SDVM^R: A Scalable Firmware for FPGA-based Multi-Core Systems-on-Chip

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

Abstract: As the main scope of mobile embedded systems shifts from control to data processing tasks high performance demand and limited energy budgets are often seen conflicting design goals. Heterogeneous, adaptive multicore systems are one approach to meet these challenges. Thus, the importance of multicore FPGAs as an implementation platform steadily grows. However, efficient exploitation of parallelism and dynamic runtime reconfiguration poses new challenges for application software developement. In this paper the implementation of a virtualization layer between applications and the multicore FPGA is described. This virtualization allows a transparent runtime-reconfiguration of the underlying system for adaption to changing system environments. The parallel application does not see the underlying, even heterogeneous multicore system. Many of the requirements for an adaptive FPGA-realization are met by the SDVM, the scalable dataflow-driven virtual machine. This paper describes the concept of the FPGA firmware based on a reimplementation and adaptation of the SDVM.

1 Introduction

Multicore systems are no longer restricted to the area of high performance computing. Today, as the main scope of embedded systems shifts from control oriented applications to data processing tasks even mobile phones contain chipsets which provide multiple, heterogeneous cores. To satisfy market needs those mobile devices must provide audio and video processing capabilites and therefore have high performance demands despite living on a tight energy budget.

Multicore systems consisting of heterogeneous components are one way to tackle these conflictive design goals. Further freedom of design space exploration is provided if these systems can be reconfigured at runtime and therefore adapt to changing needs of the application. For example, to optimize the power management the number of active, or even existing, cores can be adapted dynamically to the current workload. Therefore, the importance of multicore FPGAs as an implementation platform steadily grows as they allow to design parallel systems with most of the components placed on-chip. With multiple processor cores, both as fixed hardware (hardcore) or implemented using the configurable logic (softcores), and their ability to reconfigure they provide a good basis for parallel, scalable and adaptive systems.

To efficiently exploit parallel and adaptive systems the application software must cope with the changing number of existing cores and manage the hardware reconfiguration. As these features are shared by most applications running on such a system it is beneficial to provide a virtualization layer which hides the – due to runtime reconfiguration – changing hardware system from the application software. The Scalable Dataflow-driven Virtual Machine (SDVM) is such a virtualization of a parallel, adaptive and heterogeneous cluster of processing elements [Aut05, Aut04]. Thus, it is well suited to serve as a managing firmware for multicore FPGAs and to fulfill the above mentioned requirements. This paper covers the description of the main features and the general concept of the firmware.

This paper is structured as follows: Section 2 describes the fundamentals of FPGAs and gives an outline of the firmware concept. Section 3 discusses the realization of the firmware using the SDVM and the developement system architecture it is tested on. Section 4 gives an overview of related work. The paper closes with a brief conclusion in Section 5.

2 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 or packet filtering in a network router, their design has to address a multitude of secondary requirements. These requirements are important for most systems, merely the weighting differs. The introduction of FPGAs as a target platform for SoCs adds an other important requirement: The runtime reconfiguration ability of some FPGAs provide additional flexibility to the system. To make optimal use of these reconfigurable systems an efficient management of the reconfiguration process is necessary.

The list of 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 supporting features. This lightens the burden of the designer who can concentrate on the primary functions of the SoC.

To avoid an increase in complexity, provide flexibility, and improve portability and code reusability through different hardware the division of the functionality into several layers is a possible solution. The aforementioned generic module should therefore be implemented

System	FPGA	Slices	RAMB16
1 MicroBlaze	Virtex4 FX20	2,025 (23 %)	36 (52 %)
2 MicroBlaze	Virtex4 FX20	3,958 (46 %)	68 (100 %)
1 PPC	Virtex4 FX20	1,158 (13 %)	36 (52 %)
1 PPC + 1 MB	Virtex4 FX20	3,067 (35 %)	68 (100 %)
2 PPC	Virtex-II Pro 30	2,096 (15 %)	38 (50 %)
4 MicroBlaze	Virtex-II Pro 30	7,513 (54 %)	132 (97 %)
2 PPC + 2 MB	Virtex-II Pro 30	5,768 (42 %)	132 (97 %)

Table 1: Resource requirements of multi-processor systems implemented on different Xilinx FPGAs

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. Thus, such a middleware is specifically well suited to be used as a firmware for FPGAs.

2.1 Dynamically reconfigurable platform FPGAs

A generic FPGA architecture consists of three major components: Configurable logic blocks (CLBs), input/output blocks (IOBs) and a configurable routing network that connects all IOBs and CLBs. The CLBs can be used to implement arbitrary logic functions while the IOBs provide the physical interface to the off-chip environment. Today, FPGAs are no longer limited to these basic components. They incorporate additional specialized function blocks like embedded memory or processor cores. Thus, modern FPGA can implement complete parallel system architectures on-chip, so-called microgrids.

Any function block which is implemented on an FPGA using dedicated silicon area and therefore uses no CLBs is called a hard macro. If the function block realizes a processor core it is more accurately called a *hardcore*. In contrast, processor cores which are implemented solely using the CLBs are called *softcores*. However, hardcores may require additional support logic which has to be implemented in CLBs to be fully usable.

The vast amount of CLBs enables the designer to add several softcores. As seen in Table 1 even the second smallest device of the Virtex-4 FX family can host two MicroBlaze softcores including an FPU and dedicated RAM each and still has more than 50 % free logic resources that can be used to implement application specific functions. Larger FPGAs can support systems with four cores and still have 42 % of their logic elements unsused.

2.2 The Middleware concept

To efficiently use a dynamic reconfigurable FPGA as an implementation architecture for multi-core systems the middleware has to support a number of different features.

Today, even small FPGAs can host multiple cores (See Table 1). Besides vendor-supplied softcores or embedded hardcores system designers often add application specific function blocks like digital signal processors (DSP) or specialized datapath units. Unless a distinction is necessary these cores and function blocks will be referred to as processing elements (PE) in the following.

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 being the external memory. As FPGAs typically have only up to some hundred kilobytes of internal memory – the smaller ones actually provide less than one hundred kilobytes – 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. It should be possible to execute a task on general purpose processors of different architectures and on custom function units if applicable.
- 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. To avoid bottlenecks each PE should have its own memory both for program and data. On the other hand, applications for shared memory are much easier to design than applications for message passing systems. Thus, the distributed memory should be transparent to the applications.
- Manage the reconfiguration of the FPGA, i.e. keep track of the current usage of the FPGA resources and available alternative partial configurations. Furthermore, an adequate replacement policy has to be defined.
- Monitor a number of system parameters to gather information the configuration replacement policy depends on.
- Adjust the number of active PEs at runtime. For example, this can be used to meet power dissipation or reliability targets.
- The previous feature requires the firmware to hide the actual number of PEs from the application to ease programming.
- As the user software does not know the number and architecture of the PEs the firmware has to provide dynamic scheduling as well as code and data distribution.

One of the fundamental decisions in the design process of the firmware is whether each PE 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 like coping with errors in the fabric or reducing hotspots.

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 on the FPGA.

In addition, the firmware depends to a much lesser extent on the number and type of cores in a cluster if it runs on each core independently and communicates using hardware-independant messages.

Therefore, the middleware will be implemented as a firmware running on each core of the FPGA based system. The parallel cluster is created by the communication of each firmware instance with the other instances.

3 Realization

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

3.1 The Scalable Dataflow-driven Virtual Machine (SDVM)

The Scalable Dataflow-driven Virtual Machine (SDVM) [Aut05, Aut04] is a dataflow-driven parallel computing middleware (see Fig. 1). 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* (of any size). The code fragments and the *frames* (a data container for parameters needed to execute them) will be spread automatically throughout the cluster depending on the data distribution.

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

The SDVM is a convenient choice as a middleware (virtual layer) 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.

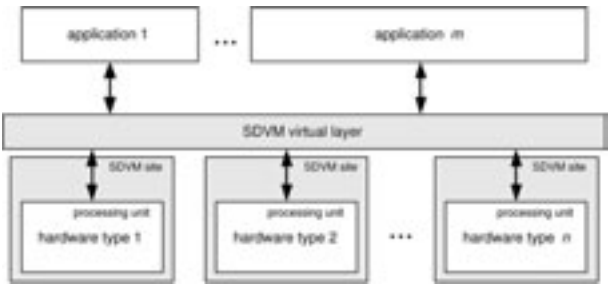

Figure 1: The processing units are encapsulated by an SDVM site each. The sites form the SDVM virtual layer. The applications don't see the underlying (possibly heterogeneous) hardware.

During a reconfiguration cycle a site drops out of the cluster, its logic estate gets reconfigured and the newly created core joins the cluster. The displacement of data and code objects is managed by the SDVM middleware. Besides changes of the sites' architectures the cluster resize mechanism can be used to adapt the cluster to different grades of parallelism of the currently running applications. On an excess of computing power sites can drop out of the cluster shutting down the affected area of the FPGA to minimize power consuption. When the demand for computing power rises free areas of the FPGA can be used to deploy new sites thus increasing the capacity of the cluster.

The integration of adaptive features into the middleware which can be used to adjust the behaviour of the system to a variety of targets requires knowledge gathered at runtime. For example, a power management policy using the SDVM was developed which can be used to improve the reliability of a multicore chip [Aut06]. Independent selection of the core's power states in conjuction with dynamic parallelism is used to minimize temperature changes of the chip thereby improving the reliability significantly while sacrificing less performance than simpler power management policies. The policy requires the collection of serveral runtime parameters like the amount of workload and current core temperature which are gathered and distributed by the SDVM.

All in all the SDVM offers the following features that are beneficial for the implementation of an FPGA middleware:

- undisturbed parallel computation while resizing the cluster
- dynamic scheduling and thereby automatic load balancing
- distributed scheduling: no specific site has to decide a global scheduling and therefore any site can be shut down at any time
- participating computing resources may have different processing speeds
- participating computing resources may use different hardcores and softcores
- applications may be run on any SDVM-driven system, as the number and types of the processing units do not matter
- no common clock needed: the clock is locally synchronous but globally asynchronous.
- support for distribution of knowledge gathered at runtime

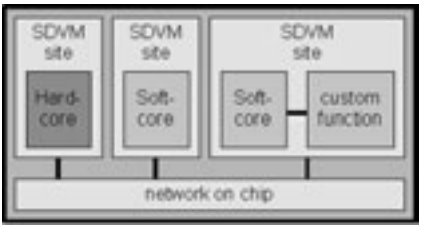

Figure 2: Each core is encapsulated by a SDVM site. The sites are implemented as software running on the cores.

3.2 Different realization schemes

One important question when implementing a parallel system on an FPGA is, how the reconfigurable area provided by the FPGA is used. There are two primal possibilities:

1. The available resources on the FPGA are used up by configuring additional processing units. Thus, the SDVM^R cluster consists of more sites and a higher parallelism can be achieved. The number of sites can be changed by reconfiguration to adopt to the available parallelism of the application as far as FPGA resources permit.

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 to adapt the system to the needs of the application.

The different approaches can be combined (see Fig. 2). 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 some of the data processing is shifted to specialized hardware (i.e. the logic fabric). The realization of the middleware functions as hardware modules is an option for the future.

3.3 Developement system architecture

For the developement of the SDVM^R firmware a scalable system has been created using the Xilinx EDK 8.1 software. The system is based on IP blocks supplied by Xilinx which are included in EDK. As the processing element the MicroBlaze softcore is used. It is supported by a timer and an interrupt module connected to a local On-Chip Peripheral Bus (OPB) to allow for the execution of the Xilinx XMK realtime operating system. The MicroBlaze core has 64 KB of embedded memory blocks connected to its Local Memory Bus (LMB) to store and execute the firmware. These four IP blocks build the basic processing element of the system.

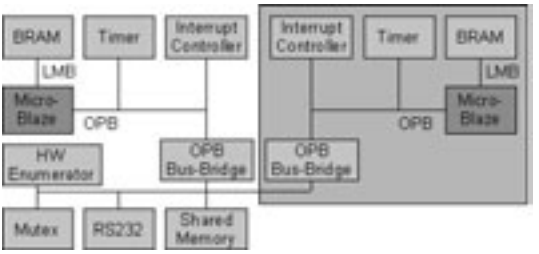

Figure 3: The system architecture of a dual core MicroBlaze system. The grey-shaded box is omitted for a single core system; for multicore systems it is replicated for each additional core.

The communication between the processing elements is done using a shared memory connected to the system OPB. To allow for mutual exclusion of the access to this memory and to the RS232 interface a hardware mutex modul is attached to the system OPB. The multiple cores are attached to the system using OPB-to-OPB bridges (See Fig. 3).

The PowerPC based system is basically the same. The MicroBlaze and its memory is replaced by the PowerPC core and a PLB-connected memory block. The resource requirements of different system configurations are listed in Table 1. The MicroBlaze is implemented with all features and FPU enabled but without caches and debugging support.

As the hardware basis a Virtex4 FX20 populated evaluation board was chosen due to its fine-grained reconfiguration features and embedded PowerPC core. Resources, especially embedded memory blocks, are quite limited on the V4FX20 to implement systems with more than two cores, so a Virtex-II Pro 30 based system is used for tests with 2 PowerPCs and up to 4 MicroBlaze cores. The busses and MicroBlazes cores of all systems run at 100 MHz clock frequency; the PowerPC cores are clocked at 300 MHz.

3.4 Preliminary performance results

The current developement version of the SDVM^R has been tested with a parallel application calculating a mandelbrot set. The application is mapped to the execution model of the SDVM^R in such a way that the calculation of a row of the mandelbrot set is done in one code fragment. Thus, the parallelism of the application increases with the number of rows to calculate. The disadvantage of this straight-forward approach is that the runtime of each instance of this code fragment differs. This is due to the mandelbrot algorithm which requires a different amount of iterations depending on the current point it is calculating.

The application has been tested on the one, two and four core systems including combinations of PowerPC and MicroBlaze cores described in Section 3.3. The results are shown in Table 2.

Despite the lower clock frequency of the MicroBlaze (100 MHz) compared to the PowerPC cores (300 MHz) the latter ones are much slower due to a missing FPU. The dualcore PowerPC system scales noticeably better that the dualcore MicroBlaze system with an

System	FPGA	CPU-Clock	Runtime	Efficiency
1 MicroBlaze	V4FX20	100 MHz	7181	—
2 MicroBlaze	V4FX20	100 MHz	3826	0.94
4 MicroBlaze	V2Pro30	100 MHz	2290	0.78
1 PPC	V4FX20	300 MHz	65574	—
2 PPC	V2Pro30	300 MHz	33045	0.99
1 PPC + 1 MB	V4FX20	300/100 MHz	6800	—

Table 2: Runtimes of a single-precision floating point mandelbrot set

efficiency of 0.99 compared to 0.94. For the four core MicroBlaze system the efficiency drops to 0.78. As the MicroBlaze system is more than 8 times faster it is more sensitive to the way the workload is distributed. As long as a site has nothing to do it keeps asking other sites for work every once in a while. Currently, this request rate is fixed. Therefore some sites may sit idle for some time even if new work is available.

4 Related work

The middleware presented in this paper uses hardware abstraction based on a virtual layer to exploit runtime reconfiguration and parallelism of currently available FPGAs on system level. Most of related work in this field either focuses on the virtualization of an FPGA on the level of the logic fabric, or on techniques for runtime reconfiguration. Furthermore the discussion of adaptive features is most often detached from concrete implementation technologies. An overview of related work in a broader sense confirm this.

Roman Lysecky et al. developed techniques for dynamic hardware/software partitioning based on online profiling of software loops and just-in-time (JIT) synthesis of hardware components called WARP [LV05]. They also present a dynamic FPGA routing approach which can be used to solve the routing and placement problem of reconfigurable components at runtime [LVT04]. However, their approach relies on a special, to our knowledge not yet implemented, FPGA architecture called Configurable Logic Architecture [LV04].

To overcome the need for this special FPGA architecture Roman Lysecky et al. developed a virtualization layer for FPGAs [LMVV05] which is placed on top of the logic fabric of a real FGPA. This virtualization layer emulates the Configurable Logic Architecture and enables the use of JIT techniques on existing FPGAs but leads to a 6X decrease in performance and a 100X increase in hardware requirements. In contrast the concept presented in our paper uses softcores and functional units specifically designed to the target FPGA family thus the overhead should be much lower.

The usage of adaptive features to tackle the complexity of modern SoCs is extensively covered by Gabriel Lipsa et al. [LHR+05]. Their paper proposes a concept that applies autonomic or organic computing principles to hardware designs. The paper does not present any kind of implementation, neither as software nor as hardware. The SDVM^R as a firmware for FPGAs is a software-realization of these autonomous principles.

5 Conclusion

In this paper a middleware providing a virtualization layer for FPGAs is presented. The virtualization layer hides the changing hardware of a reconfigurable FPGA-based system from the application software. Thereby, it allows 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. The work is currently under developement and some preliminary performance results are presented.

References

[HDHW06] Jan Haase, Markus Damm, Dennis Hauser, and Klaus Waldschmidt. Reliability-aware power management of Multi-Core Processors, 2006. DIPES 2006, Braga, Portugal.

[HEKW04] Jan Haase, Frank Eschmann, Bernd Klauer, and Klaus Waldschmidt. The SDVM: A Self Distributing Virtual Machine. In *Organic and Pervasive Computing – ARCS 2004: International Conference on Architecture of Computing Systems*, volume 2981 of *Lecture Notes in Computer Science*, Heidelberg, 2004. Springer Verlag.

[HEW05] Jan Haase, Frank Eschmann, and Klaus 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, April 2005.

[LHR+05] Gabriel Lipsa, Andreas Herkersdorf, Wolfgang Rosenstiel, Oliver Bringmann, and Walter Stechele. Towards a Framework and a Design Methodology for Autonomous SoC. In Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Christian Hochberger, Thomas Martinetz, Christian Müller-Schloer, Hartmut Schmeck, Theo Ungerer, and Rolf P. Würtz, editors, *ARCS Workshops*, pages 101–108. VDE Verlag, 2005.

[LMVV05] Roman L. Lysecky, Kris Miller, Frank Vahid, and Kees A. Vissers. Firm-core Virtual FPGA for Just-in-Time FPGA Compilation (abstract only). In Herman Schmit and Steven J. E. Wilton, editors, *FPGA*, page 271. ACM, 2005.

[LV04] Roman Lysecky and Frank Vahid. A Configurable Logic Architecture for Dynamic Hardware/Software Partitioning. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 10480, Washington, DC, USA, 2004. IEEE Computer Society.

[LV05] Roman Lysecky and Frank Vahid. A Study of the Speedups and Competitiveness of FPGA Soft Processor Cores using Dynamic Hardware/Software Partitioning. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 18–23, Washington, DC, USA, 2005. IEEE Computer Society.

[LVT04] Roman Lysecky, Frank Vahid, and Sheldon X.-D. Tan. Dynamic FPGA routing for just-in-time FPGA compilation. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 954–959, New York, NY, USA, 2004. ACM Press.