Teaching The Principles Of System Design, Platform Development And Hardware Acceleration

Tim Kranich, Thomas Schuster, Matthias Hanke, Mladen Berekovic
Institut fuer Datentechnik
Technische Universitaet Braunschweig
{kranich, schuster, hanke, berekovic}@ida.ing.tu-bs.de

Ralf Goettsche
Intel GmbH Braunschweig
Germany
ralf.goettsche@intel.com

***Abstract*—Development of embedded systems composed of tightly coupled microelectronic devices and several stacks of software layers adapted to it has evolved into a more platform based design methodology. We offer a lecture in the field of ESL design that provides a broad spectrum of theoretical background. It is escorted by a laboratory where starting from a C source code a platform with a hardware accelerator for JPEG compression is designed. Small tutorials teaching basic skills of hardware modelling with SystemC and TLM are offered. Hence, we provide a course split into three teaching forms that imparts knowledge about state-of-the-art platform design together with the qualification of understanding industrial system modeling standards and tools.**

I. Introduction

Embedded system computers increasingly ease our everyday life. Cell phones, vehicles, medical equipment or home entertainment devices are unrecognizably steered by an increasing number of electronic chips. Embedded systems and their software have become the dominant driver for microelectronics and systems engineering.

Platform-based system designs are common for the class of devices mentioned above. Platforms offer advantages like short time to market, high reusability and better reliability compared to full-custom solutions. Thus the cost for consumer devices with complex but standardised technologies is kept low [1].

Our course imparts knowledge about designing these platform-based systems for students related to the field of computer science. It illustrates common techniques and mechanisms not only for design but also for implementation and verification. Hence, they are taught in every field a design is involved with.

Section II explains our teaching philosophy, which splits the corse up into three different presentation forms. The developed laboratory framework is described in section III with its detailed schedule in section IV. The student's feedback is summarized in section V, which is followed by a conclusion in section VI.

II. Teaching Philosophy

Students attending this class are already advanced in their study of computer science, information technology or electrical engineering. We offer a course with a wide variety of teaching in the field of system design and especially platform-based design. Platforms are the most common way of developing modern systems. The content is taught in three different ways: a lecture, dedicated tutorials and a laboratory (see figure 1). This differentiation ensures an optimized transfer for the theoretical and practical knowledge as described in the following three subsections.

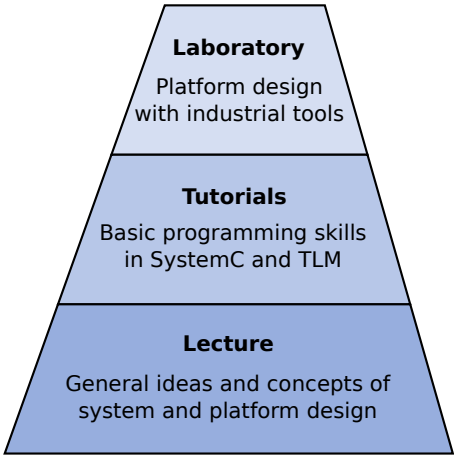

Figure 1. Course Structure

A. Lecture

Our lecture focus on teaching general concepts for system design and platforms. This is the theoretical part of our offering. It covers the various aspects of electronic-system-level (ESL) design like system-on-chip (SoC) design, bus communication, application-specific instruction set processors (ASIP), multi processor System-on-chip (MPSoC), networks-on-chips (NoC), low-power system design, test, verification and debug.

Every major topic comes along with references of state-of-the-art literature and implementation examples. These examples, either good or bad realizations, help developing a better and deeper understanding of the theoretical teaching content.

B. Tutorials

Our tutorials help to clarify the concept of the modeling language SystemC [3], [4], [5]. SystemC is one of most

common industrial modeling language and also the introduced tool Platform Creator from Coware is based on it. We experienced that getting to know how SystemC works also helps understanding how complex modelling systems like the Platform Creator are build. And this again allows developing new components for the Platform Creator like it is done during our laboratory.

Eight well chosen tutorials guide our students through the basics. The complexity of the programming tutorials increases with each new tasks. It starts with developing a 8bit counter equivalent to a HDL implementation. More advanced topics are test bench and stimulus generators. When the students get adapted to the new model language, we proceed with transaction-level-modeling (TLM). These exercises are concentrating on how communication between sources and sinks can be abstracted by invoking SystemC channels like FIFOs or buses.

C. Laboratory

The laboratory is organized in teamwork like real design teams. The idea is to enable the group of students to develop their own functional platform with the help of the Coware Platform Creator, an industrial design tool [2]. The point of origin is a generic C implementation of a JPEG compression. Integral elements of the final platform are an ARM processor, a memory mapped system bus, a system memory and a self-written hardware accelerator. The compression algorithm is tweaked so that it is executed on the ARM core together with the developed accelerator.

The students collect industrial-like experiences in many fields of system and platform design. For example the Platform Creator imparts their tooling skills with a modern design tool with industrial relevance. Additionally the Coware's tool environment allows developing the platform in a shorter time than a development from scratch.

III. Laboratory Framework

In the laboratory, our students are challenged to design and evaluate a platform for a special purpose with the industrial tool Platform Creator from Coware. In our special case we are focusing on accelerating a JPEG compression algorithm with a dedicated DCT Unit (see Figure 2). In order to achieve this goal the laboratory is divided into four working phases: tool-handling, platform design, software adaptation and JPEG acceleration.

A. Tool-Handling

When students join our laboratory they have no knowledge about the tool Platform Creator. Therefore at the beginning we offer exercises focusing on different aspects of the tool. At the beginning a sample platform is generated. This minimized system is utilized to explore the process of compiling and running an application. Since we develop the hardware as well as the software, debugging is required for both sides. It is important to mention that the tool handling is the main focus of the exercises and not developing the system.

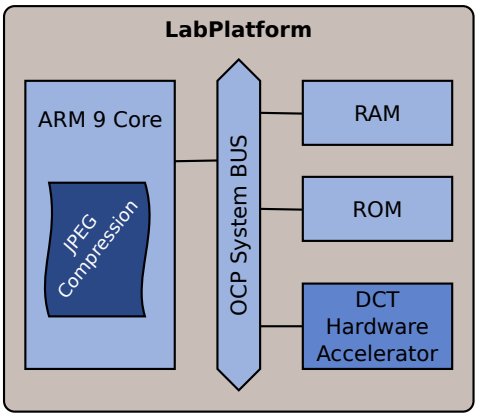

Figure 2. Platform for JPEG Compression with DCT Acceleration

B. Platform Design

Platform Creator enables developing the platform on the system level in a very elegant way. It offeres a wide range of predefined IP blocks like an ARM core or bus structures. During the first design phase the hardware accelerator is suppressed. The ARM core is connected via an OCP system bus to two memories. In order to guarantee that the platform is actually working a simple test program is cross-compiled and downloaded to the microcontroller.

C. Software Mapping

We provide a pure C implementation for our students which is not meant to run on the platform built in the previous phase. Their task is to adopt the source code to the new environment. Major distinction is the fact, that no file system is present on the target platform. Hence, an important step is to model all I/O accesses as memory requests. This includes that the input data have to be prepared for the software modification. The students are asked to realize their own approaches.

The Platform Creator allows to debug the hardware system as well as the downloaded software. A SystemC Debug interface and a GDB server accompany themselves, which gives the designer/programmer full control and transparency of its actions.

D. JPEG Acceleration

The last step in the laboratory is to accelerate the DCT for a JPEG compression. Here the students develop a strategy of how to communicate with the accelerator, modify the software code and design the actual accelerator.

For this task, the attendees extend the IP pool from Platform Creator. The tool supports two implementation styles. A C-style and a HDL-style. In both cases the functionality of the the IP block is encapsulated from the communication with the other components on the platform. The benefits are obviously. The students can start with the algorithms' partitioning without caring about the HDL implementation of the hardware accelerator. When the system executes successfully, the C model is replaced by a cycle accurate HDL implementation.

Duration	Section	Details
4 Sessions	Tool Handling	QuickStart: First Platform - Assamble a sample system - Instanciate a OCP bus SystemC Explorer - Using the build-in hardware debugging tool SystemC Shell - Commandline tool for SystemC simulation Component Wizzard - Introduction to the library approach - Creation of user-defined components
1 Session	Platform Design	Designing a platform for an ARM9 core - Understanding the design constraints - Composing the initial acceleration platform
3 Sessions	Software Mapping	Programming the Core - Writing test program - In-system debugging with GDB Algorithm Adoption - Introduction to JPEG compression - Source code modification
5 Sessions	JPEG Acceleration	Modification of Hardware and Software - Encapsulating the DCT from the software - Extending the bus system - Designing SystemC acceleration component - Implementing software call-routins Hardware Acceleration - Design hardware block for DCT - Integrate hardware block in the acceleration platform

Figure 3. Laboratory Schedule

IV. Laboratory Schedule

The laboratory is organized in weekly sessions respectively three hours. During each session the students are supervised and have the opportunity to ask questions. But they have also access to the laboratory beyond the guided sessions in order to complete their coding and debugging. At the beginning of each session tasks and goals are defined. This mechanism helps us to keep every group on track and shows the students if they work in time and if they achieve the expected progress.

We organized all sessions in four sections. The topic of these sections are tool handling, platform design, software development and JPEG acceleration. The schedule for each topic is given in figure 3.

The first section covers tool handling. This section tends to be very steered by the supervisor because we try to get our students as fast as possible productive with the new development environment. We figured out that the tool is too complex to let students explore its capabilities themselves. The required time is simply too large. But it is also imported to collect first user experiences. Thus we decided to create small challenges for this first section. They are not directly reusable for the actual accelerator platform design but demonstrate tool specific development techniques and strategies.

In the beginning we ask our students generate a first and very simple platform. It's a timer connected to a reset logic with a testbench unit. They shall understand the working directory structure and the method for design entry. The communication via busses and its memory map is introduced by adding a OCP bus connecting the timer with the testbench unit. Coware provides two tools for simulation and debugging. The SystemC Explorer is a GUI for platform debugging. It provides many visual representation like value traces, TLM port traces and process traces for checking the platform's execution. The build-in SystemC simulator can also be accessed via a GUI but Coware also allows to control the simulation via a SystemC shell (scsh). Our students learn to employ these tools on their simple platform.

Equipped with these skills the independent working effort

increases drastically. They shall composed an minimal platform for an ARM9 core. We provide an ARM specification that contains information about how the outer system has to look like concerning the address mapping and memory organization. In order to get the software properly working this specification needs to be fully adopted.

The software mapping is subdivided into two parts. First, our students are asked to program the ARM core with a pice of self-written C code. This program should be short and simple. It mainly serves as test for the cross compilation and software debugging on the virtual platform. Second, we introduce JPEG compression to compensate potential knowledge gabs in the field of image processing. The detailed understanding of JPEG is essential for the next step. We provide a C implementation of a functional reduced JPEG compression. The source code is meant to run on a normal PC with a terminal and a complex file system. The students need to identify parts in the source code that rely on these aspects and have to find alternatives that are feasible on the virtual platform.

The final section of the laboratory is JPEG acceleration. We chose the DCT for to be released because it is highly parallel and well portioned in the provided source code. Thus our students do not face too many problems with encapsulating the DCT from the remaining compression algorithm. The bus has to be extended for the new component and the SystemC component itself has to be designed and instantiated. For the first version the provided c-code is reused at the acceleration component. For the final version the software code shall be replaced by synthesisable VHDL code. The compression algorithm on the ARM core requires call-routines for read and write access to and from the accelerator in order to get working as intended.

V. Feedback from Students

The prevailing feedback is very positive. The project complexity and freedom to implement their own solution motivates our students. At the beginning we often have to discuss the necessarily of the detailed tool introduction. Our students tend to underestimate the effort of getting productive with an mature development framework. But our first year experience tells us that this time consuming introduction is necessary if our students shall be able to get all the work done during the mandatory time for the laboratory.

From time to time students do not see the necessity to have extra tutorials where basics of SystemC are touched. From our point of view this fundamental knowledge is important for an universal education. Without this manual coding experiences it is hard to estimate work effort on a higher level of abstraction.

VI. Conclusion

With this separation in three different teaching forms we are able to provide a high quality background during our lecture session paired with challenging implementation and evaluation tasks in our tutorials and laboratory at the same time. The attendees get to know implementation techniques for modern ESL design in theory and are asked to develop an own platform in our laboratory. The fundamentals of TLM with SystemC are detached from the complex platform development to ensure a deeper comprehension in both fields.

Acknowledgment

The authors would like to thank Intel Braunschweig who are sponsoring the professorship for our VLSI education. We also would like to thanks partners Coware and Europractice who have provided tools and licenses for this course.

References

[1] D. Densmore, A.Sangiovanni-Vincentelli, R. Passerone, A platform-based taxonomy for ESL design, IEEE Des. & Test Comput., Vol. 23, No 5, 2006, pp. 359-374.

[2] http://www.coware.com

[3] http://www.systemc.org

[4] D. C. Black, J. Donovan, "SystemC: From the Ground Up", Springer, 2005

[5] T. Groetker, S. Liao, G. Martin, S. Swan, "System Design with SystemC", Springer, 2001