routable, no matter how we redo the placement step.

As can be noticed from the facts presented above, there has to be a balance between:

- *generating as few cells as possible for reducing the complexity of the placement step*, and
- *generating enough cells in order to ensure the routability of the mapped design*.

5. PLACEMENT

The only metric in the cost function in traditional partitioning algorithms, applied for the placement of FPGA circuits, is the cut size. However, the cut size alone is not a good metric for architectures with limited routing resources, such as FPGA's and CPLD's. Since the algorithm tries to place connected blocks close together, it is likely to generate a placement with congested areas, where a feasible routing is difficult to find. In other words, it is possible to obtain a partition with a small cut size, with one portion being heavily connected and the other being very sparse. For FPGA applications, min-cut based placement algorithms must be modified to take into account not only the sizes of the two portions, the size of the network crossing the cut-line, but also the distribution of interconnections within the two portions.

We describe a placement procedure based on a modified min-cut bi-partitioning algorithm, that not only balances the size of the two portions, but also evenly distributes the connections among them [2]. We consider that multiple terminal nets are represented by a hyper-graph model. In general, to connect k terminals, $max\ \{k\text{-}1, 0\}$ connecting paths are needed. We define the *unbalancing number* of a net to be the number of connecting paths needed to connect all the terminals in the left portion minus the number of connecting paths needed in the right portion. The unbalancing number of a bipartition is defined to be the sum of the unbalancing numbers of all nets. The absolute value of the unbalancing number of a bipartition counts the

difference between the number of connecting paths needed in the left portion and the right portion [9].

Given an initial bipartition, we can compute its unbalancing number in O(|*T*|) time by examining all the nets, where *T* is the set of all terminals. We assume that there are more interconnecting paths in the left portion than in the right portion, that is, the unbalancing number of this bi-partition is positive. If a node *v* is moved from the left portion to the right portion, we can compute *f*(*v*, *e*), the amount by which the unbalancing number of the net *e* decreases, by examining the set of all neighbors of *v*, *N*(*v*).

The following cost function is used to incorporate the effect of congestion distribution in a given partition:

$$Cut\_size + WEIGHT \times Unbalancing\ number$$

where *WEIGHT* is a constant. If *WEIGHT* is set to zero, then the algorithm is the same as the conventional min-cut partitioning algorithm. By setting the value of *WEIGHT* appropriately, we can control the importance of balancing the congestion.

The above partitioning algorithm is applied recursively. The quadrature placement procedure [8] is used for the sequence of cutlines, until the FPGA layout is divided into slots, one slot corresponding to a cell.

6. ROUTING

The first step of the routing program consists of constructing a graph, and then selecting the specific routing segments for each graph. Therefore the allocation of routing resources is strongly dependent of the path chosen by the global router. Given that the FPGA device used here is not of high structural complexity (it includes only local buses, express buses, repeaters and logic cells), the chosen routing algorithm includes the detailed routing inside the global routing at each hierarchical level.

After the global routing, the connections assigned to each sub-channel are known. If the detailed router fails to route all the connections assigned to a sub-channel, then the channel's capacity is correspondingly reduced and the global routing at this level is redone. The advantage of this approach is that the preliminary estimation in the global routing can be corrected immediately.

It is recommended for the algorithm to reserve the bus resources for signals that go over longer distances (more than five cells), so that for these signals the advantages of the express bus become visible. The express buses are not directly connected to the cells, thus they have small capacities, are faster than the local buses, and it is indicated to use them as often as possible for increasing design performance. Also, by using an express bus instead of the local bus, the local bus is released for other necessary routings.

Substituting a local bus by an express bus is not possible in the following situations [1]:

- when directly connecting two cells
- when using a bidirectional signal
- when making 90° turns

For increasing design performance, it is indicated to limit the number of local bus segments which carry a signal and to go beyond the limit of the repeater only if it is necessary.

Branching the express bus signal to the local bus at each repeater may be beneficial when using more than eight signals or when the signal passes over more than one repeater. Figure 2 presents an example of ramification.

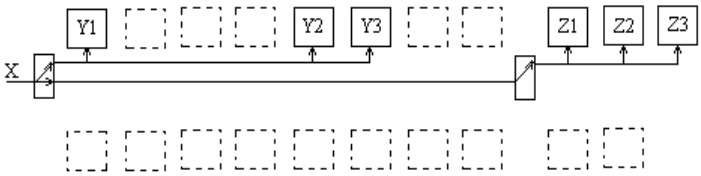

Figure 2. Example of signal ramification inside the FPGA structure.

Signal *X* is routed over the express bus and the branching to the repeaters of the local bus segment are leading to the *Z* and *Y* cells. If signal *X* would have been routed over the local bus to the *Y1*, *Y2* and *Y3* cells and over the repeaters (local bus to local bus) to the *Z1*, *Z2* and *Z3* cells, the load of the *Y* cells would have affected the speed of the signals which are branched to the *Z* cells.

Another key problem of the routing is the selection of the connection. When two or more connections pass over a common routing channel, there may appear competitions for the routing resources in that channel. Because of the limited connectivity in the *Atmel 6002* FPGA, it is essential to resolve these conflicts.

The main problem of FPGA routing is that the choice made for one connection may block another connection [4]. Figure 3 shows two positions of the same section of an FPGA device. Each section offers routing options for either *A* or *B* connection. In the figure, the logic cell is denoted by *L*, the connectivity points by ×, the wire segment (local bus) by a solid line and the possible routing by a dashed line.

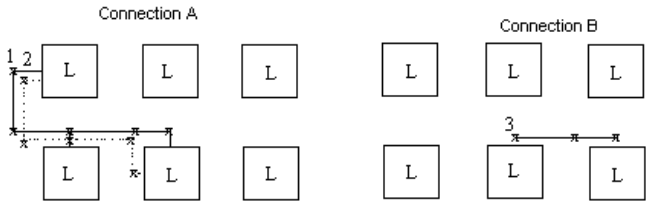

Figure 3. Routing conflicts.

Let us assume that the router first performs the connection *A*. If the wire segment number 1 is chosen for *A*, then connection *B* cannot be routed because *B* relies on the single left option, that is the wire segment number 1. The correct solution for the router is to select the wire segment number 2 for connection *A*; then connection *B* can also be routed. This is a simple example which illustrates the essence of the problem which appears because of the limited routing resources inside the FPGA device.

The algorithm cannot consider all the connecting possibilities inside the FPGA in a single phase because it has to save memory and execution time [3]. This is the reason why it uses an

iterative approach. In the first step it considers only those possible paths for a connection which correspond to the minimal cost (C_d) necessary for the algorithm to find out connecting paths. If these paths fail (are in conflict with the connections already routed), the algorithm continues its search starting with this failed path cost.

In the first step, the connectivity graph is built on the *Atmel 6002* FPGA device structure, and the routing graph is built from the results obtained after the technology mapping and placement steps. By *direct links* we denote here those links which do not imply the use of any bus.

In the second step, the connection list contains all the connections to be routed with the respective linking paths, corresponding to the shortest distance possible. There may exist more than one routing possibility with the same cost. This minimal distance is used as a cost which is computed according to the FPGA structure. The algorithm tries to break the connections in direct links if it does not have to pass over more than five cells, using the fact that the FPGA cells can also be used for routing.

In the third step, the algorithm sorts the connection list by two criteria:

1) the number of stored links
2) the cost of the competition

Each wire segment has two associated costs: the *distance cost* (C_d), which reflects the routing delays associated with the wire segment, and the *competition cost* (C_c), which counts the links' competition to the same wire segment.

Each path in the connection list has also two associated costs: the *sum of the distances cost* (S_d) of the wire segments in the path, and the *sum of the competition cost* (S_c) of the wire segments in the path.

The connection list is sorted by the number of stored links to determine what connection is essential. The essential connection must be identified for making the selection for a connection which has a minimal number of S_d-cost connecting alternatives. The algorithm first sorts the connection list by the C_d cost (also because it tries to make an area and speed optimization). By choosing first the connections with minimal S_d cost, it forces the long lines to choose the express buses, which are much faster. From all possible connections, the one with minimal S_c cost is chosen.

At the routing step, the algorithm selects the linking alternatives with minimal S_c cost. At the rerouting step, it tries to find other possible paths starting from the minimal distance, based on the updated connectivity graph. In the connectivity graph, at this step, the connectivity points used by the connections already routed are marked. The driving cell (the one which generates the signal on the connection) is also marked in the connectivity graph, because we need signal branching in the local bus to become possible. Therefore at the rerouting phase the linking alternatives starting from a connectivity point which is used for other connections are already taken into consideration.

7. CONCLUSIONS

In this paper we presented a CAD system for implementing digital systems using the *Atmel 6000* series FPGA circuits. The design input is a textual description in the *ABEL* hardware description language. This description is compiled into a set of equations, from which an in-

ternal representation of the digital system is generated. Then, the CAD system performs the technology mapping, placement and routing steps, and generates a file for configuring the FPGA circuit. The structure of the configured circuit can be displayed graphically.

In order to reduce the complexity of the placement step, the technology mapping algorithm tries to reduce the number of cells generated. In order to ensure the routability of the design, the algorithm duplicates the cells whose fan-out is greater than the maximum number of channels of the circuit.

The placement step uses a partitioning-based algorithm. The bi-partitioning algorithm not only balances the size of the two portions, but also evenly distributes the connections among them. The algorithm produces good results, in small amounts of CPU time.

The routing algorithm implemented has the advantage that the preliminary estimation of the global routing can be immediately and appropriately corrected. The algorithm may consider the side effects of the routing decisions made for one connection on the others, thus resolving the routing conflicts. According to the sorting of the connection list, we can have two kinds of optimizations: *area optimization* and *speed optimization*.

The CAD system is implemented in the C programming language, under the *Windows 95* operating system.

8. REFERENCES

[1] Atmel Corp., (1995) "Configurable Logic. PLD, FPGA, Gate Array", Data Book.

[2] Baruch, Z., Creţ, O., Pusztai, K., (1997) "Partitioning for FPGA Circuits", in Proceedings of MicroCAD'97 International Computer Science Conf., p113-116, Miskolc, Hungary.

[3] Baruch, Z., Creţ, O., Pusztai, K., (1998) "Routing for FPGA Circuits", in Proceedings of A&Q'98, pQ214-Q219, Cluj-Napoca, May 1998.

[4] Brown, S. D., (1992) "Routing Algorithms and Architectures for Field-Programmable Gate Arrays", Ph.D. Thesis, University of Toronto, Canada.

[5] Chen, C. D., Lee, Y. S., Wu, A. C. H., Lin, Y. L., (1995) "TRACER-fpga: A Router for RAM-based FPGA's", in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 3/95, p371-374.

[6] Lemieux, G. G., Brown, S. D., (1993) "A Detailed Routing Algorithm for Allocating Wire Segments in Field-Programmable Gate Arrays", in Proceedings of ACM/SIGDA Physical Design Workshop, p215-226.

[7] Mailhot, F., De Micheli, G., (1993) "Algorithms for Technology Mapping Based on Binary Decision Diagrams and on Boolean Operations", in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 5/93, p599-620.

[8] Sait, S. M., Youssef, H., (1995) "VLSI Physical Design Automation. Theory and Practice", McGraw-Hill Book Company.

[9] Sun, Y., (1994) "Algorithmic Results on Physical Problems in VLSI and FPGA", PhD Thesis, University of Illinois.

[10] Sun, Y., Wang, T. C., Wong, C. K., Liu, C. L., (1997) "Routing for Symmetric FPGA's and FPIC's", in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1/97, p20-31.

[11] Trimberger, S. M., (1994) "Field-Programmable Gate Array Technology", Kluwer Academic Publishers, Boston.