Teaching Heterogeneous Computing with System-Level Design Methods

Luca P. Carloni, Emilio G. Cota, Giuseppe Di Guglielmo, Davide Giri
Jihye Kwon, Paolo Mantovani, Luca Piccolboni, and Michele Petracca*
luca@cs.columbia.edu
Department of Computer Science
Columbia University in the City of New York, New York, NY 10027

ABSTRACT

We present our work at Columbia University teaching the design and programming of heterogeneous computing architectures with SLD methods. Over the past eight years, we have developed a new course, *System-on-Chip Platforms*, with the main goal of preparing students to contribute to the new economy of heterogeneous computing and open-source hardware. The course was one of the first nationwide to introduce the use of commercial high-level synthesis tools for the design of application-specific hardware accelerators. We also introduced the idea of structuring the final project as a design-space exploration contest that combines aspects of collaborative engineering and design for reusability.

KEYWORDS

system-on-chip (SoC), system-level design (SLD), design reuse, accelerators, open-source hardware, collaborative engineering.

1 INTRODUCTION

Heterogeneity is the best answer to the challenges that computer architecture has faced over the past fifteen years. First the end of Dennard's ideal CMOS scaling [14] and then the accelerating slowdown of Moore's Law [22, 24] have forced computer architects to design processors by combining growing numbers of components of different natures. General-purpose processors, special-purpose processors, application-specific hardware accelerators, reconfigurable hardware blocks, and even analog/mixed-signal components are increasingly combined into what used to be a microprocessor chip and is now a *system-on-chip (SoC)*.

This evolution has happened across different computer classes and different application domains. The design of high-performance microprocessors for both personal computers and servers has seen the migration of more and more components from the board into the chip as well as the transfer of critical functionality from software to specialized hardware [15, 36]. For example, in 2012 Intel presented Ivy Bridge as a "22nm IA multi-CPU and GPU System-on-Chip" [27], which integrates up to four high-performance Intel Architecture (IA) cores, a power/performance-optimized GPU, as well as memory, PCIe, and display controllers in the same die. An SoC lies at the core of each smartphone and the heterogeneity of these SoCs has been rising with each new product generation. For example, the analysis of die photos of three generations of Apple SoCs, which empower the iPhone product line, shows that more than half of the chip area is dedicated to specialized intellectual property (IP) blocks that are neither CPUs nor GPUs [63]. Traditionally, SoCs have been the dominant processor architectures in most embedded system domains, from automotive electronics [65] and avionics [40, 62] to the Internet-of-Things [57, 59]. Recently, however, heterogeneous computing and SoC architectures are expanding from the edge to the cloud [21, 39]. For example, the importance of having SoCs that are optimized for the proprietary workloads in the data center has brought many top players in the Information Technology industry—companies that were originally focused on software development—to start new initiatives for designing their own chips [13, 38, 48].

In this context, opportunities for computer engineers, system architects, and integrated circuit designers are expected to continue to grow. In their 2018 Turing Award's lecture "A New Golden Age for Computer Architecture," Hennessy and Patterson identified some opportunities for innovation in the development of domain-specific hardware and programming languages, open instruction set architectures, enhanced security, and agile chip-design methodologies [35]. They remarked *"The good news for architects is that modern electronic computer aided design (ECAD) tools raise the level of abstraction, enabling agile development, and this higher level of abstraction increases reuse across designs."*

Raising the level of abstraction in the design process has been the main motivation of our CAD research in the System-Level Design Group at Columbia University for the past decade [16, 17]. In turn, it has informed our teaching activities, particularly through the development of a new course called *"System-on-Chip Platforms"*. First offered in 2011 as a graduate-level seminar, the course has been regularly taught every year since. It evolved into an undergraduate upper-level course and, in 2016, became part of the curriculum for the Computer Engineering Program, which is jointly supported by the Departments of Computer Science and Electrical Engineering.

SoC Platforms is a course on the programming, design, and validation of SoCs with the methods of *system-level design (SLD)* [16, 60]. It is the first course to introduce state-of-the-art commercial SLD tools to Columbia students, in particular tools for *high-level synthesis (HLS)* [47]. The students use HLS to optimize the design of accelerators from specifications that they have made in high-level languages like SystemC [37, 52].

A *platform* is a combination of a flexible architecture and a companion design methodology [17]. In teaching SoC Platforms, we put particular emphasis on SoCs for high-performance embedded applications. We believe, however, that the course provides a broader

*M. Petracca is now with Cadence Design Systems.

WCAE'19, June 22, 2019, Phoenix, AZ, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6842-1/19/06…$15.00
https://doi.org/10.1145/3338698.3338893

foundation on the principles and practices of reasoning about any heterogeneous computing platform in a compositional way and with a system perspective.

2 THE GOALS OF THE COURSE

In SoC Platforms, students learn the principles of:

(1) raising the level of abstraction in the SoC design process;
(2) mastering the hardware and software aspects of integrating heterogeneous components into a complete computing system through a compositional approach;
(3) designing new components that are reusable across different systems, product generations, and implementation technologies (e.g., FPGAs and standard cells); and
(4) evaluating designs in a multi-objective optimization space that includes both logical and physical properties.

Raising the level of abstraction in the design process means:

- for design specification, move from hardware-description languages like Verilog or VHDL to high-level programming languages like C/C++ or SystemC;
- for design simulation, move from register-transfer level (RTL) simulators to virtual platforms that enable faster simulation times under more significant environment conditions and early start of the software development with more detailed hardware models, which do not need to be manually "translated" into RTL.
- for design optimization, move from logic synthesis to high-level synthesis, which provides a rich set of knobs for a better exploration of the design space.

Notice that with these "moves", we do not advocate the abandoning of RTL design under all possible circumstances. We believe that there are still some important parts of any digital circuit that are best designed with Verilog/VHDL, simulated at the RTL level, and optimized with logic synthesis. We argue, however, that for the design of hundreds of heterogeneous components, which are integrated into a billion-transistor SoC, working at the proposed higher level of abstraction is a better way of investing the bulk of the design effort.

To simplify the integration of heterogeneous components, we propose the use of customizable libraries of hardware/software interfaces that provide modularity and flexibility by decoupling computation tasks from communication tasks. This follows the "Protocols and Shells Paradigm" of *latency-insensitive design* [16, 19].

We expect that future SoC will be realized by reusing a growing number of pre-designed and pre-validated components, which may be available in-house from prior SoC design efforts or may be licensable as third-party IP blocks. This trend could be further enhanced by the emergence of *open-source hardware (OSH)* [33], which holds the promise of creating unique opportunities for academia and entrepreneurship. Hence, when we teach the design and optimization of a component like an application-specific accelerator, we emphasize the importance of *design for reusability* by explaining how to obtain many alternative accelerator implementations—each offering a unique cost/performance trade-off point—from a given specification. A richer set of implementations maximizes the chances that the result of the design effort is reused across different projects in the class, today, and across different products in the market, tomorrow.

	CSEE 4868 System-on-Chip Platforms
Times	Tuesday and Thursday: 11:40am - 12:55pm
Credits	3
Prerequisites	COMS 3157 and CSEE 3827
Description	Design and programming of System-on-Chip (SoC) platforms. Topics include: overview of technology and economic trends, methodologies and supporting CAD tools for system-level design, models of computation, the SystemC language, transaction-level modeling, software simulation and virtual platforms, hardware-software partitioning, high-level synthesis, system programming and device drivers, on-chip communication, memory organization, power management and optimization, integration of programmable processor cores and specialized accelerators. Case studies of modern SoC platforms for various classes of applications.

Table 1: Bulletin description for the Fall 2018 semester.

A fundamental idea that inspires SoC Platforms is that engineers should excel in the evaluation of the competing objectives of the design, under multiple constraints. Our approach to giving a sound foundation to this "Art of the Compromise" in the context of SoC design is based on the concept of *Pareto-optimal frontier*. This concept provides a formal metric to guide the exploration of the design space with the goal of balancing the traditional objectives and constraints of hardware design with the goal of maximizing reusability [17]. Since it can be applied both to individual components and to the complete system, this metric takes a central role in the semester-end project of SoC Platforms.

3 THE STRUCTURE OF THE COURSE

SoC Platforms has been designed as a course for senior undergraduate students majoring in one among Computer Engineering, Computer Science, and Electrical Engineering. It is also taken by students who enrolled in the MS programs in one of these disciplines as well as by first-year PhD students. Table 1 reports the bulletin description of the course for the Fall 2018 edition.

At Columbia, each course is assigned a four-digit number, where the first digit signifies the level of the course; e.g. advanced undergraduate courses are numbered in the 3000s and 4000s. The prerequisites for SoC Platforms, whose number is CSEE 4868, are two courses that Columbia undergraduates typically take during their junior year:

- *COMS 3157 Advanced Programming*, which is a Computer Science course that covers thoroughly the C programming language and Unix systems programming, together with C++ fundamentals and the basics of TCP/IP networking.
- *CSEE 3827 Fundamentals of Computer Systems*, which is a Computer Engineering course on digital logic design and computer organization, covering the material that is presented in such textbooks as those written by Hennessy and Patterson [53] or Harris and Harris [34].

All students majoring in Computer Engineering or Computer Science are required to take these two courses, while those majoring in Electrical Engineering must take CSEE 3827.

During the first week of classes, the students who have registered for SoC Platforms are invited to complete two self-assessment assignments, which are meant to provide them with the means to discover possible gaps in their knowledge of the prerequisite

Month	Principles Track	Practice Track
Sep	• Technology and economic trends • Methodologies and CAD tools for system-level design and verification • Models of computation • The synchronous model of computation	• Basics of SystemC, modules, methods, and signals • Threads and clocked threads • Channels, ports and interfaces • Transaction-Level Modeling (TLM)
Oct	• Latency-Insensitive Design (LID) • Petri nets and marked graphs • Performance analysis • Power-constraint design	• SoC modeling, proprietary interfaces • Authoring SystemC for high-level synthesis (HLS) • HLS with fixed- vs. floating- point
Nov	• SoC architecture, IP blocks • Processors, accelerators and memory • HW/SW Interface and device driver programming • Virtual platforms	• IP-block integration, FPGA prototyping • Design-space exploration with HLS • Loop transformations • Accelerator memory optimization
Dec	• On-chip communication • Memory hierarchy and cache coherence	• Focus on project

Table 2: Schedule of the two tracks for the Fall 2018 semester.

material. One assignment consists in a set of selected exercises on digital logic and computer organization. The other assignment is a simple exercise on C programming that requires each student to set-up a personal account on the class servers and on the `git` repository, which is based on the Gogs self-hosted `git` service. The class servers host the software environment and all the SLD tools that are needed to do the homework assignments and the project. The repository is used to collect the work of each student throughout the semester, while encouraging the class to follow the best practice of version control during source code development.

In Fall 2016, the course requirements included five homework assignments (accounting for 30% of the final grade), a midterm exam (15%), a project (20%) and a final exam (35%). The project offers substantial opportunities for extra-credit work.

As of now, there is no single textbook that covers adequately the topics of the course. The book "SystemC: From the Ground Up" [12] is not required, but is recommended as a companion for the lectures on SystemC. Lecture notes are accompanied by various reading assignments, which may include conference and journal papers as well as tutorial materials on using SystemC to develop hardware specification for HLS, on programming device drivers, and for learning the commercial CAD tools used in the course.

The course is structured along two main tracks, which run in parallel throughout the semester: the *Principles Track* and the *Practice Track*. Table 2 shows the organization of the topics across the two tracks during the Fall 2018 semester. The next two sections describe in more detail the topics for each track.

4 THE PRINCIPLES TRACK

In the Principles Track, the lectures cover the foundations of system-level design, models of computation (MoC), latency-insensitive design (LID), virtual prototyping, design-space exploration (DSE), HW/SW co-design, and SoC architectures.

After providing an overview of technology and economic trends, the first part of this track covers the principles of modeling, analyzing, and optimizing computing systems, and their components, with different MoCs. We first present the synchronous MoC as the basis of digital hardware design. Then we discuss in detail its crisis in the design of integrated circuits due to the "Wire Problem" and the transformation of chips into distributed systems, which exacerbate the challenges of timing closure [20]. LID is proposed as an attempt to resolve this crisis with a compromise: preserve the synchronous hypothesis while relaxing the time constraints during the early phases of the design process when correct measures of the delay paths among the modules are not yet available [16]. We explain the theory of LID [19] formally as an application of the Tagged-Signal Model framework to reason about MoCs [42]. We explain the practice of LID and its Protocols and Shell Paradigm, with a detailed discussion of the circuits for shell interfaces and relay stations as well as the protocols on which they are based [18]. Petri nets [49] are first introduced as a MoC to reason about concurrency and then applied to the issue of performance analysis in LID [23]. We explain how the task of DSE can be guided by the concept of Pareto's optimality both at the component level and the system level. We also show how the impact of the optimization of a component on the performance of the overall system can be quantified with the help of Amdahl's Law.

During the second part of the semester, the lectures cover the main components of SoC platforms which are illustrated with case studies from state-of-the-art industrial products and research projects from academia. In particular, the processor core is discussed through the presentation of the ARM and RISC-V instruction set architectures. Data-level parallelism and GPUs are discussed through the example of the NVIDIA Xavier SoC. Bus-based communication is explained with the example of the AXI protocol [8]. Cache coherence is first explained in the context of bus-based implementations. The concept of an accelerator is introduced as a consequence of the end of Dennard's scaling, discussed with respect to the issue of coupling with processors [26], and illustrated with various examples, including the open-source NVDLA [51]. The integration of accelerators into an SoC is explained from both the hardware and the software viewpoint. In hardware, we explain why LID is widely used for the communication among SoC components [41] and how it is a natural fit for interfacing circuits synthesized with HLS [44, 66]. In software, we dedicate a lecture to device driver programming [25] and another to explaining how to handle the shared memory address space for accelerators [46] and how to support different cache-coherence models [32].

5 THE PRACTICE TRACK

The lectures and the homework assignments of the Practice Track are focused on SLD with SystemC, hardware accelerator design with HLS tools, HLS-driven DSE, and accelerator HW/SW integration with device driver programming.

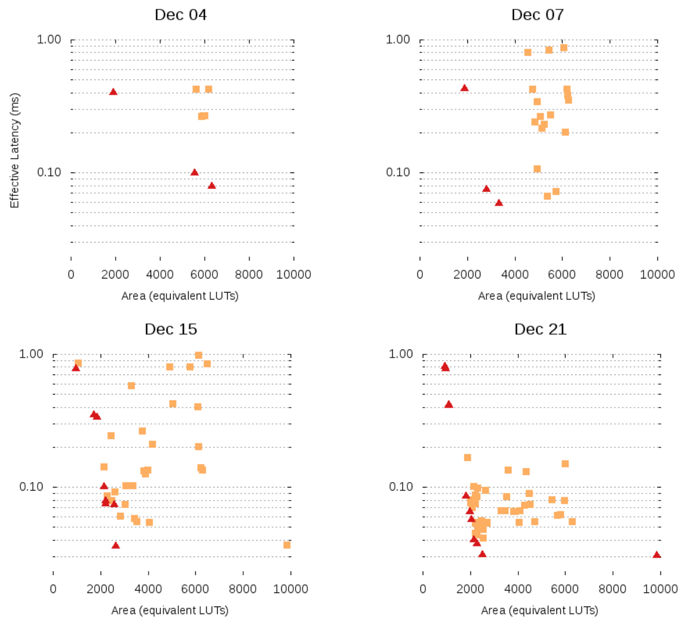
The beginning of the course aims to help students build familiarity with SystemC, which is an IEEE-standard programming language built as a supportive library of C++ [37, 52]. SystemC enables hardware design by supporting (i) *hardware data types*, such as bit vectors, logic vectors, arbitrary-precision integers, fixed-point values as well as floating-point values, (ii) *hardware concepts*, e.g. modules, processes, ports, signals, and channels, and (iii) *concurrency*, by supporting a simulation environment where three types of processes (SC_THREAD, SC_CTHREAD and SC_METHOD) can be concurrently executed. These are the SystemC counterpart of the concepts of "process" and "always" statements in VHDL and Verilog, respectively. Building and combining the knowledge gained in the prerequisite courses (Section 3), students learn to design hardware at a higher level of abstraction. With SystemC, in fact, they can combine the concepts they have learnt for software programming (COMS 3157) with those learnt for digital design (CSEE 3827). They practice these concepts through a series of homework assignments with increasing levels of difficulty, finally reaching the expertise required to work on the semester-end project (Section 6).

SystemC comes with additional libraries such as the one for Transaction-Level Modeling (TLM), which decouples communication and computation [10, 31]. In addition, CAD tool vendors provide libraries of TLM primitives that offer abstracted functions to specify the communication and synchronization mechanisms among computation processes at the system level as well as synthesizable implementations of these mechanisms, which can be combined with the implementation of SystemC processes in a modular fashion [30, 61]. These libraries follow the Protocols and Shells Paradigm in using point-to-point channels, which are inherently latency insensitive, combined with interfaces, which can be instanced to connect processes to channels. By applying TLM and these primitives, in the Practice Track students learn the practical aspects of LID, whose theory they have studied in the Principles Track.

SystemC can be used to design different types of hardware components, including general-purpose processors, memories, peripherals, and specialized accelerators. The students of SoC Platforms focus particularly on *loosely-coupled accelerators*, which execute coarse-grain tasks, are designed independently from processor cores, and reside outside these cores [26]. By following the *Embedded Scalable Platform (ESP)* approach [17], these accelerators have *modular socket interfaces* that allow them to be integrated into an communication infrastructure, which can be implemented as a bus or a network-on-chip. The interface decouples the accelerator design from its integration into the SoC, thereby enabling late-stage modifications/optimizations as part of DSE decisions and promoting its reuse across different systems. In traditional hardware design courses, the optimization and integration of a hardware block like an accelerator is taught at the register-transfer level. Instead, the students of SoC Platforms start by analyzing a software application that is provided in a language like C/C++ (the so-called "programmer's view"), identify the computational kernels that are good targets for hardware acceleration, and create a preliminary model of the accelerator in SystemC that has the same behavior of the programmer's view. This model is not yet synthesizable, but it can be verified by pairing it with a *test bench*, which must be also written in SystemC at the same level of abstraction. This enables fast simulation and thus fast convergence to behavioral correctness. Furthermore, the test bench can be reused later in the design flow to validate the RTL implementation generated with HLS through co-simulation. The students learn the importance of developing a test bench that abstracts well the basic environment in which the accelerator could operate and are typically encouraged to complete such development before starting the design of the accelerator. The SystemC code of the accelerator is later revised through a sequence of refinement steps until it meets the requirements imposed by the "SystemC Synthesizable Subset" that is supported by the given HLS tool. For example, this refinement requires the removal of dynamic memory allocation, pointer arithmetic, and other operations that are not usually supported by HLS. Over the years, in SoC Platforms we have been using two HLS tools from Cadence: C-to-Silicon and Stratus. However, based on our experience and conversations with alumni, we believe that the principles and practice learnt by the students prepare them also for the use of other commercial HLS tools such as Mentor Catapult, Xilinx Vivado HLS, and Bluespec as well as academic tools such as Bambu, Dwarv, and LegUp. Nane *et al.* completed a survey and evaluation of many of these HLS tools with particular emphasis on FPGA implementations [50].

After the students have practiced the basics of running the HLS tool on a synthesizable model of the accelerator, we teach them how to perform a DSE by exploiting the *configuration knobs* provided by the tool [43, 55]. The HLS knobs allow hardware designers to explore many alternative RTL implementations. The most important knobs operate on the loops and function calls that are present in the SystemC specification. For example, the application of the "loop unrolling" knob leads to RTL implementations that are more parallelized, thereby delivering higher performance in exchange for large costs in terms of area and power. Another example is the "loop breaking" knob, which allows the sharing of hardware resources by executing operations sequentially. This has the effect of decreasing the area and power cost in exchange for lower performance. In addition to the HLS knobs, students need to explore other DSE aspects, such as I/O bandwidth, the design of the accelerator private local memory (PLM) in the context of the overall on-chip memory organization [56], and the trade-offs between the granularity at which data is transferred and processed by an accelerator and the size of its PLM [54]. These are other fundamental aspects that permit to increase the diversity of the set of RTL implementations that can be obtained from a single high-level specification. All these HLS-synthesized implementations are not strictly equivalent from an RTL viewpoint (because they do not produce exactly, i.e. clock by clock, the same sequence of output signals for any valid sequence of input signals) but they are members of a latency-equivalent design class [16, 19]. Hence, thanks to the common socket interface, they can be seamlessly replaced with each other in the architecture of the SoC, depending on performance and cost targets.

After the lectures on DSE, the students are taught the basics of writing a device driver that manages the execution of the accelerator and exposes it to the software applications. They also learn



Number	Description
21	Number of student teams
661	Number of improved designs across all teams
31.5	Average number of improved designs per team
1.5	Average number of improved designs committed each day per team
99	Total number of changes of the Pareto curve over the project period
11	Final number of Pareto-optimal designs
26×	Performance range of Pareto curve
10×	Area range of Pareto curve

Figure 1: Progression of the DSE for the Fall 2015 project. The x axis reports the cost metric in terms of area occupation measured as the number of equivalent FPGA look-up tables (LUT), while the y axis reports the effective latency (in ms) that each accelerator takes to execute the Gradient algorithm. The Pareto-optimal designs are denoted with triangles. The final plot contains a Pareto frontier with eleven designs that span a range of about 26× in performance and 10× in area cost, thus providing a rich set of alternative implementations for this accelerator.

how to program a software application that interfaces with the device driver to request the execution of the accelerator through a system call. This application is programmed as a refinement of the test bench used during the design of the accelerator. The device driver, which runs on top of Linux, is responsible for reserving the accelerator, setting its configuration registers, starting its execution and then suspending itself until resumed by the arrival of an interrupt that the accelerator sends upon the completion of its task. Thanks to the common socket interface used for the design of the accelerator, only a small part of a generic device driver needs to be modified for the given specific accelerator (less than 3% of the code). Students can then combine the application, the device driver, and the accelerator to test and evaluate their work on a virtual platform or our FPGA infrastructure [45].

In summary, the Practice Track gives students the skills to complete the entire SLD flow. Starting from a high-level specification, they develop a hardware accelerator that can be synthesized with HLS, integrated into the system through a hardware/software interface, and invoked through a system call from a software application running on a processor. Hence, they become proficient with both the hardware and software aspects of heterogeneous computing.

6 THE SEMESTER-END PROJECT

During the second part of the semester, the focus of the students' homework activity shifts to the completion of a project in team effort. The project is centered on the design, optimization and integration of one hardware accelerator for a given application. It typically requires combining the skills that the students acquired with the homework assignments during the first part of the semester, including designing synthesizable hardware specifications with SystemC, using HLS to explore the design space of an accelerator, and programming a device driver to integrate an accelerator into an SoC. By this time, the series of homework assignments completed by the students have allowed them to practice these skills as well as to gain familiarity with the tools and underlying software infrastructure.

The Project as a Design Contest. The project is structured as a design contest among student teams. Most teams consist of two students, while there is also the option of working individually. For instance, in Fall 2015 twenty-one teams competed in designing a hardware accelerator for the Gradient algorithm, a computer-vision kernel from the Perfect Benchmark Suite [11]. Each team specified the hardware design in SystemC, programmed a device driver to integrate it with the application running on an embedded processor, and validated the hardware/software co-design using an in-house virtual platform. The teams explored the design space with the commercial HLS tool Cadence C-to-Silicon, targeting a Xilinx FPGA platform.

For each team, the goal is to obtain three distinct implementations of the given accelerator by the semester-end deadline. The three implementations must correspond to three different trade-off points in terms of two main (competing) objective functions, which, in the case of the Fall 2015 project, were the area occupation of the synthesized hardware of the accelerator and the effective latency (in ms) that the accelerator takes to execute the Gradient algorithm. Ideally, each implementation should not be Pareto-dominated by one of the other two implementations in the context of the design

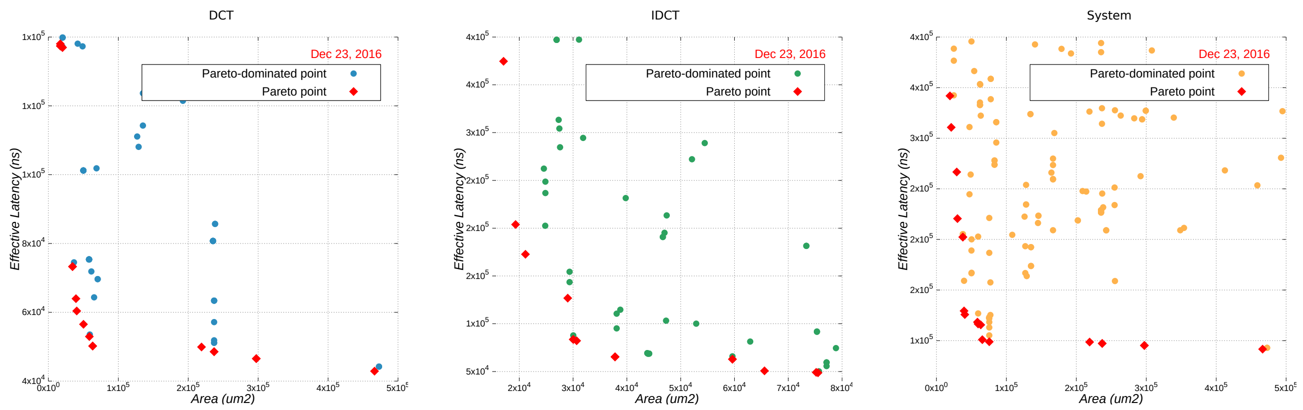


Figure 2: The Fall 2016 Project: Introducing the concept of IP-block reuse.

space delimited by the two objective functions. For example, the points denoted with triangles in the plot labeled "Dec 04" of Fig. 1 correspond to three Pareto-optimal implementations, while the four points denoted with squares correspond to Pareto-dominated implementations. The rationale of this requirement is to encourage the students to apply SLD methods, and particularly the configuration knobs provided by the HLS tool, to optimize the accelerator design for reusability. Instead of focusing on optimizing just one particular implementation of the accelerator, the students should aim at returning three distinct implementations that distinguish a (possibly segmented) line whose projections on the two axes of the design space are as long as possible. In this way, an SoC architect who may want to use an implementation designed by these students may choose between alternative solutions that offer clearly different trade-off choices in terms of performance versus cost.

The quality of each final implementation is evaluated in the context of the work done by the entire class: at the class level, Pareto-optimal implementations receive the highest score, while the penalty for Pareto-dominated implementations is proportional to the distance from the Pareto frontier. This part of the grading is meant to drive the design-optimization efforts by sparking some competition among the student teams.

Balancing Competition and Self-Improvement. A mechanism of incentives encourages the students to keep improving their work by committing new versions into the design repository: specifically, each team receives extra credits for each day when it commits at least one new implementation point that represents a clear improvement in at least one of the two directions of the objective function (measured by some minimum quantum) compared to the prior work of the same team. This part of the grading is meant to reward the self-improvement efforts of each team, independently from the efforts of the other teams.

Throughout the one-month duration of the project, a live *Pareto-efficiency plot* reporting the current position of the three best accelerator implementations of each team in the bi-objective design space is made available on the course webpage. This allows students to continuously (and privately) assess their performance with respect to the rest of the class. Privacy is guaranteed because each given design point is identifiable through a label that is known only to the instructors and the corresponding team members. The plot is automatically generated by a script running on the class servers based on the designs that the students have committed on the `git` repository. The script checks that the submitted design is functionally correct before synthesizing and evaluating it.

Fig. 1 reports the snapshots of the status of the plot taken at the end of the day for five distinct days during the Fall 2015 semester as well as the final statistics for the class. On average, over the one-month period of the Fall 2015 project, each team committed over 30 design improvements. The Pareto frontier changed every day and a total of 99 times across the project duration. The final plot contains a Pareto frontier with eleven designs that span a range of about 26× in performance and 10× in area cost, thus providing a rich set of alternative implementations for this accelerator. These final results can be seen as the outcome of the exploration of the design space performed collectively by the class *as if* it was a single design team: with this perspective, it can be argued that the combination of the contest and self-improvement aspects of the project made possible to harvest the aggregate expertise of all students to achieve superior results within the given time constraints.

Balancing Competition and Collaboration. As the course enrollment grows, the project can be scaled up to include the design of multiple accelerators for a target SoC, while balancing competition and collaboration aspects. This can be done, for instance, by assigning the design of each accelerator to a subset of the student teams and offering incentives for collaboration across teams working on different accelerators. Based on this idea, in Fall 2016 we partitioned the student teams in two subsets and asked them to compete in the design of two accelerators for two distinct algorithms, respectively. The algorithms were the DISCRETE COSINE TRANSFORM (DCT) and the INVERSE DISCRETE COSINE TRANSFORM (IDCT). In addition, every team across the two subsets competed in the realization of a system that *uses* the accelerator the team designed (e.g. the DCT) and *reuses* one of the accelerators designed by some other teams (e.g. an IDCT). The grading mechanism is similar to the one for the Fall 2015 project, with the implicit addition that teams are rewarded with credits also for delivering a component that becomes part of one or more Pareto-optimal system implementations. This allows the students to experience immediately the benefits of designing components that are highly reusable across SoCs implementations. The designs of the two components are "decoupled" from an RTL-design viewpoint because they are connected through LID interfaces. Fig. 2 reports the project results at

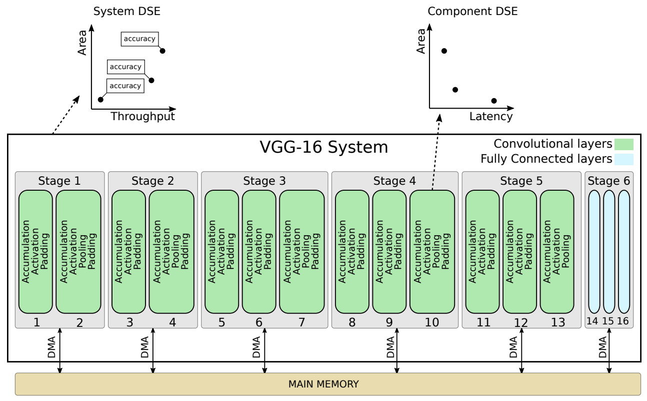


Figure 3: The Fall 2017 Project: Competitive and collaborative system-level DSE of a CNN accelerator.

the end of the semester, i.e. the outcomes of the DSE for the two stand-alone components and for the system featuring both.

Scaling up the Project Complexity through an SLD Ecosystem. As the course infrastructure continues to progress each year, we are aiming at scaling up the complexity of the system project without increasing the workload for the student teams. In addition to providing students with the opportunity to design and evaluate directly a component in the context of a larger and more interesting system, scaling the project scope follows one of the recommendations emerged from the CCC Workshop Series on Extreme Scale Design Automation, i.e.: "As custom design is being displaced by more automated design styles, university courses should highlight abstractions (e.g., system-level design) needed to manage designs that far exceed those built at universities" [9]. In Fall 2017, we organized the project as a competitive and collaborative system-level DSE of an accelerator for the VGG16, a very deep convolutional neural network (CNN) for large-scale image recognition [64]. Fig. 3 shows a high-level block diagram of this CNN, which consists of 6 stages: 5 stages are bundles of convolutional layers while the last stage is a group of fully connected layers. The layers are decoupled from an RTL-design viewpoint because they are connected through LID interfaces. Overall, the system can process data in a streaming fashion because it can work as a pipeline at the granularity of a layer: for example, while Layer 2 is working on the inference of a dog image, Layer 1 could be working on the image of a cat and Layer 3 on one of a monkey.

Each subset of student teams was asked to compete on the reusable design of a given stage of this CNN. As shown in Fig. 3, the result of this part of the contest is a *component-level* DSE for each stage, where the design space is defined by two objective functions: area occupation and effective latency. A grading mechanism that combined fixed deadlines and extra-credit incentives encouraged the students to make available their implementations as frequently as possible. In addition, every team across the team subsets was asked to realize and deliver three combinations of its stage designs with the designs for the other stages, which were made available by the other teams so that that they could be "licensed" in order to assemble a complete implementation of the overall CNN (i.e. the system). As shown in Fig. 3, the result of this part of the contest is a *system-level* DSE for the whole CNN, where the design space is defined by three objective functions: area occupation, data-processing throughput, and accuracy of image recognition. In this way, a team working on one stage is competing with the other teams of the same stage in order to provide the best solutions for the teams working on the other stages, thereby enabling collaborative engineering across stages. The rationale is to create, in the class settings, a small-scale copy of an SLD ecosystem based on IP-block reuse [17].

7 RELATED WORK

Most of the existing courses on SoC design are based on the RTL abstraction. Examples include *"Advanced VLSI Design" (ELEC 522)* [2] taught by J. Cavallaro and *"Introduction to VLSI System Design" (ECE 425)* [1] taught by C. Coats and V. Kindratenko. These courses have some lectures on HLS, but overall SLD methods play a marginal role in them. Other courses on hardware design focus on specific application domains. For example, *"Hardware/Software Co-Optimization for Machine Learning" (CSE599S)* [5], taught by L. Ceze and T. Moreau, is a course on designing systems optimized for machine learning by using GPUs and FPGAs. *"Reconfigurable Logic - Technology, Architecture and Applications" (ECE 18-643)* [7], taught by J. Hoe, covers the fundamental concepts of hardware design at RTL, with an emphasis on FPGA prototyping and the use of domain-specific tools and languages like Spiral [58].

Some courses are closer in concept to SoC Platforms, as they focus more on SLD methods. In *"Complex Digital Systems" (6.375)* [6], Arvind teaches digital design with BlueSpec [28], which can be seen as an alternative approach to HLS. For the project, students are grouped in teams and each team designs a different digital system. *"High-Level Digital Design Automation" (ECE5775)* [4], taught by Z. Zhang, involves the design of hardware accelerators with HLS. The course teaches also the principles of CAD, HW/SW co-design, and SoC design. Students design their own accelerator and compete to obtain extra credits. *"System-on-a-Chip Architecture" (ESE532)* [3], taught by A. DeHon, covers the design, programming and optimization of SoC architectures. It includes a project where students start from a given application, which needs to be accelerated, and develop an SoC targeting an FPGA platform that combines soft-core processors and accelerators. *"System-on-Chip Design" (EE382M.20)* [29], taught by A. Gerstlauer, targets the design of highly-integrated SoCs. Similarly to our course, the main project requires the development of an accelerator starting from a high-level specification (C++) and targeting an FPGA platform. It involves also HW/SW co-design and DSE. With respect to all these courses, a distinguished difference of "SoC Platforms" is the major emphasis that we put on combining aspects of team competition, design reuse, and collaborative engineering in the course project.

8 CONCLUSIONS

We presented the System-on-Chip Platform course that we have developed at Columbia over the past eight years to teach students the design and programming of heterogeneous components with system-level design methods. While we put particular emphasis on system-on-chip architectures for high-performance embedded applications, we believe that the course provides a broad foundation on the principles and practices of heterogeneous computing.

Acknowledgments. The authors thank present and past members of the SLD Group at Columbia. They gratefully acknowledge the support that the SLD Group has received over the years from DARPA, the NSF, Cadence, Qualcomm, and Xilinx.

REFERENCES

[1] ECE 425. 2019. Introduction to VLSI System Design. https://courses.engr.illinois.edu/ece425/sp2019/lectures.html.
[2] ELEC 522. 2018. Advanced VLSI Design. https://www.clear.rice.edu/elec522/.
[3] ESE 532. 2018. System-on-a-Chip Architecture. https://www.seas.upenn.edu/~ese532/.
[4] ECE 5775. 2018. High-Level Digital Design Automation. https://www.csl.cornell.edu/courses/ece5775/.
[5] CSE 599s. 2018. Hardware/Software Co-Optimization for Machine Learning. https://courses.cs.washington.edu/courses/cse599s/18sp/.
[6] MIT 6.375. 2016. Complex Digital Systems. http://csg.csail.mit.edu/6.375/.
[7] ECE 643. 2017. Reconfigurable Logic - Technology, Architecture and Applications. http://users.ece.cmu.edu/~jhoe/course/ece643/.
[8] ARM. 2018. AMBA AXI and ACE Protocol Specification. http://infocenter.arm.com/help/topic/com.arm.doc.ihi0022d/.
[9] I. Bahar et al. 2013. "Scaling" the Impact of EDA Education — Preliminary Findings from the CCC Workshop Series on Extreme Scale Design Automation. In *Intl. Conf. on Microelectronic Systems Education (MSE)*. 64–67.
[10] B. Bailey et al. 2010. *TLM-driven Design and Verification Methodology*. Lulu Enterprises.
[11] K. Barker et al. 2013. *PERFECT (Power Efficiency Revolution For Embedded Computing Technologies) Benchmark Suite Manual*. PNNL and GTRI. http://hpc.pnnl.gov/perfect.
[12] David C. Black, Jack Donovan, Bill Bunton, and Anna Keist. 2009. *SystemC: From the Ground Up, Second Edition*. Springer.
[13] Bloomberg News. 2018. Facebook To Design Own Chips, Cut Back On Intel, Qualcomm Reliance. https://www.investors.com/news/technology/facebook-building-chips/.
[14] M. Bohr. 2007. A 30 Year Retrospective on Dennard's MOSFET Scaling Paper. *IEEE Solid-State Circuits Society Newsletter* 12, 1 (Winter 2007), 11–13.
[15] S. Borkar and A. Chen. 2011. The Future of Microprocessors. *Communication of the ACM* 54 (May 2011), 67–77. Issue 5.
[16] L. P. Carloni. 2015. From Latency-Insensitive Design to Communication-Based System-Level Design. *Proc. of the IEEE* 103, 11 (Nov. 2015), 2133–2151.
[17] L. P. Carloni. 2016. The case for Embedded Scalable Platforms. In *Proc. of the Design Automation Conf. (DAC)*. 17:1–17:6.
[18] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli. 1999. A Methodology for "Correct-by-Construction" Latency Insensitive Design. In *Proc. of the Intl. Conf. on Computer-Aided Design (ICCAD)*. 309–315.
[19] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. 2001. Theory of Latency-Insensitive Design. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 20, 9 (Sept. 2001), 1059–1076.
[20] L. P. Carloni and A. L. Sangiovanni-Vincentelli. 2003. On-Chip Communication Design: Roadblocks and Avenues. In *Intl. Conf. on Hardware/Software Codesign and System Synthesis*. 75–76.
[21] A. M. Caulfield et al. 2016. A Cloud-Scale Acceleration Architecture. In *Proc. of the Intl. Symp. on Microarchitecture*. 1–13.
[22] R. K. Cavin, P. Lugli, and V. V. Zhirnov. 2012. Science and Engineering Beyond Moore's Law. *Proc. IEEE* 100, Special Centennial Issue (May 2012), 1720–1749.
[23] R. Collins and L. P. Carloni. 2008. Topology-Based Performance Analysis and Optimization of Latency-Insensitive Systems. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 27, 12 (Dec. 2008), 2277–2290.
[24] R. Colwell. 2013. End of Moore's law. *IEEE Computer* 46, 12 (Dec. 2013), 49.
[25] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. 2005. *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc.
[26] E. Cota et al. 2015. An Analysis of Accelerator Coupling in Heterogeneous Architectures. In *Proc. of the Design Automation Conf. (DAC)*. 202:1–202:6.
[27] S. Damaraju et al. 2012. A 22nm IA Multi-CPU and GPU System-on-Chip. In *ISSCC Digest of Technical Papers*. 56–57.
[28] N. Dave, Arvind, and M. Pellauer. 2007. Scheduling as Rule Composition. In *Intl. Conf. on Formal Methods and Models for Codesign (MEMOCODE)*. 51–60.
[29] EE382M.20. 2018. System-on-Chip Design (EE382M.20). http://users.ece.utexas.edu/~gerstl/ee382m_f18/.
[30] Michael Fingeroff. 2010. *High-Level Synthesis Blue Book*. Mentor Graphics Corp.
[31] F. Ghenassia. 2006. *Transaction-Level Modeling with SystemC*. Springer.
[32] D. Giri, P. Mantovani, and L. P. Carloni. 2018. Accelerators and Coherence: An SoC Perspective. *IEEE Micro* 38, 6 (Nov-Dec 2018), 36–45.
[33] G. Gupta et al. 2017. Kickstarting Semiconductor Innovation with Open Source Hardware. *IEEE Computer* 50, 6 (June 2017), 50–59.
[34] David Harris and Sarah Harris. 2012. *Digital Design and Computer Architecture* (2nd ed.). Morgan Kaufmann Publishers Inc.
[35] John L. Hennessy and David A. Patterson. 2019. A New Golden Age for Computer Architecture. *Commun. ACM* 62, 2 (Jan. 2019), 48–60.
[36] M. Horowitz. 2014. Computing's Energy Problem (and What We Can Do About It). In *ISSCC Digest of Technical Papers*. 10–14.
[37] IEEE. 2012. SystemC Standardization Working Group. 1666-2011 - IEEE Standard for Standard SystemC Reference Manual.
[38] E. Jhonsa. 2018. Why Tech Giants Like Amazon Are Designing Their Own Chips – And Who Benefits. https://www.thestreet.com/opinion/why-tech-giants-are-designing-their-own-chips-14807638.
[39] N. P. Jouppy et al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proc. of the Intl. Conf. on Computer Architecture (ISCA)*. 1–12.
[40] D. Keymeulen et al. 2018. High Performance Space Computing with System-on-Chip Instrument Avionics for Space-based Next Generation Imaging Spectrometers (NGIS). In *NASA/ESA Conf. on Adaptive Hardware and Systems*. 33–36.
[41] B. Khailany et al. 2018. A Modular Digital VLSI Flow for High-Productivity SoC Design. In *Proc. of the Design Automation Conf. (DAC)*. 72:1–72:6.
[42] E. A. Lee and A. Sangiovanni-Vincentelli. 1998. A Framework for Comparing Models of Computation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 17, 12 (Dec. 1998), 1217–1229.
[43] H.-Y. Liu, M. Petracca, and L. P. Carloni. 2012. Compositional System-Level Design Exploration with Planning of High-Level Synthesis. In *Conf. on Design, Automation and Test in Europe*. 641–646.
[44] D. Lockhart, G. Zibrat, and C. Batten. 2014. PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research. In *Proc. of the Intl. Symp. on Microarchitecture*. 280–292.
[45] P. Mantovani et al. 2016. An FPGA-Based Infrastructure for Fine-Grained DVFS Analysis in High-Performance Embedded Systems. In *Proc. of the Design Automation Conf. (DAC)*. 157:1–157:6.
[46] P. Mantovani et al. 2016. Handling Large Data Sets for High-Performance Embedded Applications in Heterogeneous Systems-on-Chip. In *Intl. Conf. on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. 1–10.
[47] G. Martin and G. Smith. 2009. High-Level Synthesis: Past, Present, and Future. *IEEE Design & Test of Computers* 26, 4 (Aug. 2009), 18–25.
[48] C. Metz. 2018. Amazon's Homegrown Chips Threaten Silicon Valley Giant Intel. https://www.nytimes.com/2018/12/10/technology/amazon-server-chip-intel.html.
[49] T. Murata. 1989. Petri Nets: Properties, Analysis and Applications. *Proc. of the IEEE* 77, 4 (April 1989), 541–580.
[50] R. Nane et al. 2016. A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 35, 10 (Oct. 2016), 1591–1604.
[51] NVIDIA. 2018. NVDLA Primer. http://nvdla.org/primer.html.
[52] Open SystemC Initiative (OSCI). [n.d.]. The SystemC Language Reference Manual. http://www.systemc.org/.
[53] David A. Patterson and John L. Hennessy. 2013. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc.
[54] L. Piccolboni et al. 2017. Broadening the Exploration of the Accelerator Design Space in Embedded Scalable Platforms. In *IEEE High Performance Extreme Computing Conference (HPEC)*. 1–7.
[55] L. Piccolboni et al. 2017. COSMOS: Coordination of High-Level Synthesis and Memory Optimization for Hardware Accelerators. *ACM Trans. on Embedded Computing Systems* 16, 5s (Sept. 2017), 150:1–150:22.
[56] C. Pilato et al. 2017. System-Level Optimization of Accelerator Local Memory for Heterogeneous Systems-on-Chip. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 36, 3 (March 2017), 435–448.
[57] Y. Pu et al. 2018. A 9-mm2 Ultra-Low-Power Highly Integrated 28-nm CMOS SoC for Internet of Things. *IEEE J. of Solid-State Circuits* 53, 3 (March 2018), 936–948.
[58] M. Püschel et al. 2004. Spiral: A Generator for Platform-Adapted Libraries of Signal Processing Algorithms. *Int. J. High Perform. Comput. Appl.* 18, 1 (Feb. 2004), 21–45.
[59] F. Samie et al. 2016. IoT Technologies for Embedded Computing: A Survey. In *Intl. Conf. on Hardware/Software Codesign and System Synthesis*. 8:1–8:10.
[60] A. L. Sangiovanni-Vincentelli. 2007. Quo Vadis SLD: Reasoning about Trends and Challenges of System-Level Design. *Proc. of the IEEE* 95, 3 (March 2007), 467–506.
[61] J. Sanguinetti, M. Meredith, and S. Dart. 2012. Transaction-Accurate Interface Scheduling in High-Level Synthesis. In *ESLsyn Conference*. 31–36.
[62] S. Schreiner et al. 2014. Autonomous Flight Control Meets Custom Payload Processing: A Mixed-Critical Avionics Architecture Approach for Civilian UAVs. In *Intl. Symp. on Object/Component/Service-Oriented Real-Time Distributed Computing*. 348–357.
[63] Y. S. Shao et al. 2015. The Aladdin Approach to Accelerator Design and Modeling. *IEEE Micro* 35, 3 (May-Jun 2015), 58–70.
[64] K. Simonyan and A. Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Intl. Conf. on Learning Representations*. 730–734.
[65] G. P. Stein et al. 2005. A Computer Vision System on a Chip: a Case Study from the Automotive Domain. In *Conf. on Computer Vision and Pattern Recognition (CVPR'05)*. 130–130.
[66] R. Zhao et al. 2016. Improving High-level Synthesis with Decoupled Data Structure Optimization. In *Proc. of the Design Automation Conf. (DAC)*. 137:1–137:6.