Current Trends in Resource Management of Reconfigurable Systems

Mojtaba Sabeghi, Koen Bertels
Computer Engineering Laboratory
Delft University of Technology
{sabeghi, k.l.m.bertels}@ce.et.tudelft.nl

Abstract — **Considering multiple applications on a system which are executing concurrently, there should be mechanisms and policies which manage the competition for resources between them and resolve the conflicts. In a traditional system, these management activities can be summarized as storage management for saving the required data and I/O management to interact with the outside world. Theoretic foundations of these activities have been fully explored in literature. In view of reconfigurable systems, additional management tasks would be imposed which include FPGA logic area allocation, placement, routing, and network on chip management. This paper presents those management activities.**

Index Terms — **Operating systems, Reconfigurable architectures, Resource management, Scheduling**

I. INTRODUCTION

Advances in reconfigurable computers (RC)[10] including field-programmable gate arrays (FPGA)[5] made them a practical computing platform for lots of computation demanding applications. The idea of FPGA has been first proposed in [14] but due to technological constraint, this idea could not be realized until recently.

An FPGA has been agreeably described as an array of uncommitted configurable logic blocks (CLBs) surrounded by a periphery of input/output blocks (IOBs), which are interconnected by configurable routing resources, whose configuration is controlled by a set of memory cells that lies beneath[16]. A brief introduction to reconfigurable computing that adequately covers all the aspects of FPGA technology can be found in[9].

There are several benefits of using reconfigurable systems. Reconfigurable architectures present an inherently good solution for applications consisting of a large number of small processing units. Another advantage of reconfigurability is the reusability of resources. Furthermore, it brings fault-tolerance. Reconfigurability also makes developing and testing hardware systems cheaper and faster. However, the most important benefit of them is the ability to use the hardware performance while retaining the flexibility of software[28].

Despite all of advantages of FPGA-based reconfigurable systems, application developers still decline to develop application for this platform because of the substantial problems involved. Their programming is cumbersome and required specialists using some difficult to understand programming languages like VHDL or Verilog. Programming is done at the gate level, that is, at the very lowest level of information processing with NAND and NOR gates[19]. The main problem arises when we want the hardware to meet the software. Hardware and software are developed using quite different models of computation. Systems that comprise a mixture of hardware and software are difficult to design because it is hard to relate C components to VHDL components [30].

One of the major trends toward solving this problem is to create compilers that can automatically detect the parts of the program that can be accelerated in hardware. These compilers then will produce both binary executable and a bitstream file for the reconfigurable fabric. There are many lines of research currently running on this trend [7, 8, 13, 17, 18, 26, 27, 35]

The most important drawback of this method is that the compiler should know the exact structure of the hardware and the produced bitstream is useful just for that hardware which means any changes in the underlying hardware will leads to a recompilation process[15].

These compilation tools, however, are usually tied to traditional placement and routing back-ends and have relatively slow compilation times. They also provide little or no run-time support for dynamic reconfiguration[25].

So far, researchers tried a lot to make use of the current programming languages and compilation tools in order to utilize the potential of reconfigurable devices. In most of these efforts, a single control threads has been assumed which is executing on general purpose processor (GPP) and controls the hardware modules. However, this model is mainly based on the coupling strategy between the GPP and

reconfigurable fabric. Generally, there are four different trends toward the coupling of the reconfigurable fabrics with the standard general purpose processor [1, 10].

First, the reconfigurable unit can only execute functional units on the main microprocessor datapath. Actually, the reconfigurable unit is considered as a custom instruction which is more powerful and complex than a normal instruction. There are some registers with which one can send parameters to the function and to receive the output [10].

The second trend considers the reconfigurable fabric as a coprocessor which is more independent than a functional unit. The GPP here initializes the coprocessor and provides the information on where the necessary data can be found in memory. This approach reduces the overhead in comparison with the first one. [20, 29, 39] are some of the researches following this approach.

In the third approach, reconfigurable unit is assumed to be as an additional processor. The communication between reconfigurable unit and GPP will be done through system primitives. This type of reconfigurable hardware allows a great deal of independent computation over reconfigurable device which means it can execute a large part of the application without GPP supervision. [24, 36] are among those project which investigate this trend. The last model considers the reconfigurable fabric as a standalone processing unit and can communicate with GPP using a network[10].

Each of these styles has distinct benefits and drawbacks. The tighter the integration of the reconfigurable hardware, the more frequently it can be used within an application or set of applications due to a lower communication overhead. However, the hardware is unable to operate for significant portions of time without intervention from a host processor, and the amount of reconfigurable logic available is often quite limited. The more loosely coupled styles allow for greater parallelism in program execution, but suffer from higher communications overhead. In applications that require a great deal of communication, this can reduce or remove any acceleration benefits gained through this type of reconfigurable hardware[9].

In this paper we will present the various management activities for reconfigurable computers. These activities can be carried out either statically by compilers and design tools or dynamically using operating systems or other runtime support mechanisms like virtualization.

II. RESOURCE MANAGEMENT

Considering multiple application on a system which are executing concurrently, there should be mechanisms and policies that manage the competition for resources between different applications and resolve the conflicts. In a traditional system, these management activities can be summarized as storage management for saving the required data and I/O management to interact with the outside world. Theoretic foundations of these activities have been fully explored in [31, 33, 34].

In view of reconfigurable systems, additional management activities are necessary some of which will be discussed in this paper.

A. FPGA Logic area allocation and relocation

Allocating one application to one FPGA device is the easiest way to tackle the allocation problem and has been investigated in RACE [32] and Dynamically Reconfigurable System [22] projects.

Brebner [3] was among the first who proposed an operating system approach for partially reconfigured hardware. He suggested the idea of swappable logic units (SLUs), which are position independent tasks that can be swapped in and out by the operating system. SLUs are FPGA logic segments (rectangles) of equal size which the application could be allocated to[4]. Since it is difficult to breakdown an application to the segments of equal size, Brebner proposed the SLUs with various rectangular dimensions[2]. This idea has been extended in [6] in a way that allows various geometric shapes.

When a new arriving task cannot be allocated immediately it might be possible that it can be placed onto the FPGA after a proper rearrangement of a subset of the executing tasks. In [12] three methods are proposed for finding such rearrangements. The goal is to increase the rate at which waiting tasks are allocated while minimizing disruptions to executing tasks that are to be moved. Two of the methods for finding a partial rearrangement are deterministic heuristics, which are referred to as local repacking and ordered compaction, while the third method is an evolutionary approach making use of a genetic algorithm.

Traditionally, the FPGA design involves static placement of the logic elements. To do that, the designer or the design tool should fix the location of the logic elements at the design time. Although, this approach results in a quick loading of the module but, it is obvious that it lacks the flexibility in case of faulty or occupied FPGA surface.

As an example, one can consider two partial configurations which were placed onto an overlapping physical location statically and they are repeatedly using one after the other at runtime. As it is obvious, it will reduce the performance and increase the reconfiguration overhead dramatically. A rearrangement may be necessary to get enough contagious space to efficiently implement incoming hardware modules.

Another problem that may arise during application execution is the FPGA surface area fragmentation[11]. Over-

time, as partially reconfigurable modules load and unload, the empty area of the FPGA may become fragmented and as a results, the maximum available size to be allocated for a hardware module decreases.

Compton et al. in [11] presents a hardware solution to provide relocation and defragmentation support with a negligible area increase over a generic partially reconfigurable FPGA, as well as software algorithms for controlling this hardware.

Gericota et al. proposed a novel active replication mechanism for configurable logic blocks (CLBs), able to impalement on-line rearrangement, defragmentation the available FPGA recourses without disturbing those functions that are currently running[16]. This has been done by the introducing the concept of the active CLB replication which means relocating the functionality of a given module to other CLBs without disturbing function execution.

B. Allocation Scheduling

After finding a suitable allocation, the task partitions should be schedule to be allocated in a way that minimize the total computation time. In [37] an online scheduling system was proposed that schedules tasks according to several non-preemptive and preemptive policies.

C. Routing

The routing between the logic blocks within the reconfigurable hardware is also of great importance. Routing contributes significantly to the overall area of the reconfigurable hardware. Yet, when the percentage of logic blocks used in an FPGA becomes very high, automatic routing tools frequently have difficulty achieving the necessary connections between the blocks. Good routing structures are therefore essential to ensure that a design can be successfully placed and routed onto the reconfigurable hardware.

Arbitrary relocated hardware modules need to communicate with each other and with I/O devices. To do that, an online routing mechanism is necessary to enable this communication. JRoute [23] is a set of java classes to route Xilinx FPGA devices.

D. Network on chip

Reconfigurable systems that are composed of multiple FPGA chips interconnected on a single processing board have additional hardware concerns over single-chip systems. In particular, there is a need for an efficient connection scheme between the chips, as well as to external memory and the system bus. This is to provide for circuits that are too large to fit within a single FPGA, but may be partitioned over the multiple FPGAs available. Because of the need for efficient communication between the FPGAs, the determining the inter-chip routing topology is a very important step in the design of a multi-FPGA system.

III. RUN-TIME REQUIREMENTS

With development of reconfigurable computers containing FPGAs with millions of systems gates, it is now feasible to consider the possibility of serving multiple concurrent applications executing on a shared logic area. This will improve the resource utilization and reduce the costs. However, it will increase the degree of complexity in order to manage the shared resources. Needless to say, dynamic and partial reconfiguration[21] are important factors in sharing the FPGA logic area and allow to take advantage of the hardware virtualization. Run-time reconfiguration provides the ability to change the configuration not only between applications, but also within a single application[11]. For example, applications that are not able to fit onto the fabric at once can be partitioned and to be loaded into the FPGA at different points in time.

Generally, to run an application on a specific hardware platform, one needs to have a run-time environment in which an execution environment plus some services and APIs has been provided.

A. Run-time Abstractions

The operating system kernel provides an execution environment in which application may run. Therefore the kernel must implement a set of services and corresponding interfaces. Applications use those interfaces and do not usually interact with hardware resources.

The kernel offers several subroutines or functions in user space, which allow the end-user application programmer to interact with the hardware.

The main abstraction is the *hardware task* which captures application functionality in as much as possible device-independent way[38].

B. APIs and Services

Here is a list of the APIs that the OS should provide to applications: Creating hardware/software tasks, Loader, Destroying hardware/Software tasks, Task to task communication, Suspend hardware/software task, Resume hardware/software task, Query Task states, Scheduling.

IV. SUMMARY

In this paper we presented the various resource management activates that should be done for a reconfigurable computer. System developer can address these issues either at run-time or compile time.

REFERENCES

[1] Barat, F. and Lauwereins, R. Reconfigurable Instruction Set Processors: A Survey. *Proceedings. 11th International Workshop on Rapid System Prototyping*.

2. Brebner, G. Automatic Identification of Swappable Logic Units in XC6200 Circuitry. in *Lecture Notes In Computer Science*, Springer, London, 1997, 173-182.

3. Brebner, G., The swappable logic unit: a paradigm for virtual hardware. in *Proceedings of 5th IEEE Symposium on FPGA-Based Custom Computing Machines*, (1997), IEEE Computer Society.

4. Brebner, G. A Virtual Hardware Operating System for the Xilinx XC6200 *6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, Springer-Verlag, 1996.

5. Brown, S., Francis, R.J. and Rose, J. *Field-Programmable Gate Arrays*. Springer, Boston, US, 1992.

6. Burns, J., Donlin, A., Hogg, J., Singh, S. and Wit, M.d. A Dynamic Reconfiguration Run-Time System *IEEE Symposium on FPGAs for Custom Computing Machines*, 1997.

7. Cardoso, J.M.P. and Neto, H.C. Compilation for FPGA-based reconfigurable hardware. *Design & Test of Computers, IEEE*, *20* (2). 65-75.

8. Cardoso, J.M.P. and Neto, H.C. Fast hardware compilation of behaviors into an FPGA-based dynamic reconfigurable computing system. *The XII Symposium on Integrated Circuits and System Design*. 150–153.

9. Compton, K. and Hauck, S. An Introduction to Reconfigurable Computing. *Invited Paper, IEEE Computer*.

10. Compton, K. and Hauck, S. Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys*, *34* (2). 40.

11. Compton, K., Li, Z., Cooley, J., Knol, S. and Hauck, S. Configuration relocation and defragmentation for run-time reconfigurable computing. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, *10* (3). 209-220.

12. Diessel, O., ElGindy, H., Middendorf, M., Schmeck, H. and Schmidt, B. Dynamic scheduling of tasks on partially reconfigurable FPGAs. *Computers and Digital Techniques, IEE Proceedings-*, *147* (3). 181-188.

13. Dimitroulakos, G., Kostaras, N., Galanis, M.D. and Goutis, C.E. Compiler assisted architectural exploration for coarse grained reconfigurable arrays. *Proceedings of the 17th great lakes symposium on Great lakes symposium on VLSI*. 164-167.

14. Estrin, G., Bussel, B., Turn, R. and Bibb, J. Parallel Processing in a Restructurable Computer System. *IEEE Transactions on Electronic Computers*, *12* (5). 747-755.

15. Fu, W. and Compton, K. An execution environment for reconfigurable computing *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2005.

16. Gericota, M.G., Alves, G.R., Silva, M.L. and Ferreira, J.M. On-line Defragmentation for Run-Time Partially Reconfigurable FPGAs. *Proc. 12th International Conference on Field Programmable Logic and Applications*. 302-311.

17. Goldstein, S.C., Schmit, H., Budiu, M., Cadambi, S., Moe, M. and Taylor, R.R. PipeRench: a reconfigurable architecture and compiler. *Computer*, *33* (4). 70-77.

18. Hammes, J., Rinker, B., Bohm, W., Najjar, W., Draper, B. and Beveridge, R. Cameron: High Level Language Compilation for Reconfigurable Systems. *Department of Computer Science, Colorado State University, Conference on Parallel Architectures and Compilation Techniques, Oct*. 12-16.

19. Hartenstein, R. Basics of Reconfigurable Computing. *Embedded Computing-A Low Power Perspective, Springer-Verlag*.

20. Hauser, J.R. and Wawrzynek, J. Garp: A MIPS Processor with a Reconfigurable Coprocessor *IEEE Symposium on FPGAs for Custom Computing Machines*, IEEE Computer Society, 1997.

21. Hutchings, B.L. and Wirthlin, M.J. Implementation Approaches for Reconfigurable Logic Applications *International Workshop on Field-Programmable Logic and Applications*, 1995.

22. Jean, J., Tomko, K., Yavgal, V., Cook, R. and Shah, J. Dynamic Reconfiguration to Support Concurrent Applications *"IEEE Symposium on FPGAs for Custom Computing Machines*, 1998.

23. Keller, E. JRoute: A run-time routing API for FPGA hardware *7th Reconfigurable Architectures Workshop*, Springer-Verlag, Mexico 2000.

24. Laufer, R., Taylor, R.R. and Schmit, H. PCI-PipeRench and the SWORDAPI: a system for stream-basedreconfigurable computing *IEEE Symposium on Field-Programmable Custom Computing Machines*, 1999.

25. Lechner, E. and Guccione, S.A. The Java Environment for Reconfigurable Computing. *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications, FPL*. 284-293.

26. Luk, W., Shirazi, N. and Cheung, P.Y.K. Compilation tools for run-time reconfigurable designs. *IEEE Symposium on Field-Programmable Custom Computing Machines*. 56–65.

27. Panainte, E.M., Bertels, K. and Vassiliadis, S. The Molen compiler for reconfigurable processors. *ACM Transactions on Embedded Computing Systems (TECS)*, *6* (1).

28. Radunovic, B. and Milutinovic, V.M. A Survey of Reconfigurable Computing Architectures. *Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications, From FPGAs to Computing Paradigm*. 376-385.

29. Rupp, C.R., Landguth, M., Garverick, T., Gomersall, E. and Holt, H. The NAPA Adaptive Processing Architecture *IEEE Symposium on FPGAs for Custom Computing Machines* 1998.

30. Satnam, S. Integrating FPGAs in high-performance computing: programming models for parallel systems -- the programmer's perspective *Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*, ACM Press, Monterey, California, USA, 2007.

31. Silberschatz, A. *Operating System Concepts 7th Edition*. John Wiley & Sons, 2005.

32. Smith, D. and Bhatia, D. RACE: Reconfigurable and Adaptive Computing Environment *International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, 1996.

33. Stallings, W. *Operating systems : internals and design principles*. Prentice-Hall, 2001.

34. Tanenbaum, A. *Modern Operating Systems, 2nd edition*. Prentice Hall PTR, 2001.

35. Venkataramani, G., Najjar, W., Kurdahi, F., Bagherzadeh, N., Bohm, W. and Hammes, J. Automatic compilation to a coarse-grained reconfigurable system-opn-chip. *ACM Transactions on Embedded Computing Systems (TECS)*, *2* (4). 560-589.

36. Vuillemin, J., Bertin, P., Roncin, D., Shand, M., Touati, H. and Boucard, P. Programmable Active Memories: Reconfigurable Systems Come of Age *IEEE Transactions on VLSI Systems*, *4* (1). 14.

37. Walder, H. and Platzner, M. Online scheduling for block-partitioned reconfigurable devices. *Design, Automation and Test in Europe Conference and Exhibition, 2003*. 290-295.

38. Walder, H. and Platzner, M. Reconfigurable Hardware Operating Systems: From Design Concepts to Realizations. *Proceedings of the 3rd International Conference on Engineering of Reconfigurable Systems and Architectures (ERSA)*. 284–287.

39. Wittig, R.D. and Chow, P. OneChip: an FPGA processor with reconfigurable logic *IEEE Symposium on FPGAs for Custom Computing Machines*, 1996.