Teaching System-on-Chip design with FPGAs

Erno Salminen and Timo D. Hämäläinen
Tampere University of Technology
P.O. Box 553, FIN-33101
Tampere, Finland
{erno.salminen, timo.d.hamalainen} @ tut.fi

ABSTRACT

This paper presents our experiences in using FPGA in teaching System-on-Chip design in Tampere University of Technology. We had a major reform on our courses and, most notably, chose a common HW platform which is used in 11 courses. It has proved good that most exercises are mandatory and bonus points are awarded for good work. In order to manage the schedules, larger projects have been partitioned by the teachers into smaller tasks and pairwork is allowed. Automated testbenches, reuse, startup examples were very useful. As a result, we observed increased motivation among students and better learning outcomes. The schedule slippages were reduced, although both teachers and students still underestimate the required time and effort. Moreover, we introduce 15 student projects where FPGA platform was also used. Some of the most innovative topics were suggested by students themselves, such as games. In the future, more effort is needed is finalizing the project works for easier reuse and setting up a common repository.

Categories and Subject Descriptors

K.3 [**Computers and Education**]: Computer and Information Science Education—*curriculum*; B.7 [**Integrated Circuits**]: General; C.0 [**Computer Systems Organization**]: General

General Terms

Experimentation, Design

Keywords

education, system-on-chip (SoC), field-programmable gate array (FPGA), exercise works

1. INTRODUCTION

Modern System-on-Chip (SoC) devices are very complex devices containing about tens of reusable IP (intellectual property) cores per chip, which are heterogeneous [1–3]. They are widely used for example, in mobile phones and other consumer electronics, cars, and medical devices. Most SoCs utilize multiple programmable processors, embedded memories, hardware accelerators, and numerous external interfaces. In addition to HW, amount of embedded software increases very rapidly. Consequently, the design teams are big (e.g. tens to hundreds of engineers), development cycles are long (e.g. half to few years), and design challenges are numerous (e.g. requirement capture, HW design, SW design, verification...). The fundamental problem in universities is how to familiarize students to the design process of such complex systems, since it is evident that full-blown, detailed design projects are out of the question. Nevertheless, the most important topics can be taught and improvements in FPGA technology allow building affordable multiprocessor chips in a lab within a semester.

This paper presents the experiences of using FPGAs in SoC education. In 2008, a project was launched to modernize and enhance System-on-Chip education at Department of Computer Systems in Tampere University of Technology (TUT), Finland [5]. The department is nowadays part of the Department of Pervasive Computing which provides education e.g. in digital systems design, software engineering, and human-computer interaction [4]. The department has a faculty of 11 professors and about 120 other employees.

The reform concentrated the laboratory exercise works in our courses. A common hardware platform, Altera/Terasic DE2 board [10], was chosen and used in several consecutive courses. Moreover, students may borrow a personal unit to implement exercises and hobby projects also outside the classrooms. The aim is to motivate the students by providing experiences of tangible working hardware, since pencil-and-paper designs or simulations alone are not sufficient. We discuss here our experiences so far.

The rest of paper is organized as follows. Sections 2 and 3 present briefly the learning objectives, related works, and the chosen exercise platform. Sections 4 and 5 describe the structured course exercises and the projects where students have much more freedom. Finally, we draw conclusions.

2. RELATED WORKS

The learning objectives for Bachelor and Master degree at TUT state that student understands scientific method and attains a competence to work in expert, development and managerial jobs in the ICT field. Roughly speaking Bachelors are able to implement a system according to specifica-

Table 1: Courses using FPGAs in Tampere University of Technology. PP denotes pen-and-paper exercises.

#	Course	Exercise	Estim. hours	Design style	Notes
1	TKT-1100 Basic digital design	7-seg ctrl	4 - 8	Schem.	opt., +16h PP
2	TKT-1202 Digital design	Pocket calc. + misc.	24 - 50	Schem, FSM	+ 8-16h PP
3	TKT-1212 Impl. of digital systems	Audio synth.	30 - 75	VHDL	+ 2-6h PP
4	TKT-3200 Computer architecture	Paint program	3 - 20	IP blocks, C	incl. mouse
5	TKT-1230 Computer arithmetics	MUL, area opt, SW prof.	6 - 15	Schem, C	+ 8-16 PP
6	TKT-1400 ASIC design	Alarm clock	50 - 140	VHDL	incl. LCD
7	TKT-2431 SoC design	Video enc. (1xCPU+acc)	40 - 100	IP blocks, C	FPS competition
8	TKT-3541 SoC platform	Video enc. (2xCPU+acc)	60 - 120	IP blocks, C	incl. OS
9	TKT-1410 System verification	Test SW for mem + acc	4 - 10	C	+ 8-15h UVM
10	TKT-3520 Processor design	SW prof. + opt C2H	2 - 5	C	+ 20-40h TCE
11	TKT-1570/2520 BSc/Project work	Varying topic	80 - 220	VHDL, IP, C...	See next section

tion and Masters are able to write the specification as well.

Many universities teach HW and embedded system design with FPGAs, see for example [6–9]. However, most publications mention just a single course or at most a couple. For example, Shi et al. have presented a comprehensive course that covers embedded systems and processors, HDL, real-time OS, and GUI programming with VGA [7]. Digital signal processing using Labview and FPGA, VHDL, both floating and fixed-point computation was presented in [8]. A 3-course curriculum from basic logic to 8-bit CPU with Verilog, and to timing analysis and test is presented in [6]. Closest to ours are the two project courses which are based on a generic multiprocessor FPGA platform: one concentrates in real-time embedded system programming and the other more HW-oriented dealing with integration of an accelerator unit [9].

To our knowledge, our curriculum with 11 courses and scope of topics (digital logic, VHDL, C, IP integration, verification etc.) is the largest.

3. COMMON FPGA PLATFORM

The platform must have enough capacity to facilitate meaningful applications. We chose the Altera/Terasic DE2 development and education board [10] that is part of Altera's university program. It has Altera Cyclone II EP2C35 FPGA which features over 30 000 logic elements (LE), 480 kb RAM, embedded multipliers and Phase-locked loops (PLLs). FPGA is large enough to host e.g. several synthesizable soft-core processors allowing one to build own multiprocessor-system-on-chip. Operating frequency is usually around 100 MHz.

The development board includes several external memories: 8 MByte SDRAM, 512 KByte SRAM, 4 MByte Flash. The user interface uses switches, buttons, LEDs, LCD and 7-segment displays; whereas the other interfaces include RS-232, infrared, video/audio in/out, PS/2, Ethernet, USB, general-purpose signal pins.

The package includes Altera's Quartus II synthesis tool, and HW design entry can be VHDL/Verilog description or schematics. The FPGA configuration is either programmed via USB interface or stored in the flash memory, which enables stand-alone operation. The functionality of the implementation can be inspected by a synthesizable logic analyzer called SignalTap. There are many FPGA development boards and the exact type is not critical. Similar projects can be carried out with other boards but we found at least DE2 very suitable.

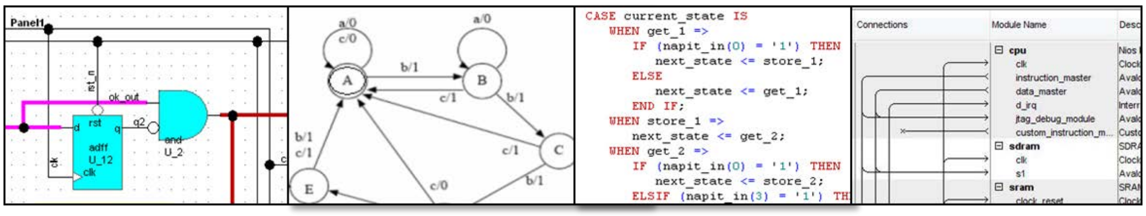
4. OUR COURSES

The courses that utilize the common hardware platform or the related design tools are summarized in Table 1. Both authors have taught on these courses for many years, especially on $\#2-3, 7-8$. Most courses are 5 credit points as defined in European Credit Transfer and Accumulation System (ECTS), and their duration is one semester (12-16 weeks). Hence, expected time usage is around 100-120 hours per student per course and we allocate approx. half to the exercises. Pairwork is allowed in most cases and also encouraged as it teaches so called metaskills, such as communication, scheduling, delegation, and reporting. However, the required effort has been balanced not to overwhelm an individual student.

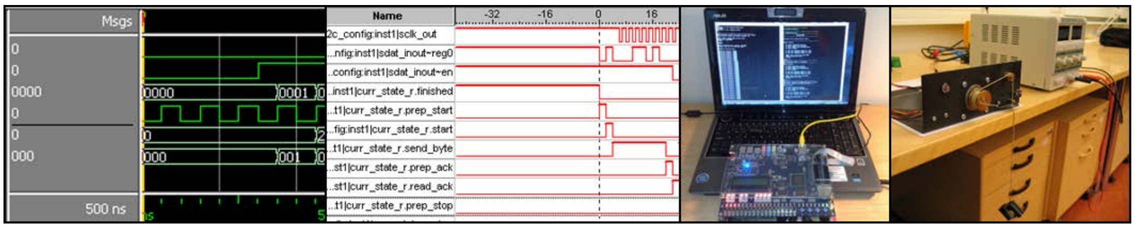
The first courses belong to bachelor studies, the rest to master studies. Bachelor thesis seminar and project work naturally have varying topics, not necessarily using FPGAs, and hence listed as last. Bachelor level courses are part of one study module but master level has a couple of modules. Most courses have both pen-and-paper and computer exercises. Table lists estimated hours that one student spends for FPGA and EDA exercises. The logic design exercises start from simple combinatorial circuits, to finite state machines (FSM), to block level design. Many exercises utilize Altera's Nios II processor core, as illustrated in Fig.1(a). Similarly, several verification and debug means are utilized, for example simulation, logic analyzer, and debug print messages, as illustrated in Fig 1(b).

4.1 Bachelor level

The first computer exercise in course #1 is a state-machine for blinking student number on 7-segment display. It is captured using gate-level schematic entry and synthesized, whereas simulation is optional. The main purpose is to introduce FPGAs for the freshmen and encourage them for pursuing HW studies.


```
CASE current_state IS
  WHEN get_1 =>
    IF (napit_in(0) = '1') THEN
      next_state <= store_1;
    ELSE
      next_state <= get_1;
    END IF;
  WHEN store_1 =>
    next_state <= get_2;
  WHEN get_2 =>
    IF (napit_in(0) = '1') THEN
      next_state <= store_2;
    ELSIF (napit_in(3) = '1') TH
```

(a) Design capture styles



(b) Debug styles

Figure 1: Various styles for design capture and debug

The next courses #2 − 3 teach more about simulation as well as higher level design capture, such as state machines, hierarchical design, and VHDL. Both pocket calculator and audio synthesizer are built piece by piece. Some exercises are first planned on pen-and-paper and then completed with computer. The latter interfaces with Wolfson audio codec.

Computer architecture #4 has a small FPGA exercise demonstrating development of embedded SW that interfaces with mouse and VGA. Students are given a half-ready program template which must be completed to create a simple paint program. Other exercises are done with pen-and-paper and use different simulators for ISA, pipelining and caches. Computer arithmetic course #5 was augmented with computer exercises to allow better verification and also to motivate students. Exercises demonstrate different multiplier structures and teach logic optimization. Even tens of percents of area can be saved by tuning the internal bit widths, pipelining, and turning multiplication into shift operations. The last exercise measures the execution time differences between variable types in SW (e.g. *int* vs. *double*), operations (e.g. +,−, *sin*()), whether the CPU includes a HW multiplication or division unit.

4.2 Master level

The courses #6 − 8 have the key content in ASIC and SoC design. ASIC design offers more freedom than earlier courses since students write the technical specification of the digital stopwatch/alarm clock themselves. They also review other groups' specifications and implement their own system on an FPGA board.

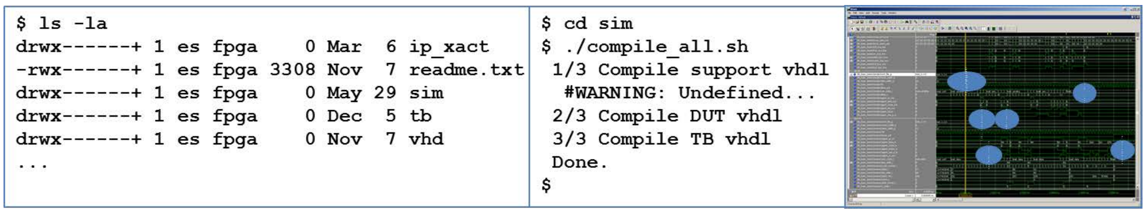
SoC design is pre-requisite for the SoC platforms, but distinction is not strict. The most important concepts and phases in system design are introduced, especially Intellectual Property (IP) reuse, platform-based design, and HW/SW co-design. A video encoder is first implemented on workstation, then on an embedded CPU, and finally an accelerator unit is integrated. SoC platform course goes further in abstraction, IP reuse, and embedded SW including OS. The platform has two CPUs and the design is captured in Kactus2 integration tool [15, 16] to promote reuse and proper IP packetization. Kactus2 uses IP-XACT standard and students also describe their own IPs with it. Other main lessons here are writing SW drivers for an accelerator, writing multithreaded program, and configuring the DMA controllers. This is one of the hardest courses but also closest to real-life design problems.

Most parts of system verification course (#9) deal with SystemVerilog but students also write simple system-level test SW to verify the connections, memory, and an accelerator in an FPGA system. Here the accelerator is broken on purpose and students must locate the bug and circumvent it with SW (design a 'workaround'). The main emphasis in processor design (#10) is freely available[1] TCE ASIP framework [17] but FPGA is used for profiling applications on Nios. Hours for FPGA works are listed for these courses.

4.3 Findings

We redesigned tens of exercise tasks and tied them better to the lectures. The importance of EDA tools and utilization of FPGA was pronounced compared to previous course implementations. Time needed to study the manuals of diverse platforms and tools was successfully reduced and education can concentrate towards core contents. Our self-assessment together with hour reports and feedback from students confirmed that reforms performed rather well, even

[1] `http://tce.cs.tut.fi/index.html`



```
$ ls -la
drwx------+ 1 es fpga    0 Mar  6 ip_xact
-rwx------+ 1 es fpga 3308 Nov  7 readme.txt
drwx------+ 1 es fpga    0 May 29 sim
drwx------+ 1 es fpga    0 Dec  5 tb
drwx------+ 1 es fpga    0 Nov  7 vhd
...
```

```
$ cd sim
$ ./compile_all.sh
 1/3 Compile support vhdl
  #WARNING: Undefined...
 2/3 Compile DUT vhdl
 3/3 Compile TB vhdl
 Done.
$
```

1. After checkout you should have these files... 2 . Compile the source codes by... 3. Simulate and you should see...

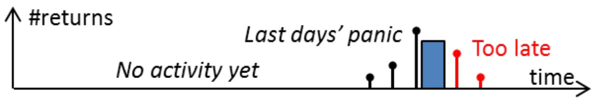
Figure 2: An example of easy startup instructions for a reusable IP component.

if some courses take quite much time. FPGAs were not the only factor affecting success but they played a major role. Next we discuss a few key observations and examples.

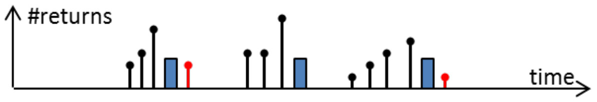
System design is a craft which cannot be learned solely from a text book but it requires lots of hands-on experience. Most our exercises are mandatory; either the students select a certain number of tasks from a list or all of them must be completed before attending an exam. Bonus points can be earned for doing little extra work or doing the work exceptionally well. Bonuses help getting a better grade and motivate the students to dig deeper into the problems at hand. For most cases we prefer numerical grades over the *pass - no pass* grading, which unfortunately seems to increase minimum effort attitude among students. In SoC courses the groups with the fastest video encoder (largest frames per second, fps) earn extra points. This has stirred friendly competition among the most active students and, even better, they have come up with some optimizations that the assistants did not think of. Clearer learning objectives, reasonable workload, and guest lectures from industry were also part of the motivational reforms.

An example of motivational hands-on is a simple skip-and-jump chicken game implemented in VHDL and students must implement a keyboard interface to play it. Similar keyboard interface could be checked by just printing the letters but that would be severely less fun and motivating. In general graphics and sounds are more interesting and satisfying than blinking LEDs or printed console messages. One exercise has even a small step-motor elevator, shown in Fig. 1(b), to really see the impact of design choices (and bugs). Here students must debug a half-ready PWM and other elevator controller units. These cases emphasize an important designer skill: how to figure out what a system or IP - implemented by others - does and how to use and modify it.

We have observed two important issues. Ready-made testbenches are excellent in basic courses and executable examples are excellent everywhere. Testbenches reduce the burden of a teaching assistant when checking the answers and teach the students the benefits of automated tests. Moreover, HW and SW designs are rather "strict" disciplines in general compared to essays and there should be very little discussion whether the student's design is good enough or not. Therefore, an agreed-upon correct behavior is best captured with automated testbench. Reusable testbenches are naturally out of the question when lots of freedom is allowed in implementation.



(a) Single deadline for the whole exercise,



(b) Three deadlines: two for sub-units and one for integration.

Figure 3: Conceptual difference between one large exercise and the split one.

Another thing in reuse is that easy-start examples for IP blocks are extremely helpful. It is very frustrating when a new "ready" system or IP does absolutely nothing on the first try, users have never seen it working, and have no clue how to fix it. Surely an IP provider can write tens of pages of documentation (few people do and nobody updates them) but they are hard to read, and especially hard to convert into something executable.

Simple *Hello World* -like examples verify that at least all the files are present and uncorrupted, all EDA tools, licenses, firewalls are properly configured etc., and they show how the most basic operation look like in wave window/command promtp/etc. Note that functional testbenches are different concept and often hard to understand when starting with a new IP. We have often used screen captures from command prompt and simulator's wave window. They are annotated with text boxes highlighting the most important points, as illustrated in Fig. 2. New user gets immediately a sense how things should look like when everything goes well. Clean compilation is desired but not always possible, e.g. some 3rd party codes might produce harmless but nasty-looking warnings. It is a good custom to give the new user a heads up that such might appear. Moreover, once the example has been successfully repeated - hopefully in a matter of minutes - one can imitate or copy-paste it to create something original. IP vendors and designers must also pay attention to code quality when we keep in mind that developers actually spend twice the time reading codes compared to writing them [11].

Table 2: **Examples of students projects at TUT**

#	Topic	CPU	Audio	VGA	SD card	Notes
1	Doom + bootloader	1	x	x	x	+keyboard, `https://github.com/japeq/nios-doom`
2- 3	Tetris, Breakout	1	x	x	x	+keyboard
4	Xvid player	1	-	x	x	uses open src codec
5	Oscilloscope	1	x	x	-	-
6	WAV player	1	x	-	x	-
7-8	Ogg Vorbis, mp3 players	1	x	-	x	use open src codecs
9	Guitar effects	0	x	-	-	distortion, delay
10	Guitar effects	1	x	-	-	distortion, flanger
11	NES+C64 audio chip	0	x	-	-	same group in both
12	NoC traffic generator	0	-	-	-	`http://www.tkt.cs.tut.fi/research/nocbench/download.html`
13	GPS datalogger	0	-	-	-	incl. UDP/IP + Eth
14	Reading inertial sensor	0	-	-	-	incl. UDP/IP + Eth
15	Interpolated FIR	0	-	-	-	incl. UDP/IP + Eth

Our course personnel has partitioned the exercises into smaller chunks, e.g. weekly tasks each requiring 1-5 hours of work and each task is part of the larger entity. Students have found it motivating to get something non-trivial working in the end, especially when it is on FPGA and not just a simulator.

Surprisingly many student return their work just before the deadline and unexpected (yet very common) problems cause deadline violations, especially when many courses have their deadlines near semester end. Fig. 3 sketches the conceptual difference between one large assignment and one split into 3 pieces. Smaller milestones spread the work more uniformly to different weeks, reduce the chances of going totally astray and minimize schedule slippages. It also reminds that "last night miracles" seldom work in real design projects. We had limited statistics about this from one course. There the acceptance rate for student reports returned on the last night was only 40% whereas it was about 80% for others. Moreover, we recommend that the most demanding exercise is not the last one but perhaps second last. Then students have still time to make corrections before the exam and the deadline does not interfere so much with other courses. Asking students to estimate their time usage and to make small initial plans encourages them to start working earlier. However, very exact partitioning leaves less room for innovative solutions. Teachers must seek a good balance and allow more freedom in the advanced courses.

Hour estimates and reports from 7 courses showed that both teachers and students *practically always* underestimate the required effort. Actual time usage was usually $10 - 80\%$ larger than estimated by the students and sometimes even $2x - 3x$. Interestingly, students tolerated quite large workloads well if the work was considered useful, e.g. optimizing own solution instead of fighting against EDA tools. Moreover, very large differences were observed between student groups: the fastest group needed about $0.5x$ time of an average group and slowest needed $2x$. Differences were even larger in individual exercises and each course had at least one exercise where the slowest groups needed over $10x$ time compared to the fastest. Therefore, teachers and project managers must be careful with schedules, hour reports are necessary, and students must learn to assess their own skills.

We prioritized the course goals and removed some topics considered less relevant in order not to overwhelm students. Better instructions for simulators and other tools were prepared to let students concentrate on the "real" content.

5. EXAMPLE STUDENT PROJECTS

A selection of 15 student projects using FPGA is summarized in Table 2. They are categorized roughly according to their type and the utilized external components. Most are implemented as BSc thesis projects (8 cr.). About half are pairwork. Most topics were proposed by course personnel but some were students own ideas, such as Doom game on Nios [12] and modeling of Nintendo Entertainment System audio chip [13]. Even the proposed topics had much freedom on which features to include and how to implement them. Some works were part of a research project, e.g. Traffic generator [14], GPS datalogger, and sensor interface.

5.1 Games and video projects

Doom is a classic action game, originally developed for IBM PC in 1993. Source codes - around 70 000 lines of C - were published in 1999 under the GPL license. Two students implemented Nios II system on DE2 using SOPC builder and the game reached 30 fps refresh rate [12]. Moreover, a boot loader program was developed and permanently stored in the flash memory, enabling stand-alone operation of the system. Programs can be downloaded from portable SD (Secure Digital) memory devices that utilize the FAT file system.

Handling of the SD card and FAT have been reused in other projects, such as video and audio players. Xvid player was a SW-only prototype that concentrated on the integration and reuse instead of raw performance. Tetris and Breakout, were notably simpler one-person projects developed from scratch. The paint exercise from computer architecture course served as an introduction to VGA graphics for these and also for the oscilloscope work. It uses ADC of the audio chip to capture incoming signal from LINE IN at 96 $kSPS$ (kilosamples per second) and stores it to SDRAM. The signal can be shown on VGA or downloaded to PC via UART. Despite the modest sample rate and undesired filtering in audio chip, all the major concepts were demonstrated and this could perhaps be turned into a course exercise.

5.2 Audio projects

The first audio project (#6) implemented a WAV player using Nios and SD card. The SD card drivers were reused from Doom but all other SW was written from scratch. The more advanced Ogg Vorbis and mp3 players were able to reuse earlier works and had a quick start. Therefore, they could better concentrate on the intricacies of the open source SW codecs (tens of thousands source lines of code). Both also implemented special instructions to Nios II to speedup the decoding. The results were very good: $4x - 6x$ speedup with modest effort. All these projects made small additions or corrections to SD card drivers which simplified the later projects.

Two projects implemented guitar effects, such as distortion, delay and flanger. The first one was entirely on HW whereas. The latter was SW which reused earlier interface units and hence could it concentrate more on the effects themselves.

Nintendo Entertainment System (NES) is a classical 8-bit game console from the 1980's and its audio chip was modeled in FPGA. It includes four 4-bit square wave channels, one 4-bit triangle wave, one 4-bit noise, and one 7-bit deltamodulation channel. The implementation took about 2 000 lines of VHDL, mere 830 LUTs, and the audio quality was very close to the original (both European and US NES). Later, the same designer modified the model to act like Commodore 64 audio chip which has somewhat more complicated functions, such as envelope generation. Again reuse paid off very clearly and the live demos were very entertaining.

5.3 Other projects

Project #12 implemented a synthesizable traffic generator that can be used for benchmarking network-on-chip designs in simulator and FPGA [14]. The generators were configured and results were gathered via Ethernet using UDP/IP protocol. The UDP unit turned out useful also by itself. For example, GPS datalogger uses general-purpose I/O pins to read coordinates from a commercial GPS chip and sends them to PC via UDP for further analysis. Very similar project reads inertial sensor chip to get the acceleration and gyroscope values.

Interpolated finite impulse response (IFIR) filter work concentrated first on the filter itself and was tested in simulator. The final demonstrator sends both coefficients and source data from Matlab via UDP to FPGA, filters the data, and returns the results to PC. They are compared to Matlab's built-in functions and the differences due to finite word widths and rounding can be analyzed.

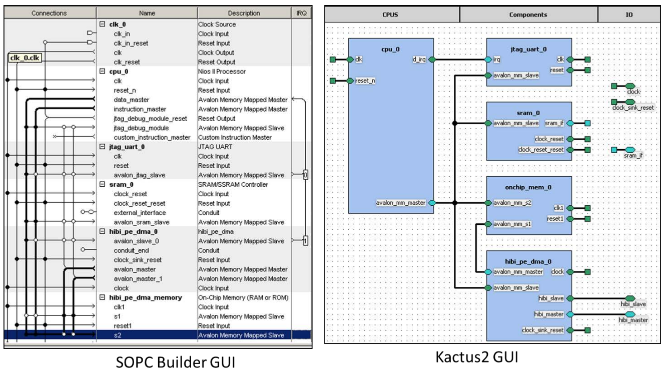
There have been also few other very innovative projects based on microcontrollers, which were left out for brevity. For example, a homemade film laboratory had one controller for a step motor rolling the film and the other for keeping developer liquids in constant temperature. Other fascinating examples are automated bicycle gears and a driving computer for a farm sowing machine which detects power train malfunctions and seed level running low.

5.4 Findings

Project are closer to real-world design problems than courses and therefore recommended, especially they teach information gathering, problem solving, and verification. We have noticed two things needing improvements.

Although the designs worked, no effort was spent making reuse easy for the next user. In hindsight, separate finalization and packetization phase would have been rather easy and fast to do, and should be required from now on. We have invested lots of effort in Kactus IP-XACT tool that is meant to speedup reuse and embedded system design [15, 16]. It offers efficient GUI, shown in Fig 4, for design capture and product data management. Abstracting the bus interfaces of HW components and column-based 2D layout make design view clear compared to, e.g. SOPC builder. Moreover, IP-XACT is preferred format since it is vendor-independent standard. Kactus supports managing not only SoC HW and SW, but also higher levels such as board, software and mapping.

Moreover, too often the earlier works have been spread around to students' and supervisors' hard drives. Common repository should be used in order to get full benefits. That helps also maintaining the older works and providing bug fixes and feature updates.



Connections	Name	Description	IRQ
	clk_0	Clock Source	
	clk_in	Clock Input	
	clk_in_reset	Reset Input	
	clk	Clock Output	
	clk_reset	Reset Output	
	cpu_0	Nios II Processor	
	clk	Clock Input	
	reset_n	Reset Input	
	data_master	Avalon Memory Mapped Master	
	instruction_master	Avalon Memory Mapped Master	
	jtag_debug_module_reset	Reset Output	
	jtag_debug_module	Avalon Memory Mapped Slave	
	custom_instruction_master	Custom Instruction Master	
	jtag_uart_0	JTAG UART	
	clk	Clock Input	
	reset	Reset Input	
	avalon_jtag_slave	Avalon Memory Mapped Slave	0
	sram_0	SRAM/SSRAM Controller	
	clock_reset	Clock Input	
	clock_reset_reset	Reset Input	
	external_interface	Conduit	
	avalon_sram_slave	Avalon Memory Mapped Slave	
	hibi_pe_dma_0	hibi_pe_dma	
	avalon_slave_0	Avalon Memory Mapped Slave	1
	conduit_end	Conduit	
	clock_sink_reset	Reset Input	
	avalon_master	Avalon Memory Mapped Master	
	avalon_master_1	Avalon Memory Mapped Master	
	clock	Clock Input	
	hibi_pe_dma_memory	On-Chip Memory (RAM or ROM)	
	clk1	Clock Input	
	s1	Avalon Memory Mapped Slave	
	reset1	Reset Input	
	s2	Avalon Memory Mapped Slave	

Figure 4: The graphical user interfaces of SOPC Builder and Kactus2

6. CONCLUSIONS

Our course reform concentrated on using a common FPGA platform and EDA tools on many courses and proved successful, because students' satisfaction and motivation increased, and number of late returns decreased. Reuse was emphasized and offered notable speedup is student projects. Ready-made system partitioning, testbenches, and working for a larger device piece-by-piece were found good approaches. More emphasis is still needed in finalizing design projects to simplify and speedup reuse.

ACKNOWLEDGEMENT

The work was supported by the Technology Industries of Finland Centennial Foundation. We thank all teachers in the project and of course our innovative students as well.

7. REFERENCES

[1] A. Sangiovanni-Vincentelli. Quo Vadis SLD: Reasoning about Trends and Challenges of System-Level Design.

Proc. IEEE, Mar 2007, Vol. 95, Iss. 3, pp. 467-506.

[2] F.R. Wagner *et al.*, Strategies for the integration of hardware and software IP components in embedded systems-on-chip, Integration, the VLSI Journal, Sep 2004, Vol. 37, Iss. 4, pp. 223-252.

[3] W. Wolf, A.A. Jerraya, and G. Martin. Multiprocessor system-on-chip (MPSoC) technology. IEEE Tran. Computer-Aided Design of Integrated Circuits and Systems, Oct 2008, Vol. 27, Iss. 10, pp. 1701-1713.

[4] Department of Pervasive Computing, [Online] Available: `http://www.tut.fi/en/about-tut/departments/pervasive-computing/`

[5] O. Vainio, E. Salminen, and J. Takala, Teaching Digital Systems Using a Unified FPGA Platform, in BEC, 2010, 4 pages.

[6] J.D. Lynch, D. Hammerstrom, and R. Kravitz, A cohesive FPGA-based system-on-chip design curriculum, in MSE, June 2005, pp.17-18.

[7] Qingsong Shi, Lingxiang Xiang, Tianzhou Chen, and Wei Hu, FPGA-based Embedded System Education, in ETCS, 2009, pp. 123-127.

[8] N. Kehtarnavaz and S. Mahotra, FPGA implementation made easy for applied digital signal processing courses, in ICASSP, 2011, pp. 2892-2895.

[9] A. Kumar, S. Fernando, and R.C. Panicker, Project-Based Learning in Embedded Systems Education Using an FPGA Platform, to appear in IEEE Tran. Education, 2013, 9 pages.

[10] DE2 Development and Education Board, Altera Corporation, [Online] Available:`http://www.altera.com/education/univ/materials/boards/de2/unv-de2-board.html`

[11] A.J. Ko, B.A. Myers, M.J. Coblenz, H.H. Aung , An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks, IEEE Tran. Software Engineering, Dec. 2006, Vol. 32, No.12, pp. 971-987.

[12] J. Kulmala and J. Järvinen, Doom on the FPGA Development Board, B.Sc. thesis, Tampere University of Technology, 2009, 28 pages (in Finnish).

[13] J. Helkala, NES console's audio processing unit on a FPGA chip, B.Sc. thesis, Tampere University of Technology, 2012, 28 pages (in Finnish).

[14] J. Nieminen, Synthesizable traffic generator for becnhmarking network-on-chips, B.Sc. thesis, Tampere University of Technology, 2010, 30 pages (in Finnish).

[15] L. Matilainen, A. Kamppi, J-M. Määttä, E.Salminen, T.D. Hämäläinen, "KACTUS2: IP-XACT/IEEE1685 compatible design environment for embedded MP-SoC products", TUT Report 37, 2011, 49 pages.

[16] Kactus2 project page, [Online], Available: `http://sourceforge.net/projects/kactus2/`.

[17] O. Esko *et al.*, Customized Exposed Datapath Soft-Core Design Flow with Compiler Support, in FPL, Aug-Sep 2010, 6 pages.