Behavioral Partitioning with Synthesis for Multi-FPGA Architectures under Interconnect, Area, and Latency Constraints *

Preetham Lakshmikanthan **, Sriram Govindarajan,
Vinoo Srinivasan ***, and Ranga Vemuri
{plakshmi, sriram, vsriniva, ranga}@ececs.uc.edu
Department of ECECS, University of Cincinnati, Cincinnati, OH 45221

Abstract

This paper presents a technique to perform partitioning and synthesis of behavioral specifications. Partitioning of the design is done under multiple constraints – interconnections and device areas of the reconfigurable architecture, and the latency of the design. The proposed Multi-FPGA partitioning technique (FMPAR) is based on the Fiduccia-Mattheyses (FM) partitioning algorithm. In order to contemplate multiple implementations of the behavioral design, the partitioner is tightly integrated with an area estimator and design space exploration engine.

A partitioning and synthesis framework was developed, with the FMPAR behavioral partitioner at the front-end and various synthesis phases (High-Level, Logic and Layout) at the back end. Results are provided to demonstrate the advantage of tightly integrating exploration with partitioning. It is also shown that, in relatively short runtimes, FMPAR generates designs of similar quality compared to a Simulated Annealing partitioner. Designs have been successfully implemented on a commercial multi-FPGA board, proving the effectiveness of the partitioner and the entire design framework.

1 Introduction

Partitioning is essential when designs are too large to be placed on a single device or because of I/O pin limitations. Partitioning of a design can be done at various levels - behavioral, register-transfer level (RTL) or gate-level. Behavioral partitioning is a pre-synthesis partitioning while RTL partitioning is done after high-level synthesis. Various studies [1] show the superiority of behavioral over structural partitioning.

A behavioral partitioner has no a priori knowledge about design parameters such as area and latency. The partitioner must be guided by a high-level estimator that provides the required information. Efficient estimation techniques [2, 3] have been developed for this purpose. The approach presented in [2], presents an efficient design space exploration technique that can be performed dynamically with partitioning. A partitioner can effectively control the trade-off between the execution time and the design space

* This work is supported in part by the US Air Force, Wright Laboratory, WPAFB, under contract number F33615-96-C-1912, and under contract number F33615-97-C-1043

** Currently at *Cadence Design Systems* Inc., MA. Work done at University of Cincinnati.

*** Currently at *Intel* Corporation, CA. Work done at University of Cincinnati.

explored. We show the effectiveness of integrating the partitioner with a design-space exploration engine in generating constraint satisfying solutions.

There has been a lot of research in multi-FPGA partitioning, as presented in the survey by Alpert and Kahng [4]. In particular, Sanchis [5] extended the FM for multiway partitioning by repeatedly applying standard bi-partitioning. This work attempts to minimize the sum of all the *cutsets* across all partition segments. For a multi-FPGA RC, it is imperative that the pin constraints of the devices are individually satisfied. Therefore, this method of minimizing a summation of cutsets may not produce a constraint satisfying solution. Our goal is to minimize each cutset individually for pin-constraint satisfaction. We present a technique called FMPAR which is an extension of the Fiduccia-Mattheyses algorithm [6]. The results of partitioning are compared against a Simulated Annealing (SA) partitioner that forms part of the SPARCS [7] framework.

The rest of the paper is organized as follows. Section 2 describes the partitioning and synthesis framework. Section 3 presents the FMPAR algorithm in detail and the interaction of FMPAR with an exploration engine. Finally, Section 4 presents results demonstrating the effectiveness of this work.

2 Partitioning and Synthesis Framework

The framework for partitioning and synthesis is shown in Figure 1.

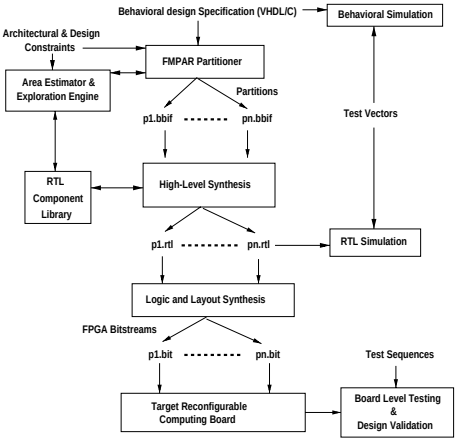

Fig. 1. Partitioning and Synthesis Framework

It consists of the FMPAR partitioner at the front-end and various synthesis phases (High-level, Logic, and Layout) at the back-end. The input behavioral designs are specified in subsets of either VHDL or C. The design descriptions are translated into an equivalent Control-Data Flow Block Graph (CDFG), where the *blocks* contain a simple data-flow graph that captures computation, and the edges between blocks represent both data and control flow.

The FMPAR partitioner views a block in CDFG as an atomic element that cannot be partitioned onto multiple FPGAs. The edges between various blocks are the set of *cutset* constraints for the partitioner. The user can specify any number of logical memory segments modeled as *dummy* blocks in the CDFG. The FMPAR partitioner automatically maps the logical memory blocks onto the physical memory banks. The core of the entire flow is the iterative FMPAR partitioner coupled with an area estimator and exploration engine. The exploration engine performs effective resource sharing across blocks and provides the partitioner with accurate area estimates.

The partitioned behavior segments generated by FMPAR are automatically synthesized by an in-house high-level synthesis tool to generate equivalent RTL designs. Further, the RTL designs are taken through commercial logic (*Synopsys Design Compiler*)

and layout (*Xilinx M1*) synthesis tools to generate FPGA bitstreams for the target board. Note that, the communication signals routed across devices are always registered in the RTL designs to ensure that the board interconnect delay does not affect the clock period of the partitioned design.

3 The FMPAR Partitioner with the Exploration Engine

Like the FM, the FMPAR also allows only one block to be moved at a time and the locking option of cells in the standard FM is incorporated here. A block can be moved across the FPGAs, a user-specified number of times, after which it is locked and cannot be moved. We now present the terminology and details of the FMPAR algorithm.

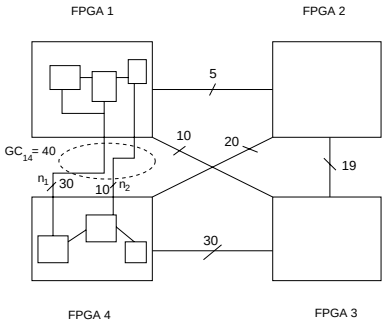

Fig. 2. Cutsets between FPGAs

Global Cut (*GC*) : This is defined as the *cut-set* between the partitions assigned to two FPGAs. Consider the example shown in Figure 2. The RC board contains 4 FPGAs and it is a fully connected board. There are six global cuts, for example GC_{14} denotes the global cut between FPGAs 1 and 4, and $|GC_{14}|$ denotes the *size* of the global cut.

Current_Max : is the greatest value among all the global cuts. In Figure 2, $|GC_{14}|$ = 40 is the Current_Max.

Current_Min : is the least value among all the global cuts. In Figure 2, $|GC_{12}|$ = 5 is the Current_Min.

Net Cut (n_i) : Each global cut is in turn composed of a set of nets that are cut $\{n_1, n_2, \ldots, n_k\}$. Consider the Current_Max value ($|GC_{14}|$ = 40) in Figure 2. It is contributed to by 2 net-cuts n_1 and n_2 of *size* 30 and 10 respectively.

Priority : The net-cuts are prioritized in decreasing order of their sizes. The size of a net-cut is the bit-width of the net. In trying to reduce any global cut, we attempt to eliminate the net-cuts, one at a time, in the sorted order.

Net-Cut Elimination & Move Types : The moves are contemplated such that the worst GC ($Current\_Max$) is reduced. For this purpose, the highest priority net-cut of the worst GC is considered. Moves are contemplated on the connected blocks to this net-cut. For example in Figure 2, n_1 is the highest priority net-cut in the worst global cut GC_{14}. Three possible moves can be contemplated to eliminate this net-cut: (1) Move the connected blocks in FPGA 1 into FPGA 4 or vice-versa, (2) Move the connected blocks in FPGA 1 or 4 into FPGA 2, and (3) Move the connected blocks in FPGA 1 or 4 into FPGA 3.

We call Option 1 as *1-degree move* of a netcut, Option 2 as *2-degree move* of a netcut and Option 3 as *3-degree move* of a netcut. In general, for 'n' FPGAs, a 1-degree move is between the pair of FPGAs (say F_i and F_j) associated with the highest priority net-cut. The remaining 'n-2' FPGAs (other than F_i and F_j) are sorted in decreasing order of available *free space*, F_2, F_3, $\ldots$, F_{n-1}. A *k-degree move* ($2 \leq$ k $\leq$ n-1) is defined as one where blocks on either F_i or F_j are moved to the corresponding k'th FPGA. We

define *free space* as the difference between the device area and the estimated area of the partition segment.

3.1 The FMPAR Algorithm

Algorithm 3.1 presents the outline of FMPAR, the proposed multiway FM partitioning technique. The inputs are the design described as a CDFG block graph (BG), the number of FPGAs (N_{fpga}) on the board, the size ($gc\_size[][]$) of interconnections between each pair of FPGAs, the area of each FPGA ($dev\_area[]$), the block locking factor ($lock\_fact$), and the design $latency$. Unlike a standard FM, our algorithm performs a user-specified number of runs (N_{runs}) from different initial partition solutions. During each run, the FM-loop (outer repeat-until loop) is executed until no improvement in cutset is observed for K successive iterations.

Algorithm 31 (FMPAR Algorithm) **FMPAR**(*BG, N_{fpga}, gc_size[][], dev_area[], lock_fact, latency, K, N_{runs}*)
Begin
  Max ← *Prev_Max* ← ∞ ; *Current_Max* ← *0;*
  For *FM_runs = 1 to N_{runs}*
    New Partition ← *Generate a legal initial partition;*
    Repeat */* Run FM-loop until no improvement */*
      Calculate GCs for all pairs of FPGAs;
      If *($1 \le i, j \le N_{fpga}$, $|GC_{ij}| \le$ gc_size(i,j))* **Then**
        Output (Constraint Satisfying Solution) and Exit;
      EndIf;
      Repeat */* Until no moves are possible */*
        Calculate Current_Max, Current_Min and order all the GCs;
        If *(Current_Max < Max)* **Then**
          Max ← *Current_Max;*
        EndIf;
        Move ← *Choose_A_Move(N_{fpga}, Current_Max, Current_Min, dev_area[], lock_fact);*
        If *(Move = Φ)* **Then**
          If *(Max < Prev_Max)* **Then**
            Prev_Max ← *Max;*
            Save current partition as **best partition**;
          EndIf;
          Break; */* Out of inner repeat-until loop */*
        Else
          Make the move and Increment that block's move_count;
        EndIf;
      Until*(False);*
      If *(Prev_Max hasn't changed over the last K runs)* **Then**
        *Output (***best partition** *solution obtained);*
        Break; */* Out of outer repeat-until loop */*
      EndIf;
      Reset move_count of all the blocks to zero;
      New Partition ← **best partition**;
    Until*(False);*
  EndFor; */* Restart FM with a new initial partition */*
End.

During each run of the FM algorithm a *legal* initial partition is generated. A partition is said to be *legal* if and only if all partition segments satisfy the area constraints posed by the individual devices. During each iteration of the FM-loop, all GCs are computed and ordered. If a *constraint satisfying* solution is obtained the entire *FMPAR* algorithm terminates. A *constraint satisfying* solution is a legal partition that satisfies the interconnection constraints as well. *Current_Max* is the worst cutset between all FPGA pairs, and is calculated every time a move is made. *Max* represents the least value of $Current\_Max$, over all moves that have been made. *Prev_Max* is the least value of Max over all iterations of the FM-loop.

During each iteration of the FM-loop, several *legal moves* are made until no further moves are possible. A move is *legal move* only if it leads to a legal partition and does not exceed the locking factor. The *locking factor* is a user-defined upper limit on the number of times a block can be moved. For selecting a legal move, the algorithm contemplates several possible moves in the procedure $Choose\_A\_Move()$. The contemplated moves are called $k - degree$ moves as explained earlier. The goal is to minimize the worst cutset ($Current\_Max$). If none of the moves decrease the worst cutset, then the least cutset *violating* move is accepted. If no legal move is possible the procedure returns Φ. This terminates the move-making process for one iteration of FM. At this point, each block is *unlocked* (move_count is set to zero) and the *best partition* obtained so far is used as the new partition for the next iteration of the FM.

3.2 Interaction between FMPAR and Exploration Engine

The partitioner is tightly integrated with a high-level exploration engine. The partitioner always communicates any change in the partitioned configuration to the exploration

engine and both the tools maintain an identical view of the partitioned configuration. The exploration engine effectively uses the partitioning information by *dynamically* generating implementations that maximize sharing of resources within each partition segment. Further details of the exploration engine can be found in [2].

The partitioner dynamically controls the trade-off between the execution time and the design space explored. The exploration technique provides an Exploration Control Interface (ECI) that facilitates tight integration with the partitioning algorithm. This interface consists of a collection of *exploration methods* that generate new implementations, and *estimation methods* that simply re-compute the design estimates for a modified partition configuration. Algorithm 32 presents the template for the FMPAR algorithm with calls to the exploration engine enclosed in boxes. The FMPAR partitioner calls the area estimator and exploration engine at two places : (1) When moves are being evaluated (line-6), and, (2) when the configuration is reset to the best partition (line-10). A detailed study was conducted to make appropriate usage of the ECI functions at crucial points of the partitioning process.

Algorithm 32 (FMPAR with dynamic exploration)

```
FMPAR()
Begin
1     Generate random initial partition of blocks;
2     Repeat
3       Unlock all blocks;
4       While (∃ movable blocks) Do
5           Select a block;
6           [ Estimate Move; ]
7           Make a move and lock;
8       EndWhile;
9       Reset to the best partition;
10      [ Explore Design for best partition; ]
11    Until (No Cutset Improvement);
End.
```

The *Estimate Move* method evaluates the effect of moving a block from a source partition to a destination partition without performing exploration and hence is not expensive in time. Whereas, the *Explore Design* method attempts to generate area and latency satisfying implementations at the expense of compute time. This way the calls to the exploration engine effectively utilize the trade-off between the exploration time and the amount of design space explored.

Essentially, the partitioner takes care of the interconnection constraints, while the area and the latency constraints are handled by the area estimator and exploration engine. Thus, each time the solution is acceptable in terms of interconnection constraint, the exploration engine ensures the best area and latency satisfying solution.

4 Experimental Results

We first present results to show the effectiveness of the FMPAR algorithm integrated with the exploration engine. Then, the FMPAR is compared with a simulated annealing partitioner. Finally, we report results obtained for designs that were successfully implemented on the *Wildforce* [8], a commercial multi-FPGA board.

4.1 Effectiveness of Dynamic Exploration with FMPAR

We developed two versions of FMPAR, one performing dynamic exploration and another that does not. In the latter case, the exploration engine is used only to obtain area estimates without exploring multiple implementations. For experimentation, we considered the two large DSP benchmarks - the Discrete Cosine Transform (DCT) and the Fast Fourier Transform (FFT). The FFT benchmark has 18 blocks with 2 loops, 152 operations, 1418 nets (data bits) across the blocks, DCT has 66 blocks with 8 loops, 264 operations and 2401 nets and both examples have an extremely large number possible implementations.

We have gathered results by fixing two of the three constraints (design *latency* (L) and RC interconnection *cutset* (C)) and varying the third (device *area* (A)). The results are presented as plots where the x-axis represents the constraint varied (device area) and the y-axis represents the *fitness* value. Fitness is defined as, $F = \frac{1}{(1+CutsetPenalty)}$, $where,\ \ CutsetPenalty = \frac{\sum UnroutedNets}{TotalDesignNets}$ The unrouted nets is the summation of all the nets contributing to GCs that exceed the board cutsize. Fitness is a measure of the solution quality, ranging between 0 and 1. A fitness value of 1 denotes a constraint satisfying solution, while a lower value denotes a poor quality solution because of a violation of cutset constraints.

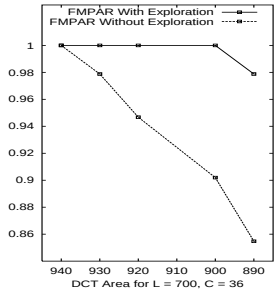

Fig. 3. Plot for DCT

We chose a representation of the *Wildforce* architecture with four FPGA devices and a cutset constraint of 36 interconnections between each pair of FPGAs. Figure 3 plots the fitness of generated solutions for the DCT benchmark. Both versions of the partitioner (with and without the dynamic exploration) generate constraint-satisfying solutions for all area constraints at and greater than 940 CLBs. As we gradually decrease the design area we see that the FMPAR version with dynamic exploration continues to generate constraint-satisfying solutions ($F = 1$), while its counterpart fails ($F < 1$), even after running on a large number random initial configurations.

We have made similar observations for the FFT benchmark, presented in [2]. This clearly demonstrates the effectiveness of interfacing the partitioner with the area estimator and exploration engine.

4.2 Comparison of FMPAR against a Simulated Annealing Partitioner

In this section, we compare the results of the FMPAR algorithm to that of a Simulated Annealing (SA) partitioner that a part of the SPARCS [7] design environment. The SA was also interfaced with the area estimator and design space exploration engine. Both algorithms were implemented and run on the same workstation – a twin processor *UltraSparc* with 384 MB RAM and clocking at 296 Mhz.

The table in Fig.4 provides a comparison of designs partitioned by the FMPAR and SA partitioners. For each design example, both partitioners were run on the same set of device area and design latency constraints. The comparison metrics are: (i) the number of *unrouted nets* (# UN) in the resulting solution and, (ii) the *run time* for each partitioner. The N_{devs} in the first column represents the number of devices on the RC, provided as a constraint to the partitioner. The interconnection constraint ($CutSet$) between each FPGA pair was fixed at 36. The last column in the table presents the *speedup* factor of the FMPAR partitioner over the SA partitioner.

Both the FMPAR and SA partitioners found constraint-satisfying solutions in five cases (Rows 1,2,4,5 and 7). The designs satisfied the cutset constraints as evidenced by '0' unrouted nets. At the same time, we see that the FMPAR algorithm *always* has much lesser run times than that of the SA.

Both partitioners did not find a constraint-satisfying solution for three designs – ELLIP (Row 3), FFT (Row 6) and DCT4x4 (Row 8). This is because the partitioners

failed on the a tight cutset constraint. For the DCT4x4 example which is the largest, the SA was run with a slow cooling schedule for 2 hrs and 24 mins and a solution with 21 unrouted nets was obtained. It is observed that for this example, the FMPAR partitioner produced a higher quality solution (only 14 unrouted nets) in a much lesser time (33.4x speedup). In case of the FFT design and the ELLIP examples, the resulting solutions of both partitioner are comparable, yet the *FMPAR* finishes quicker.

From the results, we conclude that FMPAR produces partitioned solutions whose quality is similar to that of the SA, but, in much lesser runtimes. This is because the SA is a stochastic, hill-climbing approach as opposed to the FMPAR which is a move-based algorithm that quickly converges to a constraint satisfying solution. Although FMPAR is highly dependent on the initial solution and could stop at a local optimum, the results are as good as the SA for the constraint satisfaction problem.

Design Name, (N_{devs})	FPGA Area (clbs)	Dsgn Lat. (clks)	Simulated Annealing			FMPAR			Spd Up
			Partn. Areas (clbs)	# UN	Run Time (h:m:s)	Partn. Areas (clbs)	# UN	Run Time (m:s)	
ALU (4)	150	18	60 , 123 146, 0	0	0:00 :04	22 , 147 146, 43	0	0: 01	4x
STATS (2)	324	44	287, 60	0	0:00 :10	49, 318	0	0: 02	5x
ELLIP (2)	450	61	337, 362	7	0:00 :55	441, 252	10	0: 12	5x
ELLIP (2)	600	61	536, 92	0	0:00 :12	596, 26	0	0: 02	6x
FIR (3)	290	93	242, 178, 0	0	0:00 :15	23, 86, 288	0	0: 03	5x
FFT (4)	540	104	446, 317 530, 0	4	0:04 :08	387, 494 500, 484	4	0: 16	15x
FFT (4)	580	104	0 , 580 550, 0	0	0:00 :31	480, 564 353, 423	0	0: 09	3x
DCT4x4 (4)	3600	415	3188, 3303 3468, 3266	21	2:24 :09	3338, 3534 3241, 3531	14	4: 19	33x

Fig. 4. Comparison of SA and FMPAR generated designs

4.3 On-Board Implementations

Two designs were executed on the board after logic and layout synthesis. The designs ALU and STATS were successfully implemented and tested on the Wildforce [8], a commercial multi-FPGA board. The ALU is a simple arithmetic unit that has four 16-bit operating modes: addition, subtraction, multiplication and sum of squares of two input operands. The STATS is a statistical analyzer that computes the mean and variance of eight 16-bit numbers.

Information about the synthesized designs are shown in Figure 5. We compare the estimated area and performance measures against the actual values after layout synthesis. Columns 3 and 4 show the estimated and actual area of each partition. In general, we observed in our experiments that the estimated areas are within 10-20% of accuracy. Columns 5 and 6 compare the latency constraint to

Design Name	**Partition Number**	**Area (CLBs)**		**Latency (Clks)**	
		Estimated	**Actual**	**Constraint**	**Actual**
ALU	P_1	22	30	18	19
	P_2	147	139		
	P_3	146	179		
	P_4	43	54		
STATS	P_1	49	66	44	46
	P_2	318	335		

Fig. 5. Designs down-loaded onto RC boards

the actual latency of the partitioned design obtained from board-level simulation. We observe that our framework satisfies the latency constraint within a deviation of 5%.

In order to check for functional correctness, the results generated on board were verified against the simulation results. The partitioned designs executed successfully and the results matched that of the simulation.

5 Summary

This paper presents a framework for multi-FPGA partitioning of behavioral designs and their synthesis onto reconfigurable boards. An FM based multiway partitioner was presented, which is integrated with an area estimator and design space exploration engine. By efficiently performing dynamic exploration with partitioning, the partitioner produces good quality solutions in a reasonable amount of time. A limitation of the partitioner is that it can currently handle only fixed interconnection architectures. In the future, we plan to integrate the partitioner with interconnect estimation techniques [9] that can handle programmable interconnection architectures.

Results are provided to demonstrate the advantage of tightly integrating exploration with partitioning. Also, it is shown that the FMPAR produces constraint-satisfying solutions of similar quality to that of the SA, in much lesser run-times. Designs taken down to the Wildforce board proves that the FMPAR algorithm maintains the functionality of the design after partitioning and also shows the effectiveness of the partitioning and synthesis framework.

References

1. F. Vahid. "Functional Partitioning improvements over Structural Partitioning for Packaging Constraints and Synthesis : Tool Performance". In *ACM Transactions on Design Automation of Electronic Systems*, volume 3, pages 181–208, April 1998.
2. S. Govindarajan, V. Srinivasan, P. Lakshmikanthan, and R. Vemuri. "A Technique for Dynamic High-Level Exploration During Behavioral-Partitioning for Multi-Device Architectures". In *Proc, of the 13th IEEE Intl. Conf. on VLSI Design*, January 2000.
3. F. Vahid and D. Gajski. "Incremental Hardware Estimation During Hardware/Software Functional Partitioning". In *IEEE Transactions on VLSI Systems*, volume 3, September 1995.
4. Charles J. Alpert and Andrew B. Kahng. "Recent Directions in Netlist Partitioning". In *Integration, the VLSI Journal*, 1995.
5. L. A. Sanchis. "Multiple-way network partitioning". In *IEEE Transactions on Computers*, pages 62–81. 38(1), January 1989.
6. C. M. Fiduccia and R. M. Mattheyses. "A Linear Time Heuristic for Improving Network partitions". In *Proceedings of the 19th ACM/IEEE DAC*, pages 175–181, 1982.
7. I. Ouaiss, S. Govindarajan, V. Srinivasan, M. Kaul, and R. Vemuri. "An Integrated Partitioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures". In *Proc. of Reconfig. Arch. Workshop (RAW98)*, pages 31–36., March 1998.
8. Annapolis micro systems, inc. http://www.annapmicro.com/amshhomep.html.
9. V. Srinivasan, S. Radhakrishnan, R. Vemuri and J. Walrath. "Interconnect Synthesis for Reconfigurable Multi-FPGA Architectures". In *Proc. of RAW99*, pages 597–605., April 1999.