Resource Management for the Heterogeneous Arrays of Hardware Accelerators

Zdenek Pohl and Milan Tichy
Department of Signal Processing
Institute of Information Theory and Automation
182 08 Prague, Czech Republic
Email: {zdenek.pohl,tichy}@utia.cas.cz

***Abstract*—The resource management (RM) of heterogeneous hardware cores within the large-scale embedded systems have been one of the key implementation and research subjects in last years. In future systems, consisting of growing number of heterogeneous hardware cores, the RM have to be implemented to control the energy drain, exploit maximally the performance potential, and also to improve robustness of the system by bypassing failing components. We present the resource management providing an API for parallel execution of operations on a heterogeneous set of resources. The RM controls the tasks execution and monitors the performance. The information collected by the RM is used to optimize the execution. In this paper, we propose and present the SW implementation of the RM for the MicroBlaze soft-core processor running the Linux OS and accelerated by the heterogeneous array of hardware accelerators.**

***Keywords*-resource management; FPGA; embedded systems; hardware acceleration; power management; scalability**

I. Introduction

The number of transistors-per-area available in future embedded system products is increasing. It is expected that in a near future, integration of tens or even hundreds of cores on a chip will be possible. Similarly, applied to the FPGA field it is expected that the area of the device allows presence of not only multi-core microprocessors but also many various hardware accelerators capable to solve specific problems power efficiently.

The devices will accommodate many competing applications, such as phone, television, movie player, or the Internet integrated in one low-power, wireless handheld device. Some of the applications can run sequentially and some of them in parallel, interacting with each other.

Such architectures will contain heterogeneous subsystems as well as homogeneous parts. It is expected that the programing of such systems will be more difficult by order of magnitude.

The growing trend in number of processor cores and other hardware components in one embedded device can be clearly seen in today's architectures. Such platforms are demanding the implementation of the *resource management (RM)*, which will help to overcome technological difficulties that come from increased power density and power consumption. The RM will be able to monitor system conditions and to adapt the system to the available resources, increasing its performance, reliability and fault recovery.

The aim of this paper is to analyze the implementation of the resource manager for the MicroBlaze [1] microcontroller closely coupled with hardware accelerators to improve performance of the embedded system. We focused on the following issues: the scalability of the system, the power management, the design time reduction of user applications, and the improvement in the utilization of processing elements.

II. Hardware Accelerators in the RM

The architecture used for accelerating the MicroBlaze system is based on the *UTIA master-worker platform*. The platform exploits a set of heterogeneous hardware accelerators to improve the MicroBlaze's computational performance, particularly in calculation of vector operations. The architecture of the platform is described in detail in [2]–[5]. Similar approach to improving performance of a microprocessor using vector-oriented hardware accelerators can be found in [6]–[9]. Different approach focusing on the OS integration and partial dynamic run-time reconfiguration of dedicated HW cores is used in [10]–[13].

From the resource management point of view, the hardware accelerators can be divided into *families*, where each family represents a group of compatible accelerators sharing common arithmetic or purpose. The second parameter that describes the accelerator is the *SIMD* mode, which corresponds to the number of data contexts processed independently by the accelerator. The last accelerator parameter is the so called *capabilities (caps)*. The capabilities represent functions implemented by the accelerator. For example, the complex accelerator can include hardware support for calculation of the dot-product, while another can contain support for the cumulative summation or magnitude of the complex vector. Thus, these features are represented as the special capability of such accelerator.

In perspective of the resource management, the computational resource A can be defined as the triplet:

$$A \equiv (family, simd, caps)\,, \tag{1}$$

where the $family$ is the family ID number, the $simd$ is the SIMD mode and the $caps$ is the bit-array representing the

accelerator's capabilities.

The computation resource definition (1) is used to introduce the *resource compatibility* to the RM in our concept. The computational resource compatibility can be formulated as follows: the resource A is compatible with the resource B if all operations to be performed on the resource A can be mapped to the resource B. It can be defined as:

$$A \rightarrow B \iff \begin{cases} family(A) & = family(B) \\ simd(A) & \leq simd(B) \\ caps(A) & \subseteq caps(B) \end{cases} \quad (2)$$

It is evident that this concept of compatibility allows to map tasks on the desired resource or on more "capable" resources of the same family. The incremental set of accelerator capabilities allows to achieve better hardware utilization by the effective use of the resource management.

III. Resource Management

The resource manager provides a user application with the abstraction layer to isolate an application programmer from the hardware. The user API allows to request, call, release, reserve, and allocate hardware resources that form the embedded platform. Main features that are addressed by the RM include: hardware abstraction, unified interface, performance and power control, resource sharing, resource compatibility, load balancing, semi-automatic handling of parallelism and synchronization, isolation of data flows from the control, and low RM overhead.

With respect to the RM requirements, we can define the resource management *task* (to be run on an accelerator). Such task is considered as atomic operation. The task is defined by: input and output data, input and output state, parameters and constraints, and firmware. In current implementation of the RM, the tasks are assumed to be static. The firmware is represented by the user-defined microcode controlling and executing a more complex atomic operation in accelerators. Note that even if the hardware accelerator operations are stateless, the tasks are not. The consequent execution of the task on hardware can alter its state. Another parameter related to each task is its processing priority and the reference to its predecessor, the task whose internal state must be used for the current execution, and to its successor, the task that uses the result of previous one. The task structure is schematically depicted in Figure 1(a). The last parameter of the task is the description of the requested accelerator required for its execution. The requested accelerator is described using the form given by (1). Such task definition allows the resource manager to optimize the task execution regarding its priority, local data preservation in the hardware accelerator, parameters preservation, and state synchronization.

The top-level structure of the RM implementation is depicted in Figure 1(b). The resource management is interfaced from a user application via its entry point represented by the *Global Resource Manager (GRM)* layer. Within this layer, the ability to accept and process the task is checked and the task is passed to the appropriate *Local Resource Manager (LRM)* layer entry point. If more LRM candidates can process the task, the GRM implements the load balancing based on the minimization of weighted counts of processed task with exponential forgetting, while respecting the task location constraints.

The LRM layer is represented by one or more LRM instances. Each LRM instance is created for a cluster of homogeneous HW accelerators, i.e. for HW accelerators of the same type. Such concept allows the scheduler inside the LRM instance to schedule incoming tasks to the homogeneous array of identical hardware accelerators.

The scheduling strategy strives to preserve the data locality by sending each task to the same accelerator as its predecessor. If the task has a successor, the scheduler waits for it by exclusion of a given accelerator from the load distribution. The tasks without predecessor are distributed evenly to the remaining accelerators.

IV. Case Studies

For the benchmarking of the proposed resource management, several application scenarios were implemented. In this paper, we present two applications: the physical layer of the digital audio broadcasting (DAB) receiver [14] and the radio spectrum sensing application. [1]

To implement these applications using our platform, accelerators of two families were used: the complex family and the Fast Fourier Transform (FFT) accelerators. The complex accelerator provides a native support for vector floating-point (FLP) operations, such as vector copy, addition, multiplication, dot-product, multiply-accumulate, cumulative summation, etc. in the complex domain. The FFT accelerator supports FFT and inverse FFT of orders 8 to 2048.

V. Evaluation of the RM Performance

The resource management behavior has been investigated under different working conditions using a number of application scenarios and hardware configurations. We focused mainly on the scalability, the design time reduction (portability), power consumption, and the system performance.

A. Scalability and Portability

As mentioned above, the resource manager provides a user application with the abstraction layer to isolate an application programmer from HW. Using the RM framework, the user application can be implemented to follow its goals, exploiting its parallelism.It is evident that the portability of

[1]The sensing application has been developed by Thales within the project SCALOPES as a common test case. It constitutes the evaluation use case, not an optimized sensing algorithm for cognitive radio.

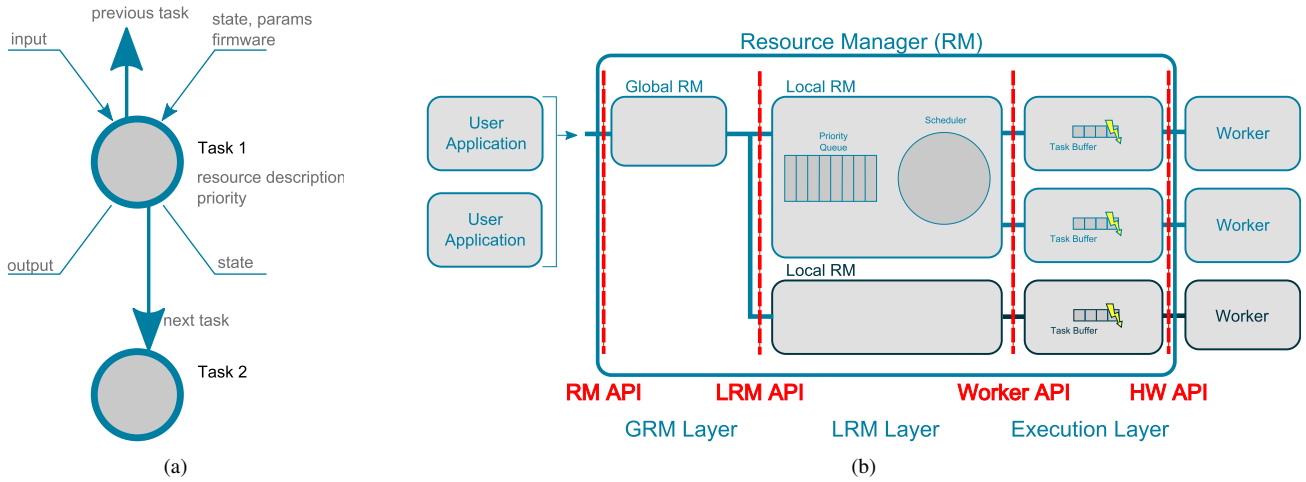

Figure 1. The task structure (a) and the resource management structure (b)

any user applications becomes an easy task once the RM has been adapted to the available hardware resources.

The portability is closely related to the scalability of the system. Generally, a significant re-design effort must be invested in order to adapt the algorithm onto a platform with different number of hardware accelerators, or with different but compatible accelerators. The resource management is able to exploit all the resources automatically.

B. Power Consumption

To evaluate the power reduction, the Xilinx Spartan-3A DSP 1800A development board has been used. This board has been chosen specifically for the power consumption evaluation as it allows measuring of the input current on its power supply jumper. The mixed signal oscilloscope, the Agilent MSO 6034A scope equipped with the N2783A current probe have been used for measuring.

The power control of hardware resources is one of the key points to achieve success in parallel architectures as the multiple processor or accelerator cores can ruin the consumption limits normally posed on the embedded, battery powered systems.

The possibility to turn off all accelerator cores not in use was integrated into the resource management. The cores can be technically turned off by employing the technique of clock gating [15]. The accelerator cores are isolated from the accelerator local memories and can still be turned off while the data communication with accelerator local memories is pending.

The measured power consumption reduction is 13.34% for the sensing application and 20.12% for the DAB application. This particular power savings correspond to a minimalist case when one complex and one FFT accelerators were used. The presented reductions are minimal in comparison to configurations where more hardware accelerators are present in the system.

The impact of clock gating can be viewed in two perspectives. Naturally, the best power savings are related to accelerators in their idle periods. When accelerators are not in use, the reduction of power consumption can be up to 40%. Less power savings are related to accelerators in their busy states, where the clock gating can be applied only during data communication. It is evident that the power reduction is strongly application dependent.

C. Performance Scaling

The concept of the resource management allows to distribute the tasks to an arbitrary number of accelerators, limited only by parallelism employed in the user applications. By adding more hardware (more parallel accelerators in our case) to the system, the performance automatically increases if the resource management is in use.

By detailed analysis of the results, it was found that adding one complex accelerator improves performance by 42.7%. Another complex accelerator brings additional 6.5% performance boost.

D. Resource Management Overheads

Naturally, the resource manager introduces overheads on the execution of tasks in hardware accelerators. The overheads are represented by the time consumed by load balancing, task scheduling, accelerator operations control etc. For the evaluation of overheads, the profiling information is collected by the RM for each processed task. Such information is represented by the time stamps created at individual processing stages as well as by the times related to data communications and hardware execution. Similar information is also used by the RM scheduler for planning the hardware accelerated operations. Based on the profiling information it was evaluated that the resource management overhead is 2% for the radio spectrum sensing application and 6% for the DAB application.

We have also compared the reference radio spectrum sensing application (non-RM, manually mapped on the UTIA platform) to the one exploiting the resource management. We found that the RM overhead is negligible with respect to the reference design running on the same number of parallel workers. That is, the resulting implementation exploiting features of the resource management almost completely suppresses the overheads related to the RM.

VI. Conclusions

The SW implementation of the resource management provides a user application with the abstraction layer to isolate an application programmer from the hardware and controls the execution of tasks on available hardware accelerators.

To fully exploit the features of the resource manager, the following requirements have to be met:

1) The accelerators have to support the clock-gating control and to use interrupts allowing successful integration to the large-scale hardware platform controlled by the resource manager.
2) The user application has to define its operations in the form of formalized tasks accepted by the RM. The tasks are executed analogically to the creation of another thread in the user application process, i.e. the execution has to be started by the "create" operation and the end of operation has to be synced by the "join" operation.

The implementation of the user applications employing the resource manager is, to some extent, independent of the hardware resources. Thus, the applications are easily portable among various hardware configurations. Using such feature, the resource management significantly reduces the design time necessary for customization to a different but compatible accelerators.

The significant power savings have been achieved by the application of the resource management. We presented the minimal power reduction 13.34% for the radio spectrum sensing application and 20.12% for the DAB application. We have shown that the power reduction increases with the number of accelerators in the system.

The ability of the resource manager to distribute the tasks to available hardware resources directly leads to performance scaling of the user application. We have presented that the automatic distribution of tasks on more accelerators bring the performance boost up to 49.2% for the radio spectrum sensing application.

Acknowledgements

This work was supported and funded by the ARTEMIS Joint Undertaking under the grant agreement No. 100029 and by the national funding project No. MSMT 7H09005.

References

[1] *MicroBlaze Processor Reference Guide*, Xilinx, Inc., Nov. 2010, v11.4.

[2] J. Kadlec, R. Bartosinski, and M. Danek, "Accelerating MicroBlaze floating point operations," in *Int. Conf. Field Prog. Logic and Applications*. IEEE, 2007, pp. 621–624.

[3] J. Kadlec, M. Danek, and L. Kohout, "Proposed architecture of configurable, adaptable SoC," in *The Inst. of Eng. and Tech. Irish Signals and Systems Conf., ISSC*, Galway, IE, 2008.

[4] J. Kadlec, "Design flow for reconfigurable Microblaze accelerators," in *Int. Workshop Reconfigurable Communication Centric System-on-Chips*, 2008.

[5] M. Danek, J. Kadlec, R. Bartosinski, and L. Kohout, "Increasing the level of abstraction in FPGA-based designs," in *Int. Conf. Field Prog. Logic and Applications*. IEEE, 2008, pp. 5–10.

[6] J. Cho, H. Chang, and W. Sung, "An FPGA based SIMD processor with a vector memory unit," in *Int. Symp. Circuits and Systems*, 2006.

[7] H. Yang, S. Wang, S. Ziavras, and J. Hu, "Vector processing support for FPGA-oriented high performance applications," in *IEEE Comp. Soc. Annual Symp. VLSI*, 2007, pp. 447–448.

[8] S. Chen, R. Venkatesan, and P. Gillard, "Implementation of vector floating-point processing unit on FPGAs for high performance computing," in *Canadian Conf. Electrical and Comp. Eng.*, 2008, pp. 881–886.

[9] J. Yu, C. Eagleston, C. H.-Y. Chou, M. Perreault, and G. Lemieux, "Vector processing as a soft processor accelerator," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, pp. 12:1–12:34, June 2009.

[10] H. Walder and M. Platzner, "Reconfigurable hardware operating systems: From design concepts to realizations," in *Int. Conf. Eng. of Reconf. Syst. and Arch.*, 2003, pp. 284–287.

[11] D. Andrews, D. Niehaus, R. Jidin, M. Finley, W. Peck, M. Frisbie, J. Ortiz, E. Komp, and P. Ashenden, "Programming models for hybrid FPGA-CPU computational components: A missing link," *IEEE Micro*, vol. 24, pp. 42–53, 2004.

[12] P. Garcia and K. Compton, "Shared memory cache organizations for reconfigurable computing systems," *Symp. Field-Programmable Custom Comp. Machines*, pp. 239–242, 2009.

[13] H. K.-H. So and R. W. Brodersen, "BORPH: an operating system for FPGA-based reconfigurable computers," Ph.D. dissertation, EECS Department, UC Berkeley, Jul 2007.

[14] *ETSI EN 300 401 V1.4.1: Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers*, Jun. 2006, reference REN/JTC-DAB-36.

[15] A. Hermanek, M. Kunes, and M. Tichy, "Reducing power consumption of an embedded DSP platform through the clock-gating technique," in *Int. Conf. Field Prog. Logic and Applications*. IEEE, 2010, pp. 336–339.