Real-scale Product Development on System Design Course

Timo D. Hämäläinen[1], Panu Sjövall, Janne Virtanen, Sakari Lahti, Jarno Vanne

Tampere University of Technology, Department of Pervasive Computing
Tampere, Finland

Conference Key Areas: Engineering Skills, "I feel brilliant", Continuing Engineering Education and Lifelong Learning
Keywords: System Design, Design Flow, Product Development

1 INTRODUCTION

This paper presents the ideas and experiences on a new course, TIE-50506 System Design [1], for teaching system design for embedded systems. The target audience is the master's students in their last semester. The goal is to learn the most current tools, methodologies, and workflows used in industry. The main idea is to use an exercise project of such a large scale that students cannot comprehend every detail of it, which is the case in a real product development project for a new engineer. Still they should understand how the HW and SW parts contribute to the product and make right design decisions when exploring alternatives. This approach is quite the opposite compared to traditional teaching, which uses small toy problems as examples.

The outcome of our new course exercises is a complete streaming HEVC video encoder [2] running on a SoC-FPGA platform [12]. A unique feature is that the application was brought right into teaching while still being developed in a research project. In practice, the research staff collaborated to implement new features and fix bugs, and at the same time the teaching staff developed weekly exercise tasks for students. As far as we know, this is the fastest possible deployment of research results into teaching.

Figure 1 The course outcome: Live streaming HEVC encoder on SoC-FPGA.

[1] Corresponding author
timo.d.hamalainen@tut.fi

2 RELATED WORK

Embedded system design methodologies and tools have been intensively researched during the last years, motivated by autonomous cars and other cyber-physical systems. Many universities have developed their own tools and brought them to teaching, which naturally biases the scope of the courses. However, common to all is the adoption of *model based design* and *simulation* as the main verification approach. UC Berkeley has a long track record on system abstraction and modelling, thus they use Ptolemy tools, e.g., in EE249B [3]. SpeC was developed at UC Irvine and it is used, e.g., in EECS222A [4], as well as in UT Austin in EE382N.23 [5]. European universities have also adopted industry standard methods like SystemC, for example, ETH Zurich in course 227-0778-00 [6].
Likewise, we have developed HW/SW design tools and used them in our teaching. In addition, we have used Gajski's textbook [7] in similar courses. However, our approach has been more practical compared to the related courses. Some of them focus only on modelling at very high level or deal with languages and their properties, which is closer to research than teaching. Another difference is the scale of exercises. We use SoC-FPGA (processor+FPGA) board at the final stage, but most of the other courses use only simulators. We have moved to industry standard methods and no longer use own custom languages or non-standard tools. As a whole, our new course TIE-50506 covers more completely the design flow from specification and modelling to real implementation.

3 OVERVIEW OF THE COURSE

TIE-50506 is a 5 cp course for Master's students in their last semester. The prerequisites are digital design, computer architecture, and programming skills. The minimum is basics of C/C++ and creating FPGA implementations using VHDL/Verilog. Understanding basics of Linux and generally operating systems is also preferred. However, the course reviews the essentials of the prerequisites, which helps students with less knowledge of a specific field.
The course consists of 14 two-hour lectures. Ten exercises contribute to the final exercise product. Tasks are carried out in groups of two students in a computer classroom that has one PC and SoC-FPGA board per group. Each task has a return deadline, and groups can work on the tasks at any time the classroom is free. However, assistants are available every week in the classroom for help, and groups must sign up for the events.
Each task is credited 1-2 points and one bonus point. To pass the course, at least 80% of the total points must be collected. The course can be passed without exam by completing 100% of the tasks without bonus, which gives the lowest acceptable mark 1. Bonus points are added to the exam points, thus students may get the highest grade 5 by a combination of the tasks and exam.
The lectures and independent study for them takes in total 42 hours. The contact hours and independent study for exercises is 96 hours. These include the exam and self-study for tools, and the course size is in total 148 hours (5 cp).

4 COURSE INFRASTRUCTURE AND MATERIALS

Figure 2 depicts an overview of the tools used to manage the execution of the course. POP portal is used for scheduling all learning events, delivering lecture notes as pdf-files, and general messaging. KAIKU is a feedback system, with which students give their assessment of the courses. A traditional web page is used for exercise instructions, since they change most during the course. The well-known code development framework Gitlab [8] is the heart of the exercise management. It

offers a front end to a Git repository, and includes tools like issue tracker for the everyday chores between the collaborating persons. Repolainen is a dashboard that is used to create the student groups, GitLab projects, and Git repositories for each group. Repolainen is also used to populate and update the projects with exercise tasks, code and other information, and manage returns. The course staff can access all student group projects and repositories, but not the groups between themselves.

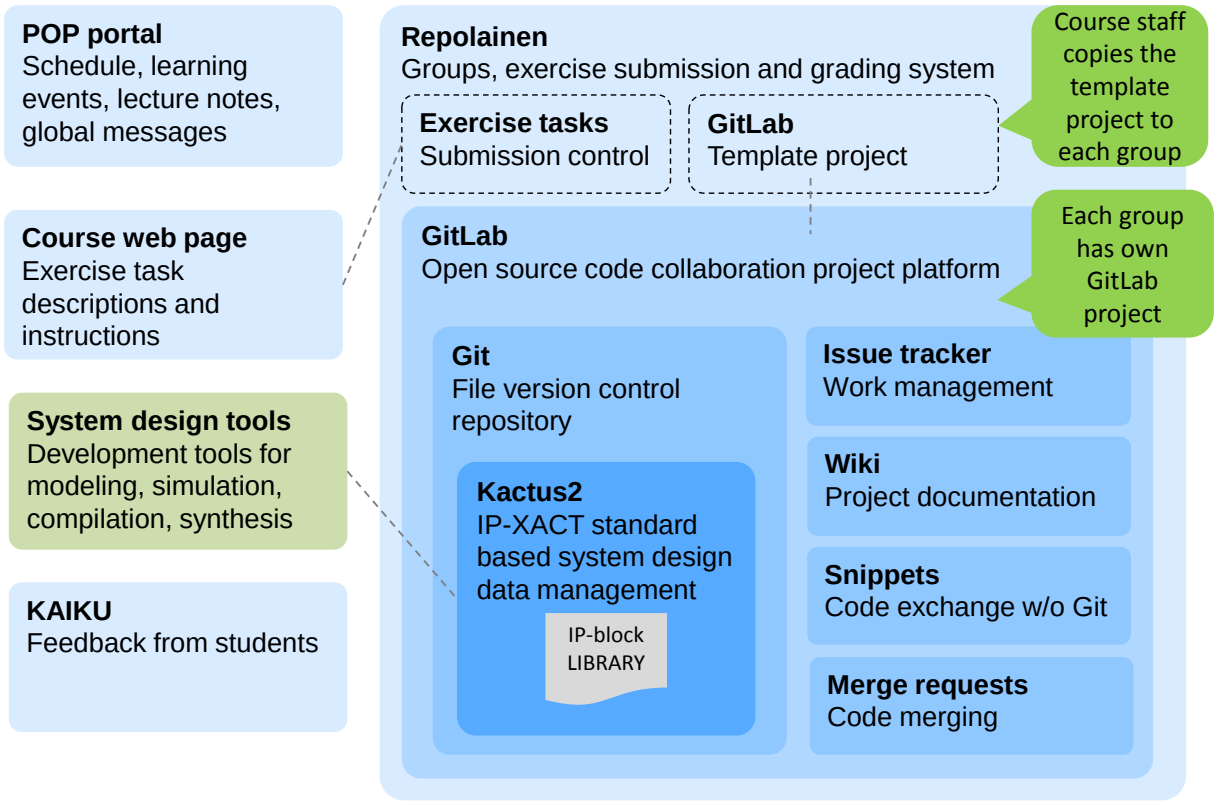

Figure 2 The course infrastructure.

Kactus2 [9] is an open source IEEE-1685 IP-XACT tool that is used to manage the product structure and launch design automation tools. The tasks are carried out in a Windows PC classroom. Each PC has a CentOS Linux virtual machine dedicated to this course. We use 28 tools in total, including, e.g., SystemC simulator, Catapult-C and FPGA tools, compilers, and custom shell scripts. In addition, Git repositories are populated with source code required in the exercises. Large files like test video sequences and Linux image are available outside Git in a shared folder for all students.

The lecture notes are developed by the course lecturer and they are based on several research articles, white papers, tutorials, manuals, and textbooks. The official textbooks are [7], [10] and [11], but only the lecture notes are required for the exam. The notes consist of 435 presentation slides in 2016.

5 DESIGN OF THE COURSE

The background of the new course is our previous courses on HW/SW co-design and model based design. They were overlapping, but students did not get enough training if they did not take both of them. In addition, the exercises and lectures were outdated and the scope gradually shifted from what companies require today and in the future. One major new target was fast deployment of top research into teaching, thus requiring more agile exercises than before.

The main goal is to teach system design with a large-scale real product case. We used our experience on the past and ongoing research projects with companies to get the requirements. The risky decision was to change everything compared to previous courses: the physical platform, operating system, modeling language, and most of the design tools. This also meant that the lectures and exercises had to be developed during the course and could not be fully completed beforehand.
The base for the course is contemporary system design flow. Since there is not a standard for it, we first designed the flow itself based on solid textbooks and our experience. After that, we created the lectures and exercises around it.
Figure 3 depicts the main steps of the design flow. For brevity, not all tools and details are disclosed. The green shaded steps are related to general modelling or SW, and the blue shaded to HW issues. For each step, the related exercise is depicted on the right column. Some steps, denoted by “S” in Figure 3, were performed by staff on behalf of students to avoid overloading.
The complexity of the exercise work can be depicted as follows. The application, Kvazaar HEVC encoder, includes over 20k lines of C code. Completely understanding HEVC takes about 3 months for a full-time engineer. Three SystemC models with different abstractions are created for getting familiar to the modeling and to explore potential HW/SW architectures for HEVC. To succeed in this, students need to know both the performance requirements of the HEVC and available performance of the platform. This is achieved by profiling and benchmarking.
The SW part includes preparing embedded Linux for the SoC-FPGA platform as well as developing kernel drivers for camera and HW accelerated parts. The HW part includes generating HW modules from C code using CatapultC [13] High Level Synthesis (HLS) tool. This is followed by creation of the FPGA project and integrating generated HW modules into it. The SoC-FPGA platform itself is very complex. Even booting it to “hello world” requires good knowledge. Thus, the high number of tools and numerous steps in the design flow are very demanding, for which reason the students are firmly guided through the flow. The exercise tasks include instructions what to do in each step. The returns include answers to specific questions, code, figures or values, design files, spent hours, and feedback.

6 EXECUTION OF THE COURSE

The course has been given twice: in spring 2015 and 2016. 105 students in total were signed up for the course implementations, out of which 85 passed the course after the first exams. It was noticed that most of the dropped students gave up right in the beginning. The common reasons were experienced feeling of difficulty and other overlapping courses.
Table 1 gives an overview of the weekly learning events without examination and holiday calendar weeks. The topics and order of the lectures and exercises is dictated by the design flow (Figure 3). Two of the lectures were self-study by video tutorials, and three were guest lectures from companies. Two lectures were compulsory: the first to get organized and the last to review and discuss with all staff and students present.
The staff consisted of a lecturer and three assistants. The lectures were implemented as traditional presentations, but focused on explaining the intent rather than lots of specific details.

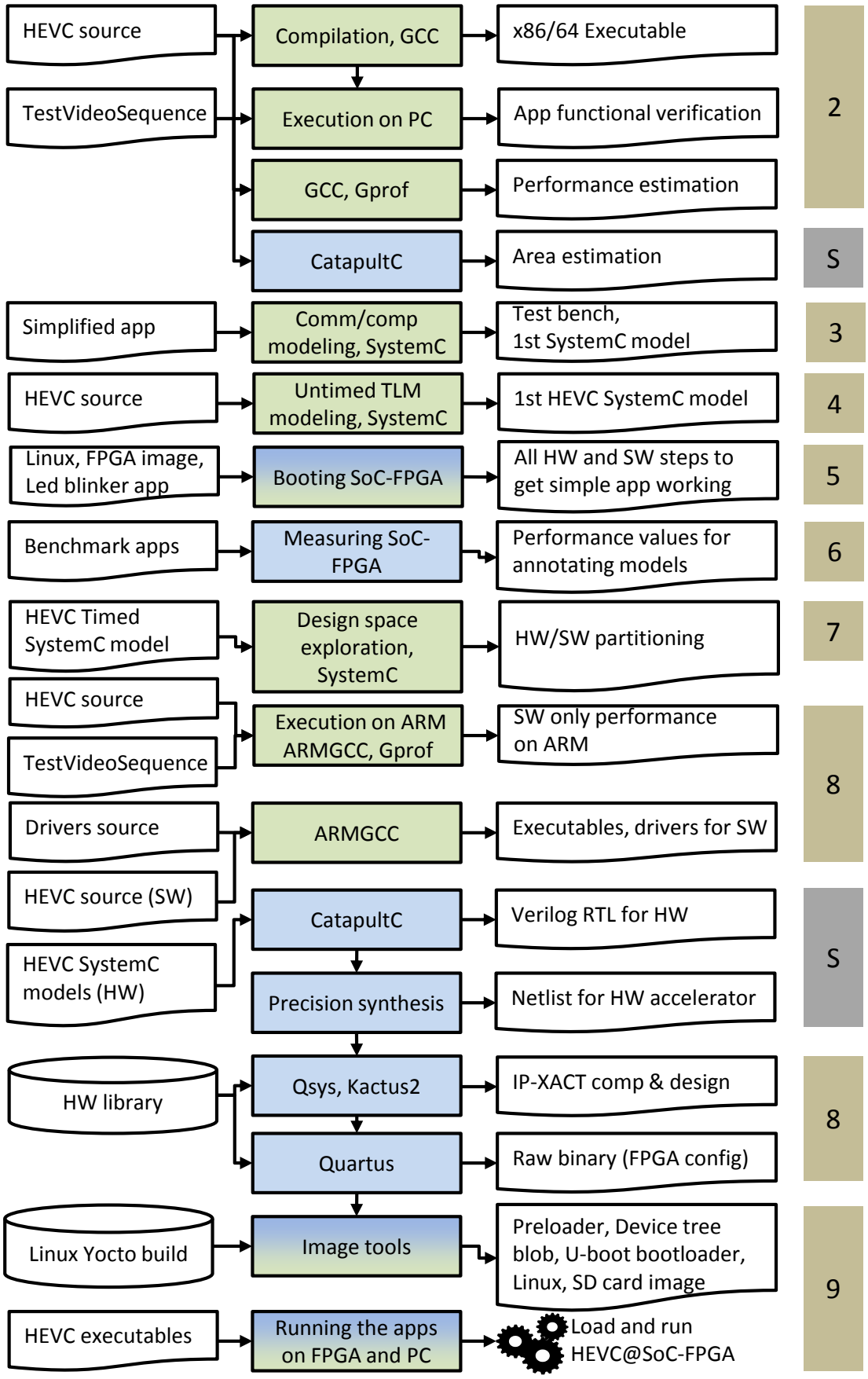

Figure 3 Exercises in the system design flow.

Three weekly parallel exercise groups were formed. The PC classroom occupies ten groups at the same time, for which reason a signup was required.
The exercise tasks were developed during the course, because the application and its FPGA implementations were developed in the research team at the same time. However, the staff performed peer reviews and testing before publishing the exercises to students. We clearly explained the ongoing development to the students, and committed them to a joint effort with staff. In addition, the difficulty level of each task was probed every week by asking spent hours and freeform feedback. This was very successful and we get regular feedback from all groups.

Table 1 Course schedule.

Wk	**Lecture topic**	**#**	**Exercise topic**
1	Intro to embedded system design		Self study: course infrastructure
2	System design flow	1	Git training
3	a) Modelling and simulation b) Self-study: SystemC tutorial	2	Introduction to HEVC application
4	a) SoC platforms modeling b) Self-study: TLM2.0	3	Communication and computation modelling in SystemC
5	Verification, principles (G)	4	Untimed TLM modeling in SystemC
6	FPGA-SoC Platform, memory, buses		
7	Catapult-C High Level Synthesis demo	5	Get familiar with SoC-FPGA Bonus: kernel module
8	Verification, UVM (G)	6	Platform modeling
9	Linux kernel drivers, HW/SW interface (G)	7	TLM level design space exploration
10	SoC interconnections		
11	IP-XACT standard	8	HW/SW integration
12	Questions&Answers	9	Complete streaming HEVC encoder Bonus: camera driver II
13	Review meeting	10	Final report

Figure 4 shows the reported hours per exercise in spring 2016, excluding #1 that was a fixed two hours session for all. The min, max, and average values are from all groups per exercise. It can be seen that #4 was clearly the most demanding. After adjusting the difficulty level, #5 was even a bit too easy, and the level was slightly adjusted in later exercises. It should be noted, however, that the weekly hours are more evenly distributed since some of the tasks had 3-4 weeks to the deadline. The performance of the groups varied quite a lot. The minimum total time was 58 hours, and one group spent 132 hours for the exercises. The average was 93 hours, which is very close to the target 96 hours.
The time spent by staff was roughly 350 hours including contact teaching, setting up and maintaining the infrastructure, updating the lectures and exercises, and examinations during the course implementation.

7 FEEDBACK

KAIKU was used to collect feedback from the student afterwards. During the course, some students complain about the large effort. However, we started to publish the statistics in Figure 4 after #4, which disclosed that either the feeling was a bit wrong or the reporting of hours had problems. In the last review lecture, we get very good understanding with the students about the goals and effort required for it. This is also seen in KAIKU feedback depicted in Figure 5. We got this assessment from 50 students. For comparison, we included the answers from all courses in our department (3449 responses). In the scale, one means “light workload wrt credits”,

two means “the workload met well the credits” and three means “heavy load wrt credits”. As a whole, it seems we succeeded better that the department on average.

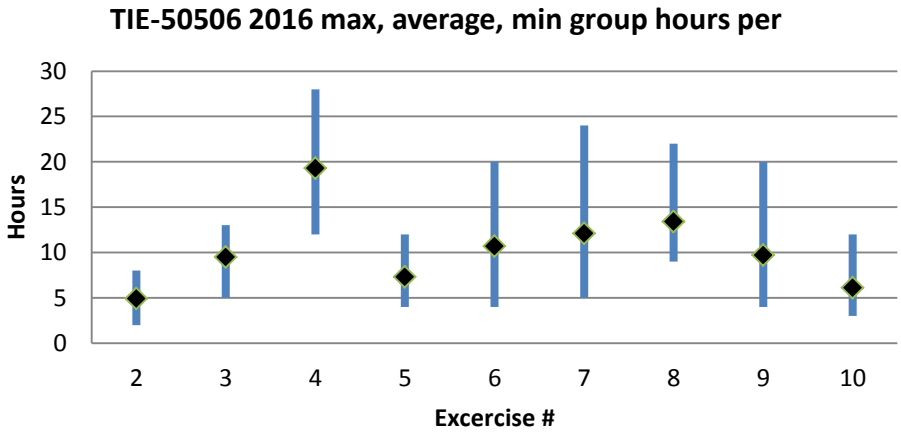

Figure 4 Reported time spent on the exercises.

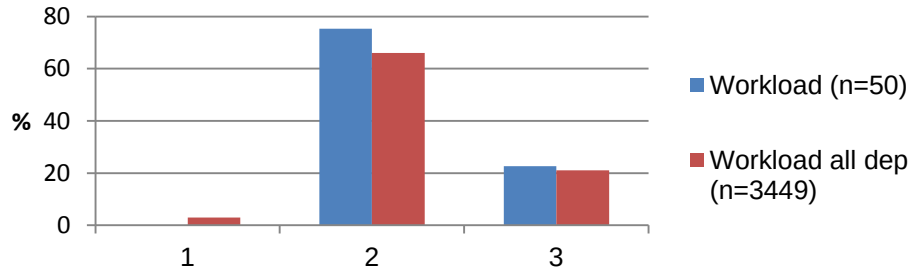

Figure 5 Experienced workload in comparison to all courses in the department.

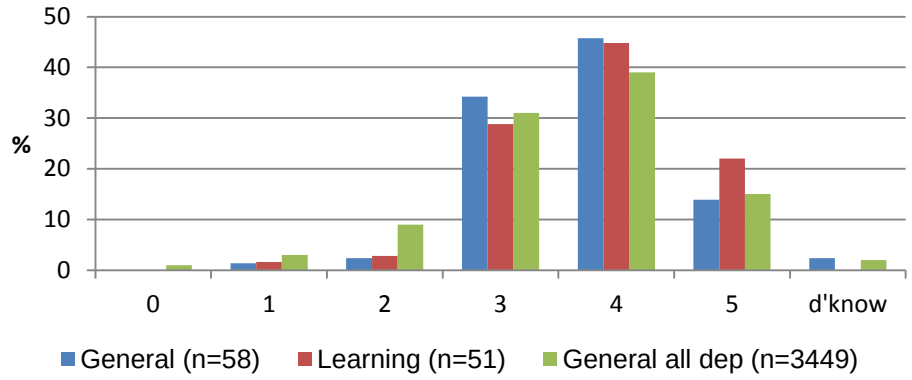

Figure 6 Results of the overall rating and learning outcomes.

Figure 6 summarizes the feedback for the overall rating and how well the course succeeded in learning outcomes. The scale is from excellent (5) to very poor (0) and “don’t know”. Compared to all courses in the department, we got better feedback especially for the learning outcomes. One student expressed this as follows: “*The course was great. I think the most important thing about it -- and what truly made the course worthwhile -- was the focus on model-based design: I had never received actual, proper training on it.*”

Most of the negative feedback concerned the tools used in the course: problems in file paths and complex user interface in many tools. However, the students were warned about those problems in the beginning, since this is a well-known “industry standard” and will not get much better in companies once the students are there. On the other hand, this helps in a way to understand the design intent first and only after that how it can be realized with the available tools at the moment.

8 CONCLUSIONS

The course succeeded in its goals as we can see from the student feedback. In addition, we were satisfied with their commitment and teaming up with staff to reach the goal. We also asked feedback from companies, which confirmed the right scale and goals. In the future, methodologies evolve and we should most likely change the platform or some other part of the whole. The agility of the design and collaboration with research teams will help in this. Finally, the students should understand the intent and methodology, and apply it to varying tools and platforms. As a conclusion, we successfully implemented the new kind of course with a large scale, real product by taking the application and its development workflow from ongoing research.

REFERENCES

[1] Tampere University of Technology, TIE-50506, http://www.tkt.cs.tut.fi/kurssit/50506, visited on May 24, 2016.

[2] Open source HEVC encoder Kvazaar, https://github.com/ultravideo/kvazaar, visited on May 24, 2016.

[3] UC Berkeley, EE249B: “Design of Embedded Systems: Models, Validation and Synthesis”

[4] UC Irvine, EECS222A: “SoC Description and Modeling”

[5] UT Austin, EE382N.23: “Embedded System Design and Modeling”

[6] ETH Zürich, 227-0778-00: “Hardware/Software Codesign”

[7] Daniel G. Gajski et al, “Embedded System Design - Modeling, Synthesis and Verification”, ISBN 978-1-4419-0504-8

[8] Gitlab community edition, https://about.gitlab.com/downloads/, visited on May 24, 2016.

[9] Kactus2, https://sf.net/p/kactus2, visited on May 24, 2016.

[10] David C. Black, Jack Donovan, Bill Bunton, Anna Keist, “SystemC: From the Ground Up”, ISBN 978-0-387-69957-8

[11] Peter Marvedel, “Embedded System Design - Embedded Systems Foundations of Cyber-Physical Systems”, ISBN 978-94-007-0257-8

[12] Terasic, http://www.terasic.com.tw/, visited on May 24, 2016.

[13] Mentor Graphics, https://www.mentor.com/hls-lp/, visited on May 24, 2016.