Platform Tuning for Embedded Systems Design

Frank Vahid and Tony Givargis, University of California, Riverside

Chip capacity continues to double roughly every 18 months, as Moore's law predicted. Today's chips hold millions of transistors, replacing chips from the 1980s that held thousands of transistors. System-on-chip technology can now integrate entire digital systems that used to occupy one or more boards—systems consisting of numerous components like microprocessors, memories, co-processors, and peripherals—onto a single chip.

SOCs have enabled the proliferation of powerful embedded computing systems such as cell phones and digital cameras. Component vendors must now market not only integrated circuits (ICs), but also intellectual property (IP), also known as cores. A core is an electronic-file representation of a processing-level component, like a microprocessor or peripheral. A designer integrates numerous cores to build a system, then fabricates a single SOC IC.

Finding cores with the appropriate functionality, quality level, and support can be challenging. Licensing cores often involves complex legal arrangements, while interfacing cores often poses difficult timing problems. Simulating a multicore system requires extensive modeling of its complex environment, and the resulting environment often is still incomplete.

Even after developers find, license, and interface the cores and model the system environment, extremely slow simulation still poses a problem: A day's simulation work may cover only milliseconds of real time. Thus, designers find many errors and omissions only after obtaining a first IC. These problems typically cause multiple time-consuming and costly IC fabrication iterations.

UCR's Dalton Project shows that platform tuning can increase performance and reduce power consumption for system-on-chip embedded platforms.

SOC PLATFORMS

Vendors increasingly sell *SOC platforms* to help designers overcome these problems. An SOC platform is a predesigned, multicore system in which the cores have already been acquired, licensed, and integrated.

A platform typically targets a class of applications, like television set-top boxes, digital cameras, or network switches. For example, Figure 1 shows a sample platform that targets digital cameras, including a microprocessor, instruction and data caches, and peripherals like analog-to-digital and digital-to-analog converters, universal asynchronous receivers and transmitters, pixel processors, direct-memory-access controllers, and various communication buses.

Platforms can also possess large regions of field programmable logic. Vendors often accompany their platforms with sophisticated software development environments supporting compilation, debugging, simulation, and emulation.

Platform types

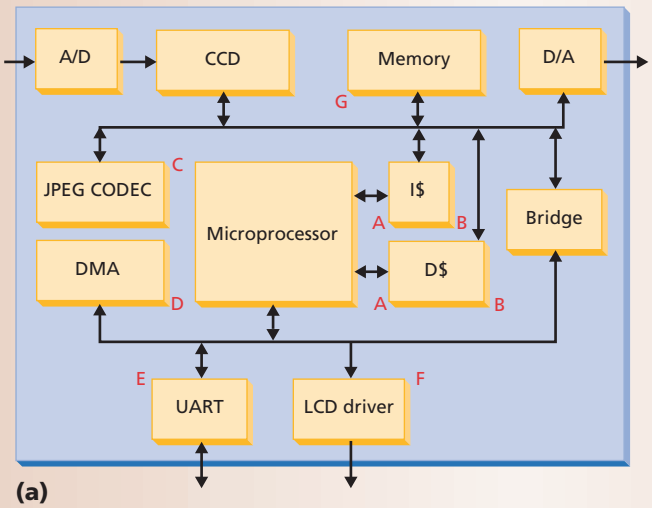
A platform can take the form of an IP, IC, or both. An *IP platform* comes in electronic-file form, such as a hardware description language. The designer using an IP platform must model the environment and then simulate and fabricate an IC, but can optimize the IP before fabrication. An *IC platform* comes as a packaged chip. The designer programs the platform and executes it directly in a real environment, skipping the environment modeling and simulation tasks. IC platforms come in two formats:

- *Product-oriented*. Small and inexpensive, these platforms target a small class of applications.
- *Prototype-oriented*. Too big and costly for incorporation into a final product, these platforms are useful in a variety of applications and support extensive debug features.

Prototype-oriented IC platform vendors often provide an accompanying IP platform, which mirrors the IC platform, for eventual optimized IC fabrication.

Platform tuning

The top-down, describe-and-synthesize methodology that developers commonly use to design single-purpose processing cores requires describing the initial functionality up front as an algorithm, then synthesizing it to a logic-gates implementation. In contrast, platforms require a highly iterative, spiral-like methodology because developers cannot completely describe the entire complex systems up front. Thus, in the increasingly common configure-and-execute methodology, a



Parameter	Description
A	Cache capacity (1-256K byte)
B	Cache line size (32, 64, 128 byte)
C	CODEC resolution (12-16 bit)
D	Data block size (32, 64 byte)
E	UART buffer size (2-16 byte)
F	LCD duty cycle (60, 100 Hz)
G	Memory size (1-16M bytes)

(b)

Figure 1. Sample SOC platform for digital cameras: (a) Block diagram with shaded letters showing sample parameters and (b) table describing architecture parameters. In SOC platforms, the cores have already been acquired, licensed, and integrated.

designer configures the existing platform by mapping programmed functionality onto it, and then executes the platform. The designer repeats these steps tens or hundreds of times, refining existing functionality and adding new functionality throughout the procedure.

Configuration goes beyond mapping functionality to a platform. A second configuration task is *tuning*. To increase sales volume and amortize design costs, vendors parameterize IC and IP platforms to accommodate a wider range of applications. For example, the platform in Figure 1 can have the following parameters: cache capacity and line size, pixel processing resolution, direct-memory-access block-transfer size, LCD duty cycle, and on-chip memory size. Tuning adjusts the platform's parameters and the application's functionality to one another. Whereas mapping gets the platform to operate correctly, tuning gets it to operate efficiently.

We define two types of tuning:

- *Application-constraint* tuning adjusts for external constraints on power, performance, size, and so forth. For example, the designer can reduce a platform's voltage for an application that requires low power or can increase the voltage for a different application that requires high performance.
- *Application-profile* tuning adjusts for the particular application's run pattern. For example, if a particular application doesn't need a cache bigger than 8 Kbytes, then the designer can reduce an IP platform's cache size or shut off some of an IC platform's cache regions.

Designers can perform tuning before or after IC fabrication. For IP platforms, setting parameter values before fabrication optimizes the IC. For IC platforms, tuning is done by setting parameter values in configuration registers.

THE DALTON PROJECT

UCR's Dalton Project focuses on IP-related design issues. One project goal is to develop methods that support tuning parameterized IP and IC platforms.

Using a simple version of the platform shown in Figure 1, we sought to develop a method to quickly but accurately evaluate the power consumed by an application running on a particular platform parameter configuration. Gate-level hardware-description-language simulation can accurately evaluate power consumption, but it is too slow for SOCs, which may require tens of hours per configuration. Register-transfer-level simulation is faster, but still too slow. Therefore, we developed a system-level simulation approach. We used C++ (or Java) to model each core as an object, then we modeled the complete system as communicating objects. The C++ system-level simulation runs orders of magnitude faster than the gate-level simulation, requiring seconds rather than hours.

We create each core's object so that whether the core is a microprocessor, memory, or peripheral, it executes a fixed number of instructions, which differ depending on the device's function:

- For microprocessors, the instruction set defines the instructions.
- For memories, an instruction is a memory access.
- For peripherals, an instruction is a high-level action that the peripheral performs, like sending serial data or compressing a video frame.

For each instruction, we perform low-level power analysis of a gate-level implementation of the core, using a variety of parameter configurations and instruction data values. We back-annotate each instruction of a core's C++ object with this power information. Thus, when executing the complete C++ model for a particular

configuration, the designer obtains accurate power and performance information.

Our simulations ran 10,000 to 100,000 times faster than gate-level simulations, with only a 5 percent error. We have also investigated further speedups by generating instruction traces for each core once, then using C++-based trace simulators during exploration rather than full-function simulators for each core.

Pareto-optimal points provide a uniquely best configuration in terms of power and performance.

Given this fast power evaluation approach, we investigated methods for efficient exploration of the configuration space. For the platform in Figure 1, the configuration space is greater than 10^{25} because of the platform's parameters. To examine every configuration, even assuming just one second per configuration, would take longer than the universe has existed. However, we are really only interested in Pareto-optimal points, which provide a uniquely best configuration in terms of, in our case, power and performance.

After careful investigation of the parameters and their dependencies, we found that a specific ordering of the configurations could prune away most non-Pareto-optimal points and thus generate all Pareto-optimal points in a reasonable time. In our investigations using such pruning, we obtained all Pareto-optimal points for four different applications mapped to the architecture in Figure 1 in less than one hour.

FUTURE DIRECTIONS

The Dalton Project is currently investigating new directions in platform tuning. Until now, our platform has included fairly standard parameters such as cache size and associativity, peripheral buffer sizes, and so on. We are investigating a class of more aggressive low-power transformations to an IP platform to tune a microprocessor's architecture to its application program. Example transformations include partitioning one microprocessor into two based on the application's particular profile, and moving critical software into custom hardware or field-programmable logic.

Another direction involves developing support environments for tuning. To know where to focus tuning efforts, a designer needs an environment that quickly provides information about how much power each platform part and each application instruction consume. Further, we plan to build suites of tuning transformations and investigate design patterns that describe how to apply those transformations in given situations. It may be possible to automate each transformation's use individually or as part of a tuning heuristic.

A third direction involves self-tuning IC platforms. Developers could extend a parameterized IC platform to include components that analyze an existing application and determine a new parameter configuration—or even apply transformations—that reduce power or improve performance.

Parameterized SOC platforms solve numerous problems that face today's system designers. With platforms ranging from IP to IC variants and tuning tasks ranging from simple parameter configuration to aggressive architectural transformations, platform-tuning issues can be daunting. Our early results show great promise, but we need extensive future research in this area. ✸

Frank Vahid *is an associate professor in the Department of Computer Science and Engineering at the University of California, Riverside. Contact him at vahid@cs.ucr.edu.*

Tony Givargis *is a PhD student in the Department of Computer Science and Engineering at the University of California, Riverside. Contact him at givargis@cs.ucr.edu.*

Editors: Jerzy W. Rozenblit, University of Arizona, ECE 320E, Tuscon, AZ 85721, jr@ece.arizona.edu; and Sanjaya Kumar, Motorola, 1501 W. Shure Dr., Arlington Heights, IL 60004, sanjaya.kumar@motorola.com