An Enhanced Static-List Scheduling Algorithm for Temporal Partitioning onto RPUs

João M. P. Cardoso and Horácio C. Neto jmpc@acm.org, hcn@inesc.pt
University of Algarve/INESC, IST/INESC

Key words: Temporal Partitioning, Reconfigurable Computing, Dynamic Reconfiguration, Field-Programmable Gate-Arrays (FPGAs)

Abstract: This paper presents a novel algorithm for temporal partitioning of graphs representing a behavioral description. The algorithm is based on an extension of the traditional static-list scheduling that tailors it to resolve both scheduling and temporal partitioning. The nodes to be mapped into a partition are selected based on a statically computed cost model. The cost for each node integrates communication effects, the critical path length, and the possibility of the critical path to hide the delay of parallel nodes. In order to alleviate the runtime there is no dynamic update of the costs. A comparison of the algorithm to other schedulers and with close-to-optimum results obtained with a simulated annealing approach is shown. The presented algorithm has been implemented and the results show that it is robust, effective, and efficient, and when compared to other methods finds very good results in small amounts of CPU time.

1. INTRODUCTION

The availability of RPUs (reconfigurable processing units), such as the new FPGAs, with lower reconfigurable times and partial-reconfiguration capability, has made possible the concept of "virtual hardware" [1]: the hardware resources are supposed unlimited and implementations that oversize the RPU area are resolved by temporal partitioning. Then, the partitioned solution is executed by time-sharing the device such that the initial functionality is maintained. This concept promises to be an efficient solution to save silicon area. One of the applications is the switch among functionalities that

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35498-9_57
L. M. Silveira et al. (eds.), *VLSI: Systems on a Chip*

have mutual exclusiveness on the temporal domain, such as the context-switching between coding/decoding schemes in communication, video or audio systems. However, temporal partitioning algorithms able to exploit efficiently the new concept are needed. They must consider trade-offs among parallelism, communication costs, latency and reconfiguration times. The nodes of a given graph have to be scheduled in time slots to be executed in each temporal partition. Temporal partitioning must preserve the dependencies between nodes (that are already temporal dependencies) such that a node **B** dependent on node **A** cannot be mapped to a partition executed before the partition where node **A** was mapped.

Although, the FPGAs themselves, such as the Xilinx™ XC6200 family [2], do not have mechanisms to implement efficiently temporal partitions and the time of reconfiguration of the overall FPGA is still quite high, the importance of the "virtual hardware" concept has already been demonstrated with computationally complex applications [3]. Industrial efforts are under way to further improve the capability of the devices to handle multiple-configurations by storing several on-chip configurations and permitting the context-switching in few nanoseconds [4][5][6]. The trend to have on-chip configurations instead of more logic cells is explained by the fact that the area of SRAMs to store configurations for each cell is much lower than the cell itself.

Efficient mechanisms of communication between temporal partitions have also been actively researched such as the micro-registers in [5]. The majority of the efforts considers FPGA registers that maintain the same state between contexts whenever wanted.

As referred, research efforts are under way on both new RPU architectures [7] and on the automation of the temporal partitioning process. Our efforts address the temporal partitioning of behaviors during the synthesis steps. This paper presents a new temporal partitioning algorithm that effectively takes into account, among other aspects, the inter-communication costs, while maintaining a small computational complexity. Besides, it is sufficiently flexible to permit the consideration of various target architectures. Results are compared to a number of alternative constructive algorithms and, in order to show how far they are from close-to-optimum solutions, comparisons to a simulated annealing (SA) [8] approach are shown.

From now on we refer to temporal partitions and temporal partitioning simply as partitions or partitioning respectively, since this paper neither considers spatial partitions nor spatial partitioning.

The paper is divided in the following sections. Section 2 summarizes the related work on temporal partitioning. Section 3 describes the computational models considered by the approach proposed in this paper. Section 4 explains the algorithm and the heuristics used. Results are shown in section 5,

where the new algorithm is compared to simple heuristics and to results obtained by the SA implementation. Finally, conclusions are enumerated and future work is envisaged.

2. RELATED WORK

The development of temporal partitioning algorithms was firstly considered in [9][1]. The similarities of both scheduling on high-level synthesis [10] and temporal partitioning allow the use of common scheduling schemes for partitioning. However, an important factor that must be considered is the inter-communication (communication among partitions) cost, because it can impose an unacceptable overhead on the overall latency.

In [11] a static-list scheduling (SLS) approach is used for partitioning trying to minimize the number of nets among partitions. It works on the netlists (4-input-LUTs) of combinational circuits and uses the path-to-end's length of each node as a priority function and the size of the fan-out of each node as a tiebreaker. The approach is suitable to RPU architectures with inter-buffering (on-chip buffers that maintain the state among partitions). These RPUs have small inter-communication costs and the overall optimization problem resumes to the minimization of the critical path length. [12] presents an enhanced force direct scheduling algorithm which considers communication costs and which is able to process sequential circuits.

In [13], a variation of the SLS followed by an optimizer is used to perform partitioning of netlists. The algorithm uses three scheduling rules to select among the ready nodes and is tailored to the Time-Multiplexed FPGA [5].

[14][15] present a network-flow based method for multi-way partitioning of netlists. In [14] the algorithm is also targeted to the Time-Multiplexed FPGA and the results out-perform the SLS approach [13] in terms of communication costs (number of nets between partitions). The algorithm uses the max-flow min-cut computation iteratively to find k-partitions. [15] shows improvements over the enhanced force direct scheduling of [12] with respect to communication costs. Results comparing the latency of the solutions are neither presented nor examined.

The above approaches are all based on the netlist of the final circuit previously mapped to the library of the target FPGA. They can be efficient approaches to rapid prototyping but suffer from the impossibility to exploit partitions at the behavioral level. This has more importance when considering the integration of partitioning into the *reconfigware* compilation from behavioral descriptions. Moreover, these approaches suffer from the heavy number of nets and nodes that must be manipulated. At the behavior level

the operations encapsulate the tricky connections and only the groups of nets transporting operands are visible by the algorithms.

Some authors, such as in [9] and [16], have considered the partitioning at behavioral levels having in mind the integration of synthesis. In [9], a heuristic based on the SLS enhanced to consider dynamic area constraints is presented. The approach does not consider the inter-communication costs. In [16] the partitioning problem is modeled in a specified 0-1 non-linear programming (NLP) model [17]. Due to the computational complexity of the approach, heuristic methods must be developed to permit feasible executions on large input examples. In [18], a partitioning algorithm based on the levelized nodes obtained by the "as soon as possible" (ASAP) scheduling algorithm is used. The algorithm fills the available area of the RPU in the increasing order of the ASAP levels. The selection of nodes in the same level is arbitrary and the algorithm switches to another partition when it encounters the first node that does not fit in the current partition. In [19], partitioning algorithms based on the extension of the ASAP or "as late as possible" (ALAP) leveling algorithms with the selection of a node, in the same ASAP or ALAP level, by a local priority function based on the nodes' mobilities have been considered. [19] also shows an algorithm that searches recursively in the list of ready nodes so that if a node cannot be mapped to the current partition, other nodes can be considered.

However, all the above approaches do not consider both the inter-communication costs and the latency, and the majority of them are tailored to a specific target architecture. Therefore, new efforts to integrate the inter-communication costs and the latency of the solutions in a temporal partitioning algorithm working at the behavioral level are presented in this paper.

3. RECONFIGURABLE COMPUTING MODEL

Our model assumes that the CPU has access to the *reconfigware* and is responsible for the reconfiguration of the RPU(s) that integrate the *reconfigware* part. We also consider without loss of generality that the CPU has access to the memory attached to the RPU. The partitions are mapped to the *reconfigware* and the CPU store/load primary input/outputs directly to/from the memory, or the FPGA when this is supported. The data transfered between partitions can be stored in the memory by the *reconfigware*, in special registers that are maintained during reconfigurations, or collected by the CPU.

At least three schemes of interface mechanisms between partitions can be considered. The use of registers and a task running on the microprocessor to load and store operands between partitions is one possibility. This scheme is

well suited to boards of RPUs with a processor or a micro-controller that can also control the reconfiguration of partitions, or to boards without any buffer scheme to maintain results among partitions. With the advent of the integration of RPUs in processor cores this can be an efficient scheme because of the lower communication overhead on these systems. The second scheme is the use of a set of registers in the RPU when the device can be partially reconfigured. The registers will be configured once in a region of the RPU and shared between partitions (this can include the advent of new RPUs tailored to time-sharing). The third scheme uses a macro-cell to access memory locations controlled by a hardware unit. This is well suited to types of RPUs where there is no processor (only local memory) or the communication between the host and the RPUs is time-consuming.

From the above considerations it is clear that feasible partitioning schemes must consider different inter-communication costs due to different interface mechanisms.

4. PROBLEM FORMULATION

In order to be independent from a particular software language (e.g., C/C++ or Java) the input program is represented as a hierarchy of PDGs (program dependence graphs) in which the bottom level is formed by DAGs (directed acyclic graphs) where each node represents an operation. A node in the PDG can be a group of statements or a single statement. A loop is represented by a special node in the PDG which encapsulates the PDG of the loop body. Herein, we only consider nodes with deterministic delays (known at compile time), and recursive constructions are not allowed. Thus, a behavioral description is represented by a graph, G = (V, E), which is an ordered, directed and acyclic graph with |V| nodes, $\{v_1,v_2,\ldots,v_{|V|}\}$ and |E| edges, where each node v_i represents a single behavior. Each edge $e_{i,j} \in E$ represents a dependency between nodes v_i and v_j. A dependency can be only a precedence-dependency or a transport-dependency due to the transport of data between the two nodes.

The communication cost associated with an edge $e_{i,j}$ representing a transport-dependency is calculated by the number of bytes to transfer ($d_{i,j}$) divided by the maximum bandwidth of each atomic transfer (with the result rounded to the next big integer). In the majority of the communication mechanisms for each connection between different partitions the data must be stored by the partition that defines it and loaded by the partition that uses it. Non-transport dependencies (precedence-dependencies) have a zero $d_{i,j}$.

We assume that an estimation of the execution time and the *reconfigware* size (CLBs, cells, FUs, etc.) of each node is available at compile time.

The objective of the partitioning algorithms is to create partitions in the temporal domain such that the cost is minimum and that each partition fits in the *reconfigware* area physically available. Each partition Π_i is a non-empty subset of V. A graph G partitioned in k subsets is correct if:

- The set of partitions $\wp = \Pi_1 \cup \Pi_2 \cup \ldots \cup \Pi_k \supset V$.
- $\forall \Pi_i \in \wp$, Area(Π_i) $\leq$ MaxArea; Each temporal partition fits in the RPU resources.
- $\forall e_{i,j} \in E$, $\wp(v_i) \rightarrow \wp(v_j) \vee \wp(v_i) \equiv \wp(v_j)$; $\rightarrow$ indicates the order of execution. All dependencies are met (necessary condition to obtain the same functionality).

A correct set of partitions guarantees the same behavior of the original graph. However, we are also interested on the minimization of the overall latency. The cost that reflects the latency of a graph in a time-multiplexed RPU can be estimated by the equation (1). τ_{cycles} is the number of clock cycles for each load/store. In(Π_i) and Out(Π_i) correspond to the number of inter-communications without considering the primary input/outputs. The second term represents the overall critical path delay without inter-communication costs.

$$\Omega(G, RWModel_A) = \sum_{i=0}^{NumPart-1} \left[\tau_{cycles} \leftarrow \left[In(\Pi_i) + Out(\Pi_i)\right]\right] + \sum_{i=0}^{NumPart-1} \left[Max_{Delay}(\Pi_i)\right] \quad (1)$$

5. THE ENHANCED-STATIC LIST SCHEDULING ALGORITHM

The partitioning algorithm proposed (ELS), as can be seen in *Figure 1*, is an extension of the SLS algorithm. It starts by computing the ASAP and ALAP values of the nodes in the graph. Then a cost computed with equation (2) is assigned to each node of the input graph. Each term of the equation has a multiplication factor to give more weight to the communication cost (α), to the critical path (β), or to a tradeoff between them (η). The first term (3) gives emphasis to the communication costs. A large difference between the input and output edges of a node assigns a greater priority to that node. Also, a large number of nodes from a given node to the sink can produce more communication costs. The middle term (4) tries to give emphasis to the existent parallelism by assigning more weight to the nodes with lower ASAPs (giving the opportunity to place ready nodes in parallel to the nodes of the critical path already scheduled). This factor increases with the decreasing of the communications' weight. The third term (5) is the starting fine-grain ALAP of each node and permits to sort the nodes by ascending order of their

ALAPs. The scale factor is set to the ratio of the maximum number of levels of the ASAP leveling and the delay of the critical path. Experimentally we have found that the weight η expressed by $\beta/(\alpha+1)$ generally conduces to good results. However, another independent weight instead of the previous expression can be used to unconstrain the exploitation.

$$W[v_i] = \alpha \leftarrow \psi_{comm}(v_i) + \eta \leftarrow \psi_{comm/delay}(v_i) + \beta \leftarrow \psi_{delay}(v_i) \tag{2}$$

$$\psi_{comm} = scale \leftarrow [In(v_i) - Out(v_i) - MaxLevels + LevelALAP(v_i)] \tag{3}$$

$$\psi_{comm/delay} = -scale \leftrightarrow ASAP_{start}(v_i) \tag{4}$$

$$\psi_{delay} = -scale \leftrightarrow ALAP_{start}(v_i) \tag{5}$$

```
ESL(MaxArea, G(V, E), α, β, η) {
  Compute ASAP(G); // compute the fine-grain ASAP for each node
  Compute ALAP(G); // compute the fine-grain ALAP for each node
  int CurrentArea=0, SchedNum=0, NumNode=0, NumSchedNodes=0;
  for each v_i ∈ V { Cost_vi=ComputeCost(v_i, α, β, η);} // compute the cost for each node
  SortedList = Create a sorted List of ready nodes to be scheduled; // based on the Costs
  BitSet SchedNodes = new BitSet(|V|); // all nodes marked as unscheduled
  while (NumSchedNodes < |V|) { // while there are unscheduled nodes
    Node A = SortedList.ElementAt(NumNode); // peek a node
    Boolean Fit = ((CurrentArea + Area(A)) <= MaxArea);
    if (Fit) { // the node A fits in the current temporal partition
      SortedList.removeElement(A); // remove A from the SortedList
      // schedule A in the partition, update the current area and mark A as scheduled
      ScheduleElement(A, CurrentArea, SchedNum, SchedNodes);
      NumSchedNodes++; // increment the number of scheduled nodes
      if (SortedList.Update(G, A, SchedNodes)) NumNode = 0;
      else if (NumNode ≥ SortedList.size()) NumNode—;
    } else {
      if (NumNode < SortedList.size()-2) { // try another node in the SortedList
        NumNode++;
      } else {
        SchedNum++; NumNode = 0; CurrentArea = 0; // another temporal partition
      }
    }
  }
}
```

Figure 1. The enhanced static-list scheduling algorithm.

The next step of the algorithm is to compute the nodes ready to be mapped to the current partition and to sort them by descending order of the

costs. Then, the algorithm starts the loop by considering each node of the sorted list of ready nodes. The algorithm checks if a node can be mapped. If the node was mapped the algorithm tries to update the list of ready nodes by considering the sink nodes. Then the algorithm starts the loop considering another node. If a node cannot be mapped to the current partition the algorithm searches for a feasible node on the list before the creation of a new partition.

The ASAP and ALAP scheduling algorithms compute with graph traversal and have a runtime complexity of $O(|V|+|E|)$. The ELS algorithm has a worst-case runtime complexity of $O(|V|^2+|E|)$.

To have an idea of how much improvement can be obtained an SA approach has been implemented. It starts from a feasible partitioning solution and tries to improve it by moving nodes (considering the probabilistic selection of randomly valid moves) among adjacent partitions. Moves that can violate the maximum area available on the destination partition, but do not violate any temporal precedence, are considered valid. The approach can exploit results considering more partitions than those of the initial solution by adding empty partitions in the beginning of the execution.

6. RESULTS AND COMMENTS

All the algorithms presented have been implemented with the Java™ language. To permit a statistical comparison a random graph generator has also been implemented.

All the results attributed to SA are close-to-optimum (the best result of several executions with different parameters was collected). Herein Ω is computed with the equation (1) in clock cycles units and Δ_{comm} and Γ_{exec} correspond to the 1st and 2nd terms respectively. All the results neither consider the store/load of primary input/outputs nor the possibility to interleave execution and inter-communications. $\mathbf{S_1}$ refers to the algorithm presented in [18] and $\mathbf{S_2}$ refers to the leveling of the nodes by the ALAP scheme. $\mathbf{S_3}$ and $\mathbf{S_4}$ refer to the algorithm presented in [19] oriented by the ASAP or ALAP levels respectively. $\mathbf{S_5}$ refers to a version of $\mathbf{S_2}$ with the nodes sorted by the ascending ALAP levels and the ascending ASAP levels as a tiebreak. $\mathbf{S_6}$ is a version of $\mathbf{S_1}$ with a list created with the nodes of an ASAP level sorted by the ascending order of their ALAP step time. $\mathbf{S_7}$ refers to an SLS approach with the nodes in the list sorted by the ascending order of their ALAP step time.

In *Table 1* and *Table 2* **E** is the relative improvement cost of the SA over the ELS solution. The constructive approaches obtained each solution in less than 1ms.

The 1st example to be considered is the loop body of the HAL example [20] (all operands with 16-bit width). The example has a total area of 4,384 cells and a critical path delay of 58 cycles. The results presented in *Table 1* show small improvements of the SA over the ELS.

The 2nd example is the AR filter [21]. It has 16 multipliers and 12 adders contributing to a total area of 16,960 cells and a critical path delay of 90 cycles (all operands with 16-bit width). The results are shown in *Table 2*.

Table 1. Results for the HAL example (τ_{cycles}=2; **A**: MaxArea = 2,457; **B**: MaxArea = 4,096).

CASE/#Part	Measure	S_1	S_2	S_3	S_4	S_5	S_6	S_7	ELS, α=2	SA time	E (%)
A/5	Ω	70	86	86	80	80	70	80	66, β=20	62 7.8 s	6.1
B/2		66	86	66	84	84	66	84	66, β=1	60 5.7 s	9.1

Table 2. Results for the AR filter (**A's**: MaxArea =4,096; **B's**: MaxArea =16,384).

CASE/#Part	τ_{cycles}	Measure	S_1	S_2	ELS	(α,β)	SA time	E (%)
A1/5	2	Ω	222	202	202	(2,20)	194 26.4 s	3.9
B1/2	2	Ω	134	142	134	(2, 1)	98 90 s	26.8
A2/5	1	Ω	186	166	166	(1, 1)	162 25.4 s	2.4
B2/2	1	Ω	122	126	122	(1, 1)	94 85 s	22.95
A3/5	0	Γ_{exec}	150	130	118	(0, 1)	118 26 s	0
B3/2	0	Γ_{exec}	110	110	110	(0, 1)	90 68 s	18.18
A4/5	1	Δ_{comm}	36	36	26 (13 edges)	(1, 0)	22 19.2 s	15.38
B4/2	1	Δ_{comm}	12	16	6 (3 edges)	(1, 0)	2 66.7 s	66.7

ELS has produced results always better or equal than those obtained by the other heuristics. An accurate analysis has shown that SA has the capability to map nodes with many connections in the same partition reducing the inter-communications. The ELS approach is unable to balance the last two partitions. This problem has more impact when the number of needed partitions is small and is one of the disadvantages of constructive approaches. The results were not improved by making the SA exploit more partitions than those obtained by the constructive approaches. Also, results from experiments with random graphs confirm that the number of partitions used by the heuristics (e.g., ELS) is close-to-optimum and few cases need more partitions (one) to improve the results. More partitions have higher probability to increase the critical path delay and only in few cases can reduce the overall inter-communication cost.

Table 3 shows results obtained with random graphs. All the graphs have 50 nodes and the output edges of each node vary between 0 and (4 or 10). For each algorithm the median of relative improvements over the ASAP leveling method for different inter-communication's weights are shown. The pairs of rows (2, 3) and (4, 5) present results considering 2 and 1 clock cy-

cles for each inter-communication cost respectively. The results shown in the 6th and 7th rows were obtained considering null inter-communication costs. The 8th and 9th rows show the results obtained when only inter-communication costs are considered. The last column indicates the median of the relative improvements of SA over ELS. The last row shows the median of each column. In these tests the SA has run over an initial solution obtained by ELS.

Table 3. Results for N=100 graphs randomly generated.

	Out-edges	τ_{cycles}	S_2	S_3	S_4	S_5	S_6	S_7	ELS (α,β)	SA	E_1 (%)
E/N (%)	[0 10]	2	-0.9	9.1	1.1	-2.2	-1.4	9.2	**12.2** (2,1)	29.7	19.9
	[0 4]	2	-1.0	12.4	1.7	-3.1	-0.8	14.5	**18.5** (2,1)	38.1	24
	[0 10]	1	-0.3	5.8	0.2	-0.7	0.7	**8.7**	6.8 (1,1)	22.9	17.3
	[0 4]	1	-0.3	7.4	0.7	-1.1	3.3	**13.7**	10.6 (1,1)	28	19.5
	[0 10]	0	2.8	-8.6	-3.5	5.4	8.6	6.0	**9.5** (0,1)	13	3.9
	[0 4]	0	2.2	-11.1	-2.9	5.8	**16.2**	10.8	10.9 (0,1)	16	5.7
	[0 10]	1	-2.2	15.3	2.7	-5.1	-5.3	10.3	**29.8** (1,0)	45	21.7
	[0 4]	1	-2.5	22.9	-3.7	-7.1	-9.8	16.0	**48** (1,0)	57.8	18.8
Tot.	-	-	0.28	6.65	-0.46	-1.0	1.4	11.2	18.3	20.6	16.4

The results show improvements of ELS over the other considered heuristics. On the 4th and 5th rows ELS results would be similar to the S_7 results if only the third term of equation (2) was used. S_3 was the best algorithm of the ASAP/ALAP leveling approaches with respect to communication. The results show better solutions of SLS over the ASAP/ALAP approaches with respect to latency, as was expected due to the opportunity to execute freedom nodes in parallel with the critical path. S_6 has produced the best results of the constructive approaches when only latency has been considered. The fact is due to the consideration of all the nodes of an ASAP level before the addition of nodes from the next level (an SLS tries, for each node scheduled, to update the list of the ready nodes). When the cost of each communication was not high, S_7 has produced better results than the other constructive algorithms (rows 4 and 5).

The close-to-optimum results had about 16% of average relative improvement over the ELS (from 4 to 24%). Better results of ELS can be achieved by exploiting the values of the α, β and η weights on equation (2).

7. CONCLUSIONS & FUTURE WORK

In this paper temporal partitioning techniques have been presented and compared. A novel heuristic extension to the static-list scheduling algorithm

was presented. Being an algorithm of the family of the scheduling algorithms it can embody resource constraints without much more complexity. The low complexity of the algorithm makes it applicable on large graphs. The results show improvements over related algorithms and show that simplified algorithms can be used to resolve the temporal partitioning problem at the behavior level. The algorithm can be also applied efficiently to rapid prototyping of DFGs with much better results than the ASAP approach presented in [18] without increasing significantly the execution time.

Although just presented as a comparison term a temporal partitioning approach based on the simulated annealing has been also implemented. Based on the execution time the annealing approach does not seem, at least alone, to be a reasonable choice to *reconfigware* compilers where one of the most important objectives is fast compilation. However, efficient cooling schemes should be studied to improve the efficiency of the annealing.

The close-to-optimum experimental results show that most of times the optimal number of partitions is the minimum as was also stated in [7].

The support of the loop distribution transformation and the possibility to deal with resource sharing will be considered by future temporal partitioning schemes.

ACKNOWLEDGEMENTS

This work has been partially supported by the PRAXIS XXI Program under the scope of Project PRAXIS/2/2.1/TIT/1643/95 and by the Ph.D. program under the PRODEP 5.2 action. The authors would also like to thank Mário P. Véstias for the fruitful comments about the annealing implementation.

REFERENCES

1. X.-P.Long, H. Amano, "WASMII: a Data Driven Computer on a Virtual Hardware," in *Proc. 1st IEEE Workshop on FPGAs for Custom Computing Machines*, Napa Valley, CA, USA, April 5-7, 1993, pp. 33-42.
2. Xilinx Inc., *XC6000 Field Programmable Gate Arrays*, version 1.10, April 24, 1997.
3. R. D. Hudson, D. I. Lehn, and P. M. Athanas, "A Run-Time Reconfigurable Engine for Image Interpolation", in *Proc. 6th IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, USA, April 15-17, 1998.
4. S. M. Scalera, J. R. Vázquez, "The Design and Implementation of a Context Switching FPGA," in *Proc. 6th IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, USA, April 15-17, pp. 78-85, 1998.
5. S. Trimberger, D. Carberry, A. Johnson, J. Wong, "A Time-Multiplexed FPGA," in *Proc. 5th IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, USA, April 16-18, pp. 22-28, 1997.

6. T. Fujii, et al, "A Dynamically Reconfigurable Logic Engine with a Multi-Context/Multi-Mode Unified-Cell Architecture," in *Proc. IEEE Int'l Solid State Circuits Conference*, SA, CA, Feb. 15-17, 1999. See <http://www.nec.co.jp/english/today/newsrel/9902/1502.html>.
7. A. DeHon, *Reconfigurable Architectures for General Purpose Computing*, PhD Thesis, AI Technical Report 1586, MIT, 545 Technology Sq., Cambridge, MA02139, Sept. 1996, <http://www.ai.mit.edu/people/andre/phd.html>.
8. S. Kirkpatrick, C. Gelatt, Jr., and M. Vecchi, "Optimization by Simulated Annealing," Science, Vol. 220, No. 4598, May 1983, pp. 671-680.
9. M. Vasilko, D. Ait-Boudaoud, "Architectural Synthesis Techniques for Dynamically Reconfigurable Logic," in *Proc. 6th Int'l Workshop on Field-Programmable Logic and Applications*, Darmstadt, Germany, Sept. 23-25, 1996, LNCS, vol. 1142, Springer-Verlag, 1996, pp. 290-296.
10. G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw Hill, 1994.
11. D. Chang, M. Marek-Sadowska, "Buffer Minimization and Time-multiplexed I/O on Dynamically Reconfigurable FPGAs," in *Proc. 5th ACM Int'l Symposium on FPGAs*, Monterey, CA, USA, 1997, pp. 142-148.
12. D. Chang, M. Marek-Sadowska, "Partitioning Sequential Circuits on Dynamically Reconfigurable FPGAs," in *Proc. 6th ACM Int'l Symposium on FPGAs*, Monterey, CA, USA, February 22-24, 1998.
13. S. Trimberger, "Scheduling Designs into a Time-Multiplexed FPGA," in *Proc. 6th ACM Int'l Symposium on FPGAs*, Monterey, CA, USA, February 22-24, 1998.
14. H. Liu and D. F. Wong, "Network flow based circuit partitioning for time-multiplexed FPGAs," in *Proc. ACM/IEEE Int'l Conference on CAD*, San Jose, CA, USA, Nov., 1998.
15. H. Liu and D. F. Wong, "Circuit partitioning for dynamically reconfigurable FPGAs," in *Proc. 7th ACM Int'l Symposium on FPGAs*, Monterey, CA, USA, Feb 21-23, 1999.
16. I. Ouaiss, et al., "An Integrated Partioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures," in *Proc. 5th Reconfigurable Architectures Workshop*, Orlando, Florida, USA, March 30, 1998.
17. M. Kaul, R. Vemuri, "Optimal Temporal Partitioning and Synthesis for Reconfigurable Architectures," in *Proc. Design, Automation & Test in Europe,* Paris, France, Feb. 23-26, 1998, pp. 389-396.
18. K. M. GajjalaPurna, and D. Bhatia, "Partitioning in Time: A Paradigm for Reconfigurable Computing", in *Proc. IEEE Int'l Conference on Computer Design*, Austin, Texas, USA, October 5-7, 1998, pp. 340-345.
19. J. M. P. Cardoso, and H. C. Neto, "Macro-Based Hardware Compilation of Java™ Bytecodes into a Dynamic Reconfigurable Computing System, " in *Proc. 7th IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa Valley, CA, USA, April 21-23, 1999, Kenneth Pocek and Jeffrey Arnold, eds., IEEE Computer Society Press.
20. P. G. Paulin, J. P. Knight, and E. F. Girczyc, "HAL: A Multi-Paradigm Approach to Automatic Data Path Synthesis, " in *Proc. 23rd Design Automation Conference*, June 29-July 2, 1986, 263-270.
21. R. Jain, A. Parker, N. Park, "Module Selection for Pipelined Synthesis, " in *Proc. 25th Design Automation Conference*, 1988, 542-547.