Proceedings of the 5th WSEAS Int. Conf. on Electronics, Hardware, Wireless and Optical Communications, Madrid, Spain, February 15-17, 2006 (pp88-93)

An Incremental Temporal Partitioning Method for Real-Time Reconfigurable Systems

HAMID R. AHMADIFAR, FARHAD MEHDIPOUR*, MORTEZA S. ZAMANI*, MEHDI SEDIGHI* and KAZUAKI MURAKAMI**

Engineering Faculty
Guilan University,
Rasht, p.b. 41635-1844
IRAN

*IT and Computer Engineering Department
Amirkabir University of Technology
#424 Hafez Ave.,Tehran
IRAN

**Department of Informatics, Graduate School of Information Science and Electrical Engineering
Kyushu University
6-1 Kasuga-Koen, Kasuga,
Fukuoka 816-8580
JAPAN

Abstract: In this paper, a temporal partitioning algorithm is presented which partitions data flow graphs in a real-time domain. Timing constraint is a critical factor in temporal partitioning of real-time reconfigurable design. An incremental algorithm is presented to partition data flow graphs while meeting the timing constraints by obtaining the target number of partitions. In addition, the proposed algorithm attempts to minimize the logic resources used for implementing the real-time application. In this algorithm, selecting the appropriate nodes and moving them between subsequent partitions results in more area balanced partitions and less number of partitions.

Key-Words: Reconfigurable computing system, Temporal partitioning, Real-time system, Data flow graph.

1 Introduction

Recently, field-programmable gate arrays (FPGA) have played an important role in the realization of real-time reconfigurable applications. We discuss here the partitioning problem for run-time reconfigurable systems (RTR). In the task of implementing an algorithm on reconfigurable hardware, two approaches could be distinguished [1]. In the first case, we have to fit an algorithm with an optional time constraint in an existing system made from a host CPU connected to a reconfigurable logic array. In this case, the goal of an optimal implementation is to minimize one or more of the following criteria: processing time, memory bandwidth, number of reconfigurations and power consumption. In the second case, we have to implement an algorithm with a required time constraint on a system throughout the design exploration phase. The design parameter is the size of the logic array that is used to implement the data-path part of the algorithm. Here, an optimal implementation is the one that leads to the minimal area of the reconfigurable array.

Previous works in the field of temporal partitioning and synthesis for RTR architecture [2–10] assume that the reconfigurable resources are limited. In this case, the goal is to minimize the processing time and/or the memory bandwidth requirements. Among the previous works, there is a GARP project called GARP [2]. The goal of GARP is the hardware acceleration of loops in a C program by the use of a data path synthesis tool namely GAMA [3] and the GARP reconfigurable processor. GARP is a processor tightly coupled to a custom FPGA-like array and designed specially to speed-up the execution of general case loops. The logic array has a DMA feature and is tailored to implement 32-bit wide arithmetic and logic operations with control logic. All this allows to minimize the reconfiguration overhead. GAMA is a fast mapping and placement tool for the data-path implementation

on FPGAs. It is based on a library of patterns for all possible data-path operators.

SPARCS [4,5] is a CAD tool developed for application development on multi-FPGA reconfigurable computing architectures. Such architectures need both spatial and temporal partitioning; a genetic algorithm is used to solve the spatial partitioning problem. The main objective used here is the data memory bandwidth. Other works propose a strategy to automate the design process that considers all possible optimizations (partitioning and task scheduling) that can be carried out from a particular reconfigurable system [6,7]. Shirazi et al. and Luk et al. [8,9] proposed both a model and a methodology to take advantage of common operators in successive partitions. We proposed a similarity-based temporal partitioning algorithm which integrated temporal partitioning and physical design to address the long reconfiguration overhead time [10].

In this paper, we propose an incremental temporal partitioning which considers the real-time domain time constraints and the amount of logic resources used to implement configurations. First we try to find the minimal area that can meet the timing constraint. This is different from searching the minimal memory bandwidth or execution time to meet the resources constraint.

In Section 2 of this paper, the problem formulation is presented. Section 3 explains the temporal partitioning algorithm proposed for partitioning of a data flow graph considering the real time constraint. In Section 4, details of our proposed temporal partitioning algorithm are presented. Section 5 discusses experimental results and finally, Section 6 concludes the paper.

2 Problem Formulation

Temporal partitioning can be stated as partitioning a data flow graph into a number of partitions such that each partition can fit into a target device and also, dependencies among the graph nodes are not violated. Currently, the temporal partitioning is often done at boundaries of algorithm operations [11]. Temporal partitioning algorithms usually take a data flow graph (DFG) as input. The proposed methods are often based on elementary arithmetic and logic operations of the algorithm (such as adder, subtractor, multiplier etc.). In other words, the nodes of DFG represent pre-designed modules in a library.

Temporal partitioning leads to a register transfer level (RTL) decomposition of the data flow graph (DFG). The partitioning for a real-time application could be considered as a time constrained resource allocation problem.

To formulate the problem, firstly, the algorithm can be modeled as an acyclic DFG where the set of vertices corresponds to the arithmetic and logical operators and the set of directed edges represents the data dependencies between operations. Secondly, the application has a critical time constraint T. The problem to be solved is:

For a given FPGA family, one must find the set of partitions or sub-graphs of DFG where: $\sum_{i=1}^{n} P_i = DFG$ (n is the number of partitions and P_i is the ith partition) and the data dependencies modeled by the set of vertices and also requires the minimal amount of FPGA resources. Our method, which partitions a DFG of a real time application and attempts to use minimal area, is depicted in Fig. 1.

3 Temporal Partitioning for a Real-Time Application

In order to reduce the search domain, we first estimate the minimum number of partitions and the appropriate target device size. To do this, we use a library of modules based on the target device. The main constraint for real-time applications is the need for real-time processing. For a given FPGA device with reconfiguration time of μ_{Full} , the following inequality has to be satisfied:

$$T > (n-1)\mu_{Full} + T_P \qquad (1)$$

Where n is the number of partitions (n-1 is the number of reconfigurations), T is the upper limit of processing time (time constraint) and T_P is the total execution time of DFG. T_P could be determined by a full FPGA implementation of a DFG with no area constraint (Fig. 1).

Using the above inequality, the maximum number of partitions is obtained as follows:

$$n = \left\lfloor \frac{T - T_P}{\mu_{Full}} + 1 \right\rfloor \qquad (2)$$

In addition we obtain the corresponding minimum device size:

$$DeviceSize = \frac{DFGSize}{n} = \sum_{i=1}^{N} Size(Module(i)) \quad (3)$$

where N is the number of modules in DFG and *Size(Module(i))* is the number of CLBs [1]for *Module(i)* in the target-dependent library.

On the other hand, the minimum FPGA size should be larger than the size of the largest module in the library plus its required memory size. Therefore, the initial minimum partition size can be determined as follows:

$$DeviceSize = Max(\frac{DFGSize}{n}, Size(Module(k)) + Memory(Module(k)) \quad (4)$$

Where k is the index of the largest module in DFG and *Memory(Module(k))*is the required number of CLBs for transferring data between module k and its dependent descent modules in DFG.

From the analysis of the value of n, in order to realize the implementations, proper decision can be made, therefore different cases could be considered:

1) $n > 1$: This means that it is possible to realize a temporal partitioning. In this case, reduction of logic area is possible.

2) $n = 1$: In this case, only a full FPGA implementation without reconfiguration can meet the constraints.

3) $n < 1$: This means that the reconfigurable implementation of DFG cannot meet the time constraint.

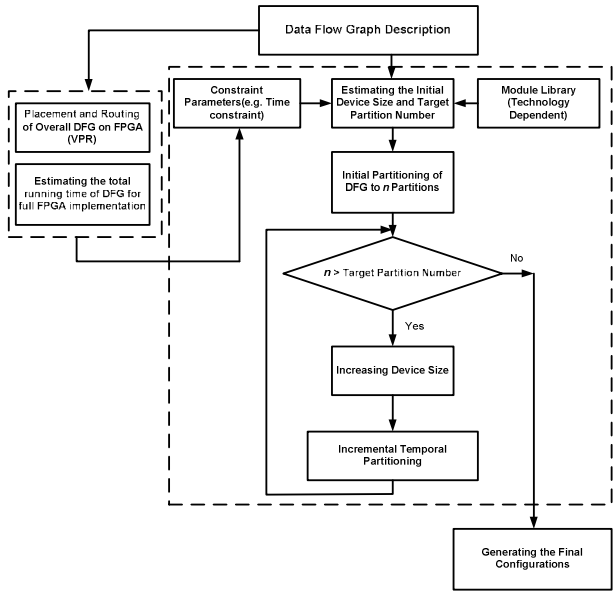

Fig.1. General outline of our incremental temporal partitioning method

[1]- The target FPGA contains a square array of logic blocks called configurable logic blocks (CLB's) embedded in a uniform mesh of routing resources.

4 Our Temporal Partitioning Algorithm

In a reconfigurable system, configurations are swapped in/out at run-time duration. The number of reconfigurations is an important factor to determine the overall run-time. Reconfiguration overhead time is usually the dominant time factor. We propose an incremental temporal partitioning algorithm for real-time reconfigurable applications. In this algorithm, first we determine the maximum number of partitions. Appropriate device size is determined with respect to the number of partitions. Then, an iterative partitioning process tries to perform the partitioning to obtain desired the number of partitions after an initial partitioning. This incremental algorithm increases the device size for the next iteration until the requested number of partitions is achieved. Finally, a post processing algorithm attempts to minimize the required device size. The general outline of our proposed method is depicted in Fig.1.

4.1 Initial Partitioning

In the first stage of the proposed method, an initial temporal partitioning algorithm is performed. In order to ensure that all computations will be performed correctly when the circuit is decomposed into stages, certain temporal constraints must be satisfied [12]. For example, a node can be executed if all of its predecessors have already been executed. In [12, 13], in the first stage, a level assignment is performed according to the as soon as possible (ASAP) algorithm. ASAP schedules a data flow graph in an attempt to minimize latency by topologically sorting of the nodes of the graph. In the partitioning stage of the DFG, the level number of modules, their sizes and the size of the target hardware are the most important factors which should be considered.

4.2 Incremental Temporal Partitioning

Initial partitioning of the DFG is done based on the required number of partitions which is the critical constraint in a real-time system. After the initial temporal partitioning, an incremental algorithm tries to achieve the desired number of partitions. Initial partitioning is done based on the initial partition size determined in the first stage and the target device size constraint. Therefore, it is possible that the number of partitions obtained is more that the requested number. In the second stage, the appropriate device size is

determined based on the initial partition size and the extra logic area to reach the target partition number. For example, if the number of partitions achieved from the initial current partitioning is n' and the desired number of partitions is n, the new device size of the next iteration can be computed as follows:

$$DeviceSize_{new} = DeviceSize_{current} + \frac{\sum_{i=n+1}^{n'} P(i)}{n} \quad (5)$$

In the next iteration, an incremental temporal partitioning is done by swapping modules between subsequent partitions. This algorithm attempts to increase the area balancing of partitions which results in less number of partitions. In the proposed incremental temporal partitioning algorithm, candidate nodes in each partition are first selected and then, they are moved to the previous partition until the area constraint is met. In addition, a post processing is done to minimize the device size used for the implementation. Our proposed temporal partitioning algorithm pseudo code is depicted in Fig. 2.

Temporal partitioning can be done iteratively which consume more compilation time. Using the incremental method takes much less time for generating the partitions.

Temporal Partitioning Algorithm for Real-Time Applications:

-Determine target number of partitions (n) according to Equation 2;
-Determine minimum and maximum node size (MaxNodeSize and MinNodeSize);
-Determine total size of DFG based on Equation 3 (DFGSize);
- DeviceSize= Initial Device Size (Equation 4);
*-Align DeviceSize to MinNodeSize (DeviceSize= (DeviseSize/MinNodeSize +1)*MinNodeSize);*
- Perform the Initial temporal Partitioning
- While (number of partitions obtained > n)
- Compute the new device size according to Equation 5
- Align DeviceSize to MinNodeSize;
- Perform the incremental temporal partitioning process and attempt to preserve the area balancing of partitions.

Incremental Temporal Partitioning Algorithm:

```
For i = 0 to number of partitions -1
  For j= i+1 to number of partitions
CandidatesNodeNo= FindCandidates(j) // This function returns the number of candidate nodes and a list of candidate nodes of partition j.
For k= 0 to CandiatesNodeNo
  Add kth candidate node from candidate node list to partition i
  Determine partition size of partition i
  if size of partition i is larger than DeviceSize
      remove the node currently added from partition i
FindCandidates (j){
For i= 0 to number of members of partition j
  if all parents of a node which is member of partition j are partitioned
      add the node to candidates list
      increase the candidates node number
return candidates node number and candidates nodes list
}
```

Fig. 2. Pseudo code for incremental temporal partitioning algorithm presented for real-time applications

5 Experimental Results

In our presented temporal partitioning algorithm the input is taken as a DFG. A library consisting of the required modules has been developed. Fig. 3 illustrates the CAD flow we used for generating the modules. At the beginning, each module was described in VHDL and was then synthesized by Leonardo Spectrum synthesis tool to obtain a structural description of the module based on logic gates. The SIS synthesis package [14] was used to perform technology-independent logic optimization of each module circuit. Next, each circuit was technology-mapped into 4-LUTs and flip flops by FlowMap [15]. The output of FlowMap is a netlist of LUTs and flip flops in .blif format. T-VPack [16,17] then packed this netlist of 4-LUTs and flip flops into more coarse-grained logic blocks, and generated a netlist in .net format. We used VPR [16] as a popular tool for placement and routing of the configurations.

The architecture of the target programmable device was chosen to be a Xilinx Virtex (XCV100) FPGA. VPR uses an architecture profile in which the architecture details can be specified. Table 1 shows some of the modules stored in the library and their sizes in terms of the number of CLB's used. To our knowledge, there is not any common use or standard benchmarks for static data flow graphs. We chose five static data flow graphs [12] and applied our tool to them from.

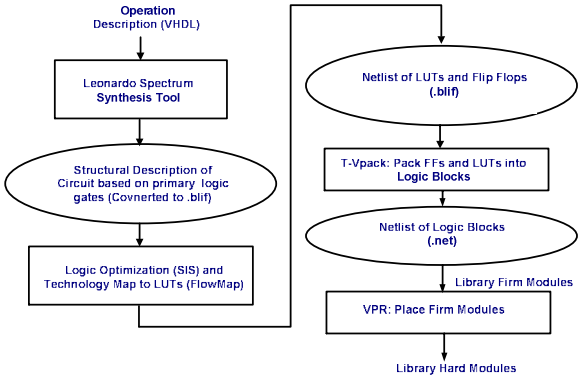

Fig. 3. Generating library cells in .net format

We implemented the proposed temporal partitioning algorithm and fed the data flow graphs to it. In our experiments, we assumed that the upper time limit to application execution is 30 microsecond and the time needed for full reconfiguration is 10 microsecond. Table 2 shows the results of the experiments. In this table, the total DFG size, the number of target partitions to be achieved, the initial device size, the initial number of partitions, the number of iterations for incremental algorithm, different device sizes during the incremental partitioning algorithm, final device size, average partitions size, and deviation from average partition size is depicted. We consider the deviation from average partition size as a balancing factor for the configurations generated. Smaller deviation represents the more balanced partitions which can result in less number of configurations.

Table 1. Some 16-bit operations and the number of CLB's used in Xilinx Virtex100 FPGA

Operation Type	No. of CLB's Used
ADD	11
SUB	10
XOR	4
CMP	5
MUX	4
MULT	195
ROT	4

6 Conclusion

Timing is the main constraint in real time implementation of a reconfigurable system. Temporal partitioning is an important stage of reconfigurable system design which attempts to partition a DFG by considering the timing and resource constraints and dependencies between nodes in the DFG. In this paper, we proposed an incremental temporal partitioning algorithm. This method starts with estimating the initial number of partitions and initial device size and then runs incrementally to achieve the target number of partitions to meet the real-time system constraint. In addition, this algorithm tries to obtain a minimum device size. The proposed incremental temporal partitioning algorithm increases the device size until the target number of partitions is obtained. This algorithm attempts to increase the balancing of partitions by moving the candidate nodes between subsequent partitions. This results in less deviation from average partition size and less number of partitions.

Table 2. Results of reconfigurable real-time implementations of DFGs

Data Flow Graph	DFG1	DFG2	DFG3	DFG4	DFG5
Total DFG size	1079	943	1124	1297	984
Number of target partitions	3	3	3	3	3
Initial device size	360	304	370	410	320
Initial number of partitions	5	4	5	5	5
Number of iterations	2	3	2	2	3
Device sizes during algorithm running	360, 510	304, 380, 456	370, 510	410, 555	320, 390, 470
Final device size	510	456	510	555	470
Average Partition Size	359	314	374	432	328
Deviation from average size (Balancing factor)	91	71	105	10	94

References

[1] X. Zhang, K.W. Ng, A review of high-level synthesis for dynamically reconfigurable FPGA's, *Microprocessors and Microsystems, Elsevier*, vol. 24, 2000, pp.199–211.

[2] T.J. Callahan, J. Hauser, J. Wawrzynek, The GARP architecture and C compiler, *IEEE Computer*, vol. 33, no. 4, 2000, pp. 62–69.

[3] T.J. Callahan, P. Chong, A. DeHon, J. Wawrzynek, Fast module mapping and placement for data paths in FPGA's, *Proceedings of the ACM/SIGDA Sixth International*

Symposium on Field Programmable Gate Arrays, Monterey, CA, February 1998, pp. 123–132.

[4] S.V. Srinivasan, S. Govindarajan, R. Vemuri, Fine-grained and coarse-grained behavioral partitioning with effective utilization of memory and design space exploration for multi-FPGA architectures, *IEEE Transactions on VLSI Systems*, vol. 9, no. 1, 2001, pp. 140–158.

[5] M. Kaul, R. Vemuri, Optimal temporal partitioning and synthesis for reconfigurable architectures, *International Symposium on Field-Programmable Custom Computing Machines*, April 1998, pp.312–313.

[6] R. Maestre, F. Kurdahi, M. Fernadez, R. Hermida, N. Bagherzadeh, H.Singh, A framework for reconfigurable computing: task scheduling and context manegement, *IEEE Transactions on VLSI Systems*, vol. 9, no. 6, 2001, pp. 858–873.

[7] M. Karthikeya, P. Gajjala, B. Dinesh, Temporal partitioning and scheduling data flow graphs for reconfigurable computer, *IEEE Transactions on Computers,* vol. 48, no. 6, 1999, pp.579–590.

[8] N. Shirazi, W. Luk, P.Y.K. Cheung, Automating production of runtime reconfiguration designs, in: K.L. Pocek, J. Arnold (Eds.), *Proceedings of IEEE Symposium on FPGA's Custom Computing Machines, IEEE Computer Society Press*, 1998, pp. 147–156.

[9] W. Luk, N. Shirazi, P.Y.K. Cheung, Modeling and optimizing runtime reconfiguration systems, in: K.L. Pocek, J. Arnold (Eds.), *Proceedings of IEEE Symposium on FPGA's Custom Computing Machines, IEEE Computer Society Press*, 1996, pp. 167–176.

[10] F. Mehdipour, M. Saheb Zamani, M. Sedighi, An Integrated Temporal Partitioning and Physical Design Framework for Static Compilation of Reconfigurable Computing System, *The International Journal of Microprocessors and Microsystems*, Elsevier, 2005, Accepted for publishing.

[11] C. Tanougast, Y. Berviller, P. Brunet, S. Weber, H. Rabah, Temporal partitioning methodology optimizing FPGA resources for dynamically reconfigurable embedded real-time system*,* *The International Journal of Microprocessors and Microsystems*, vol. 27, 2003, pp. 115-130.

[12] C. Bobda, Synthesis of dataflow graphs for reconfigurable systems using temporal partitioning and temporal placement, Ph.D thesis, Faculty of Computer Science, Electrical Engineering and Mathematics, University of Paderborn, 2003.

[13] G.D. Micheli, Synthesis and optimization of digital circuits, McGraw-Hill, 1994.

[14] Sentovich E M, SIS: A system for sequential circuit analysis, Tech. Report No.UCB/ERLM92/41, University of California, Berkeley, 1992.

[15] J. Cong, Y. Ding, Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs, *IEEE Transactions on CAD*, 1994, pp.1-12.

[16] V. Betz, J. Rose, A. Marquardt, *Architecture and CAD for deep-submicron FPGAs*, Kluwer Academic Publishers, 1999.

[17] V. Betz, VPR and T-VPack1 user's manual (Version 4.30), http://www.eecg.toronto.edu/~vaughn, 2000.