Specialization for Energy Efficiency using Agile Development

Borivoje Nikolić, Jonathan Bachrach, Elad Alon, Krste Asanović, and David Patterson
Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, USA
bora@eecs.berkeley.edu

Designing hardware specialized for a target application domain dramatically improves energy efficiency. However, the design of specialized hardware is hampered by high cost, dominated by the effort needed to design, validate, and verify the custom integrated circuit. An agile design approach, discussed here, relies on hardware generators coupled with rapid design of iterative prototypes and open source to reduce the cost of hardware specialization.

A limited set of 13 computational motifs have been found to provide the basis for most of the relevant algorithms in important applications [1]. Examples of these motifs are dense and sparse linear algebra, transformations into spectral domain, or graph traversals, a subset of which will be represented in a particular application domain. Implementing specialized hardware for efficient computation of the representative motifs in a target application domain, along with the effective methods for mapping the software onto the specialized hardware [2], is the key to achieving efficiency while maintaining programmability.

However, the practicality of mapping specialized functions to custom silicon is limited by designers' productivity and the overall cost of the design. The hardware design process needs to change to provide a path to achieving energy-efficient, cost-effective, high-performance designs for future applications. The complexity of modern software systems far exceeds the complexity of the hardware on which they run. Software productivity has dramatically improved with adoption of the agile software development process [3], which relies on building a sequence of working but incomplete prototypes, the use of higher levels of abstraction, and modularization done by small design teams. Hardware design can become more efficient by adopting the principles of software development, as outlined in this presentation.

Some of the key principles that enable productive customized hardware design are: (1) small integrated teams building a series of prototypes, (2) use of higher-level descriptions to accelerate design, aid verification and enhance reuse, (3) develop generators not instances for better reuse and scalability of the design, (4) rapid design flows to enable agile design validation, and (5) leverage open-source designs for common hardware infrastructure.

Small, integrated teams building prototypes. Rather than extensive up-front design and working in distinct phases by using different engineers for architecture exploration, RTL design, and verification, Agile relies of a sequence of working but incomplete prototypes built by a small, group of engineers that all do the design, verification ("built the thing right?"), and validation ("built the right thing?").

Higher-level description. Chisel is a hardware construction language aimed at designing hardware by using parameterized generators [5]. Chisel is based on the Scala programming language, and is built by extending Scala, and supports good software engineering techniques via object-oriented and functional programming. Chisel code is compact, due to its higher level of description than traditional hardware description languages. Hardware development in Chisel enables the agile development practices.

Designing generators, not instances. The key to efficient design and reuse is in using generators, that is, parameterized descriptions of a set of design choices. The use of generators enables design-space exploration through sweeping the parameters of the design. For this to work, language and tool support are needed that enable designers to naturally implement generators, rather than particular instances of a function [4-6]. Fortunately, languages for constructing generators, such as Chisel, Genesis2, and Spiral are emerging to aid the designers.

Rapid design flows. Agile hardware development relies on a rapid design flow that maps a description of hardware into layout. Modern synthesis and place-and-route tools are robust enough that they can be scripted to produce a clean layout with a push-button flow. Such flows enable design exploration from a high-level description and by changing generator parameters. Even if each mapping of the design into silicon is suboptimal in terms of performance and energy efficiency, often the overall design is highly optimized through automated search of architectural tradeoffs. Producing complete prototypes rapidly enables iterative agile validation with the intended user to ensure the design actually meets user needs.

Open-source hardware. A large portion of the cost of a modern chip is in re-implementing and re-verifying the myriad interfaces necessary for seamless integration into complex products, yet these standard features add little differentiated value to the part, resulting in proposals to push these onto separate standardized chips in a multi-chip module [7]. Many interfaces are simple, and fully commoditized, such as SPI and I2C, but there are also more complex interfaces such as USB, WiFi, or Bluetooth. Development of standard open-source hardware implementations of these and other common blocks presents a path to align design effort with the added value of a chip project. The open-source model has revolutionized software development in the past ~15 years, and opening the source for hardware designs provides a similar opportunity.

Processors are ubiquitous in complex chip designs, and multiple processor cores are often embedded in a typical modern SoC. The vast majority of embedded processors are based on proprietary instruction-set architectures (ISAs), but much broader innovation is enabled with a free ISA supported with a full open-source software stack. While there are a few free ISAs in use, the RISC-V eco-system is provocative, as it closely follows the principles of open-source development [8]. RISC-V is a modern ISA, with a small number of instructions, designed for extensibility. It supports the IEEE-754-2008 floating-point standard, compact instruction encoding, and 32-, 64- and 128-bit address spaces. A complete software tool stack is available including GCC, Linux, LLVM, and a verification suite. Chisel generators (e.g. a "Rocket Chip generator") to build RISC-V cores are freely available, and a number of other open-source implementations of RISC-V cores are available online.

An example design that relies on many of the principles outlined in this paper is the RISC-V microprocessor with integrated switched-capacitor DC-DC converters [9] based on the Rocket Chip generator. The use of generators throughout makes the design scalable and its components reusable. The 64-bit RISC-V processor was built in a 28nm, fully-depleted, silicon-on-insulator (FD-SOI) process with ultra-thin body and buried oxide (UTBB). The DC-DC converter is supplied with two external voltages, a 1.0V core and 1.8V I/O, and generates four dynamically reconfigurable average output voltages: 1.0V, 0.9V, 0.67V, and 0.5V. An adaptive clock generator adjusts the clock period of each cycle based on the instantaneous converter output voltage. All custom mixed-signal blocks—DC-DC converters, clock generator, and custom low-voltage memories—have been characterized for use in an automated flow. The complete system has been synthesized and automatically placed and routed using the automated flow, where the run-time from an RTL change to the LVS-clean layout was less than 14 hours. The resulting design runs Linux and software applications and achieves a peak energy efficiency of 27GFLOPS/W including the DC-DC convertor.

Acknowledgment:

This work was in part supported by DARPA PERFECT Award Number HR0011-12-2-0016 and Intel. The chip fabrication was donated by STMicroelectronics. The authors acknowledge BWRC, and ASPIRE students, faculty and staff, and their respective industrial member companies.

References:

[1] K. Asanović, et al, "The landscape of parallel computing research: A view from Berkeley", Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, December 2006.

[2] Selective Embedded Just-In-Time Specialization: http;//sejits.eecs.berkeley.edu

[3] A. Fox, D. Patterson, *Engineering Software as a Service: An Agile Approach Using Cloud Computing*, Strawberry Canyon, 2014.

[4] M. Puschel, *et al*, "SPIRAL: Code generation for DSP transforms," *Proc. IEEE*, vol.93, no.2, pp.232-275, Feb. 2005.

[5] J. Bachrach, *et al*, "Chisel: Constructing hardware in a Scala embedded language," *49th ACM/EDAC/IEEE Design Automation Conference (DAC), June 2012*, pp.1212-1221.

[6] O. Shacham, *et al*, "Rethinking digital design: Why design must change," *IEEE Micro,* vol.30, no.6, pp.9-24, Nov. 2010.

[7] S. Sutardja, "The future of IC design innovation," *2015 IEEE International Solid- State Circuits Conference, Digest of Technical Papers,* Feb. 2015.

[8] RISC-V website: http://www.riscv.org

[9] B. Zimmer, *et al*, "A RISC-V vector processor with tightly-integrated switched-capacitor DC-DC converters in 28nm FDSOI," *2015 Symposium on VLSI Circuits*, June 15-19, 2015, pp. C316-C317.