Multi-Way FPGA Partitioning by Fully Exploiting Design Hierarchy

Wen-Jong Fang and Allen C.-H. Wu
Department of Computer Science
Tsing Hua University
Hsinchu, Taiwan, 300, Republic of China
{chunghaw@cs.nthu.edu.tw}

Abstract

In this paper, we present a new integrated synthesis and partitioning method for multiple-FPGA applications. This method first synthesizes a design specification in a fine-grained way so that functional clusters can be preserved based on the structural nature of the design specification. Then, it applies a hierarchical set-covering partitioning method to form the final FPGA partitions. Our approach bridges the gap between HDL synthesis and physical partitioning by fully exploiting the design hierarchy. Experimental results on a number of benchmarks and industrial designs demonstrate that I/O limits are the bottleneck for CLB utilization when applying a traditional multiple-FPGA synthesis method on flattened netlists. In contrast, by fully exploiting the design structural hierarchy during the multiple-FPGA partitioning, our proposed method produces fewer FPGA partitions with higher CLB and lower I/O-pin utilizations.

1 Introduction

Because of their reprogrammability, Field Programmable Gate Arrays (FPGAs) have become the most popular Application-Specific Integrated Circuits (ASICs) for rapid system prototyping. In addition, the development of reconfigurable systems by integrating FPGAs and Field Programmable Interconnect Chips (FPICs) has become the new trend in design verification, rapid prototyping, and computation-intensive applications [1, 2, 3].

In general, the traditional design flow to map designs onto a multiple-FPGA system consists of the following two phases. In the first phase, a synthesizer is used to transform a design specification into a CLB-based netlist by performing HDL compilation, RTL/logic synthesis, and CLB-based technology mapping tasks. In the second phase, a circuit-level partitioner is used to partition the CLB netlist into FPGA chips. One of the crucial tasks for the above design flow is to partition a design onto a set of FPGAs. The problem of FPGA-based partitioning is quite different from the classical ASIC partitioning problem. Because FPGA chips have fixed and limited amounts of logic units (CLBs) and I/O pins, mapping a design onto a set of FPGAs is usually predominately constrained by pin limitations. This often results in FPGA partitions with very low logic utilizations.

In the past several years, many partitioning approaches and algorithms [4, 5, 6, 7] have been proposed to solve the FPGA partitioning problem. All of the above approaches perform circuit-level partitioning on flattened gate-level or CLB-level netlists. They do not take into account design-hierarchy information. Furthermore, none of the above approaches exploit the interrelationship between design synthesis and partitioning tasks. In a recent study [8], Schmit et al. experimented with multiple FPGA partitioning methods at behavioral and structural levels. In [9], a hierarchical functional structuring and partitioning method was proposed for multi-FPGA implementations. Both methods demonstrated that better FPGA partitions can be achieved when design structural information is considered during the partitioning process.

In this paper, we present a new integrated synthesis and partitioning method for multiple-FPGA applications. This method first synthesizes a design specification in a fine-grained way so that functional clusters can be preserved based on the structural nature of the design specification. Then, it applies a hierarchical set-covering partitioning method to form the final FPGA partitions. Experimental results on a number of benchmarks and industrial designs are reported to demonstrate the effectiveness of the proposed approach.

2 Problem Description

One commonly used partitioning approach for large designs is to first apply some clustering scheme to reduce circuit complexity and then apply a set-covering method to reduce the number of required chips. Based on this approach, several algorithms [7] have been developed to produce good results on large industrial designs. However, this approach performs partitioning on flattened circuits which does not take advantage of circuit hierarchy information to alleviate the IO-limitation problem. Furthermore, the selection of cluster sizes may be very sensitive to the complexity and quality of the partitioning procedure. This motivates us to investigate how to use a synthesis technique to generate clusters based on the structural nature of the design specification and pre-

†This work was supported by the National Science Council of R.O.C. under Grant NSC 86-2221-E-007-047.

DAC97, Anaheim, California

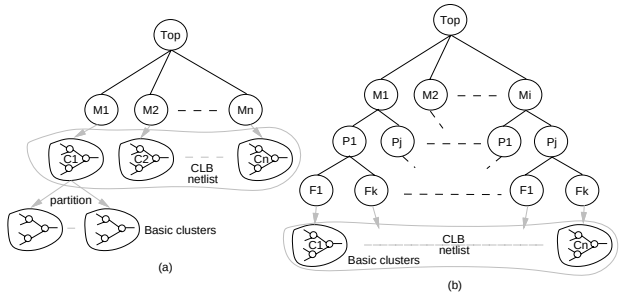

Figure 1: Two synthesis approaches: (a) module-based, (b) fine-grained.

serve the circuit hierarchy so that the partitioning procedure can fully exploit the design hierarchy.

First we describe the relationship between the synthesis and the partitioning design tasks as follows. An industrial design is usually represented as a mixed RTL/logic/gate-level description in High-level Descriptive Languages (HDLs) such as Verilog and VHDL. The HDL description of a design is described as a set of interconnected modules. During the synthesis process, each module is synthesized into an independent circuit. The final circuit of the design is the composite circuit of all modules, as shown in Figure 1(a). This synthesis approach can preserve the circuit hierarchy between the top-level design, its modules, and their corresponding netlists. If all the modules are small enough, then each module can be treated as a basic cluster. However, in most industrial designs a module often contains a complex function, such as an MPEG algorithm or a floating-point multiplier, which may be too large to be covered by a single FPGA chip. In this case, we have to apply circuit-level partitioning on the module to decompose it into smaller clusters. However, the drawback of this approach is that we will experience much the same problem as determining the cluster size when performing partitioning on flattened circuits. Furthermore, a module, itself, is too coarse to be used as the basic cluster for partitioning. One possible solution to overcome the above problem is to synthesize designs in a fine-grained way such that the circuit is generated in a multi-level hierarchy, as shown in Figure 1(b). The sub-circuit of each leaf node is small enough to be used as a basic cluster for partitioning. In this case, the design hierarchy can be fully preserved and the clusters are formed based on the design hierarchy. Subsequently, we can directly apply a set-covering algorithm to perform FPGA covering.

3 An integrated synthesis and partitioning approach

The proposed design flow consists of three steps: (1) HDL synthesis, (2) functional-based clustering, and (3) hierarchical set-covering partitioning. The input to the system is a design specification described in Verilog. In the first step, a synthesizer [10] performs RTL and FPGA synthesis tasks. The synthesizer uses a fine-grained synthesis method to generate a structural tree for the design. In the second step, the system performs a functional-based clustering procedure to form basic clusters. In the third step, a partitioner uses a hierarchical set-covering algorithm to perform FPGA-based partitioning. Finally, the system outputs multi-FPGA partitions in XNF and Verilog formats.

3.1 Functional-based clustering

We use the fine-grained synthesis method [10] to synthesize the design in a fine-grained way so that a set of clusters can be formed according to the structural nature of the design. We use a structural tree to represent the structural hierarchy of the HDL description of a design. In a structural tree, the root node represents the top-level design, each intermediate node a higher-level design such as *modules*, *processes*, and *tasks*, and each leaf node a circuit cluster.

We first discuss how to synthesize an HDL-based design in a fine-grained way as follows. A Verilog description of the design is usually described as a set of hierarchical interconnected modules. Each module may contain a set of concurrent processes. Each process is defined as an *always* clause which can be activated by an event control statement. The statements in the body of an *always* statement are repeatedly executed in sequential order. From the hardware point of view, each process can be implemented as an independent hardware block. It seems most natural to use each process as a basic cluster. However, in order to obtain clusters with a finer granularity, we should further decompose the process-level circuit into smaller clusters. Consider that a process consists of a set of statements with a set of input and output signals. The outcomes of the outputs are dependent on the executions of the statements embedded in the process. It follows that, we can generate a set of logic functions for each output such that each output is a function of a set of inputs and internal signals in the process. Furthermore, we can decompose multi-bit logic functions into sets of bit-level logic functions to produce clusters with finer granularities.

The functional-based clustering procedure is shown as follows.

Procedure Functional_Based_Clustering(V_{design}) **begin**
  Import a Verilog HDL description;
  Create a top-level structural tree;
  for (each *process* node) **begin**
    Determine the I/O ports of the process;
    Perform RTL synthesis;
    Generate logic functions for each output;
    Bit-level functional decomposition;
    Construct the leaf nodes;
  endfor
  Perform logic and FPGA synthesis on nodes;
  Perform interconnect analysis;
  Return the structural tree;
endProcedure

In the first step, a Verilog description is imported and converted into a top-level structural tree according to the hierarchy of the description. In the second step, the synthesizer performs RTL synthesis tasks to convert each process into sets of logic functions. In the third step, a bit-level functional decomposer partitions the logic functions into a set of bit-level functions to form the leaf nodes of the structural tree. In the fourth step, the synthesizer converts each leaf node into a structural design in Berkeley Logic Interchange Format and EQN (Boolean Equation) format. In the fifth step, each leaf node is converted into a CLB design by applying logic and

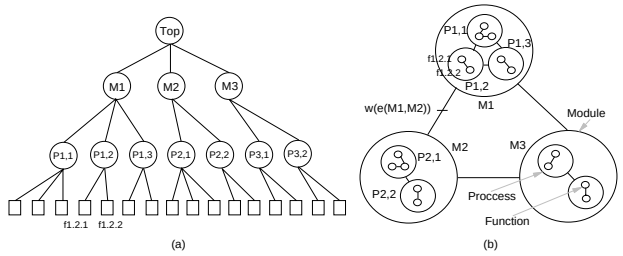

Figure 2: A hierarchical connected graph:(a) the structural tree, (b) the corresponding graph.

FPGA synthesis tasks After generating the CLB netlists for all the leaf nodes, we can further generate the CLB netlists for intermediate nodes of the structural tree by applying a collapsing technique [11]. Consequently, the required CLBs and IO pins of nodes in the structural tree can be determined. Furthermore, the number of interconnections between two nodes can be computed by matching the IO pins of the two nodes. Finally, the procedure outputs a structural tree and its corresponding CLB-based design.

3.2 Hierarchical set-covering partitioning

We formulate FPGA covering into a set-covering problem which is defined: Given a hierarchical connected graph G and the CLB/IO-pin constraint of the FPGA chips, find a minimum number of FPGAs to cover G. In our approach, we use a hierarchical set-covering method for FPGA covering. Our goal is to produce a coverage by fully exploiting the structural nature of the design.

After constructing a structural tree and forming the clusters, we next convert the structural tree into a hierarchical connected graph. Figures 2(a) and (b) show a structural tree and its corresponding hierarchical connected graph. Each node v_i has associated with two attributes $CLB(v_i)$ and $IO(v_i)$ which represent the number of CLBs and IO pins contained in it. An edge connects two nodes when there is a physical connection between them.

The basic idea of the hierarchical covering method is to start the set-covering procedure from the top-level nodes (i.e., *module* nodes). If no more feasible covers can be found in the top-level, then the set-covering process continues on the nodes at the lower level. In short, the covering process starts from the nodes with coarse granularities and then moves down to the nodes with finer granularities.

Let $G = (V, E)$ be a connected hypergraph. $V = V_m \cup V_p \cup V_f$, where V_m is a set of module nodes, V_p a set of process nodes, and V_f a set of functional nodes. C_{cover} denotes a set of FPGA chips. For a node v_i, $clb(v_i)$ and $io(v_i)$ represent the number of CLBs and IO pins of v_i. CLB and IO denote the CLB and IO-pin constraints of the FPGA chips.

The inputs to the algorithm include a connected hypergraph G and the CLB and IO-pin constraints (CLB and IO) of the FPGAs used. The output is the number of FPGA chips required to cover the design. The algorithm first computes the score after assigning the nodes at the top level into a chip. If the number of CLBs and IO pins by assigning a node v_i into a chip c_k satisfies the CLB and IO-pin constraints of the chip, then the score of the pair (c_k, v_i) is calculated as a function of the closeness between the node and the nodes already assigned to the chip and the ratio between the number of CLBs and IO pins. α and β are two parameters set by the user to express a preference of one term over the other. Otherwise, the score of (c_k, v_i) is zero.

After computing the scores for all nodes, if none of the nodes can be assigned to any chips (i.e., *all* $score(c_k, v_j) = 0$), then there are two possible solutions. First, if none of the leaf nodes (i.e., the functional nodes) can be covered by any of the allocated FPGA chips, then a new chip c_{new} is allocated and the covering procedure is continued, starting from the nodes at the top level (i.e., module nodes). Otherwise, procedure $Next\_Level()$ is invoked to expand the covering process to the nodes in the next lower level so that the covering process can be performed on nodes with a finer granularity. Second, if there are several possible covering solutions, then the one with the highest score will be selected. The algorithm stops when all the nodes are covered by FPGA chips.

Algorithm Hierarchical_Set_Covering(G, CLB, IO) **begin**
  $C_{cover} \leftarrow c_{new}$; $V_{cluster} = V$; $top\_level = V_m$;
  while ($V_{cluster} \neq \phi$) **begin**
    for (all $v_i \in V_{cluster}(top\_level)$)
      for (all $c_k \in C_{cover}$)
        if ($clb(\{c_k \leftarrow v_i\}) \leq CLB$
          and $io(\{c_k \leftarrow v_i\}) \leq IO$) **then**
          $score(c_k, v_i) = \alpha\ Conn(c_k, v_i) +$
            $\beta\ clb(\{c_k, v_i\})/io(\{c_k, v_i\})$;
        else $score(c_k, v_i) = 0$;
    if (all $score(c_k, v_i) = 0$) **then begin**
      if ($top\_level = V_f$) **then begin**
        $C_{cover} = C_{cover} \cup c_{new}$;
        $top\_level = V_m$;
      endif
      else top_level = Next_Level(top_level);
    endif
    else begin
      Select the pair of $\{c_k, v_i\}$ with the highest score;
      $V_{cluster} = V_{cluster}$ - v_i; $c_k = c_k \cup v_i$;
    endelse
  endwhile
endAlgorithm

Time complexity. Let n be the number of functional nodes and m the number of chips used. It takes $O(n * m)$ time to compute the scores. Hence, the set-covering procedure takes $O(n^2 * m)$ time.

4 Experiments

We have implemented the hierarchical set-covering algorithm in the C programming language. Presently, the algorithm is embedded in an interactive multiple-FPGA synthesis system (*ISyn*) which consists of approximately 150,000 lines of C code and runs on SUN and HP workstations.

We have tested our proposed algorithm on four designs. The first design *Design*1 is a 32-bit fifth-order elliptic filter benchmark. *Design*2, *Design*3, and *Design*4 are three industrial designs, a 32-bit floating-point multiplier, a microcontroller, and a communication chip. The four designs are described in Verilog de-

Table 1: Comparisons between the MRFM and the FHSC methods.

Design 1 (MRFM/FHSC)			
Chip Type	#CLBs	IOU	#Chips
XC4013	1,680/1,741	.92/.78	10/5
XC4010	1,680/1,799	.96/.81	12/7
XC4005	1,680/1,876	.97/.88	25/13

Design 2 (MRFM/FHSC)			
Chip Type	#CLBs	IOU	#Chips
XC4013	1,153/1,196	.93/.87	6/5
XC4010	1,153/1,228	.93/.82	10/6
XC4005	1,153/1,377	.98/.91	18/13

Design 3 (MRFM/FHSC)			
Chip Type	#CLBs	IOU	#Chips
XC4013	798/803	.90/.71	9/3
XC4010	798/829	.98/.79	12/4
XC4005	798/978	.96/.88	21/9

Design 4 (MRFM/FHSC)			
Chip Type	#CLBs	IOU	#Chips
XC4013	2,964/3,050	.96/.74	16/6
XC4010	2,964/3,247	.98/.89	23/13
XC4005	2,964/3,678	.99/.93	41/27

scriptions with 167, 404, 4,084, and 6,298 lines of code containing 9, 1, 20, and 36 modules, respectively. Using the fine-grained synthesis method, the hierarchical structural trees contain 5, 7, 30, and 167 process nodes and 595, 57, 1,095, and 5,148 functional leaf nodes, respectively.

We have compared the partitioning results produced by our proposed Fine-grained synthesis and Hier_Set_Covering partitioning method ($FHSC$) and the results produced by the Module-based synthesis and RFM partitioning method ($MRFM$). For the latter, we first performed module-based synthesis to generate a flattened CLB netlist. Then, we applied the RFM [12] algorithm to partition the netlist into multiple-FPGA chips. Table 1 shows the comparative results in which $Type$, $\#CLBs$, IOU, and $\#Chips$ represent chip types, the number of CLBs, I/O utilization, and the number of chips used, respectively. The results show that our proposed method produced partitions using fewer chips and lower I/O utilizations than that produced using the traditional method.

From the experiments, the following observations can be made. First, the results show that the fine-grained synthesis with hierarchical set-covering partitioning produced designs using in average of 10% more CLBs than that produced by the module-based synthesis with RFM partitioning. The additional design overhead resulting from our method is explained as follows. Using the module-based synthesis method, the synthesis system can apply a global optimization procedure effectively on each module of the designs, often resulting in smaller circuits. On the other hand, using the fine-grained synthesis method, the synthesis system applies optimization procedures on either module, process, or functional nodes depending on the partitioning decision. When the functional nodes of a module are assigned to several partitions, each functional node will be synthesized into an independent circuit. In this case, the synthesized circuits will be larger compared to that produced using the module-based synthesis method. Hence, using the fine-grained synthesis method may result in a larger CLB design. Nevertheless, the method can produce designs with a finer granularity which provides a much greater degree of flexibility for partitioning.

Second, when HDL synthesis and physical partitioning tasks are integrated for multiple-FPGA synthesis, the quality of the synthesized results may depend on the coding style of the HDL design descriptions. Designers can either describe a design as a single module and a single process, or a description containing a set of interconnected modules and processes. The latter approach should provide a much greater degree of flexibility for multiple-FPGA partitioning. From this study, we have observed that our proposed method is very effective for communication-oriented and control-oriented designs (e.g., *Design3* and *Design4*) which are usually described as a set of interconnected modules in HDLs and do not contain a large data path.

5 Conclusions

In this paper, we have presented a new integrated synthesis and partitioning method for multiple-FPGA applications. We have tested the proposed method on a number of benchmarks and industrial designs. Experimental results have shown that high-density FPGA partitions can be achieved when the structural hierarchy of designs is fully exploited during the partitioning process.

Future studies include developing different integrated synthesis and partitioning methods targeted to designs described in different HDL coding styles and performance-driven partitioning methods.

References

[1] M. Butts, J. Batcheller, and J. Varghese, "An Efficient Logic Emulation System," *Proceedings of ICCD92*, pp. 138-141, 1992.

[2] C. E. Cox and W. E. Blanz, "GANGLION- A Fast Field-Programmable Gate Array Implementation of a Connectionist Classifier," *IEEE Journal on Solid-State Circuits*, vol. 27, pp. 288-299, March 1992.

[3] S. Walters, "Computer-Aided Prototyping for ASIC-Based Systems," *IEEE Design and Test of Computers*, pp. 4-10, June 1991.

[4] N.-S. Woo and J. Kim, "An Efficient Method of Partitioning Circuits for Multiple-FPGA Implementation," *Proceedings of the 30th DAC*, pp. 202-207, 1993.

[5] G. Saucier, D. Brasen, and J. P. Hiol, "Partitioning with Cone Structures," *Proceedings of ICCAD93*, pp. 236-239, 1993.

[6] R. Kuznar, F. Brglez, and K. Kozminski, "Cost Minimization of Partitions into Multiple Devices," *Proceedings of the 30th DAC*, pp. 315-320, 1993.

[7] N.-C. Chou, L.-T. Liu, C.-K. Cheng, W.-J. Dai, and R. Lindelof, "Circuit Partitioning for Huge Logic Emulation Systems," *Proceedings of the 31st DAC*, pp. 244-249, 1994.

[8] H. Schmit, L. Arnstein, D. Thomas, and E. Lagnese, "Behavioral Synthesis for FPGA-based Computing," *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines 1994*, pp. 125-131, 1994.

[9] W.-J. Fang and Allen C.-H. Wu, "A Hierarchical Functional Structuring and Partitioning Approach for Multiple-FPGA Implementations," *Proceedings of ICCAD96*, pp. 638-643, 1996.

[10] W.-J. Fang and Allen C.-H. Wu, "A Fine-Grained Synthesis Method for Logic Emulation Systems." *Technical Reports*, Computer Science Dept., Tsing Hua Univ., Taiwan, R.O.C., 1996.

[11] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," *Proceedings of ICCAD91*, pp. 564-567, 1991.

[12] C. M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions," *Proceedings of 19th DAC*, pp. 175-181, 1982.