A Unified SoC Lab Course: Combined Teaching of Mixed Signal Aspects, System Integration, Software Development and Documentation

Johannes Pfau*, Richard Leys†, Marc Neu*, Alexey Serdyuk*, Ivan Peric† and Jürgen Becker*
†Institut für Prozessdatenverarbeitung und Elektronik, Karlsruher Institut für Technologie
*Institut für Technik der Informationsverarbeitung, Karlsruher Institut für Technologie
Email: pfau@kit.edu, richard.leys@kit.edu, neu@kit.edu, serdyuk@kit.edu,
ivan.peric@kit.edu, becker@kit.edu

***Abstract*—University courses for System-on-Chip (SoC) design mostly focus on particular aspects. Whereas this can provide detailed understanding of these aspects, it neglects system integration specific topics such as crossing digital and analog domains. In addition, many courses skip practical issues and do not teach Electronic Design Automation (EDA) tools. This is reasonable in the context of a specialized course, but omitting these techniques often prevents students from making active use of the learned knowledge in their own projects.**

In the following, we present our technological platform to teach SoC design in a holistic lab course: The course takes students from writing the first line of Verilog code to advanced digital and analog design. It introduces simulation of digital and analog systems, debugging methods for software and hardware, CPU bus architecture, custom peripherals and driver development. This work is prototyped using Field Programmable Gate Arrays (FPGAs) and later transferred to an ASIC target, covering standard cell synthesis and analog layout. At the end of the semester, students finalize the project with technical documentation writing. The course is built on the design of an audio peripheral, combining all topics in a single real-world system. It enables students to apply theoretical aspects from various lectures in the SoC curriculum in practice and equips them with the skills needed to dive deeper into each of the involved topics on their own.

***Index Terms*—FPGA, ASIC, SOC, Mixed Signal, Teaching**

I. Introduction

SoC curricula like [1] cover various digital and analog design courses, but often with little cross-course integration of both. Furthermore, software development for SoCs is usually excluded, or part of another isolated lecture. Lab courses like [2] and [3] add hands-on experience, but don't teach mixed-signal issues or documentation. And whereas [4] integrates a complex digital and analog design, each student only works on a small part of the system individually. Without a course integrating the lectures of the curriculum for each single student, it is difficult for students to see cohesiveness and practical application of the learned knowledge. This may demotivate students and therefore reduce learning success. To fill the gap, we conceive a lab course for our SoC teaching curriculum which combines knowledge from major lectures and applies it in a single application.

We integrate digital design and simulation as in [5], adding analog aspects, driver development, debugging and documentation writing. To motivate using a real-world application, we base the course on the design of an audio peripheral for an existing SoC. Unlike existing courses, we introduce basic concepts and focus primarily on a cohesive overview over the whole problem space. In order not to overwhelm students, we only quickly cover CPU architecture, unlike [6]. Using a RISC-V Central Processing Unit (CPU) enables students to make use of the vast landscape of RISC-V educational materials [7], [8] to obtain further knowledge on their own. We focus on Verilog as it's less complex than Chisel [9] and quicker to learn for students new to hardware description. Understanding basic concepts in Verilog enables enhanced understanding of Chisel afterwards. And whereas [10] stresses importance of integration frameworks such as Chipyard [11], we focus on development of peripherals from ground up. With this course providing the groundwork and integration knowledge, expanding on any of the covered topics will be left for advanced courses or to students own exploration.

As previous work has identified the importance of remote work [12], [13], we enable remote access. To cover physical debugging, setting up of boards and probing signals, we additionally require lab attendance in normal cases.

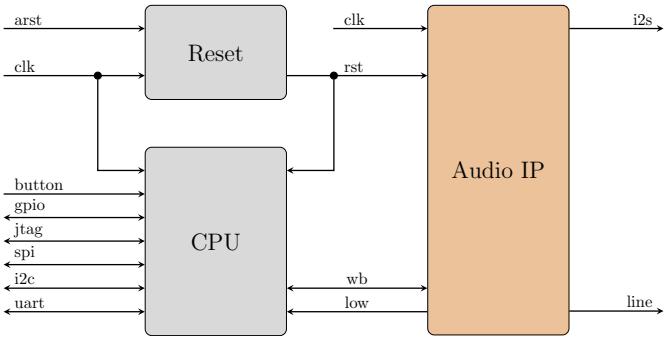

Fig. 1. Block design of the final audio player target application platform. Students are focusing primarily on the audio peripheral marked in orange.

II. Target Application Platform

Figure 1 shows the top-level block design of the lab course platform. To keep the design easy to understand for students,

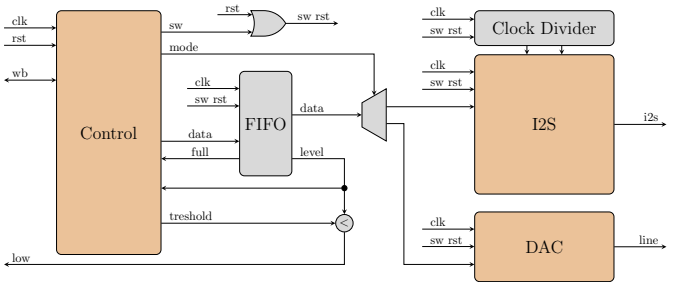

Fig. 2. Sample solution implementation of the audio peripheral. Blocks the students are either developing from scratch or extending are shown in orange.

it consists of only three blocks: A basic reset block, the main CPU block and the audio peripheral. The reset block is used to synchronize an external, asynchronous reset signal. The CPU is an instance of the open-source NEORV32 SoC [14]. It provides a CPU implementing the open RISC-V specification [15] and various peripherals. The lab platform uses the General Purpose Input / Output (GPIO) module to access LEDs, buttons and general purpose IO, the JTAG module for software debugging, the SPI module for SD card access, the I2C module for the I2C display and the UART module for text output and programming. It also makes use of the Execute in Place (XIP) module to execute code directly from an externally connected SPI flash.

The top-level design is kept generic to fit both the FPGA prototype and the final Application Specific Integrated Circuit (ASIC) target. It avoids introduction of target-specific signals and where vendor specific IP cores are required, they are abstracted to a common interface for both FPGA and ASIC. Due to the dominating amount of digital logic in the project, we adopt a digital-on-top flow, although students are introduced to the other integration possibilities as well. As the digital-on-top flow also fits the FPGA prototyping method best, the course can naturally proceed from FPGA prototyping to ASIC implementation. We use Digilent ZedBoards for the FPGA prototyping and the AMS AH18 technology for the ASIC target. For the CPU, NEORV32's main teaching benefits include extensive documentation with both user documentation and internal documentation, a simple, easy to understand implementation, an uncomplicated build process using only VHDL sources, a large ecosystem with ready to use drivers and compilers, and a well-integrated software stack. A high quality of implementation, documentation and reduced complexity is needed to enable beginner students to understand the system quickly. The NEORV32 ecosystem also provides a ready to use FreeRTOS implementation [16], which is used for a real SD card based audio player application. This application is provided for demonstration purposes, but not worked on by students in the lab.

Figure 2 shows the detailed implementation of the audio peripheral, with students working on parts marked in orange. The connection to the CPU is realized as a wishbone bus responder, which is implemented in the Control module. It implements the address space as shown in table I and

TABLE I
REGISTER SPACE FOR THE AUDIO PERIPHERAL

0x00 CTRL0 RW IP Core Control Register

31 – 4	3	2	1	0
Reserved	I2S_EN	DAC_EN	MODE	RST

I2S_EN Set to 1 to enable I2S module.
DAC_EN Set to 1 to enable DAC module.
MODE Set to 1 to select DAC output. 0 selects I2S output.
RST Set to 1 to keep peripheral in reset.

0x04 STAT0 R IP Core Status Register

31 – 3	2	1	0
Reserved	FULL	EMPTY	LOW

FULL Reads 1 if FIFO is full.
EMPTY Reads 1 if FIFO is empty.
LOW Reads 1 if FIFO level is below threshold.

0x08 FIFO_LOW RW FIFO Low Threshold Register

0x0c FIFO_LEVEL R Current FIFO Level Register

0x10 AUDIO_LEFT W Left Audio Sample Register

31	30 – 24	23 – 0
COMMIT	Reserved	DATA

COMMIT Set to 1 to commit left and right sample to FIFO.
DATA Audio data for left channel.

interacts with the corresponding control signals. This includes read-only, write-only and read-write registers, teaching the commonly used variations. The *CTRL0* register includes a software reset, enable signals for the peripherals and a mode selector, realized using the multiplexer in fig. 2. When writing this register, the values need to be passed to the peripherals. In addition, it should be possible to read back the values of the register. The *STAT0* register on the other hand is only readable and provides access to the FIFO status flags. *FIFO_LOW* realizes another read-write register which can be used to specify the level below which the low condition of the FIFO will be set. This condition is not only available in the status register, but is also exposed as a port from the audio peripheral. It is connected to the XIRQ module of the NEORV32, providing an interrupt when the audio FIFO level is below a configurable threshold. The current level can also be read from the *FIFO_LEVEL* register. *AUDIO_LEFT* and *AUDIO_RIGHT* are symmetrical write-only registers to put audio data into the FIFO. Applications are expected to write both registers in arbitrary order and set the *COMMIT* bit with the second write. The module internally buffers both 24 bit samples and forwards them as one 48 bit value to the FIFO.

To develop the system piece by piece in various lab sessions, students first start with a reduced top design, which does not include the Control and CPU modules. Instead, it consists of only the I2S module and a sine generator developed by students and a pre-provided FIFO connecting both. These modules teach students digital design aspects, including clocking, implementing state machines and driving FPGA output signals. Testbenches are provided to test the individual modules early on, but students also need to implement tests on their own to combine both modules. When both modules are

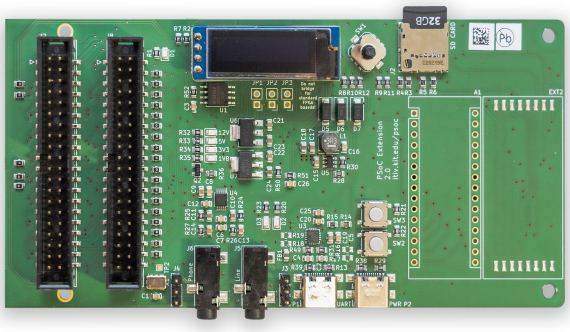

Fig. 3. Photograph showing the top side of the extension PCB. The FMC connector connecting to the base board is located on the bottom side.

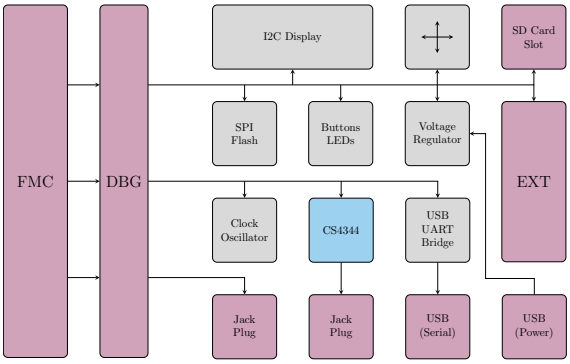

Fig. 4. Major components on the FMC extension board. Parts shown in red are connectors, the blue part is the DAC used to test the students' I2S modules.

implemented, students are taught how to prototype on FPGAs and test the system on the ZedBoard. We then introduce various debugging aids: Analyzing the generated I2S signal using logic analyzers, capturing internal signals using the Integrated Logic Analyzer (ILA) and observing the analog sine wave generated by an external Digital to Analog Converter (DAC) IC using oscilloscopes.

Once this standalone system is correctly implemented, students will proceed to package the I2S module into a reusable wishbone peripheral. They will derive the register space according to given specifications, then implement the wishbone protocol and Control module. They will learn how to design a register space and how address spaces are used in SoCs.

III. Peripheral Extension PCB

To enable students to test their I2S module in a real system, the ADAU1761 chip on the ZedBoard could be used. This approach has two drawbacks: The I2S signals are not readily available on the board for easy debugging. Furthermore, the ADAU1761 is a complex audio codec, needing to be configured via SPI. This introduces possible sources of error, which can trip up beginner students and move away the focus from the system integration aspect of the lab. We therefore designed the custom extension board shown in fig. 3, interfacing to the FMC connector on the ZedBoard. Its main components are shown in fig. 4 and are built around the CS4344 DAC [17], which as opposed to the ADAU1761 does not require initialization. All signals used on the extension board are routed to a debug connector (DBG) in addition, providing easy debugging access for oscilloscope, logic analyzer or JTAG testing.

The board furthermore provides various convenience peripherals, enabling full use of the developed SoC as an audio player: An I2C OLED display, a directional switch located next to it for navigation and an SD card slot for data storage. A voltage regulation subsystem ensures that all peripherals can either be powered from a 12 V supply provided via the FMC connector, or from externally supplied 5 V. The board is ready to be used with non-FPGA carrier boards, in which case it can also provide the generated voltages on the FMC connector. SPI flash provides a convenient way to permanently store the SoC firmware. It is programmed using NEORV32's default bootloader via a UART interface. This interface is connected to a USB to UART bridge, providing simple access to firmware programming from the lab PCs.

In addition, the board provides a local clock oscillator as main clock reference for the FPGA. We chose a clock frequency of 98.304 MHz, which can be divided evenly to the required I2S clocks. We thus avoid the need to use a Phase Locked Loop (PLL) IP core for clock generation and enable the design to operate in a single clock domain. Providing all required peripherals and voltages, the board can also be used with the taped out ASIC. This way, the ASIC can be tested in the identical environment as the FPGA, avoiding errors caused by mismatches in different board designs.

IV. Software and Driver Design

In order to conveniently use the audio peripheral in software, a device driver is needed. As the main focus of the lab is mixed-signal SoC, drivers are kept simple using a bare-metal system and plain C. They are developed following the NEORV32 Software Development Kit (SDK) as a reference.

The basic driver consists of C-header file with definitions for registers from table I, and a C-source file with the implementations of access functions. Its Application Programming Interface (API) includes an initialization function, setters and getters for registers and an audio data transmit function. Blocking and non-blocking mode are supported, where the non-blocking mode is implemented using the dedicated interrupt channel and a callback function. This function is called whenever the *FIFO_LEVEL* register value reaches the threshold, defined by the *FIFO_LOW* register. It is used to push further data into the FIFO of the audio peripheral.

Students implement their own device driver by first studying the contents of the NEORV32 SDK: linker file, startup code, common headers and existing peripherals. They will first implement the less complex blocking API and will further learn to describe their peripheral using System View Description (SVD) [18]. Although initially developed for the ARM-based SoCs, this format has been adopted by various vendors and the open source community for other architectures. In the

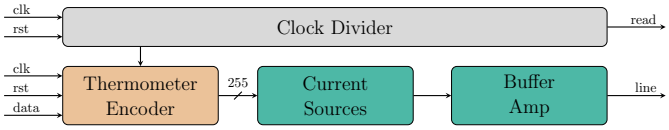

Fig. 5. Sample solution for the DAC block. Digital blocks the students are developing are shown in orange, analog blocks they are designing in green.

NEORV32 SDK, it is used to provide information about peripheral registers for debug software. Students will add the description of their peripheral, learning to use it with Joint Test Action Group (JTAG)-based debugging. The software part further teaches how to check the registers in a debugger and how to implement a non-blocking driver.

V. Mixed-Signal ASIC Design

Figure 5 depicts the system level overview of the DAC in detail. All digital signals are synchronous to the main clock and the analog output of the thermometer-coded DAC is amplified and made available to the physical interface. The clock divider module serves a special purpose in the ASIC design: A strobe signal is utilized to enable the clock input of the thermometer encoder, defining the sampling frequency of the DAC while avoiding explicit handling of clock domain crossings as in [19]. The thermometer encoder module features an 8-bit parallel bus to the FIFO module, so only most significant output bits are read. The strobe output is used as data read signal for the FIFO, propagating the used sampling rate up the signal chain. The thermometer encoder switches 255 current sources on or off, and their currents are summed and passed to a set of amplifiers. Those convert the current to a voltage, then buffer it to ensure low output impedance.

During the lab, students design and verify the functionality of the thermometer encoder module. The synthesis process is performed in Cadence Genus, the implementation in Cadence Innovus. Special emphasis is put on the differences between behavioral simulation, simulation after synthesis and simulation after implementation. In addition, the concepts of setup and hold times are explained as well as the role of clock skew in ASIC design. For the analog components, theoretic aspects such as MOSFET basics, different amplifier base structures and differences between single ended and differential amplifiers are reiterated. Students use Cadence Virtuoso to design a schematic for a differential amplifier, simulate it and perform a stability analysis. They then proceed to integrate the amplifier with a provided DAC cell and the digital thermometer encoder. After performing automatic placement and automatic routing as well as manual placement and manual routing, students use Design Rule Check (DRC) and Layout Versus Schematic (LVS) checks to validate their design.

VI. Documentation and Didactics

The lab uses modern tools to relieve students from dealing with tool setup. We use a Linux network boot system, ensuring identical setups on all available lab PCs. Students data is shared across all PCs and the Linux setup ensures that students can't mess up the installation in any way. All required software is preinstalled. For the most-commonly used tools in the lab such as Vivado, Git and Cadence tools, we provide hands-on tutorials showing real workflows. Students can work in the lab outside of course sessions and in addition we provide remote desktop access to a centralized server with identical setup.

To improve the learning experience of students, we make use of various modern didactic approaches: The lab itself is designed around a set of defined learning objectives, which are communicated to the students early on in the first lab session. The course sessions then adhere to constructive alignment [20], focusing on the defined learning objectives. The grading technique is based on the objectives as well, basing the grade on a learning portfolio [21], [22]: A folder of documents managed by students collecting all the important lab resources. To make portfolios comparable, we specify the contents: Work sheet solutions for theoretical aspects, which have to be prepared ahead of the labs, protocols for the labs themselves and a final summary. The final summary for the lab project must be given as a datasheet documenting the audio peripheral and driver, teaching proper documentation techniques. Each student's learning outcomes are assessed using this portfolio and individual short oral exams during the semester and after the course.

VII. Conclusion

We have shown how our SoC lab course teaches students to develop digital systems in Verilog, simulate and prototype them on FPGAs and debug them using ILA, logic analyzer and oscilloscopes. They learn to integrate their own audio peripheral within an existing SoC, design, implement and document the register set for the peripheral and how to connect it to the wishbone bus. In developing software drivers for the peripheral and relating address spaces in hardware and software, they also learn how to access peripherals in embedded C code in general. Students further work on a DAC and amplifier system to directly provide analog outputs for audio. They learn to design schematics, to simulate and to layout. The course then concludes with integration of analog and digital designs. This integrated approach teaches students the basics of every aspect, connecting topics from various lectures in the curriculum. It provides a bridge between those lectures and shows how the lecture knowledge can be applied in practice. On a scale of 1 (best) to 5, course evaluation by students in the winter term 2023 shows improvements in enjoyment of course attendance (1.57), importance of the experiments for further studies (1.21) and the quality of the experimental setup (1.77). The course changes described here therefore seem to better motivate students as intended.

In the future, we hope to improve the analog design and mixed-signal integration aspects and tape out our reference design. Additionally, we'd like to introduce some simple hardware/software co-design considerations, for example using NEORV32's extensions capabilities to implement special instructions.

REFERENCES

[1] J. Lynch, D. Hammerstrom, and R. Kravitz, "A cohesive fpga-based system-on-chip design curriculum," in *2005 IEEE International Conference on Microelectronic Systems Education (MSE'05)*, 2005, pp. 17–18.

[2] Y. Tang, L. M. Head, R. P. Ramachandran, and L. M. Chatman, "Vertical integration of system-on-chip concepts in the digital design curriculum," *IEEE Transactions on Education*, vol. 54, no. 2, pp. 188–196, 2011.

[3] F. Passe, M. Canesche, O. P. V. Neto, J. A. Nacif, and R. Ferreira, "Mind the gap: Bridging verilog and computer architecture," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[4] D. C. Burnett, B. Kilberg, R. Zoll, O. Khan, and K. S. J. Pister, "Tapeout class: Taking students from schematic to silicon in one semester," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.

[5] D. L. Knox, "Integrating design and simulation into a computer architecture course," in *Proceedings of the 2nd Conference on Integrating Technology into Computer Science Education*, ser. ITiCSE '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 42–44. [Online]. Available: https://doi.org/10.1145/268819.268834

[6] A. Amid, A. Ou, K. Asanović, Y. S. Shao, and B. Nikolić, "Vertically integrated computing labs using open-source hardware generators and cloud-hosted fpgas," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[7] RISC-V International, "Risc-v learn online," Oct. 2022. [Online]. Available: https://riscv.org/risc-v-learn-online/

[8] S. Wallentowitz, "Risc v in practical education of computer architecture," *Proceedings of the RISCV summit 2019*, 2019.

[9] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," in *DAC Design Automation Conference 2012*, 2012, pp. 1212–1221.

[10] B. Nikolic, E. Alon, and K. Asanovic, "Generating the next wave of custom silicon," in *ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference (ESSCIRC)*, 2018, pp. 6–11.

[11] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, and B. Nikolić, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[12] C. Monzo, G. Cobo, J. A. Morán, E. Santamaría, and D. García-Solórzano, "Remote laboratory for online engineering education: The rlab-uoc-fpga case study," *Electronics*, vol. 10, no. 9, p. 1072, May 2021. [Online]. Available: http://dx.doi.org/10.3390/electronics10091072

[13] M. Canesche, L. Bragança, O. P. V. Neto, J. A. Nacif, and R. Ferreira, "Google colab cad4u: Hands-on cloud laboratories for digital design," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[14] S. Nolting and All the Awesome Contributors, "The neorv32 risc-v processor," Sep. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.7109792

[15] RISC-V International, "Risc-v technical specifications," Oct. 2022. [Online]. Available: https://riscv.org/technical/specifications/

[16] FreeRTOS Authors, "Freertos - real-time operating system for microcontrollers," Oct. 2022. [Online]. Available: https://www.freertos.org

[17] Cirrus Logic, "Cs4344/5/8 10-pin, 24-bit, 192 khz stereo d/a converter," Jul. 2013. [Online]. Available: https://statics.cirrus.com/pubs/proDatasheet/CS4344-45-48_F2.pdf

[18] ARM-Software Authors, "System view description," Oct. 2022. [Online]. Available: https://arm-software.github.io/CMSIS_5/SVD/html/index.html

[19] A. B. Mehta, *Clock Domain Crossing (CDC) Verification*. Cham: Springer International Publishing, 2018, pp. 149–166. [Online]. Available: https://doi.org/10.1007/978-3-319-59418-7_8

[20] J. Biggs, "Enhancing teaching through constructive alignment," *Higher Education*, vol. 32, no. 3, pp. 347–364, Oct 1996. [Online]. Available: https://doi.org/10.1007/BF00138871

[21] C. Porter and J. Cleland, *The Portfolio as a Learning Strategy.* ERIC, 1995.

[22] J. Zubizarreta, *The learning portfolio: Reflective practice for improving student learning*. John Wiley & Sons, 2009.