An Integrated Partitioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures [*]

Iyad Ouaiss, Sriram Govindarajan, Vinoo Srinivasan, Meenakshi Kaul, and Ranga Vemuri

DDEL, Department of ECECS, University of Cincinnati, OH 45221.

Abstract. *This paper presents an integrated design system called* SPARCS *(Synthesis and Partitioning for Adaptive Reconfigurable Computing Systems) for automatically partitioning and synthesizing designs for reconfigurable boards with multiple field-programmable devices (*FPGAs*). The* SPARCS *system accepts design specifications at the behavior level, in the form of* task graphs. *The system contains a temporal partitioning tool to temporally divide and schedule the tasks on the reconfigurable architecture, a spatial partitioning tool to map the tasks to individual* FPGAs*, and a high-level synthesis tool to synthesize efficient register-transfer level designs for each set of tasks destined to be downloaded on each* FPGA*. Commercial logic and layout synthesis tools are used to complete logic synthesis, placement, and routing for each* FPGA *design segment. A distinguishing feature of the* SPARCS *system is the tight integration of the partitioning and synthesis tools to accurately predict and control design performance and resource utilizations. This paper presents an overview of* SPARCS *and the various algorithms used in the system, along with a brief description of how a* JPEG*-like image compression algorithm is mapped to a multi-*FPGA *board using* SPARCS.

1 Introduction

During the past few years, reconfigurable computers (RCs) based on multiple field-programmable devices (FPGAs) have become ubiquitous. In order to fully realize the performance and cost advantages offered by these architectures, it is necessary to develop architecture-independent partitioning and synthesis environments for reconfigurable computers.

Although well-developed commercial tools exist to perform the tasks of logic and layout synthesis for FPGAs, high-level synthesis and multi-FPGA partitioning, both spatial and temporal, need to be further developed before mature commercial tools can emerge. Especially ignored, even in the academic community, is the problem of temporal partitioning.

This paper presents the architecture of the SPARCS environment and describes a typical design flow through SPARCS. In addition, a brief summary of each major tool in SPARCS is provided. Whereas various algorithms and tools in SPARCS are subjects of detailed topic-specific publications in their own right, the aim of this paper is to provide a birds-eye view of the SPARCS system, with a specific focus on algorithm integration and users' view.

The rest of this paper is organized as follows: Section 2 provides an overview of the SPARCS design environment. Sections 3, 4 and 5 discuss the temporal partitioning, spatial partitioning, and high-level synthesis algorithms and techniques used in the SPARCS system. Section 6 examines a design flow in SPARCS and Section 7 contains concluding remarks.

2 Users' View of SPARCS

Figure 1 shows the architecture of the SPARCS system. A design specification is submitted in the form of four inputs to the SPARCS system:

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

1. *Application's Behavior Specification:* The SPARCS system accepts a behavioral specification of an application in the form of a set of tasks. Shared data is stored in *memories*. In addition, tasks may communicate through direct communication *channels*. Tasks are modeled in behavior-level VHDL. We assume that the specification is written such that the user provides resolution for critical regions that access shared memory.
2. RC *Architecture Specification:* SPARCS admits the specification of the target reconfigurable FPGA board. Allowed features include local memories, globally shared memory, various interconnect topologies, and features of the FPGA and memory devices used on the board. Architecture specification includes the type of FPGAs used, number of FPGAs, number of resources (function generators and flip-flops) on each FPGA, interconnect topology, sizes of memory modules, etc.
3. *Constraints:* Users of SPARCS specify a constraint on the minimum clock speed at which to clock the design. In addition, users may specify timing constraints on any straight-line block of VHDL code inside the tasks.
4. *Macro Library:* The macro library contains parameterized register-level components. These macros are used to estimate resource counts and performance of contemplated designs during high-level synthesis.

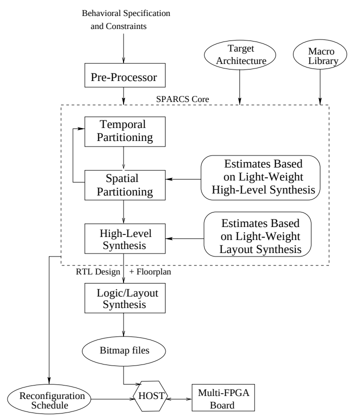

Fig. 1. SPARCS System

Typically, the reconfigurable board is attached to a host computer. The host controls the loading of the design and monitors the execution of the board. The SPARCS system produces a set of FPGA bitmap files, a reconfiguration schedule that specifies when these bitmap files should be loaded on the individual FPGAs in the RC, and in the case of programmable interconnect, a mask (configuration) of the interconnect for each temporal segment.

Preprocessing is the first stage of the SPARCS design flow. Three tasks are performed during preprocessing: (1) memory graph extraction, (2) dependency analysis for each task, and (3) resource and performance estimation for each task. A memory graph represents the relationship among the tasks and memories explicitly and will be used during partitioning. Dependency analysis captures the task-level dependencies and the dataflow among the tasks. Finally, each task is processed by an estimator that estimates the FPGA resource requirements, clock speed, and schedule length of each task, as if it were to be individually synthesized.

3 Temporal Partitioning

The temporal partitioner has an abstract view of the board resources and uses aggregate costs for partitioning. From the architectural specification described in Section 2, the overall resource constraint (C) and shared memory size (M_s) are derived. The temporal partitioner heuristically estimates the upper-bound on the number of temporal segments (N) for building a Non-Linear Programming (NLP) formulation. It uses a fast list-scheduling heuristic (a variation of [1]) for this estimation. We have incorporated a synthesis model to determine the amount of resource sharing among tasks. This requires an operation level modeling of each task for the synthesis subproblem. As a preprocessing step, we determine the mobility ranges (ASAP and ALAP values) of all the operations in the combined graph of the entire specification. The NLP model is linearized and solved by an ILP (Integer Linear Program) solver. We use the following notation:

$t_i \rightarrow t_j$, a directed edge between tasks, $t_i, t_j \in T$, represents a dependency; $Bandwidth(t_i, t_j)$, number of data units to be communicated between tasks t_i and t_j; $Fu(i)$, the set of functional units on which operation i can execute; F, the set of functional units corresponding to the most parallel schedule obtained from the high-level synthesis estimator; N, the upper bound on the number of segments which are numbered $1\ to\ N$, the index specifies the order of execution of the segments; M_s, the shared memory available for storage between segments; $FG(k)$, the number of function generators used for functional unit k; C, the resource capacity of the underlying board architecture; I, the set of all operations in the specification.

Non-Linear 0-1 Model: We describe here the variables, constraints, and cost function of our NLP model. Not all constraints have been shown in mathematical form. The variables: (1) y_{tp} models the partitioning at the task level: $y_{tp} = 1$, if task $t \in T$ is placed in segment p, $1 \leq p \leq$ N; 0 otherwise. (2) x_{ijk} models the synthesis subproblem at the operation level: $x_{ijk} = 1$, if operation $i \in I$ is placed in control step $j \in CS(i)$ and uses functional unit $k \in Fu(i)$; 0 otherwise. (3) $w_{pt_1t_2}$ models the communication cost incurred if two tasks connected to each other are not placed in the same segment: $w_{pt_1t_2} = 1$, if task t_1 is placed on any segment $1 \cdots p-1$ and t_2 on any of $p \cdots N$ and $t_1 \rightarrow t_2$: 0 otherwise. y_{tp} and x_{ijk} are the fundamental modeling variables. All other variables are secondary and are constrained in terms of the fundamental variables.

1. Temporal Partitioning Model: Temporal partitioning has the following constraints:
   a. Uniqueness constraint: Each task should be placed in exactly one segment.
   b. Temporal order constraint: A task t_1 on which another task t_2 is dependent cannot be placed in a later segment than the segment in which task t_2 is placed.
   c. Shared memory constraint: The amount of intermediate data stored between segments should be less than the shared memory M_s. The variable $w_{pt_1t_2}$, if 1, signifies that t_1 and t_2 have a data dependency and are being placed across segment p. Therefore, the data being communicated between them, $Bandwidth(t_1, t_2)$, will have to be stored in the memory of segment p.
   $$\forall p,\ \ 2 \leq p \leq N \quad : \sum_{t_2 \in T} \sum_{t_1 \rightarrow t_2} (w_{pt_1t_2} * Bandwidth(t_1, t_2)) \leq M_s \qquad (1)$$
2. Synthesis Model: The constraints due to synthesis are:
   a. Unique operation assignment constraint: Each operation should be scheduled at one control step and on only one functional unit.
   b. Temporal mapping constraint: Prevents more than one operation from being scheduled at the same control step on the same functional unit.
   c. Dependency constraint: Maintains the dependency relationship between operations.
3. Combined Partitioning and Synthesis Model: The set F, obtained initially, is an upper-bound on the number of functional units that can be used in a segment. To determine whether a functional unit has been used in a segment, we define u_{pk}; $u_{pk} = 1$, if functional unit $k \in F$ is used in segment p, $1 \leq p \leq N$, 0 otherwise.
   a. Resource constraints: We introduce resource constraints in terms of variables u_{pk}. Typical FPGA resources include function generators, combinational logic blocks (CLBs), etc. Similar equations can be added if multiple resource types exist in the FPGAs. α is a user defined logic-optimization factor in the range [0;1]. Typical values [2] of α using Synopsys FPGA components are in the range [0.6;0.8].
   $$\forall p \quad : \alpha * \sum_{k \in F} (u_{pk} * FG(k)) \leq C \qquad (2)$$
   b. Unique control step constraint: We introduce this constraint to make sure that each control step is mapped uniquely to a segment.
4. Cost Function: Minimize the cost of data transfer between segments.
   Minimize: $$\sum_{t_2 \in T} \sum_{t_1 \rightarrow t_2} \sum_{1 \leq p \leq N} (w_{pt_1t_2} * Bandwidth(t_1, t_2)) \qquad (3)$$

For the sake of brevity we have presented only a part of the NLP model. For more details about the constraints, linearization, and solution by ILP techniques refer to [3]. To reduce the amount of time required for solving the ILP model, the model may be solved to find a constraint satisfying solution rather than an optimal one.

Interaction with Spatial Partitioner: Each segment in the temporal partition must fit spatially on the reconfigurable board. Since the temporal partitioner uses aggregate estimates to model the underlying resources, the segment may fail to fit on the board. In such cases the spatial partitioner provides a feedback to the temporal partitioner by posing tightened constraints on the architectural resources to compensate for underestimations. Depending on the number of temporal segments that failed spatial partitioning, the temporal partitioner may choose to re-partition the entire task graph or may just re-partition the segments that violated the constraints.

4 Spatial Partitioning

Problem Formulation: Let $\mathcal{F} = \{f_1, f_2, \cdots f_N\}$ be the N FPGAs available on the target reconfigurable board. Each FPGA has a set of attributes associated with it. For any $f \in \mathcal{F} : C(f)$, $F(f)$, $P(f)$, and $L(f)$ denote the bounds on the number of function generators, flip-flops, uncommitted I/0 pins, and local memory size in f. CM represents the direct connection matrix. It defines the number of dedicated lines pre-routed between each pair of FPGAs. I_c denotes the number of programmable interconnection channels available on the board. All of the above constraints posed by the reconfigurable board are part of the *architectural constraints* as described in Section 2.
A *spatial partition* of a task graph, $TG = (V, M, E)$, where V is the set of task nodes, M is the set of memory segments, and E is the set of dependency edges and channels, is a binding of each task in V to a unique FPGA and each logical memory segment to a unique local/shared memory, such that all architectural constraints are satisfied. Each task graph that is partitioned, corresponds to a temporal segment which is a subgraph of the whole design.
Genetic Spatial Partitioning Algorithm: We model and solve the spatial partitioning problem through a *Genetic Algorithm* (GA). The genetic search procedure was developed by John Holland in 1975 [4], and since then has been used successfully for solving several combinatorial problems in VLSI design automation [5].
Encoding: The solution representation must capture the binding of tasks to the FPGAs and the binding of logical memory segments to local/shared physical memories. We use a simple integer array to encode the above information. Each chromosome has two integer arrays - *task array* TA and *memory array* MA. The length of the TA is equal to the number of tasks in the task graph (t) and the length of the MA is equal to the number of memory segments (m). Consider a board having N FPGAs with local memories and a shared memory. For $1 \leq i \leq t$, the variable $TA[i]$, ranging from 1 through N, represents the FPGA number to which task i is assigned. Similarly, for $1 \leq i \leq m$, the variable $MA[i]$, ranging from 0 through N, represents the memory bindings. $MA[i] = 0$ implies that the memory segment i is mapped to the shared memory.
Initial Population: The task arrays for all chromosomes in the initial population are set to random legal values. Then based on the task assignments, for each chromosome, we assign the logical memory segments to local physical memories. If the majority of the tasks which access a memory segment are assigned to FPGA k then we bind the memory segment to the local memory of FPGA k.
Crossover: We use a uniform crossover operator. A binary string, T, whose length is equal to the greater of the number of tasks and the number of memory segments, is generated. Each bit in this *template* is randomly set to either 0 or 1. Next, two parents are probabilistically selected for mating. Let pt_1, pt_2 be the task arrays and pm_1, pm_2 the memory arrays in the parents. Then ct_1, ct_2, cm_1, and cm_2, are the corresponding arrays in the two child chromosomes resulting from a crossover defined as:

$$ct_1[i] = \begin{cases} pt_1[i] \text{ if } T(i) = 0 \\ pt_2[i] \text{ otherwise} \end{cases} \quad ct_2[i] = \begin{cases} pt_1[i] \text{ if } T(i) = 1 \\ pt_2[i] \text{ otherwise} \end{cases} \quad cm_1[j] = \begin{cases} pm_1[j] \text{ if } T(j) = 0 \\ pm_2[j] \text{ otherwise} \end{cases}$$

$$cm_2[j] = \begin{cases} pm_1[j] \text{ if } T(j) = 1 \\ pm_2[j] \text{ otherwise} \end{cases} \quad \text{In these equations, } i \text{ and } j \text{ have legal values}$$

Mutation: The mutation operator randomly selects an entry from the chromosome arrays and changes its value to another legal value.

Partition Cost Estimation: The cost of each chromosome (spatial partition) is dependent on several constraint satisfaction requirements. The constraints we consider are *area constraint (A)*, *Speed Constraint (S)*, *Pin Constraints (P)*, *Interconnect Constraint (I)*, and *Memory Constraint (M)*. We use the the same multi-constraint cost cost function used in [5, 6]. In the case when the spatial partitioner cannot achieve a constraint satisfying solution, it flags a *failure* and returns tighter constraints for use by the temporal partitioner. The new aggregate constraints are based on the degree of cost violated by the best achieved partition.

5 Design Synthesis

At the core of the SPARCS system is a high-level synthesis (HLS) tool, DSS [7] (Distributed Synthesis System), which accepts a behavioral description specified in VHDL and produces an equivalent RTL design consisting of a *Datapath* and a *Controller*. A collection of light-weight layout algorithms is integrated into DSS in order to generate a floorplan along with the RTL design (Figure 2). This enables SPARCS to accurately predict the performance of the FPGA implementation. DSS accepts a clock period constraint, and tries to minimize the maximum combinational delay of any register transfer. The area constraint is satisfied by trying to minimize the size of both the datapath and the controller.

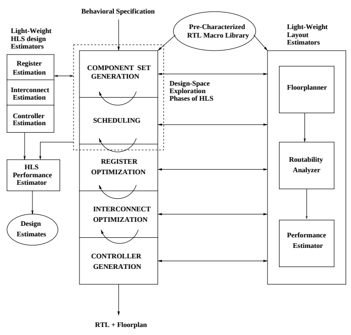

Fig. 2. HLS FlowGraph

The HLS process consists of component set generation, scheduling and performance estimation, register and interconnect optimization, and controller generation. For a detailed discussion of these phases, we refer the reader to Roy et al. [7]. For the purpose of this paper, we briefly discuss the scheduling and controller generation phases.

Scheduling and Performance Estimation: The scheduler [7] in DSS handles several constraints provided by the SPARCS system. The scheduler resolves memory access conflicts by scheduling operations that access the same memory in different time steps. Also, if the dataflow graph has user-specified critical regions, a time-constrained scheduling is performed and the estimated resources are checked with those available in the RTL component set. The SPARCS system also has a global scheduler that tries to resolve memory conflicts across tasks assigned to different FPGAs. After the scheduling phase, the SPARCS system estimates the performance of the design in one of two ways: (1) The HLS performance estimator can be invoked to get the rough design estimates; and (2) The complete RTL design can be estimated using the light-weight HLS design estimators. The estimated RTL design along with the macro library can be provided to the light-weight layout estimators to get the design estimates. The second approach is slower than the first but provides more accurate estimates.

Controller Generation: The controller for each FPGA is conceptually organized as a collection of communicating synchronous Finite State Machines (FSMs) each corresponding to a VHDL process (task). There is a privileged FSM called the root FSM that controls the execution of all other FSMs. This controller model directly facilitates the resolution of memory access conflicts between tasks. The root FSM also generates a *finish signal* after all other FSMs have reached their finish states. In order to let the host computer know the completion of a temporal segment, the HLS tool synthesizes logic in the form of an and-gate chain, across all FPGAs that tie up the individual *finish signals* to provide the *done signal* for the entire board. The HLS tool of the SPARCS system produces a datapath and controller pair for each FPGA assuming a synchronous clock for all the FPGAs on the board.

The HLS tool can be used in a lighter form in order to obtain area and performance estimates on one or more possible RTL implementations. This would involve invocation of only the initial design space exploration phases (refer Figure 2) of HLS. The *light-weight* HLS *estimator* always over-estimates the design performance, ensuring that the actual HLS process will generate only a better RTL implementation. Also, since the estimation process does not go through the entire (heavy-weight) HLS process, it will be considerably faster than the actual HLS.

6 Case Study

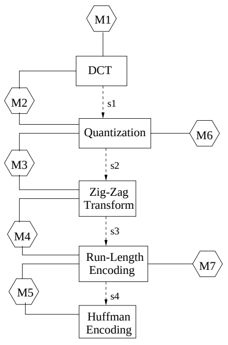

Fig. 3. JPEG

For the case study, we consider the Joint Photographic Experts Group (JPEG) [8] still image compression standard, shown in Figure 3. The five main tasks in the JPEG flow are: (1) Discrete Cosine Transform (DCT), (2) Quantization, (3) Zig-Zag Transformation, (4) Run-Length Encoding, and (5) Huffman Encoding. As a first step, the task graph of the JPEG algorithm, shown in Figure 3, was implemented in behavioral VHDL and thoroughly simulated. DSS[7] was then used to perform high-level synthesis on each individual task in the design. Following high-level synthesis, the RTL designs and their floorplans were taken through Xilinx place and route tools. The Huffman encoding task, due to its large size was transformed further into a collection of tasks, each requiring at most one FPGA.
A board consisting of 512 interconnect channels, 32KBytes of shared memory, and four XC4008 FPGAs with 4KBytes of local memory each, was considered for this experiment. Temporal partitioning of the JPEG task graph resulted in the first four tasks (DCT, quantization, zig-zag and run-length encoding) being mapped to the first temporal partition and tasks of Huffman Encoding to the second. For each of the temporal steps the spatial partitioner trivially assigned the tasks to one FPGA each, satisfying the architectural and routing constraints. The entire JPEG compression was successfully executed in two reconfigurations of the RC for several test images. The algorithm achieved an average compression factor of 30 times the original image size.

7 Summary

We presented an integrated design environment for automatically partitioning and synthesizing behavioral specifications onto multi-FPGA based reconfigurable computers. Approaches for temporal partitioning, spatial partitioning, and high-level synthesis geared towards reconfigurable architectures were presented. SPARCS system is continuing to develop towards handling large designs.

References

1. M. Vasilko and D. Ait-Boudaoud, "Architectural Synthesis Techniques for Dynamically Reconfigurable Logic", *FPL'96.*
2. M. Vootukuru, R. Vemuri, and N. Kumar, "Resource Constrained RTL Partitioning for Synthesis of Multi-FPGA Designs", *Proceedings of the 10th International Conference on VLSI Design, IEEE Press, 12 pages, 140-144, January 1997.*
3. M. Kaul and R. Vemuri, "Optimal Temporal Partitioning and Synthesis for Reconfigurable Architectures", to appear in *Design, Automation, and Test in Europe, February 98.*
4. Holland J., "Adaptation in Natural and Artificial Systems", *Ann Arbor: University of Michigan Press, 1975.*
5. Ram Vemuri, "Genetic Algorithms for Partitioning, Placement, and Layer Assignment for Multichip Modules", *PhD thesis, University of Cincinnati, USA, July 1994.*
6. V. Srinivasan, S. Radhakrishnan, and R. Vemuri, "Hardware/Software Partitioning with Integrated Hardware Design Space Exploration", *to appear in Design, Automation, and Test in Europe, February 1998.*
7. J.Roy, R.Dutta, N.Kumar, R.Vemuri, "DSS: A Distributed High-Level Synthesis System for VHDL Specifications", *IEEE Design and Test of Computers 1992.*
8. Gregory K. Wallace, "The JPEG Still Picture Compression Standard", *Communications of the ACM, pages 30–44, April 1991.*