A Design Workflow for Dynamically Reconfigurable Multi-FPGA Systems

Alessandro Panella*, Marco D. Santambrogio†‡, Francesco Redaelli†, Fabio Cancare†, and Donatella Sciuto†
* Computer Science Department - University of Illinois at Chicago, Email: apanel2@uic.edu
† Dipartimento di Elettronica e Informazione (DEI) - Politecnico di Milano,
Email: {santambr, cancare, fredaelli, sciuto}@elet.polimi.it
‡ Computer Science and Artificial Intelligence Laboratory - Massachusetts Institute of Technology, Email: santambr@mit.edu

***Abstract*—Multi-FPGA systems (MFS's) represent a promising technology for various applications, such as the implementation of supercomputers and parallel and computational intensive emulation systems. On the other hand, dynamic reconfigurability expands the possibilities of traditional FPGAs by providing them the capability of adapting their functionality while still running to cope with runtime environment changes. These two research directions are merged together in this work, that describes a methodology for designing dynamic reconfigurable MFS's. In this paper a novel MFS design flow has been described, which makes use of blocks reuse through dynamic reconfigurability to make the implementation of large systems feasible even on multi-FPGA architectures with strict physical constraints. Functional to this goal is the development of an algorithm for the extraction of the isomorphic structures of a circuit that extensively exploits the hierarchy of the design.**

I. Introduction

The use of Field Reprogrammable Gate Arrays (FPGAs) is nowadays widespread in both industry and academic research. Their computational power can be increased through the creation of clusters of chips. Besides obviously augmenting the available physical area, this also provides the possibility of massively exploiting parallel computation. Such *multi-FPGA systems* (MFS's) are currently used in supercomputing applications and logic emulation of custom circuits [1]. A computational paradigm attracting growing interest is *reconfigurable computing* (RC). An early definition given by Gerald Estrin refers to RC as the *process of altering the location or the functionality of a system element, as a response to faults, changes in the environment or explicit application needs* [2]. Due to their reprogrammability, FPGAs currently represent the leading technology for implementing reconfigurable systems. In recent years, the evolution of FPGA architectures has made it possible to further increase the degree of flexibility in the use of such chips. This innovation is represented by the possibility of having parts of the FPGA reconfigured at run-time, while others are still running, so that the execution of the system never ceases. This technique is called *partial dynamic reconfigurability*, as opposed to the standard *static reconfigurability*. A number of works about MFS design can be found in literature (e.g. [1], [3]–[5]), but only few approaches have been proposed that explore the field of dynamically reconfigurable MFS's ([6], [7]). Merging together the potential of MFS's and reconfigurability is nevertheless a promising research direction. Although the area available on MFS's is usually large, some complex applications may require even more space, thus imposing the replacement of the physical architecture with a larger one, a process that is very expensive and time-consuming. By providing a larger *virtual area*, dynamic reconfigurability allows to go beyond the physical space constraints of the architecture [8]. The presented work proposes a novel design methodology that exploits the dynamic reconfigurability of interconnections in MFS's. This allows design blocks of the application to be used more than once during execution, with the result of significant area savings. At the best of our knowledge, no other work on multi-FPGA design has explored this scenario.

The remainder of this paper is organized as follows. Sections II, III, and IV present the proposed MFS design workflow and describe in details the three phases it is composed of. Section V provides the obtained experimental results, while Section VI briefly reports previous works on MFS design, comparing them with the proposed methodology. Section VII concludes the paper providing some hints for future work.

II. Proposed Workflow and Design Extraction

The approach proposed in this paper consists of a workflow for the design of MFS's, whose abstract view is represented in Fig. 1 and is briefly described in the following.

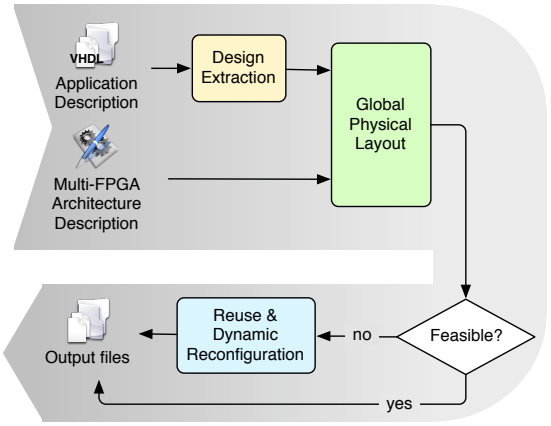

Figure 1. Outline of the proposed multi-FPGA systems design workflow.

The input of the design process consists of a VHDL description of the application and a specification of the target

multi-FPGA architecture. The VHDL code undergoes a design extraction phase, which aims at collecting the information relevant to the design structure. A global physical layout phase performs the partitioning, placement and routing of the application on the specified architecture. At this point, two situations are possible. If the application fits into the architecture, the flows ends. Otherwise, another step is undertaken, aimed at exploiting the dynamic reconfiguration of the communication infrastructure for modules reuse. The output of the workflow is a new VHDL specification, describing the modules to be instantiated on each FPGA, together with information about the reconfiguration of interconnections. This process relies on existing commercial tools (e.g. Xilinx ISE) for subsequent intra-FPGA synthesis.

A. Intermediate Representation

The VHDL specification received as input is parsed and interpreted, and the result is saved in a specifically designed intermediate representation, that maintains information both on the structure and the *hierarchy* of the design tree-like data structure. Hierarchy information is useful to our task for two important reasons. The first one depends on what makes the designer choose a particular design hierarchy when implementing the VHDL application. The designer follows some simple implicit rule in recursively aggregating – or splitting down – components. Design blocks are built based on their functionality: if two sub-components (*children*) carry out operations that are theoretically portions of a larger function, they are likely to be aggregated in a bigger component (*parent*). It is evident that such sub-components are probably strongly interconnected. Therefore, it seems natural and favorable to exploit this information in the partitioning, placement and routing of the input circuit. The second reason is rooted in the concept of regularity: if two components belong to the same *type*, they are roots of two identical sub-trees in the hierarchy. Therefore, when a given operation is carried out during the execution of some algorithm in one of these subtrees, it can be immediately replicated in the other one.

B. VHDL Preprocessing and Extraction

The extraction phase is composed of two steps. First, the VHDL specification is preprocessed to reduce it to a pure VHDL *structural* description. The resulting code contains only structural statements for the intermediate nodes of the hierarchical tree, while *behavioral* and *data-flow* instructions are allowed exclusively in leaf blocks. To obtain a pure structural description, two operations are carried out for each component in the design:

1) For every process, a leaf component is created which contains the process. The process in the original file is replaced by the instantiation of this component.
2) All data-flow instructions are turned into a leaf component, and are replaced by the instantiation of such component.

Then, the specification is parsed into the intermediate representation. The estimated FPGA area occupation in number of slices is retrieved by this step, using existing FPGA synthesis tools such as Xilinx XST [9]. Each leaf of the extracted hierarchical tree is then constituted by a single VHDL process or a group of data-flow instructions. The granularity of this structure is quite coarse, especially if compared to usual gate-level netlists. The choice of handling the circuit at a process-level granularity arises from the fact that the presented workflow performs a *global* mapping of the application on a multi-FPGA architecture, with subsequent phases taking care of fine-grained local syntheses. In this context, dealing with a low number of relatively large design modules leads to faster results.

III. Global Physical Layout

The *global layout* phase deals with the search of a feasible mapping of the parsed application on a multi-FPGA architecture received as input, while optimizing some objectives, i.e. the interconnections length. Such mapping assigns one and only one host FPGA of the architecture to each leaf block of the input application and routes interconnections between any two communicating modules assigned to different chips. The cost function to be minimized is the estimated length of the interconnections between blocks assigned to different FPGAs, measured in number of hops and weighted over connections width. Let us define $w(i,j)$ as the amount of communication in number of bits between nodes i and j and let c_i identify the FPGA node i is assigned to. The cost function to be minimized is the *Weighted Estimated Wire Length* (WEWL), computed as follows:

$$WEWL = \sum_{1 \leq i < j \leq n} w(i,j) d(c_i, c_j)$$

where n is the number of nodes in the architecture, $w(i,j)$ is the size in bits of the interconnection between nodes i and j, and $d(c_i, c_j)$ is the estimated distance between FPGAs c_i and c_j in the architecture. Off-chip wires are undesirable since ([10]): they degrade performances, constitute a source of faults, and increase the need of I/O pins. In this paper, only global partitioning and placement are addressed. Global partitioning aims at creating partitions of leaf design nodes such that their size is not bigger than the area available on the FPGAs composing the architecture and the cut-size is minimized. Placement generates a one-to-one mapping between the created partitions and the FPGAs that minimizes interconnection length. The partitioning algorithm is a *bottom-up clustering* that exploits the regularities extracted from the design hierarchy. At the beginning, each leaf of the design hierarchy is considered as a cluster and is assigned a *type*, given by the VHDL component the node is instance of. Then, the two clusters maximizing a given closeness metric are collapsed together, provided this does not violate maximum area and I/O pin count constraints. Let us define as $B = \{1, 2, ..., N\}$ the set of all leaves of the design hierarchy, $\mathcal{P} = \{P_1, P_2, ..., P_M\}$ a partitioning over B. Being P and $Q \in \mathcal{P}$, the metrics considered by the algorithm are:

- *Connection* (CONN): volume of communication between

two clusters (in bits).

$$CONN(P,Q) = \sum_{i \in P, j \in Q, i<j} w(i,j)$$

- *Communication Ratio* (CR): ratio between the communication volume internal to the resulting cluster (*Internal Communication* – IC) and the communication volume with other clusters (*External Communication* – EC).

$$IC(P,Q) = \sum_{i,j \in P \cup Q, i<j} w(i,j)$$

$$EC(P,Q) = \sum_{i \in Q \cup P, j \in B \setminus (P \cup Q), i<j} w(i,j)$$

- *Communication Density* (CD): ratio between the Internal Communication and the number of edges of an hypothetical complete graph built on the resulting cluster.

$$CD(P,Q) = \frac{IC(P,Q)}{CliqueSize(P \cup Q)}$$

Two possible cases can arise when two clusters are collapsed:

1) If the two clusters belong to the same parent in the hierarchy, other instances of the parent's type are searched in the hierarchy to apply the same transformation. The same type is assigned to these newly created clusters. In other words, a collapse operation *induces* other ones.
2) If the two clusters do not belong to the same parent, the newly created cluster is added as a child of the root node, being such cluster unique and surely not involved in any regularity patterns.

This schema, exemplified in Gif. 2, is iterated until no more clusters can be formed or only one cluster remains. Throughout the algorithm, intermediate hierarchy nodes with a single child are dropped. Notice that the use of the hierarchy information overcomes a traditional problem of clustering algorithms, represented by the locality of closeness metrics. The placement step is essentially a one-to-one mapping of the

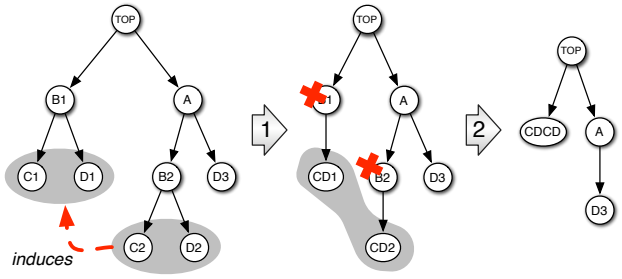

Figure 2. Two iterations of the clustering algorithm.

generated partitions on the FPGAs of the target architecture. When dealing with a small number of chips, an optimal solution can be easily found. In general, a topology-independent iterative method can be considered. A simulated annealing (SA) approach has been developed and tested, whose objective function is the expected wire length needed for inter-FPGA communication. If partition K is assigned to chip c_K, then this function is $\sum_{I,J \in \mathcal{P}} d(c_I, c_J) CONN(I,J)$, where $d(\cdot,\cdot)$ is the distance in number of hops between two chips,

IV. Reuse and Dynamic Reconfigurability

The attempt to find a static global layout may fail, due to the bounded area available on the architecture. In such a case, a design blocks reuse technique is adopted. Consider a *dynamically interconnected* circuit structure where the nets connecting the blocks can be added and dropped at run-time. In this scenario, a block can be connected to more than one net in non-overlapping time intervals. In this way two or more identical parts of the application can be implemented by a single block with dynamic interconnections. A *crossbar* topology can be used to implement this kind of circuit: the reprogrammable switch-boxes in the crossbar chip can be dynamically reconfigured to implement temporary connections, as shown in Fig. 3. Other multi-FPGA architectures (e.g. bus-based) can be considered as well. The problem of design parts

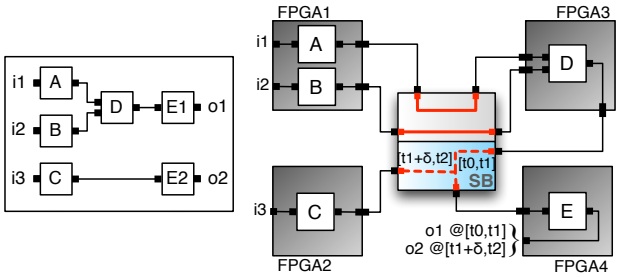

Figure 3. Input structure (a) and possible crossbar implementation (b).

reuse can be decomposed in two sub-problems: (1) what are the *parts* of the design to consider, and (2) *which ones* of those should be reused. The proposed solution to these problems is presented in the following paragraphs.

A. Isomorphic Structures

One straightforward answer to the first question is to consider each leaf in the design hierarchy as a potentially reusable part. This is not satisfying, because it leaves out any repeated pattern constituted by more than one atomic block. The concept of isomorphic structure comes to help. **Definition: Isomorphic Structures.** *Two structures C_1 and C_2 are* isomorphic *if for each block contained in C_1 there exists one block in C_2 of the same type (and vice-versa), and the interconnections among the blocks are identical.* (As a particular case, two atomic blocks having the same type are isomorphic). Consider every isomorphic structure in the design represents the perfect answer to our question. Unfortunately, the identification of all isomorphisms is known from graph theory to be an NP-complete problem, therefore the proposed technique aims at finding *some* isomorphic structures in the input circuit. These structures are provided by the clustering algorithm described in the previous section: clusters belonging to the same type are isomorphic structures.

B. Blocks Reuse Choices

Although the reuse of blocks causes a beneficial reduction of the amount of area required for implementing the circuit, it also implies extra execution time, necessary to carry out

the reconfigurations. The problem to be solved is therefore to find a blocks reuse strategy which allows the input application to fit in the architecture while minimizing the required reconfiguration time. Complicating the problem is the fact that isomorphic structures are in general overlapping: this introduces mutual constraints to be fulfilled when choosing which blocks to instantiate. In spite of that, the isomorphic structures extracted by the clustering algorithm have a peculiar nature: given two clusters, either one contains the other either they do not overlap. This is because the algorithm generates a clusters hierarchy, that can be viewed as a dendrogram, as the one shown in Fig. 4. This context implies that any cut of the

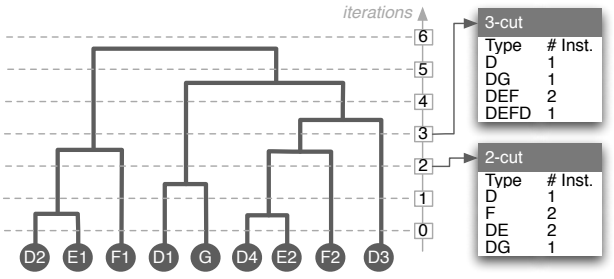

Figure 4. Example of extraction of horizontal cuts.

dendrogram represents a full, flat, *typed* specification of the input application. The number of possible cuts is exponential in the number of initial clusters, therefore only the subset given by horizontal cuts is considered by the proposed methodology: the reuse problem is individually solved for each of these cuts, and the best solution is then returned. Being $n(c_i)$ the number of occurrences of cluster type $c_i \in C$, a solution to the problem is represented by a function $m(c_i) : C \rightarrow \{1, 2, ..., n(c_i)\}$, which represents the number of instances of cluster type c_i in the resulting dynamically interconnected structure. In a partial crossbar topology as the one described above, the reconfiguration time related to a cluster can be estimated by considering that it is with good approximation proportional to the area that has to be reconfigured. In turn, such area is proportional to the width of the external interconnections of the cluster, denoted as $w(c_i)$. This quantity has to be multiplied by the number of reconfigurations implied by the solution, equal to $n(c_i) - m(c_i)$. We can conclude that the actual time needed for the reconfigurations is proportional to the following quantity, that therefore has to be minimized:

$$T_{rec} \propto \sum_{c_i \in C} \{[n(c_i) - m(c_i)] * w(c_i)\}. \quad (1)$$

The area occupied by the resulting system has to be smaller than the overall capacity of the architecture A. Mathematically, this constraint is expressed as $\sum_{c_i \in C} [a(c_i) * m(c_i)] < A$. Despite being the problem NP-complete, an Integer Linear Programmig (ILP) model is simply obtainable from these formulae and it has shown to run in acceptable time (see Section V). Considering again the nature of the isomorphic structures that are considered for reuse, obtained by a clustering process that tries to minimize the amount of external communication. Combined with (1), this fact implies that these structures are *good* with respect to the goal of minimizing the reconfiguration time.

V. Experiments and Case Study

Four VHDL test circuits have been used for validating the global layout algorithms proposed in this paper: an encryption/decryption core (3DES), a Finite Impulse Response filter (FIR), a cypher (NOEK), and a combination of the first two (3DES+FIR). Quantitative information of these circuits are reported in Table I.

Table I
Test designs characteristics.

Circuit	3DES	FIR	NOEK	3DES+FIR
Size (slices)	1613	561	958	2141
# Nodes in hierarchy	67	231	29	301
# Leaves	52	211	25	264
Leaves size (slices)				
Mean	19.21	2.66	38.32	8.11
Std. Dev.	28.5	4.94	72.45	27.36

Table II reports some numerical results[1] of the execution of the MFS design workflow proposed in this paper. In particular, results are shown for partitioning using the three different clustering metrics that have been introduced in Section III: Connection, Communication Ratio, and Communication Density. For carrying out experiments on the test circuits explained above, three hypothetical FPGA dimensions have been considered: 300, 400, and 600 slices. The partitioning quality is tested against the results obtained by using the MeTiS partitioning algorithm. For carrying out this comparison, the cutsize of the resulting partitioning is considered, which is the amount of communication – in number of bits – among different partitions. Moreover, the results for the one-to-one placement and for solving the instances of the ILP model for blocks reuse are listed in the table.

The table shows that the Connection metric (CONN) for clustering leads to smaller cutsizes in the majority of the cases, although it sometimes implies the use of one additional partition. The time required for partitioning is reasonably low, even if it grows more than linearly with the number of nodes in the design hierarchy. These results are compared with the ones obtained using the MeTiS partitioning algorithm ([11]), which currently represents the state-of-art for partitioning large flat netlists. In some cases the proposed clustering algorithm behaves better than MeTiS. This can be noticed to happen for circuits whose leaves size has a high variance-mean ratio (i.e. 3DES and 3DES-FIR), while it is not true circuits whose structure is more similar to a typical flat netlists. This shows that the proposed approach is promising, as it provides good results in partitioning hierarchical structures extracted from VHDL with large and irregular blocks dimensions. The one-to-one placement algorithm has been tested on 4-mesh multi-FPGA topologies. The running times are acceptable, and grow roughly linearly as the number of partitions increases.

The ILP model for computing the best solution in clusters reuse has been solved by actually considering a maximum

[1] All tests have been carried out using an Intel Core 2 Duo 2.2 GHz machine.

Table II
EXPERIMENTAL RESULTS FOR THE PROPOSED METHODOLOGY AND COMPARISON WITH THE METIS PARTITIONING ALGORITHM.

Circuit			**3DES**			**FIR**			**NOEK**			**3DES+FIR**		
Partition Size (slices)			300	400	600	300	400	600	300	400	600	300	400	600
Partitioning (Clustering)														
METRIC	*CONN*	*Cutsize*	547	550	349	36	50	0	1965	2061	1314	610	620	434
		# Partitions	7	5	3	2	2	1	4	3	2	9	6	4
		Time (ms)	18	17	19	970	952	974	3	3	3	2034	2019	1990
	CR	*Cutsize*	1604	1296	1193	130	197	0	2672	2478	1643	1371	1482	1344
		# Partitions	6	5	3	2	2	1	4	3	2	8	6	4
		Time (ms)	13	13	13	639	663	660	2	2	2	1165	1255	1261
	CD	*Cutsize*	915	692	417	111	53	0	1965	1995	1506	1348	1001	976
		# Partitions	6	5	3	3	2	1	4	3	2	8	6	4
		Time (ms)	13	13	14	337	334	342	2	2	2	1187	1082	1167
Partitioning w/ MeTiS - *Cutsize*			577	1536	1335	27	27	0	2128	1518	1421	2148	2826	1683
1-to-1 Placement		*WEWL*	1224	621	494	36	50	0	2287	2061	1314	2099	775	753
		Time (ms)	1335	789	540	2	2	1	655	528	2	1658	892	753
ILP Model Solving		*# Runs*	42	34	46	186	186	187	22	23	22	228	229	230
		Time (ms)	420	492	474	2103	2142	2078	2173	2325	2296	2763	2685	2753

area equal to the 80% of the one actually available on the architecture. This is a common technique used to counter-balance possible estimation errors. It can be seen that, even when the number of executions is high, the running time is reasonably low. A more detailed analysis shows that such time depends on both the size of the circuit and the number of required executions of the solver, which is equal to the number of iterations performed by the clustering algorithm (i.e. the depth of the dendrogram), that in turn is bounded by the number of leaves in the design hierarchy. In order to show how the proposed workflow works in practice, a sample case study is provided. The user needs to deploy a parallel JPEG decoder composed by two identical decoding modules on a multi-FPGA architecture. The estimated overall size of the application is 3978 slices, with a design hierarchy composed of 174 nodes, 162 of which are leaves. The architecture used within these experiments is composed by Xilinx XC3S100E[2] devices ([12]), due to their low power and low costs characteristics, each with an available area of 960 slices and 108 I/O pins. The best result is obtained by partitioning the application using the Connection metric, that produces five partitions with a cutsize of 198. The METIS algorithm gives a cutsize equal to 388.

Consider now a scenario where the multi-FPGA system under exam deploys hardware applications on-demand: it is likely that a certain sequence of received requests limits the current available area. Within this environment, assume that the total available area is bounded to 3000 slices. The blocks reuse methodology is then applied in this case. Every typed circuit specification resulting from horizontal cuts of the clustering dendrogram is given as input to the ILP model solver. The estimated reconfiguration times obtained from the executions of the ILP solver are shown in Fig. 5. The lowest reconfiguration time is obtained from cuts 14 to 18 of the dendrogram. The corresponding ILP solutions suggest two structures to be used twice during execution. One is a leave of the design hierarchy, while the other is a structured component that results from some collapsing operations in the clustering algorithm.

[2]The proposed approach has not been thought and implemented to target just one specific FPGA devices. Using different devices would only mean having different constraints and resources availability

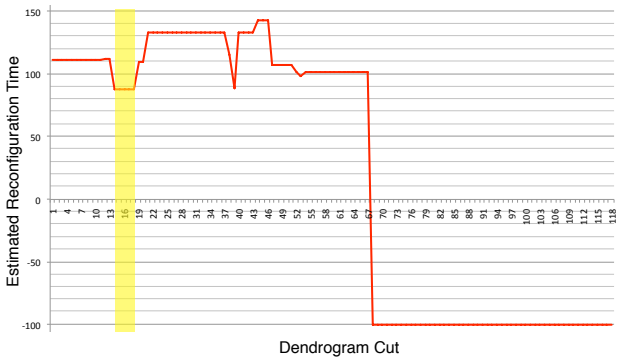

Figure 5. Case study: estimated reconfiguration time for different dendrogram cuts. The value -100 represents the impossibility to find a solution to the ILP model.

VI. RELATED WORKS

The design of MFS's is addressed in several works. In [13], [14], the authors proposed a the *Virtual wires* approach to overcome pin limitations by intelligently multiplexing each physical wire among multiple logical wires. Hauck ([1], [15]) provides a complete workflow for the design of MFS's, by proposing an integrated methodology for global partitioning, placement, and routing. The application to be mapped is recursively bi-partitioned using, in each iteration, a multilevel algorithm based on the Fiduccia-Mattheyses (FM) heuristic ([16]). The recursive bi-partitioning is driven by *partitioning orderings*: the first two obtained partitions are placed on the two least connected portions of the architecture, and so on recursively. This technique also implicitly provides the global routing of the application on the architecture.

Khalid's work ([3], [17]) focuses on the evaluation of different MFS topologies rather than on the performances of the adopted algorithms. Nevertheless, he proposes a complete MFS design workflow, in which the three global layout steps are carried out sequentially. For partitioning the input application, Khalid uses a simple recursive FM bi-partitioning algorithm. Global placement is executed differently depending on the architectural topology; for instance, a force directed approach is used for mesh topologies. To cope with the routing problem, a general topology-independent approach based on graph structures is proposed, as well as several algorithms tailored to specific topologies, that provide better results.

A direct comparison between these approaches and the proposed methodology can be carried out in terms of partitioning results. Both Hauck and Kahlid use derivations of the FM heuristic, in a multilevel process and in recursive bi-partitioning, respectively. In Section V comparisons with METIS are shown. Being a multilevel partitioning algorithm that also uses FM as a baseline, METIS represents a progress with respect to recursive FM and has been reported to outperform Hauck's approach ([18]).

Other approaches focus on parts of the MFS design flow. In [19] the authors propose a genetic algorithm with fuzzy fitness function for MFS partitioning and placement targeted to 4-mesh topologies. [20] use simulated annealing to cope with global partitioning and placement. Iterative techniques seem indeed suitable to cope with high dimensional and complex problems such as global layout. Works like [21] and [4] show the advantages of exploiting the design hierarchy of the application, instead of using a flat netlist representation, by providing heuristic algorithms that traverse the hierarchy structure. In [22] the authors include an HDL synthesis step in the MFS design flow: a Verilog description is analyzed and turned into a hierarchical tree, and a top-down set covering algorithm is applied to generate partitions.

The approaches presented so far in this section deal with the design of MFS without taking into account dynamic reconfiguration. Instead, in [6] the authors propose a partitioning and synthesis process claimed to generate dynamically reconfigurable MFS's. The input specification is transformed into a directed task-graph, which is divided in time segments (*temporal partitioning*). Then, a binary non-linear programming model performs a *spatial partitioning* over the FPGAs of the architecture for each time segment. At run-time, after a time segment is completed, execution is stopped and all the FPGAs are reconfigured, with temporary results stored in memory. Clearly, this is approach is not "dynamic" in the sense described in section I. In order to be truly dynamic, a MFS must be partially reconfigured, so that reconfiguration times are masked and the execution never ceases.

This is exactly what the methodology described in the present work proposes, by introducing partial dynamic reconfigurability in a standard MFS design workflow. In particular, inter-FPGA interconnections are *partially* reconfigured with the goal of saving area through blocks reuse, without requiring the execution of the system to cease.

VII. Conclusions and Future Works

In this paper a novel MFS design flow has been described, which makes use of blocks reuse through dynamic reconfigurability to make the implementation of large systems feasible even on mutli-FPGA architectures with strict physical constraints. Experimental results have been provided in Section V to validate the proposed methodology. Among the others, a remarkable novelty is the exploitation of the design hierarchy both producing good partitioning results and extracting isomorphic structures. Future work will deal with the improvement of the clustering algorithm by adopting more powerful clustering metrics and developing solutions to go beyond its intrinsic greediness. An algorithm for scheduling the reuse of components has to be developed, along with an effective routing methodology. Finally the proposed approach will be used in conjunction with the virtual wires approaches that has been proven [13] not to need for expensive crossbar technology while increasing FPGA utilization.

References

[1] Scott Hauck. *Multi-FPGA Systems*. PhD thesis, University of Washington, 1995.

[2] G. Estrin. Organization of Computer Systems–The Fixed Plus Variable Structure Computer. *Proc. Western Joint Computer Conf., Western Joint Computer Conference, New York*, pages 33–40, April 1960.

[3] M. Khalid. *Routing Architecture and Layout Synthesis for Multi-FPGA Systems*. PhD thesis, University of Toronto, 1999.

[4] H. Krupnova, A. Abbara, and G. Saucier. A Hierarchy-Driven FPGA Partitioning Method. *Proceedings of the 34th annual conference on Design automation conference*, pages 522–525, 1997.

[5] Ranieri Baraglia, Raffaele Perego, J. Ignacio Hidalgo, Juan Lanchares, and Francisco Tirado. A Parallel Compact Genetic Algorithm for Multi-FPGA Partitioning. *pdp*, 00:113, 2001.

[6] I. Ouaiss, S. Govindarajan, V. Srinivasan, M. Kaul, and R. Vemuri. An Integrated Partitioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures. *IPPS/SPDP*, pages 31–36, 1998.

[7] Vincenzo Rana, Marco D. Santambrogio, Donatella Sciuto, Boris Kettelhoit, Markus Köster, Mario Porrmann, and Ulrich Rückert. Partial Dynamic Reconfiguration in a Multi-FPGA Clustered Architecture Based on Linux. In *IPDPS*, 2007.

[8] Eylon Caspi, Michael Chu, Randy Huang, Joseph Yeh, John Wawrzynek, and André DeHon. Stream computations organized for reconfigurable execution (score). In *FPL '00: Proceedings of the The Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications*, pages 605–614, London, UK, 2000. Springer-Verlag.

[9] Xilinx, Inc. *XST User Guide*.

[10] Sadiq M. Sait and Habib Youssef. *VLSI Physical Design Automation*. Number 6 in Lecture Notes Series on Computing. World Scientific Publishing, 1999.

[11] George Karypis and Vipin Kumar. Multilevel k-way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.

[12] Xilinx, Inc. *Spartan-3E FPGA Family: Complete Data Sheet*, 2008.

[13] Jonathan Babb, Russell Tessier, Matthew Dahl, Silvina Zimi Hanono, David M. Hoki, and Anant Agarwal. Logic emulation with virtual wires. pages 625–642, 2002.

[14] J. Babb, R. Tessier, M. Dahl, S.Z. Hanono, D.M. Hoki, and A. Agarwal. Logic emulation with virtual wires. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(6):609–626, 1997.

[15] Scott Hauck. The roles of FPGAs in reprogrammable systems, 1998.

[16] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *DAC '82: Proceedings of the 19th conference on Design automation*, pages 175–181. IEEE Press, 1982.

[17] M.A.S. Khalid and J. Rose. Experimental Evaluation of Mesh and Partial Crossbar Routing Architectures for Multi-FPGA Systems. *IFIP IWLAS97, Grenoble, France*, pages 119–127, 1997.

[18] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. *IEEE Transactions on Very Large Scale Integration Systems*, 7(1):69–79, 1999.

[19] JI Hidalgo, J. Lanchares, and R. Hermida. Partitioning and Placement for Multi-FPGA Systems Using Genetic Algorithms. *Euromicro Conference, 2000. Proceedings of the 26th*, 1, 2000.

[20] Juan de Vicente, Juan Lanchares, and Román Hermida. Placement Optimization Based on Global Routing Updating for System Partitioning onto Multi-FPGA Mesh Topologies. In *FPL '99: Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications*, pages 91–100, London, UK, 1999. Springer-Verlag.

[21] Dirk Behrens, Klaus Harbich, and Erich Barke. Hierarchical partitioning. In *ICCAD '96: Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 470–477, Washington, DC, USA, 1996. IEEE Computer Society.

[22] Wen-Jong Fang and Allen C.-H. Wu. Multiway FPGA partitioning by fully exploiting design hierarchy. *ACM Trans. Des. Autom. Electron. Syst.*, 5(1):34–50, 2000.