Application-Driven Design of Cost-Efficient Communication Platforms

Hans-Martin Blüthgen, Christian Sauer,
Dominik Langen, Matthias Gries, Wolfgang Raab
Infineon Technologies, Corporate Research, Munich, Germany
Hans-Martin.Bluethgen@infineon.com

Abstract: We present a five phases approach for the cost-efficient design of communication platforms. We start with complete and executable models of the system function that capture all essential parts of a reference application, i.e. processing kernels, data and control flow, and timing constraints. We analyze such an architecture-independent model to narrow the design space and derive a most flexible initial platform architecture. We map the model onto the platform hard- and software, and profile the system. Through iterations, the prototype is refined, until an architecture is found that meets performance and cost constraints. The resulting virtual prototype serves as executable specification for the design of the final system. Two case studies from wireless communications (Software Defined Radio, SDR) and access networks (Digital Subscriber Line Access Multiplexer, DSLAM) illustrate our approach.

1 Introduction

Platforms are a powerful means to cope with increased pressure on time-to-market, design and manufacturing costs [1] that is more and more exploited, especially in price-sensitive application domains, such as wireless communications and access networks. The main goal is to integrate all required functionality while preserving as much flexibility as needed in the application domain to withstand communication standards and deployment trends under tight performance, and costs constraints like area and power.

We have developed an approach for communication platforms that takes the importance of the application domain into account. One major focus of our methodology is the proper characterization of the domain and implementation of a representative reference application. With their help a fully flexible platform architecture template can be customized and the complexity of the design space significantly reduced to meet costs constraints. Our methodology can be described by the following five phases:

1. **Application Domain Analysis:** Identify and define representative and comparable system-level benchmarks and realistic workloads for algorithms and protocols.
2. **System Function/Reference Application Modeling:** Implement a performance indicative system-level reference application that captures essential system functions.
3. **Architecture-independent Profiling:** Derive architecture-independent application properties by analyzing and profiling the reference application.
4. **Design Space Exploration:** Perform a systematic search of the architectural design space based on the reference application and its properties; define and iteratively refine the resulting platform architecture.
5. **Platform Implementation and Deployment:** Implement the platform and provide a code generators to ease the deployment with efficient programming abstractions.

In the following, we present the five phases of our design flow and discuss results looking at DSLAMs for access networks and SDR for wireless networks.

2 Application Domain Analysis

The availability of benchmarks is often an unresolved issue for new application domains. We need to identify and specify representative system-level benchmarks including function, traffic model, and environment specifications. The goal is to understand the application domain and derive design requirements and defined system environments.

DSLAMs connect individual customers with the broadband service provider's network. They are built modularly out of line cards aggregating xDSL lines, a backplane connecting line and uplink cards, and uplink cards providing access to the service provider network. With the convergence of voice, data, and video networks, future DSLAMs will become the central access point for all broadband services networks. Therefore, we need to develop a reference application that captures current trends like quality-of-service. In [4], more details of the domain analysis are provided.

In the SDR domain, by comparing standards like WCDMA and WLAN 802.11a it becomes evident, that functional kernels and topologies are quite diverse. At this early stage of the design flow, essential functions together with their specific features needs to be derived.

Application domain analysis includes deriving estimates on the required processing power. Based on our experience it is necessary to plan with significant headroom in computing power. During analysis it is also important to find reasonable ranges serving as stop criteria for area/power optimizations [6].

3 System Function/Reference Application Modeling

An executable reference application is necessary to explore different partitioning and mapping decisions on the system-level, i.e. the entire system function together with latency and throughput constraints needs to be captured. For this task, we prefer domain-specific tools, since they are often more efficient and natural to the designer.

We model the DSLAM in Click [2], a domain-specific framework for describing packet processing applications. Click models are modular, executable, implementation independent, and capture inherent parallelism in packet flows and dependencies among components. By using Click, a functionally correct model of the application can be derived quickly. The DSLAM reference [4] consists of the actual system function and a test bench that contains traffic sources and sinks.

In case of SDR, there are several tools available which are suited mainly for describing the signal processing part of systems, e.g., Matlab/Simulink, CoCentric System Studio, ML Designer, or Ptolemy, to name a few. For the modeling of the baseband part of wireless communication systems we use Simulink and extend it with code generation capabilities for parallel processors [7].

4 Architecture Independent Profiling

After the reference application has been defined, architecture-independent properties of the application are derived by static analysis and/or simulation. The goal of this step is to determine architecture-independent characteristics of the functionality that can help pruning the design space and finding a good starting point in the architecture space. The analysis focuses on features of the reference implementation like data types, primitive operations and sequences thereof, communication between function kernels, and storage requirements. Some of these characteristics like primitive operations and sequences thereof can be obtained by profiling the reference model using conventional profiling tools. Others like communication between function kernels have to be estimated from the profiling results due to the lack of appropriate tools.

5 Design Space Exploration

We recognize three prerequisites that are essential for this phase: First, a comprehensive model of the system function, second, an efficient solution for mapping the application onto the target architecture, potentially supported by tools, such as code generators and compilers, and third, libraries of platform components that enable rapid assembly and ease alteration of candidate architectures. With these prerequisites, an iterative guided exploration can be performed effectively, starting from the most flexible and programmable solution. By gradually specializing the platform, a virtual prototype with an efficient programming environment is implemented.

In order to map our DSLAM reference onto different embedded processors, we have developed our CRACC framework that generates embedded code from Click descriptions [5]. Using the reference application described in Click, CRACC's Click front-end is used to generate a netlist and the corresponding configurations for CRACC elements written in C. The source code can then be cross-compiled and profiled on the respective embedded platform. A subsequent performance optimization can focus on individual elements and on the partitioning of elements onto several processing cores.

Figure 1 exemplarily compares the performance of individual cores measured as packet throughput per second (Pkts/s) and their area normalized to 90nm. The figure reveals that MIPS and N-core have the best performance/area trade-off for the chosen IP-DSLAM reference application. Based on these profiling results, we determine the required number of on-chip processing elements and their communication patterns using an analytical approach [3] and thus prune the design space systematically.

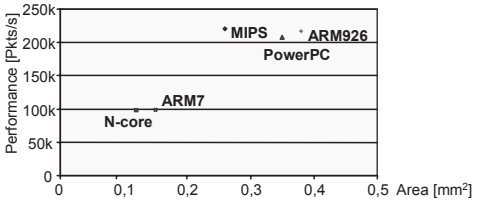

Figure 1: Performance of individual processor cores for the DSLAM upstream line card.

Apart from number and type of the processing cores, we continue the design space exploration and architecture modeling along the following axes: Instruction set extensions

that can be utilized by compilers; Tightly/loosely coupled coprocessors for standardized and complex tasks [8]; Memory hierarchy and data layout [9]; Communication architecture (such as hierarchical, globally shared, hybrid) [3]; Customized I/O interfaces that exploit domain specifics [10].

Since these axes are interrelated, the exploration as an iterative process must be based on a virtual prototype. In case of SDR the exploration phase has led to a virtual prototype based on System-C (*www.systemc.org*). This cycle and bit accurate simulator contains models for all processors, accelerators, busses, memories, and peripherals. Therefore, the same software can run on the virtual prototype and on the actual hardware. We prefer a cycle-accurate system over transaction-level models since the duration of transactions often cannot be estimated in complex inter-dependent multi-processor systems and may depend on the load of the system. The cycle-accurate SDR prototype is also used to verify whether a particular platform implementation meets real-time requirements.

For SDR, we deploy three different types of processors in order to meet real-time requirements. The first type is a general-purpose core, currently an ARM926. The second type is a DSP core with single-instruction multiple-data (SIMD) extensions. This coprocessor is highly configurable, e.g. the number of functional units within the coprocessor can be adapted to the level of data parallelism of the application. The third group are hardware accelerators for compute-intensive tasks, such as FIR filtering. The SDR virtual prototype provides several tools for analyzing the soft- and hardware. The general purpose cores can be debugged and profiled by using a graphical debugger. Internal state, instructions, and data transfers of all modules can be logged. It is also possible to track the execution status of threads on the general purpose processors. This leads to fast and very accurate analyses of the performance of an application.

6 Platform Implementation and Deployment

The virtual prototype serves as executable specification for the actual platform hardware and software implementation and serves as golden model for verification. As the functionality and timing requirements for each module have been accurately analyzed before, hardware designers are relieved from redesigning modules repeatedly due to flaws in the specification. The translation of complete modules to HDL can be straightforward, e.g. if an accelerator is modeled with all pipeline stages and precise widths of all data paths. Cycle-accurate stimuli of the virtual prototype can also be reused.

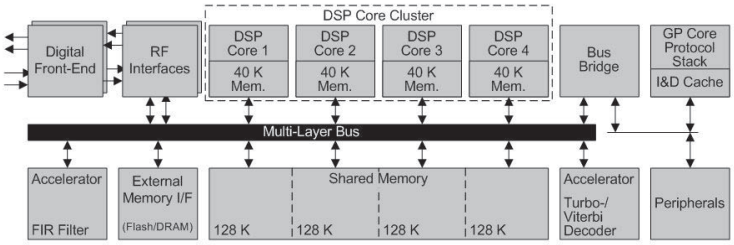

Figure 2: Multi-standard radio platform

A successful deployment of a programmable platform can only be achieved if the programmer is relieved from understanding the architecture in full detail. An approach is a

layer of firmware that provides sufficient abstraction for the customer. This way, only the firmware programmer is faced with the problem of coordinating the execution of software portions on different processing elements manually that must be optimized individually on assembly level [9]. Such a solution is feasible for platforms in closed environments, such as SDR. For the DSLAM, we plan a fully customer-programmable platform and therefore need efficient and natural programming abstractions. We already deal with code generation and programming abstractions during the exploration phase to perform mapping (semi-)automatically. After revision, this internally used programming abstraction [5] may be released together with the open platform to the customer.

7 Concluding Remarks

Our design methodology for flexible yet cost-efficient communication platforms is based on five phases. Starting from system level, a flexible platform architecture is developed and refined going down to circuit level. This iterative process is application-driven and relies on the use of virtual prototypes for analyzing and optimizing the architecture. This virtual prototype is used again in the final deployment phase to facilitate development and verification of RTL code. Case studies from two different domains demonstrated the application of different domain-specific tools throughout the phases. In each domain, tools had to be developed and extended to support the refinement process. The final goal of our effort is to automate at least each of the single phases if not the entire approach.

Acknowledgements: This work has been supported by the German government (BMBF), grants 01AK065A (PlaNetS) and 01M3062A (GigaNetIC). The authors wish to thank N. Brüls, Y. Fonin, C. Graßmann, U. Hachmann, M. Löw, U. Ramacher, M. Richter, M. Sauermann, A. Schackow, S. Sonntag, A. Troya, V. Ramakrishnan, and M. Vukoslavcevic for their contributions.

8 References

[1] A. Sangiovanni–Vincentelli, “Defining Platform-based Design”, *EETimes*, February 2002.

[2] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, “The Click modular router”, ACM Transactions on Computer Systems, 18(3), August 2000.

[3] M. Gries et al., “Exploring Trade-offs in Performance and Programmability of Processing Element Topologies for NPs”, Network Proc. Design, Vol. 2, Morgan Kaufmann, Nov. 2003.

[4] C. Sauer, M. Gries, S. Sonntag, “Modular Reference Implementation of an IP-DSLAM”, 10th IEEE Symp. on Computers and Communications (ISCC’05), Cartagena, Spain, 2005.

[5] C. Sauer, M. Gries, S. Sonntag, “Modular Domain-Specific Implementation and Exploration Framework for Embedded Software Platforms”, Design Automation Conference, June 2005.

[6] H.-M. Bluethgen, C. Grassmann, W. Raab, U. Ramacher, J. Hausner, “A Programmable Baseband Platform for Software-Defined Radio”, SDR Forum Techn. Conf., Nov., 2004.

[7] C. Grassmann, M. Sauermann, H.-M. Bluethgen, U. Ramacher, “System-Level Hardware Abstraction for Software-Defined Radios”, SDR Technical Conference, Phoenix, Nov. 2004.

[8] J.-C. Niemann, M. Porrmann, C. Sauer, U. Rückert, “Evaluation of the Scalable GigaNetIC Architecture for Access Networks”, ANCHOR-2 at ISCA’05, Jun. 2005.

[9] C. Kulkarni, M. Gries, C. Sauer, K. Keutzer, “Programming challenges in network processor deployment”, Conf. on Compilers, Archit., and Synth. for Emb. Systems, Oct. 2003.

[10] C. Sauer et al., “A flexible network processor interface for RapidIO, Hypertransport and PCI-Express”, Network Proc. Design: Issues and Practices, Vol. 3, Morgan Kaufmann, Feb 2005.