Synthesis and Floorplanning For Large Hierarchical FPGAs

H. Krupnova, C. Rabedaoro, G. Saucier

Institut National Polytechnique de Grenoble / CSI
46, Avenue Felix Viallet, 38031 GRENOBLE Cedex FRANCE

E-mail address: bogushev@atarax.imag.fr

Abstract

Because the VLSI circuits complexity growth, the trend in design is towards divide-and-conquer schemes, in which circuits are composed of blocks, standard macros or custom macros. From the other side, to allow an implementation of large digital circuits, increased capacity target FPGAs are organized hierarchically. In this paper, we present a hierarchical FPGA floorplanning method which takes into account both the hierarchy of the design and the hierarchy of the target. The method aims at minimization the timing and balancing cost of the floorplan and is based on automatic detection of macro blocks and assigning them to the target FPGA hierarchical zones.

1. Introduction

The next generations of FPGAs are highly complex and contain up to 100K gates. Their structure is commonly hierarchical. They are organized in "blocks" or "quadrants" to provide efficient routing capabilities. Connections follow the hierarchy with different speed characteristics for the different levels of the hierarchy.

In parallel, the design specification and design methods for these targets have become similar to the ones used for ASICs. Initial design descriptions are made in HDL (VHDL or Verilog) and the design methodology follows a mixed top-down/bottom-up approach using extensively macro inference. By macro inference we mean the automatic insertion in the design of optimized register-transfer level blocks of variable bit width such as adders/subtractors, counters, multipliers, etc.

The main issue is to have a good connection between the logical and topological synthesis steps. This is obtained by combining architectural synthesis, macro inference, floorplanning and timing optimization respecting the initial floorplanning feedback. The global design flow is represented in Figure 1. The first step is the architectural synthesis where macroblocks play a key role. The resulting design is mainly a network of blocks plus glue logic. A floorplanning action is required to give a first view of the topological arrangement. This is important to get any accurate timing prediction to guide the synthesis optimization phase. As said above, timing prediction strongly relies on connection delay and therefore depends on the location of the cells. These delays may differ by a factor 1 to 5 according to the placement of the cells.

This floorplanning process has to take into account both the hierarchy of the design and the hierarchy of the target. It is obvious for timing optimization that the containment of the macros within an appropriate area is a key point.

FPGA97, Monterey California USA
© 1997 ACM 0-89791-801-0/97/02 ..$3.50

Therefore, we address in this paper a floorplanning method where the design hierarchy is passed through macro containment constraint and where of course the hierarchy of the target guides the whole approach. The floorplanner definitively gets rid of the gate level view and considers the circuit as a network of macroblocks and glue logic blocks.

This paper will address successively: the introduction of hierarchical target and the design flow focusing on macro inference; the introduction of a floorplanner preserving the macros; final experimental results.

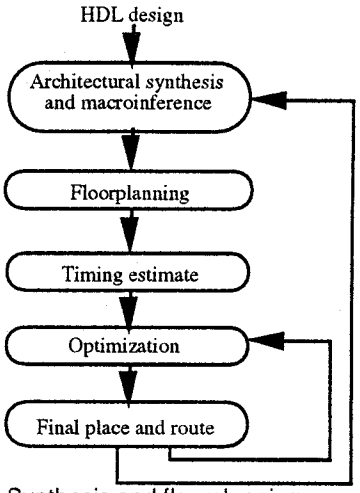


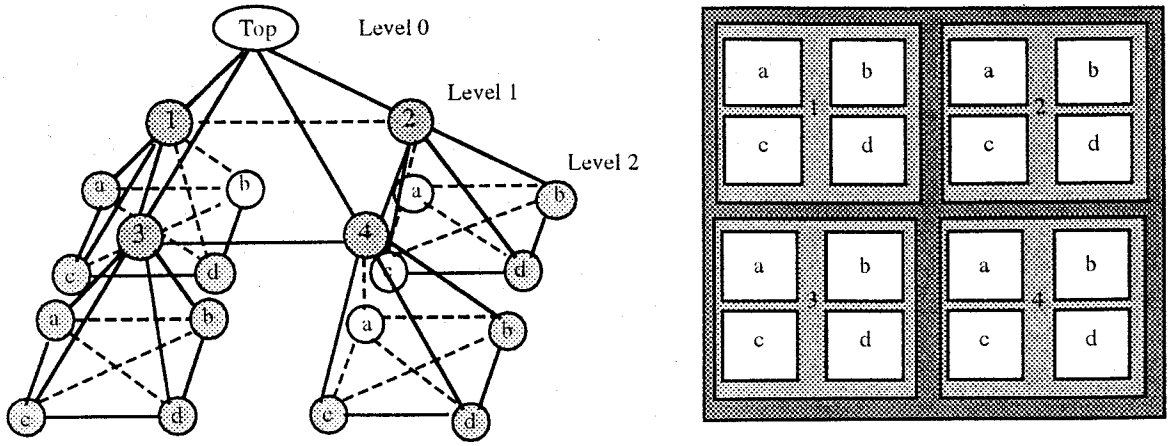
Figure 1. Synthesis and floorplanning general flow.

2. New target and new design methods

2.1 Hierarchical targets

A target architecture is characterized by a set of basic modules (such as CLBs, MUX cells) and interconnection resources. A hierarchical target is characterized by a depth corresponding to the different levels of the hierarchy. At each level the modules are organized into a subset of regions, called "quadrants" containing a fixed number of modules. Like for any hierarchy, the subsets at a given level are included in a subset associated with a higher level. Two quadrants may be connected by several types of connections with regards to timing characteristics. In addition, a limited number of long lines may exist crossing the whole circuit and affording fast connections. Figure 2 gives an example of a hierarchical FPGA structure. In the following of this paper, we take this example to illustrate our method. In Figure 2a the "hierarchy tree" represents the target example which has only three levels, and in Figure 2b the corresponding chip structure is shown.

Each node in the hierarchy tree corresponds to a quadrant region and represents the corresponding modules subset. Each non-leaf node contains four successors. Lines in horizontal planes correspond to inter-block connections. They are referred to as channels corresponding to physical connections.



a) b)

Figure 2. Hierarchical FPGA: a) hierarchy tree; b) chip structure.

We take as example a structure where at the first level connections are organized as a mesh, and at the second level - as a complete graph. In large complex chips there may exist a number of wiring layers on each level. Each type of wiring channel may be labelled with a width (w) and timing characteristics (t) (Figure 3). The width specifies the number of signals which can be carried out by a channel. It is the number of wires connecting the associated pairs of nodes. These parameters should be available from the vendors. As said above, a certain number of global horizontal or vertical wires may exist in the chip which can be used to realize any level connections. In such a system, the connection delay is a function of the physical distance between two modules.

2.2. Hierarchical netlists

2.2.1. Macro inference in HDL flow. As said in the introduction, design and synthesis is a bottom-up/top-down method where the granularity of the basic target cell is any predefined or precharacterized stock. In the past, gates and standard cells were the basic modules. Hard macros and soft macros of fixed bit width have existed early in the design history and are added to the set of basic cells. Recently the impact of "parameterized macrogenerators" has increased due to the development of synthesis tools. Parameterized generators create, on the fly, macros of any size, optimized specifically for a given target and these are automatically inserted in the final layout of the design.

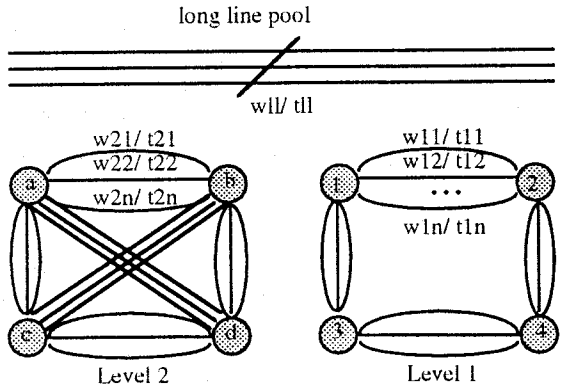


Figure 3. Inter-quadrant connections structure.

Starting from a HDL description, the calls of the macros can be done through macro instantiation, where the macro is named in the HDL design and the synthesis tools calls a prestored macro which is inserted in the layout. The macro call may also be done through macro inference, where a HDL operator or a special HDL template basic block is identified, generated on the fly by the macrogenerator and then inserted in the netlist.

Recently FPGA vendors promoted heavily the use of efficient macros as it is guarantee for efficient results. Some vendors have for a while proposed their own macrogenerators (ACTGEN for ACTEL, XBLOX for XILINX, SCUBA for ATT). A proposal has also been made to standardise the content and the description of a set of elementary macros (LPM or library of parametrized modules). Some vendors accept for place and route netlist containing the LPM. On the other hand, synthesis tools companies offer powerful multitechnology parameterized generators. In addition, big and powerful "Reuse Blocks" stored as predesigned blocks are available today to increase design productivity.

So overall today, when coming from an HDL, specifically through a powerful synthesis tool, a design may contain 80% of macros and 20% of glue. As the number of macros on a chip increases, the demand for adequate synthesis and layout tools increases.

2.2.2. User hierarchical netlist. Assume that we ignore how the netlist has been generated and that a hierarchical netlist is given to the floorplanner. The approach consists in identifying which hierarchy block should be preserved to get an efficient partitioning. The objective is to preserve at least the blocks corresponding to the macros previously mentioned. These macros have to be detected and extracted through a special macro identification procedure. The macro blocks identification process consists in :

1) identifying vendor's macros through their specific name. For example, macros generated by ACTGEN have a suffix "BODY_ACTGEN" ;
2) identifying vendor's macros as blocks which have an associated "don't touch" property;
3) identifying macros given in the associated with the design TAG file generated by a synthesis tool;
4) identifying macros manually specified by the user in a graphical mode or in a directive file;
5) identifying macros through the RTL naming, such as ADD, SUB, REG, DIV, MLT, etc;
6) identifying macros as repetitive blocks in the design. For example, if some non-primitive cell is instantiated in the netlist more than a given number of times, it will be considered as a macro block.

```
* Partitioning Hierarchy Report:
* * Top Block: Inst top_bhv,Cell top_bhv: Size=587.00,IO=342,INSTs=393.
*   BLOCK43 -> Inst ADD_SUB_RETURN_ACTGEN_38_39_inst4_0,Cell ADD_SUB_RETURN_ACTGEN_38_body:
*                                          Size=48.00, IO=48, FF=0, INSTs=18
*         #Level1. Inst ADD_SUB_RETURN_ACTGEN_38_BODY_ACTGEN_64,Cell ADD_SUB_RETURN_ACTGEN_38_BODY_ACTGEN:
*                                          Size=32.00, IO=49, FF=0, INSTs=34
*   BLOCK44 -> Inst ADD_SUB_RETURN_ACTGEN_36_37_inst4_0,Cell ADD_SUB_RETURN_ACTGEN_36_body:
*                                          Size=32.00, IO=48, FF=0, INSTs=1
*         #Level1. Inst ADD_SUB_RETURN_ACTGEN_36_BODY_ACTGEN_63,Cell ADD_SUB_RETURN_ACTGEN_36_BODY_ACTGEN:
*                                          Size=32.00, IO=48, FF=0, INSTs=34
*   BLOCK45 -> Inst MLT99u_50_inst4_0,Cell MLT99u:
*                                          Size=227.00, IO=36, FF=0, INSTs=351
*   BLOCK46 -> Inst MUX_o_49_inst3_1,Cell MUX_o_body:
*                                          Size=32.00, IO=75, FF=0, INSTs=1
*         #Level1. Inst MUX_O_BODY_ACTGEN_65,Cell MUX_O_BODY_ACTGEN:
*                                          Size=32.00, IO=75, FF=0, INSTs=32
*   BLOCK47 -> Inst COUNTER_data_15_inst2_2,Cell COUNTER_data_body:
*                                          Size=19.00, IO=9, FF=7, INSTs=2
*         #Level1. Inst COUNTER_DATA_BODY_ACTGEN_62,Cell COUNTER_DATA_BODY_ACTGEN:
*                                          Size=19.00, IO=10, FF=7, INSTs=20
*   BLOCK48 -> Inst ADD_SUB_RETURN_ACTGEN_7_8_inst1_3,Cell ADD_SUB_RETURN_ACTGEN_7_body:
*                                          Size=16.00, IO=17, FF=0, INSTs=2
*         #Level1. Inst ADD_SUB_RETURN_ACTGEN_3_BODY_ACTGEN_61,Cell ADD_SUB_RETURN_ACTGEN_3_BODY_ACTGEN:
*                                          Size=16.00, IO=24, FF=0, INSTs=18
*   BLOCK49 -> Inst ADD_SUB_RETURN_ACTGEN_5_6_inst1_3,Cell ADD_SUB_RETURN_ACTGEN_5_body:
*                                          Size=16.00, IO=24, FF=0, INSTs=1
*         #Level1. Inst ADD_SUB_RETURN_ACTGEN_3_BODY_ACTGEN_60,Cell ADD_SUB_RETURN_ACTGEN_3_BODY_ACTGEN:
*                                          Size=16.00, IO=24, FF=0, INSTs=18
*   BLOCK50 -> Inst ADD_SUB_RETURN_ACTGEN_3_4_inst1_3,Cell ADD_SUB_RETURN_ACTGEN_3_body:
*                                          Size=16.00, IO=24, FF=0, INSTs=1
*         #Level1. Inst ADD_SUB_RETURN_ACTGEN_3_BODY_ACTGEN_59,Cell ADD_SUB_RETURN_ACTGEN_3_BODY_ACTGEN:
*                                          Size=16.00, IO=24, FF=0, INSTs=18
*   BLOCK51 -> Inst ADD_SUB_RETURN_ACTGEN_1_2_inst1_3,Cell ADD_SUB_RETURN_ACTGEN_1_body:
*                                          Size=24.00, IO=24, FF=0, INSTs=10
*         #Level1. Inst ADD_SUB_RETURN_ACTGEN_1_BODY_ACTGEN_58,Cell ADD_SUB_RETURN_ACTGEN_1_BODY_ACTGEN:
*                                          Size=16.00, IO=25, FF=0, INSTs=18
*   BLOCK52 -> Inst MLT83u_47_inst1_3,Cell MLT83u: Size=58.00, IO=22, FF=0, INSTs=83
*   BLOCK53 -> Inst I1u8_44_inst1_3,Cell I1u8: Size=16.00, IO=16, FF=0, INSTs=16
*   BLOCK54 -> Inst CLT8u_43_inst1_3,Cell CLT8u: Size=11.00, IO=17, FF=0, INSTs=15
*   BLOCK55 -> Inst CLE9u_42_inst1_3,Cell CLE9u: Size=9.00, IO=19, FF=0, INSTs=16
*   BLOCK56 -> Inst CGT9u_41_inst1_3,Cell CGT9u: Size=12.00, IO=19, FF=0, INSTs=17
*   BLOCK57 -> Inst CGE8u_40_inst1_3,Cell CGE8u: Size=8.00, IO=17,FF=0, INSTs=14
```

Figure 4. Initial design hierarchy.

According to the way the macro block is identified, we can differ macros for which the preservation during the floorplanning is mandatory (1-4), and others, for which the preservation is desirable (5-6). For the first ones the identification is explicit, for the second ones the macrostatus is only a guess. An example of the initial design hierarchy extracted from the execution trace of an industrial benchmark synthesized with ACTGEN macro inference is presented in Figure 4. We can see macro cell instances generated by ACTGEN which contain suffix "BODY_ACTGEN", and implicit macro cells identified through their RTL naming (MLT83u, MLT99u).

So, the initial design is structured in a hierarchical manner and is composed of macro blocks and other hierarchy blocks created by user or high-level synthesis tool. User hierarchy may be modified during the floorplanning process - certain hierarchy blocks may be flattened. But all the macro blocks are protected and in the target FPGA device hierarchy tree will be assigned to the same leaf node. An example of the initial design hierarchy, including macro blocks labelled as M and hierarchy blocks labelled as HB is represented in Figure 5a. If all non-macro hierarchy blocks are flattened, the hierarchy contains only two levels as shown in Figure 5b.

3. The hierarchical FPGA floorplanning problem

The floorplanning objectives considered here differ from the ones commonly used for a full-custom design style. The custom design floorplanning consists in placing a given set of circuit modules on a plane minimizing (1) the area of the bounding rectangle containing all the modules; (2) the total wire length ([SuShRo91], [ViTs91], [WaWo93]). In the case of a hierarchical FPGA target, the aim is to partition the design into blocks and associate each block to a leaf quadrant of the target hierarchy tree. The following objectives are taken into consideration.

Area. We suppose that the chip area is split in quadrants as represented in Figure 2 and that the maximal size of the quadrants is limited. Area requirement is satisfied if the design size is smaller than the top-level quadrant size. Notice that routing resources may become an obstacle to successful final layout if the placed blocks density is too high.

Connectivity. To make routing possible, the maximal number of connections between blocks assigned to different quadrants should not exceed the channel width (as specified in section 2.1). To reduce the usage of costly higher level routing resources, blocks which communicate heavily should be placed close in the final floorplan. Clustering communicating modules together at each level is mandatory to get a good quality of the floorplan.

Timing. The delay is a function of distance between blocks, the number of available connections and associated with their delay (Figure 3). Notice the existence of global long lines which make the problem more complicated. As said in the introduction, if the final system delay estimated after partitioning/floorplanning process does not satisfy designer requirements, further iterations in the design cycle may be needed, and final success is not guaranteed.

We call a feasible mapping of the initial circuit to the hierarchical FPGA structure partitioning circuit components in a hierarchy of clusters in such a way that each cluster may be placed into the corresponding FPGA quadrant and a number of wires between two quadrants does not exceed corresponding summary channel length:

$$\forall i: S_i < S_{MAX}$$

$$\forall i, \forall j: Q_{ij} < \Sigma w_{ij} + e_{ij},\ e_{ij} \geq 0,\ \Sigma e_{ij} < w_{ll}$$

where S_i is the size of cluster associated with i-th quadrant, S_{MAX} is the maximal size of this quadrant, Q_{ij} is the number of common connections between clusters associated with quadrants i and j, and e_{ij} is the number of used long lines.

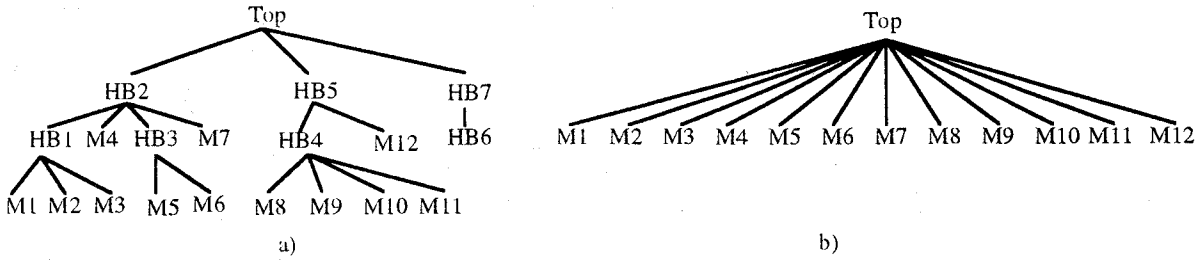


Figure 5. Initial design hierarchy: a) initial hierarchy tree; b) hierarchy tree reduced to macro blocks.

The hierarchical FPGA floorplanning problem consists in finding a feasible mapping of each block of a circuit and the quadrants of the target hierarchical structure while minimizing the timing cost.

4. Floorplanning approach description

4.1. Floorplanning phases

Taking into account all previous considerations, the floorplanning cycle steps are the following.

Step 1. Identifying macro blocks that will be preserved. Choose a hierarchical strategy - keeping the user hierarchy or flattening all non-macro hierarchy blocks.

Step 2. Calculation the circuit critical path and creating a fictive macro block containing instances which belong to the critical path.

Step 3. Recursively decompose the netlist into lower level quadrants until the leaf level quadrants are reached while minimizing cut nets number and respecting inter-quadrant routing resources limits. This step will determine which macro block will be assigned to which quadrant in the hierarchical FPGA layout.

The floorplanning process may iterate several times while feasible floorplan is obtained and timing constraints are met.

4.2. Recursive bisection

The described floorplanning problem may be resolved using a recursive netlist partitioning algorithm. The impact of the macro blocks preservation strategy on the bisection algorithm is the fact that the initial netlist contains cells of different granularity: small glue logic cells and big macro cells. The network is modelled as a hypergraph H=(V,E), where V is a set of nodes representing the macro blocks and glue logic cells, and E is a set of hyperedges which represents a set of nets.

The partitioning process follows the structural layout scheme: at each step the chip is divided through bipartitioning technique. Recursively, each subregion is further divided into four sub-regions. Using the hierarchical structure of the layout, the netlist is recursively dissected until each chip quadrant has an associated list of macro blocks.

Due to the fact that macro cells are treated as black boxes, being known their size and IO number, task complexity is significantly reduced. Macro cells require a special adaptation of the bisection method. When a cell move is performed in the iterative improvement algorithm, it is important that big macro cell move does not disbalance partitions.

As said before, to reduce the total system delay and to facilitate routing process, the number of connections between the quadrants at each level have to be minimised. On the other side, because routing depends on the placement and tight packing minimizes area but complicates routing, successful routing may be guaranteed if all hierarchical FPGA quadrants are uniformly filled. If a quadrant is saturated and others are badly filled, the routing in the first one may be difficult. To avoid this, the task is to create the uniformly saturated blocks. Or, in other words, empty space should not be minimized, like in the package partitioning, but uniformly distributed.

4.2.1. General partitioning procedure. Partitioning procedure may be realized like a classical F-M algorithm [FiMa82], [Kris84]. But the F-M algorithm is necessarily restrictive with respect to size requirement. More suitable to reflect two described upper objectives is the ratio-cut cost function of [YeCh91], which is estimated as:

$$R_{C(V_1,V_2)} = \frac{\left|C(V_1,V_2)\right|}{S(V_1)\times S(V_2)}$$

where $C(V_1,V_2)$ is a cut separating the nodes of the initial graph in two disjoint subset V_1 and V_2, $|C(V_1,V_2)|$ is the size of the cutset, and $S(V_1)$, $S(V_2)$ are correspondingly sizes of subsets V_1 and V_2 which are calculated as sums of sizes of belonging nodes in number of basic cells.

The implemented bipartitioning algorithm starts from the selection of two initial nodes: the first node is chosen arbitrarily, and the second one is a node having largest distance from the first one. Two passes searching for the best bipartition ratio are applied, similarly to [YeCh91], starting from each initial point. The bipartition providing the best ratio value is accepted as starting solution for iterative improvement step if its blocks satisfy quadrant size constraints.

The iterative improvement algorithm performs node moves based on the classical gain in number of cut nets. As solutions violating the quadrant size constraint are rejected, moves creating inacceptable partitions are disabled. Each time the move is performed, the current solution ratio is evaluated. The node to be moved is chosen according to the gain in cut nets, the criteria for the best solution is the ratio value.

An alternative is to perform node move on the base of the ratio gain: node producing best ratio gain is selected for move. The disadvantage is that gain value is not an integer number and traditional bucket structures may not be used. Every time the cell is moved, the ratio gain should be recalculated for all the cells, and the algorithm time complexity will be at least $O(c^2)$, where c is the number of cells in the netlist. However, the macro blocks preservation reduces the problem size by 50 to 90% and makes possible the application of such an algorithm. We have imposed on the base of experimental results a constraint on the initial problem size for use of ratio gain moves and use both cut set gain and ratio gain if the problem size does not exceed this limit or only the cut set size gain otherwise.

4.2.2. User hierarchy processing. Two options were explored concerning the user hierarchy processing during partitioning:

- to keep non-macro hierarchy blocks preserved as long as the quadrant size allows to contain them;

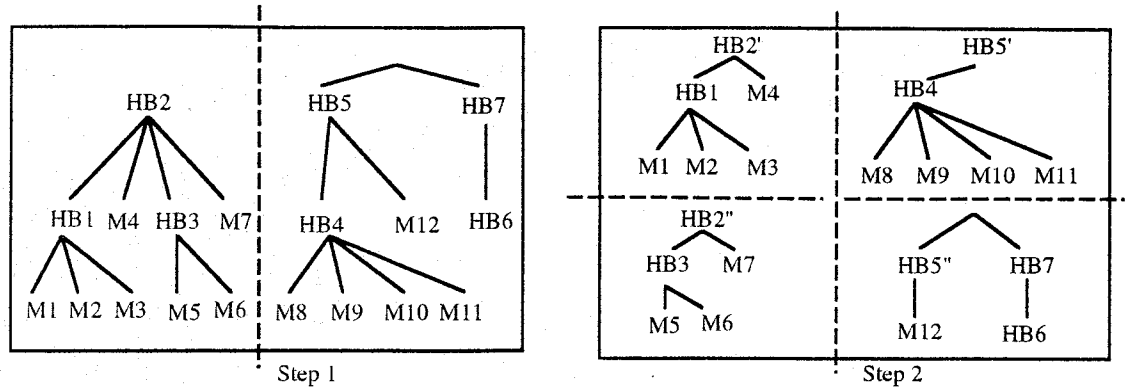


Figure 6. Hierarchy processing: hierarchy-driven method.

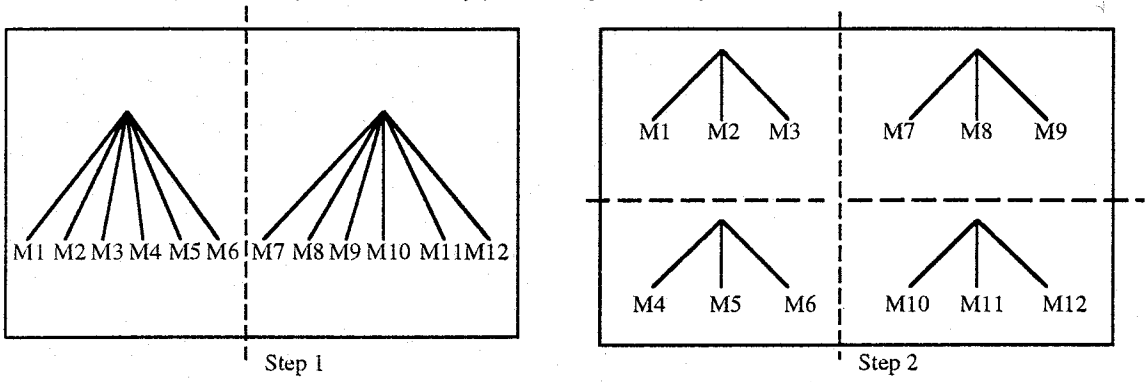


Figure 7. Hierarchy processing: macro-driven method.

- to flatten all non-macro blocks at the beginning of the partitioning, like shown in Figure 5b.

Both approaches from the initial example in Figure 5 are given in Figure 6 and Figure 7 respectively: (Step1) - the chip is split in two blocks and (Step2) - each block is split in its turn into two smaller quadrants.

In the first approach, the partitioning/placement problem complexity is reduced due to the preservation of all hierarchy blocks. But hierarchy that comes from the synthesis flow is not always suitable for floorplanning. Initial hierarchy is a logical grouping of modules. The layout hierarchy represents blocks neighbourhood and floorplanning should be based on the connectivity criteria. Thus, the floorplanning has not to pursuit initial circuit hierarchy in a mandatory way. In the second case the floorplanning/placement tool has as input a two-level netlist containing cells of different granularity: small glue logic cells and big macro cells. The second method is more flexible and is more suitable in the case when macro blocks are prevailing in the circuit. We have compared partitions quality of these two hierarchy processing methods and the results are presented in section 6.

4.2.3. Floorplanning solution quality. The quality of a floorplanning following the previous method has been evaluated according to three following characteristics:

(1) feasibility - F,
(2) timing cost - T,
(3) balancing cost - B.

Feasibility takes a value from the set (0,1). Timing cost of the solution is based on the delay of the different connection. If we use the target FPGA hierarchy described in Figure 2, the timing solution cost may be estimated as

$$T = \sum_{i=1}^{3}\sum_{j=i+1}^{4}\sum_{k=1}^{n} Q_{ij}^{k} \times t_k + \sum_{l=1}^{4}\sum_{i=l}^{3}\sum_{j=i+1}^{4}\sum_{k=1}^{n} q_{lij}^{k} \times t_k$$

where Q_{ij}^{k} is the number of connections of type k between 1st level quadrants i and j, and q_{lij}^{k} is the number of connections between the 2nd level quadrants i and j embedded into the first level quadrant l. We suppose also that there exist n different connection channels between each pair of quadrants, and the delay of k-th channel is denoted as t_k. The balancing cost of the current solution may be estimated as

$$B = f \times \prod_{i=1}^{4} S_i + \sum_{l=1}^{4}\prod_{i=1}^{4} s_{li},$$

where S_i is the size of i-th 1st level quadrant, s_{li} is the size of the i-th second level quadrant embedded in l-th 1st level quadrant, and f is the normalizing factor. The total solution cost is estimated then as

$$C = F \times \frac{B}{T}.$$

The initial mapping may be further improved by module moves between leaf nodes. The task of minimizing connections cost is similar to hierarchical tree partitioning problem described in [KuLiCh96].

5. Graphical hierarchical FPGA floorplan

Graphical hierarchical FPGA floorplan is a part of a synthesis environment. It performs FPGA partitioning, ASIC prototyping and hierarchical FPGA floorplanning. Along with the automatic partitioning/floorplanning tools it offers the commands to display - for viewing the hierarchical FPGA quadrants contents. Hierarchy display allows navigation in the hierarchy, interactive partitioning by foreassigning hierarchy blocks to quadrants, preserving and flattening hierarchy blocks.

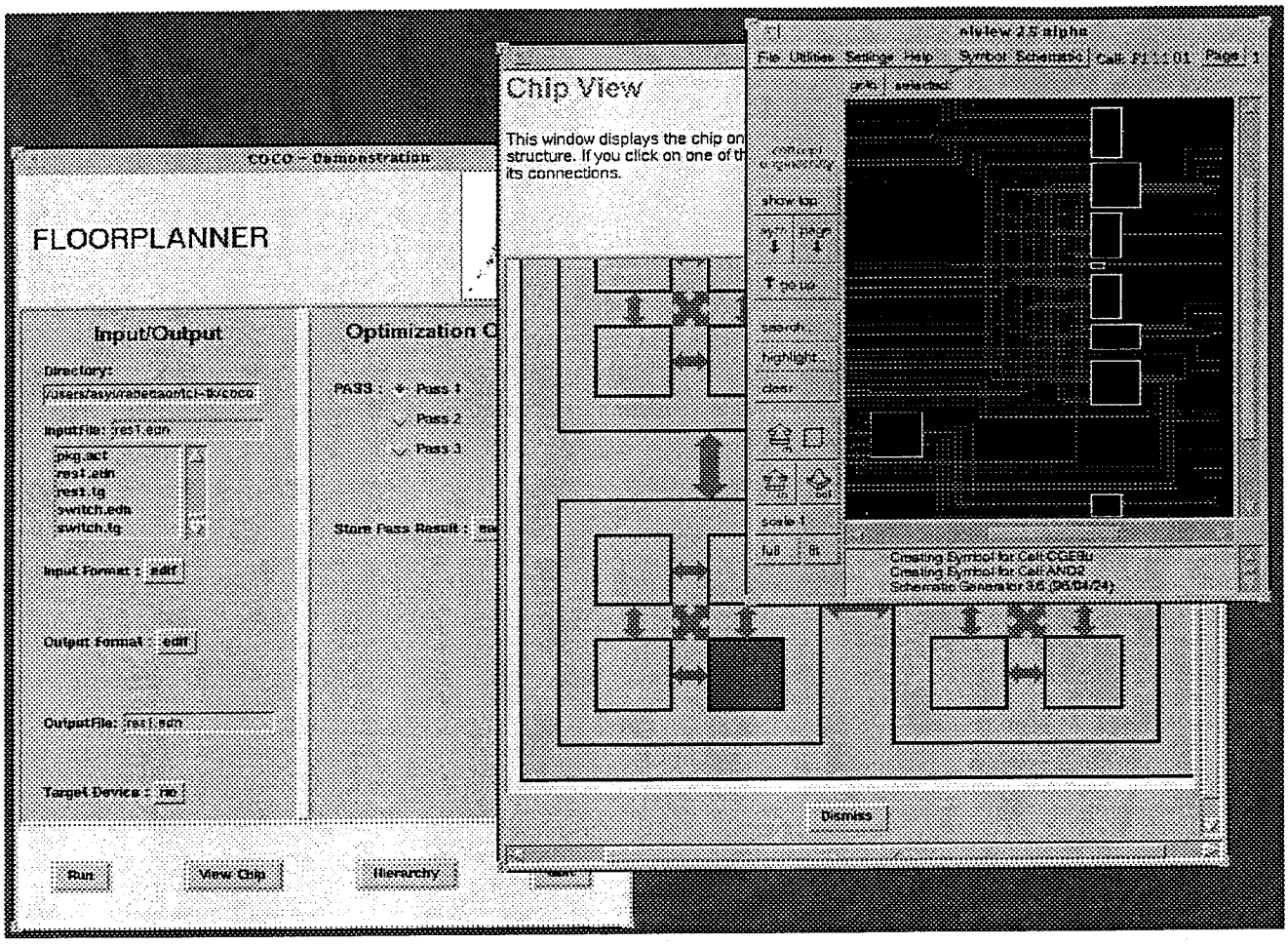


Figure 8. Floorplanning session screen view.

In Figure 8 the screen dump of the floorplanning session is presented. The netlist viewer displays preserved macro blocks (highlighted rectangles) and logic cells assigned to the highlighted in the chip view window second level quadrant.

6. Experimental Results

Floorplanning has been performed on a set of industrial benchmarks on ACTEL Eclipse SPGA family chips. The hierarchical FPGA target represented in Figure 2 was used for floorplanning. The following circuits have been used: (1) big industrial ASIC netlists migrated by ASYL+; (2) small netlists synthesized by Synopsys and by ASYL+ from VHDL descriptions using ACTGEN and ASYL+ MACRO inference. In Tables 1 and 3 the following benchmark characteristics are presented: the name of source ASIC technology, number of gates before migration to ACTEL Eclipse technology (for ASIC netlists), number of cells after migration/synthesis, number of IOs and flip-flops, the macro blocks identification strategies applied during floorplanning (listed in paragraph 2.2.1), number of macro block instances, and the percentage of macro blocks relatively to the total benchmark size.

In Tables 2 and 4 the floorplanning results are presented correspondingly for big and small circuits. For big circuits were used devices from ACTEL Reprogrammable SPGAs family ES (used device type is indicated) with capacity of 50-150K gates. For small benchmarks floorplanning results are presented on a hypothetical chip with maximal size of 1600 ACTEL Eclipse cells.

Floorplanning solutions obtained by different hierarchical strategies have been compared. In Tables 2, 4 are given the comparison results of hierarchical processing by (1) preservation of all the hierarchy blocks - HB (hierarchy blocks), (2) preservation only macro blocks - MB (macro blocks), (3) without hierarchy and macro blocks preservation - the netlist is flattened before the floorplanning starts - FLAT. The CPU time (on SPARC 20 station) and quality of solutions in terms of timing and balancing criteria were compared. For timing quality the following channel parameters were taken: (1) for the first level n=2, w_{11} = 32, t_{11} = 3ns, w_{12} = 64, t_{12} = 5ns; (2) for the second level n=3, w_{21} = 32, t_{21} = 1,2ns, w_{22} = 32, t_{22} = 1,5ns, w_{23} = 64, t_{23} = 1,8ns, and t_{II} = 6ns.

As shown in Tables 2,4, floorplanning on totally flat netlist for hierarchical design produces the worst result - it requires more CPU time, and the solution quality is worse. Preservation of macro blocks reduces the task complexity and allows partitioning to be performed more quickly. Reduction of the solution space permits to explore it better and produce smallest cutsets. Methods based on blocks preservation produce almost equal results: preservation of all hierarchy blocks is sometimes more rapid, but the method based on the preservation of only macro blocks produces sometimes better timing cost. Balancing quality depends on the percentage of macro blocks in the circuit. If some amount of glue logic is present in the netlist, it is more simple to equilibrate partitions by moving glue logic cells.

Table 1. Benchmark information - industrial ASIC netlists.

Bench-mark	Technology Source	# gates before migration	Size after migration, ACTEL Eclipse cells	#IO	#FF after migr.	Macro Identification type	#Macro Instan-ces	% Macro Blocks
Industrial1	FUJITSU cg24.lib	12420	2858	214	948	(4),(5),(6)	0	0%
Industrial2	cmos6.lib	46003	15071	88	5160	(4),(5),(6)	9	76%
Industrial3	FUJITSU cg31.lib	73320	24990	329	10369	(5),(6)	131	39%
Industrial4	FUJITSU cg31.lib	134532	42020	182	10742	(5),(6)	393	34%

Table 2. Experimental results of floorplanning with hierarchical FPGAs.

Benchmark	ACTEL Eclipse Chip	HB			MB			FLAT		
		CPU, sec	timing cost T,ns	bal. quality B	CPU, sec	timing cost T,ns	bal. quality B	CPU, sec	timing cost T,ns	bal. quality B
Industrial1	A65ES75	676	2290	2,07	8020	2214	1,47	8132	2214	1,47
Industrial2	A65ES50	2438	8237	1,47	8260	7589	0,99	24745	12813	1,02
Industrial3	A65ES150	11452	39732	1,41	22546	35784	1,85	45364	52671	1,62
Industrial4	A65ES150	12516	10584	1,03	16784	15478	1,54	47451	22478	0,98

Table 3. Benchmark information - small sized circuits synthesized from VHDL.

Benchmark	Synthesised by	Size, ACTEL Eclipse cells	#IO	#FF	Macro Identi-fication type	#Macro Instances	% Macro Blocks
Industrial5	Synopsys	1104	123	217	(5),(6)	37	78%
Industrial6	Synopsys	1476	105	381	(5),(6)	105	79%
Industrial7	Synopsys	1306	124	527	(5),(6)	61	40%
Industrial8	ASYL+	738	92	182	(1),(5)	5	2%
Industrial9	ASYL+	1512	131	428	(1),(5)	1	3%
Industrial10	ASYL+	732	35	163	(1),(5)	4	11%
Industrial11	ASYL+	1537	168	393	(4),(5),(6)	2	67%
Industrial12	Synopsys	1587	119	391	(4),(5),(6)	23	100%

Table 4. Experimental results of mapping netlists to hierarchical FPGA structure.

Benchmark	HB			MB			FLAT		
	CPU, sec	timing cost T,ns	bal. quality B	CPU, sec	timing cost T,ns	bal. quality B	CPU, sec	timing cost T,ns	bal. quality B
Industrial5	87	1079	1,02	174	1011	0,96	556	1678	0,58
Industrial6	123	745	2,17	135	930	2,14	554	573	1,33
Industrial7	98	1524	2,13	321	1316	1,51	648	1804	1,35
Industrial8	73	864	0,32	165	333	0,18	212	352	0,17
Industrial9	577	1786	2,94	510	2254	2,08	721	1754	2,20
Industrial10	31	762	1,01	106	574	0,26	163	557	0,26
Industrial11	1024	1225	1,40	1138	1450	2,18	1569	665	1,60
Industrial12	113	526	2,48	119	526	2,48	1999	1308	1,84

7. Conclusion

In this paper we addressed the hierarchical FPGA floorplanning problem. We have described the synthesis and floorplanning general flow, new hierarchical targets and new design methods based on the macro inference in HDL flow. The proposed hierarchical FPGA floorplanning method is based on the containment of the macros within an appropriate area in the chip. This is the key point for timing optimization. We have proposed also the comparison results of floorplanning solution quality using different hierarchical strategies.

8. References

[FiMa82] C. M. Fiduccia, R. M. Mattheyses, "A Linear Time Heuristic For Improving Network Partitions", DAC-82, pp. 175-181.

[Kris84] B. Krishnamurthy, "An Improved Min-Cut Algorithm For Partitioning VLSI Networks", IEEE Trans. On CAD, vol. CAD'33, no. 5, May 1984, pp. 438-446.

[KuLiCh96] Ming-Ter Kuo, Lung-Tien Liu, Chung-Kuan Cheng, "Network partitioning into tree hierarchies", 33rd DAC ACM/IEEE, pp. 477-482, 1996.

[SuShRo91] S. Suthantavibul, E. Shragowitz, and B. Rosen, "An Analytic Approach to Floorplan Design and Optimization", IEEE Transactions on Computer-Aided Design, vol. 10, pp. 761-769, June 1991.

[ViTs91] G. Vijayan, R. Tsay, "A New Method for Floorplanning Using Sizing and Enumeration", IEEE Transactions on Computer-Aided Design, vol. 10, pp. 1494-1501, December 1991.

[WaWo93] T.-C. Wang, D.F. Wong, "Graph-based techniques to speed up floorplan area optimization", INTEGRATION, the VLSI journal 15 (1993), pp. 179-199

[YeCh91] C. W. Yeh, C. K. Cheng, "A General Purpose Multiple Way Partitioning Algorithm", 28th DAC ACM/IEEE, pp. 421-426, 1991.