A Virtual Layer for FPGA Based Parallel Systems (MP-SoCs)

Andreas Hofmann and Klaus Waldschmidt
J. W. Goethe-University, Technical Computer Sc. Dep.,
Box 11 19 32, D-60054 Frankfurt, Germany
E-mail: {ahofmann,waldsch}@ti.informatik.uni-frankfurt.de

published: Proceedings of the Work in Progress Session of 10th Euromicro DSD 2007

I. Introduction

Besides performance and time to market, robustness and reliability are important design targets for modern Systems-on-Chip (SoCs). Despite these features the power consumption must be as low as possible. To meet these design goals parallel, flexible, and adaptive architectures are required [1].

Today, dynamically reconfigurable FPGAs are well suited to form a parallel architecture because they incorporate serveral hard- and softcores. To efficiently use such multicore systems a hardware independent system must be created which handles all cores. Further, optimizing the power management the number of active cores must be adapted dynamically to the current workload. To make these features manageable and augment the system with adaptivity a virtual layer is required which hides the – due to runtime reconfiguration – changing hardware system from the application software. The Scalable Dataflow-driven Virtual Machine [2] is such a virtualization of a parallel, adaptive and heterogeneous cluster of processing elements (PE). Thus, it is well suited to serve as a managing firmware for multicore FPGAs.

II. A Firmware concept for FPGAs

Besides the primary functions that a System-on-Chip (SoC) should accomplish, e.g. speech encoding in a cell phone, their design has to address a multitude of secondary requirements. These requirements are important for most systems, merely the weighting differs. Using FPGAs as a target platform for SoCs adds an other important requirement: To make optimal use of these reconfigurable systems an efficient management of the reconfiguration process is necessary. The list of these secondary requirements can be summarized as follows:

- performance and scalability
- support for parallelism
- adaptivity, robustness and reliablity
- energy efficiency
- support for runtime reconfiguration
- incorporation of heterogeneous components

As these requirements and therefore the techniques to achieve them are common to a vast number of SoCs it is beneficial to supply a generic module which manages these features. To avoid an increase in complexity, provide flexibility, and improve portability the division of the functionality into several layers is a possible solution. The aforementioned generic module should therefore be implemented as a functional layer between the system hardware and the application software thus acting as a middleware.

The middleware should provide a complete virtualization of the underlying hardware. The application has no longer to be tailored to the hardware, instead it is sufficient to tailor it to the virtual layer. This virtual layer not only provides hardware independence, it can also hide changes in the underlying hardware due to reconfiguration. Such a middleware is specifically well suited to be used as a managing firmware for FPGAs.

The Middleware concept

A fundamental decision in the design process of the firmware is whether each PE on an FPGA forms an independent building block of the parallel cluster or multiple PEs are merged in a higher-order cluster element. The latter may impose less overhead but the former eases the implementation of adaptive features; in detail:

- coping with errors in the fabric
- reducing hotspots on the FPGA
- avoiding bottlenecks.

If each PE is augmented with a complete set of the virtualization functions and therefore no PE is the sole provider of any function, the system is much more flexible. If an error is detected in some part of the FPGA the affected PE can be disabled or reconfigured to avoid the erroneous location without hampering the functionality of the cluster. Furthermore, as each augmented PE provides its share of the cluster management functionality, the number of bottlenecks is reduced. The distribution of functionality can lead to a better distribution of workload thus reducing the number of hotspots.

The logic resources and therefore the computing power of the FPGA and the internal memory blocks can be distributed evenly among all PEs. But there are resources which cannot be efficiently split. The most important one is the external memory. As FPGAs typically have only up to some hundred kilobytes of internal memory, a lot of applications require external memory. Therefore, the middleware should support a multi-level memory architecture that is transparent to the application software.

Besides external memory every interface of the FPGA system to the outside world like ethernet or PCIe cannot be allocated to every PE. The middleware must manage these resources on the cluster level.

The middleware should provide a complete virtualization of runtime reconfigurable platform FPGAs. Therefore it has to support the following primary features:

- Combine all PEs on the FPGA to create a parallel system.
- Provide task mobility between all PEs even if they are heterogenous.

- Virtualize the I/O-system to enable the execution of a task on an arbitrary PE.
- Combine the distributed memory of each PE to form a virtually shared memory.
- Manage the reconfiguration of the FPGA.
- Adjust the number of active PEs at runtime.
- Hide the actual number of PEs from the application to ease programming.
- The firmware has to provide dynamic scheduling as well as code and data distribution.

III. Realization

In this section the realization of the presented virtualization concept is described which is under developement. Due to its features which match the requirements specified in Section II, the SDVM [2] was chosen as a basis. Thus, the firmware for FPGA-based reconfigurable systems is called SDVM^R.

A. The Scalable Dataflow-driven Virtual Machine (SDVM)

The SDVM is a dataflow-driven parallel computing middleware. It was designed to feature undisturbed parallel computation flow while adding and removing processing units from computing clusters. Applications for the SDVM must be cut to convenient *code fragments*. The code fragments and their *frames* (a data container for parameters) will be spread automatically in the cluster depending on the data distribution.

Each processing unit which is encapsulated by the SDVM virtual layer is called a *site* and acts as an autonomous member of the cluster. The sites consist of a number of modules with distinct tasks and communicate by message passing. Currently, the SDVM is implemented as a UNIX-daemon to be run on each participating machine, creating a site each.

The SDVM is a convenient choice as a middleware for FPGAs due to several distinguishing features. The two most important features are that the SDVM cluster can be resized at runtime without disturbing the parallel program execution, and each site in a cluster can use a different internal hardware architecture. These two features are the basis for the runtime reconfiguration ability of the system.

A reconfiguration cycle in such a system starts with the site occupying the area to be reconfigured dropping out of the running cluster. The displacement of data and code objects is managed by the SDVM middleware. After loading the new configuration the newly created site, now having a different architecture, joins the cluster and is automatically included in the ongoing runtime distribution of the workload. On FPGAs which support continuous operation of the unchanged part of the logic fabric, the other sites' operations are not disturbed.

B. Different realization schemes

The SDVM is to be ported to run on an FPGA. There are two primal possibilities to use the reconfigurable area provided by the FPGA.

1) The available resources on the FPGA are used up by configuring additional processing units. Thus, the SDVM cluster consists of more sites and a higher parallelism can be achieved.
2) The FPGA fabric is used to implement custom function units, each attached to and therefore controlled by one of the cores. The function units conform to specific code fragments which are to be executed often. The supported functions of the custom function units can be changed at runtime by reconfiguration.

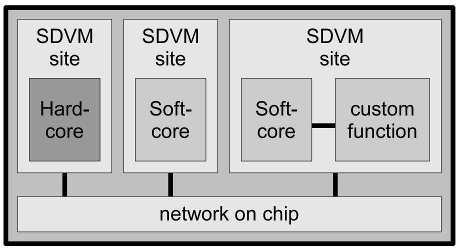

Fig. 1. Each core is encapsulated by a SDVM site. The sites are implemented as software running on the cores.

The different approaches can be combined (see Fig. 1). For a start the first possibility will be implemented. The next step will be the implementation of support for code fragments realized in hardware. This is especially aided by the fact that both the Microblaze and the PowerPC hardcore provide a fast low-level interface to the FPGA logic fabric. In this way the middleware still runs as software on the core while the data processing is shifted to specialized hardware (i.e. the logic fabric). Eventually, the realization of the middleware functions as hardware modules will be examined in future.

C. Technical challenges

As the virtualization layer will be implemented in software which runs on the PE of each site the memory footprint is a main concern. Therefore the existing SDVM C++-implementation is going to be reimplemented targeting the basic operating system kernel XMK. As the number of sites changes due to reconfiguration, a reconfiguration-aware on-chip network connecting the sites has to be developed.

IV. Conclusion

In this paper the concept of a virtualization layer for FPGAs was presented which separates applications to be run from the underlying hardware. The virtual layer is used to exploit runtime reconfiguration and parallelism of currently available FPGAs on system level. It is based on the SDVM, a middleware for computer clusters and multicore chips. Due to its features, the FPGA may reconfigure itself at runtime to adapt to changing conditions and requirements.

References

[1] G. Lipsa, A. Herkersdorf, W. Rosenstiel, O. Bringmann, and W. Stechele, “Towards a framework and a design methodology for autonomous soc.” in *ARCS Workshops*, U. Brinkschulte, J. Becker, D. Fey, C. Hochberger, T. Martinetz, C. Müller-Schloer, H. Schmeck, T. Ungerer, and R. P. Würtz, Eds. VDE Verlag, 2005, pp. 101–108.

[2] J. Haase, F. Eschmann, and K. Waldschmidt, “The SDVM - an approach for future adaptive computer clusters,” in *10th IEEE Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS 05)*, Denver, Colorado, USA, Apr. 2005.