Skiing the Embedded Systems Mountain

INGRID VERBAUWHEDE
University of California at Los Angeles, and Katholieke Universiteit Leuven
and
PATRICK SCHAUMONT
University of California at Los Angeles

UCLA teaches students how to master the steep slopes of the embedded systems mountain. The EE201A graduate course connects high-level design specification to embedded implementation. There is a long-standing and wide culture gap between system designers that create those abstract specifications and the system architects that need to implement them. In industry, the culture gap has separated software from hardware teams, and platform creators from platform users. In an embedded context, where these are very tightly connected, this leads to large inefficiencies both in design time and design results. Our course takes students to both sides of the gap and lets them look at this problem from different perspectives. For a given application, it teaches how to select target architectures, tools, and design methods. The course covers a stepwise systematic design process. It includes specification, transformation, and refinement of an application. Specifications enable systematic and structured expression of an application. Transformations rework specifications into ones that are a better match for a given target architecture. Refinements lower the abstraction level toward the target architecture. The embedded systems mountain is traversed in two directions. A vertical refinement axis covers elements such as power-memory-reduction methods or fixed-point refinement. A horizontal exploration axis covers various architecture alternatives including application-specific integrated circuits (ASIC), domain-specific processors, digital signal processors (DSP), embedded cores, programmable processors, and system-on-chip (SOC). During the course, the students also go through an extensive design project to apply the methods learned in this course. A typical embedded application is used to drive the project. In this paper it is illustrated using an embedded version of an image encoder, more specifically a JPEG encoder. Several commercial tools, design environments, and platforms have been used as alternative implementation targets for this application.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: Embedded Systems

General Terms: Design

Additional Key Words and Phrases: Education, cosimulation, design space exploration

This research was supported by NSF (Grant CCR-0310527) and SRC (Grant HJ-1116).
Authors' addresses: Electrical Engineering Department, University of California at Los Angeles, CA 90095-1594 USA; (email: {schaum,ingrid}@ee.ucla.edu), and Electrical Engineering Department, Katholieke Universiteit Leuven, B-3001, Belgium.

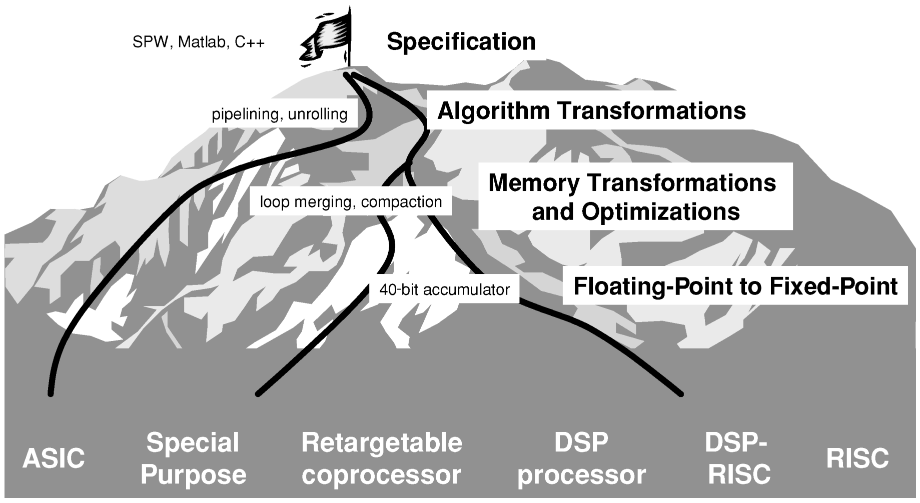

Fig. 1. Embedded system design: from application to alternative implementations.

1. INTRODUCTION

Real-time embedded systems are a challenge to implement. An embedded environment constrains the energy consumption, the memory space, and the execution time of an application. On the other hand, the complexity of the applications running on an embedded platform keeps increasing. Examples include next-generation cell phones, portable game-boys and digital assistants, smartcards, and medical monitoring devices.

There is a huge need for courses to teach students systematic, divide-and-conquer, design methods to map applications on embedded platforms. Traditional courses do not cover this topic. Courses in Electrical Engineering or Computer Science are either focused on algorithm design, such as signal processing, communications or networking, or they are focused on implementation aspects such as Very Large Scale Integration (VLSI), computer architectures, or circuits classes.

The goal of our embedded systems design course is to bridge the culture gap between applications and implementation alternatives. According to the International Technology Roadmap for Semiconductors, design complexity will limit filling the silicon foundries, not process technology limitations [ITRS 2003]. Without design methods and tools, the designer in the middle is lost between the vast algorithm space and the features offered by deep-submicron and nanotechnologies [De Man 2002; Gupta 2003].

1.1 Skiing Down the Slopes

In this graduate course, we assume that an application is given, but that design paths into target architectures need to be explored. This leads to the concept of a mountain, as shown in Figure 1. Students need to go through the design phases to obtain alternative realizations, ranging from Verilog to assembly code. This corresponds to skiing down different slopes of the mountain, starting from

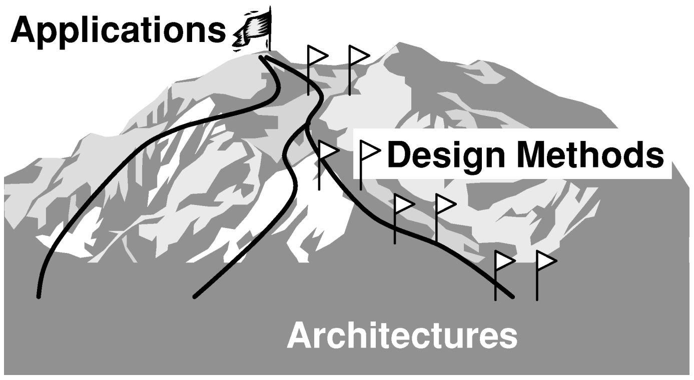

Fig. 2. Design of an embedded application requires close interaction with the architecture target and associated design methods.

the application on the top. It will allow them to compare relative complexity of different slopes, experience what a DSP compiler brings, relative to a hardware synthesis tool, and get a feeling of the overall cost factors in a snowy and seemingly uniform landscape.

The starting point of a single application with multiple implementation alternatives is a key element. In one graduate class with duration of only 10 to 12 weeks, it is not realistic to consider the entire design space of all the algorithm options and application alternatives, together with all the design and implementation options. In our school, the duration of a course is 10 weeks. Each week there are 4 hours of class lectures. Besides the class meetings, students also receive weekly homework with hands-on assignments and experiments. They typically have 1 week to turn in the assignment. Instead of individual homework that are unrelated to each other, we have divided the problem of system design in multiple steps. Each step is the focus of a homework.

This approach will prepare students well for future design jobs in industry. Indeed, in many companies, the algorithm and system designer considers a job done when Matlab code is operational and a high-level objective, such as bit-error-rate (BER), is achieved. The system architect has to pick up this specification and obtain a suitable realization, corresponding to one of the downhill slopes on the mountain. In the past, this was the job for the so-called "hardware" or "VLSI" designer. These days, however, this task has become one of platform selection, hardware/software codesign, embedded-software development, besides the traditional hardware development.

A suitable realization is one that combines or trades-off different design objectives such as performance, flexibility, energy consumption, and area, expressed in memory space or silicon gates.

The goal of the course is to give the students insight in this application development process, explore implementation alternatives, and link these with design methods and tools. The interaction between application development, architectural platforms, and design methods is illustrated in Figure 2. All three components are covered in the course. It is the key in meeting the high-level application objectives with a concrete implementation. Indeed, the application or application domain will influence and often determine the architecture platform. For example, a programmable DSP processor has specialized instructions and datapath units to efficiently implement the cellular phone baseband standards. Similarly, successful design methods and tools have been developed for particular platforms and/or for particular application domains.

Many design-tool vendors offer us the opportunity to use their tools in our course and project. This is a win–win situation, as they get people trained in new design methods and the students access the latest in electronic system design automation. Unique to a class environment is that we can put these design methods and tools in a larger context. We have the luxury to start with unbiased students and to divide the class in teams. Different teams use different environments, platforms, and/or tools. At the end of the course, the different alternatives can be compared, based on the design goals of performance, area and programmability, including design time and learning curve.

This links the theoretical part of the course with the tightly integrated design experiment part of it. Instead of using a single design environment or a single architecture platform, we divide the class in teams of 2 or 3 students. Each team starts from the same initial specification, but the target platform is different, as is some part of the design flow. For instance, for the JPEG image encoder, four different implementation platforms were targeted using multiple design flows [Sakiyama 2003a, 2003b].

1.2 Prerequisites for the Course

The course is typically taken by fresh graduate students with an undergraduate education in Electrical Engineering, Computer Engineering, or Computer Science. The students have a basic knowledge of computer architectures, circuit design, hardware description languages, and some programming experience (C, Java).

For the project, we try to make heterogeneous teams, e.g., one student with a Computer Science background will team with an Electrical Engineering student. This also mimics the reality in industry where students have to learn to work in teams of engineers with different background. We typically take a task or component of an embedded system that requires acceleration, a third-party module, or some form of hardware/software codesign. We have taken applications from signal processing and networking, as will be illustrated in Section 3.

In general, we do not require that the students know a lot about the application. We noticed, however, that those students who excel in the project often are also those who have gained a thorough insight into the application.

1.3 Overview of the Paper

In the next section, we explain the organization of the course curriculum. We explain and work out the three essential parts of the course: specification, refinement, and implementation. In Section 3, we discuss a typical sample course projects: an embedded JPEG image encoder. We conclude the paper in Section 4.

Table I. Comparative Feature List in GEZEL

Topic	Examples for Theory	Reading Examples	Assignment Examples
Specification and models of computation	Data flow vs. control flow	Lee 1987 Harel 1987	Create a golden test-bench
Transformation and refinement	Data flow transformations, HW/SW codesign	Panda 2001	Reduce memory accesses with Atomium
Implementation platforms	ASIC, FPGA, DSP, SOC.	Verbauwhede 2000	Implement on an embedded core with a HW accelerator

2. CURRICULUM

The curriculum is organized around three main topics: specification, refinement, and implementation. The first part discusses the importance of specifications and associated models of computation. In a second part, the importance of systematic transformations and gradual refinement is covered. This corresponds to different tracks down the hill. In the third part, the different implementation platforms are covered. This corresponds to the horizontal exploration at the bottom of the mountain.

2.1 Practical Organization

The lectures are evenly distributed over the three main topics. Typically, in a first lecture, some basic introductory material to the topic is covered. Then, in the follow-up lecture, more advanced research papers on the topic are studied. The advanced topics are not only presented by the instructor. In addition, papers are also distributed to the students and the students initiate the discussion by preparing slides on the topic for a follow-up lecture.

Occasionally we have invited speakers. Sample invited speakers are Prof. Gajski to speak about Spec-C [Gajski et al. 2000], Dr. Killian to speak about the Tensilica instruction-set extension approach and associated design tools, Mr. Ueda to talk about the design of the Matsushita i-Mode cell phone IC's, and Mr. Johnson to talk about the Handel-C design environment. For academics, such a presentation offers a forum for their research. For professionals, this presentation offers a forum for their tools and potential future users or employees. For the students, this brings them closer to reality and provides a glimpse of what industry offers.

2.2 Link between Theory and Practice

Next to the lectures there are practical assignments, organized as a design project. A new application was chosen every time the course is offered. During the quarter, an initial specification is transformed and implemented on different platforms. The different assignments correspond to different steps in the design process. Some examples are discussed in detail in Section 3.

A key concept of the curriculum is to tighten theory to practical assignments, as shown in Table I. The three rows correspond to the three phases in the course. The three columns associate theory with assigned reading and with assignments in the course project.

By making this link between theory and practice, we want to teach the students how to link fundamental concepts to their practical application. In the fast-moving field of embedded systems, the useful span of knowledge is not more than a few years. Consequently, embedded systems professionals are in a continuous process of retraining. From all the innovations that come out in the form of tools, methods, and architectures, they will have to learn how to select those that can solve their design problems.

The concept of a design flow as a curriculum also promotes a continuous evolution of the course material over subsequent offerings of the course. Recent evolutions in the field are introduced to the students by covering these topics in the advanced lectures or by assigning them as reading material. Therefore, the material described in the next section is not fixed.

In addition, the concept of course layering with specification, refinement, and implementation is applicable to different design domains as well, for example graphics, networking or security.

2.3 Student Evaluation

An active participation of the students is required in class. The grading is based on three main criteria: the theory, the project results, and the reporting. The theoretical part is covered during the final exam. The project results are evaluated after each design step. For instance, Figure 9 (see later) shows the reduction in memory accesses obtained by each team. The teams with the most memory access reduction (to the left of the histogram) will receive the highest grade. Originality, even if it did not necessarily result in a positive outcome, is rewarded. Reporting, documentation, and presentations are a third component of the grade. Communication skills are very important for future engineers who need to learn to work in a team setting. Each design step needs a report. Presentations are done during the class on different advanced research topics. At the end of the quarter, the teams also present their design results.

2.4 Introduction to the Course

One lecture is used to introduce the course. First, the course is situated based on Figures 1 and 2. It is illustrated with typical examples, such as cellular phones, or ambient intelligent systems. Very motivational are the short descriptions of EE-Times, called “under the hood” [Carey 2004]. Systematic design of a system as complex as an embedded system is divided in to three main tasks:

- A systematic top-down refinement from specification to implementation. This is illustrated with the Y-chart concept of Gajski and Kuhn [1983].
- A systematic exploration of alternative implementation techniques. What are the different platforms and architectures available? It includes topics such as hardware/software codesign, systems-on-chip, the role of intellectual property, etc.

• A systematic evaluation of the cost factors. Examples are energy versus flexibility and programmability and performance versus time to market.

The reading associated with this lecture includes Gupta [2003] and DeMan [2002]: both publications describe the culture gap. To illustrate the fact that there exists an implementation world next to general-purpose processing, we use the article by Strauss [2000]. It shows the importance of digital signal processing (including video, image, voice, audio, sensor, and other processed signals), in the overall computing done by embedded system platforms, and, in general, on all platforms. The different examples of "under the hood" clearly illustrate to the students that a typical embedded application is a very heterogeneous architecture with multiple different cores, programmable, and third party or intellectual property (IP) modules.

2.5 Specification and Models of Computation

A few lectures follow on the importance of specifications and executable models. The specifications of an embedded system are very heterogeneous and one model cannot capture it all. Indeed, similar to architecture design, a model that is tuned to an application domain will be more efficient. It has modeling primitives that allow accurate and fast expression of complex design problems in that domain. Often, it is also possible to design very specific supporting design methods and tools. The result is that a real embedded system will use a mixture of specifications, models, and simulation environments.

In a first part, we present a classification in specifications and models. The Tagged Signal Model [Lee and Sangiovanni-Vincentelli 1996] allows us to clearly introduce the concept of time, partial and total order of events, the difference between discrete time and discrete events, etc.

The two most popular models of computation for signal processing systems are then discussed in more detail. These are the data flow model for the data processing part and the control flow model to model the reaction to events, typically used to model the control part of an embedded system.

Explicit modeling of parallelism is important to obtain efficient implementation of many signal-processing applications. Therefore, the synchronous data flow graph model is introduced [Lee and Messerschmitt 1987] for individual or 1D signals. To model the dependencies between multidimensional arrays of signals, we use the stream model of Phideo [Verhaegh et al. 2001] and the single assignment multidimensional signal representations [Verbauwhede et al. 1996].

Simple control flow models can be represented by a finite-state machine (FSM) and more complex ones with an extended finite-state machine (EFSM) or hierarchical concurrent finite-state machine (HCFSM). This concept is introduced nicely in a visual manner through the Statechart model of Harel [1987]. A combination of control flow and data flow is available through the Finite-State Machine with Datapath (FSMD) model.

Although these models have their limitations in expressiveness, they are very instructional to provide students insight in representations that are very different from what they have learned before. Usually they have only learned imperative models, expressed in sequential languages such as C.

Because of the wide adoption of C and C++ as initial specification languages, several system design languages are developed, such as SystemC and SpecC. These languages have the advantage that only a single syntax is needed. One should bear in mind, however, that teaching languages is not a substitute for teaching models. Indeed, a language is just a representation of a model through a set of syntax rules. Often the model represented in the language is entirely different from the model desired for design. For example, developing data flow models in C++ will require a set of classes to support data flow semantics. In a graduate course where more lectures are available, this topic can be furthered elaborated and more advanced topics can be added.

2.6 Transformation and Refinement

In the second part of the course, transformations and gradual refinement are introduced. A large selection of transformations, gradual refinements, and synthesis techniques are available. In this course, we focus on the viewpoint of the designer and focus on transformations that benefit the design objectives. Next, three main categories are listed. The list is not intended to be complete; the topics were chosen because we feel they have a large impact on the final implementation results. Memory transformations are chosen because they are relatively independent of the target platform, yet they have a great influence on the final performance. Two other kinds of transformations (for performance and for fixed-point refinement) are much more closely linked to the target architecture platform. Hence they are situated closer to the bottom of the slope.

Memory Transformations. Many real-time signal processing applications operate on large arrays and streams of data. Examples are video, audio, and games. In reality, the throughput and performance of most of these applications is limited by the data transfers, either to/from memory or between software and hardware acceleration units and the size of the embedded memories. Hence, in this course, we place great emphasis on a detailed study of memory accesses, memory size, and code transformations to reduce those memory accesses and to reduce the memory size. This requires an insight into the dependenciess between production and consumption of arrays, the life-time analysis of multidimensional arrays, followed by code transformations to reduce the amount of data that needs to be stored and that needs to be accessed.

This part is closely linked to the previous one that introduces multidimensional array representations.

Code transformations for memory optimization occur high up on the mountain. As an example, code transformations to improve data locality are unrelated to the final implementation platform. The overall reduction of memory accesses will benefit the final solution regardless of the specific architecture selected.

The basic reading for this part is reference [Panda et al. 2001]. More advanced topics are taken from Catthoor et al. [1998].

Performance Transformations. Related to the previous topic are transformations to improve parallelism, pipelining, resource utilization, etc. Similar code transformations can be used to obtain the performance goals.

In the digital signal processing community, many transformations are known to improve performance. A typical reference is Parhi [1999]. From the general-purpose parallel programming community, a similar set of code transformations is known and many interesting references are available. Many of these transformations can benefit multiple platforms. e.g., solutions with multiple processors, also very large instruction word processors or behavioral level-synthesis tools.

Floating-Point to Fixed-Point Transformations. In a signal-processing-oriented design flow, data-type refinements related to floating-point and fixed-point signal representations are a very important aspect. The basics of fixed-point signal representation, such as sign, wordlength, overflow, and truncation behavior are quite easy to explain. A more complex matter is an optimization strategy to convert floating-point into fixed-point behaviour while meeting preset precision constraints (SQNR, signal-to-quantization noise ratio). The problem here is that floatingpoint to fixed-point transformation is a highly nonlinear optimization, and, on top of that, it also requires an understanding of advanced signal-processing concepts. Two concrete references that use a C++ environment are a simulation-based [Cmar et al. 1999] and a pseudoanalytical [Keding et al. 1998] approach.

This topic is closely related to the final implementation platform. In an application-specific intruction-set processor environment, the designer can fine tune the different wordlengths to the requirements of the application. Programmable digital signal processors include specific instructions and features that can be used by the designers to obtain maximum precision for the predefined wordlengths. Examples are accumulators with guard bits, block floating point, truncation, and saturation logic.

Although the design process looks like a purely top-down process, in reality, the transformations and refinements are also bottom-up driven. Those considerations originate from the availability of third-party IP modules (for hardware) and legacy code (for software).

Test and verification are a very important component in all refinements and transformations. They are not treated as a separated subject, but rather are included in each design step, as testbenches. Formal verification is not yet included. In the course, we emphasize an understanding of design representation and transformations that are at the level of the designer, rather than at the level of the tools.

2.7 Implementation

The third set of lectures focuses on the large variation of architectural platforms available to the designer. We provide two basic concepts to select an architecture platform. One is the concept of the energy-flexibility trade-off. The second one is a representation of flexibility. Flexibility is offered in the forms of programmability, reprogrammability, configuration, reconfiguration, etc. We present a systematic classification of flexibility in what we call the “reconfiguration hierarchy” [Schaumont et al. 2001].

The energy flexibility trade-off is illustrated in Figure 3. General-purpose approaches provide cost effectiveness but lack energy efficiency. Fully dedicated

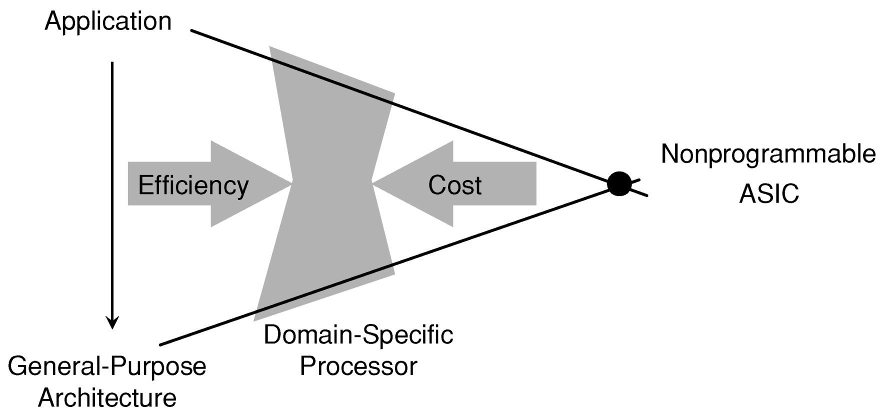

Fig. 3. Energy-flexibility trade-off.

approaches are best for energy efficiency, but are also very expensive to design. Moreover, dedicated architectures lack programmability, as they completely unify application and platform. A embedded systems platform, which typically consists of multiple heterogeneous processing elements, seeks a balance between power and cost, by selectively reducing general-purpose programmability. The reduced programmability is achieved by targeting the architecture specialization to the application. Each component of the multiprocessor system-on-chip (MPSOC) is highly optimized to its application domain. The examples of "under the hood" can be used to confirm this.

The reconfiguration hierarchy is used to determine the amount of flexibility a programmable processor, coprocessor, retargetable instruction-set processor, reconfigurable logic, and so on, offers to the designer.

Between the two extremes of a fixed application-specific integrated circuit (ASIC) and a general-purpose programmable CPU, there is almost a continuum of options available. The two extremes, ASIC and CPU correspond to the two extremes at the bottom of the mountain (shown in Figure 1).

ASICs are still used because the know-how is available in the form of library modules. Examples of such modules include two-dimensional discrete cosine transform, a cryptographic module for advanced encryption standard (AES) operations, and a Viterbi or Turbo acceleration unit for wireless communications. To increase the reuse possibilities, the designers of third-party IP blocks add features, parameters and other options to these blocks to turn them into special-purpose programmable units. For example, the AES unit can be programmed to execute encryption or decryption. Similarly, the Viterbi unit can be programmed for the number of states or samples. Programmable modules have better reuse across different designs instances.

A processor has fundamentally four basic components: the data path or execution units, the control part, the memory part, and the interconnect. This corresponds to the "reconfigurable feature" axis shown in Figure 4. Each of these components can be fixed or can be made programmable. At the same time, the granularity of programming needs to be defined. If a data path is still reprogrammed at the bit-level or at the lookup-table level, it is called

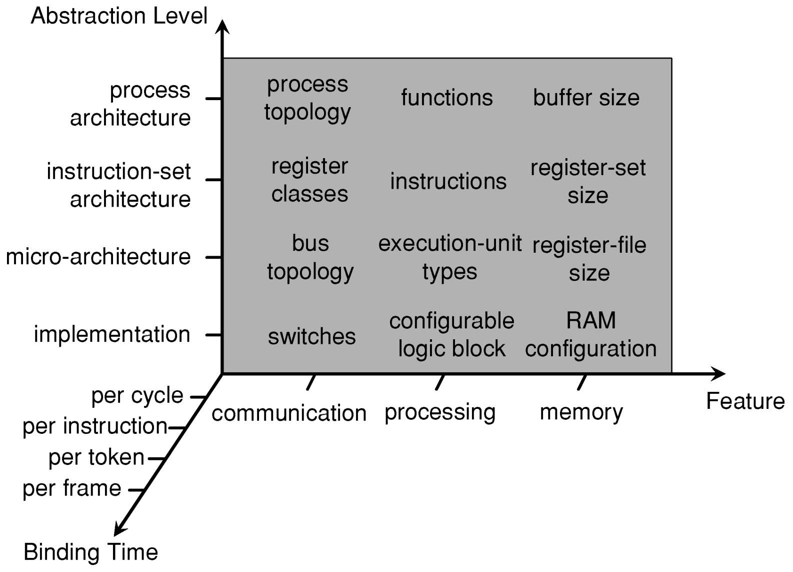

Fig. 4. The reconfiguration hierarchy.

"reconfiguration," typically on a field-programmable gate array (FPGA) or a FPGA block. If it is reprogrammed at the instruction level, it is called instruction-set reconfiguration. This corresponds to the "computation abstraction level axis" in Figure 4.

The third axis corresponds to the "binding time." It relates the relative timing of configuration to the timing of the data processing and it allows making distinction between configuration, reconfiguration, and adaptive reconfiguration.

The reconfiguration hierarchy is illustrated in the course with typical processors and design environments. For instance, DSP processors have all components of the processor architecture adapted to the application domain [Verbauwhede and Nicol 2000]. This not only includes the well-known multiply-accumulate (MAC) datapath, but also the Harvard architecture to improve the data-stream processing, the specialized addressing units, and so on.

General-purpose embedded processors, such as ARM or Leon SPARC, offer different opportunities for optimization. One option is instruction-set extension. Examples are the Tensilica [Rowen 2004] and Lisatek [Hoffmann et al. 2002]. This only changes the datapath and associated instructions. The basic architecture remains a Von Neumann machine. Instead of tightly coupled instruction set extensions, another option is to build loosely coupled coprocessors. The GEZEL environment allows students to experiment with this type of coprocessor [Gezel 2004].

At the end of this lecture series, we also provide time to discuss the important topic of communication between modules and processors. We will discuss topics such as multiprocessor systems-on-chip (MPSOC), and newer developments, such as networks-on-chip.

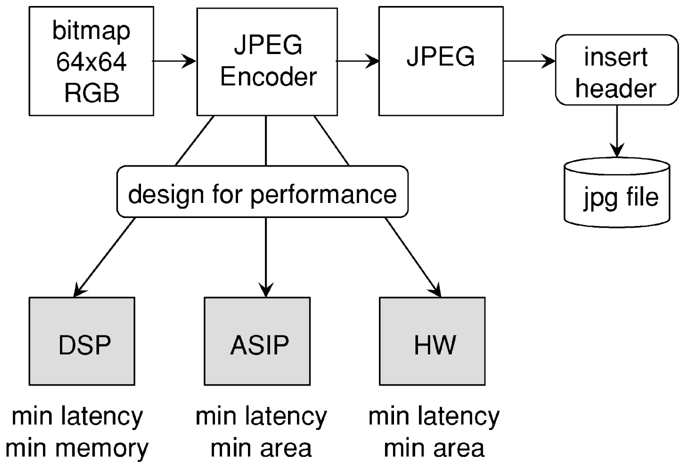

Fig. 5. Project goals for the JPEG encode project.

3. SAMPLE COURSE DESIGN PROJECTS

In this section we present a sample course project and discuss the structure and organization of the labs, along with the results obtained by the students. A JPEG encoder was chosen as a typical embedded application driver. In another class project, an embedded web server was chosen. The course organization outlined in this paper was applied in both projects. Students were grouped in small teams. Using a single-system specification, each of the teams worked toward a different target architecture in a series of design steps, including specification, optimizing transformations, and implementation. The JPEG encoder project is described next to illustrate the design project. The project materials, including assignments and sample descriptions, are available on the web.

3.1 Project Goals and Specification of the JPEG Encoder Project

In the JPEG encoder project, we started with a reference JPEG implementation in C code and targeted three kinds of platforms: digital signal processor (DSP), application-specific instruction-set processor (ASIP), and dedicated hardware. As illustrated in Figure 5, the design was done for maximum performance, with implementation cost (area, memory footprint, etc.) being a second criterion.

There are a considerable amount of variations in JPEG encoding [Pennebaker and Mitchell 1992], and we opted to implement only the simplest encoder, the so-called 422-baseline encoding scheme. The motivation for this choice was given by the amount of time one can invest to get familiar with the application. This should not require more than a single lab project. A 422-baseline JPEG encoder is a block-based encoder that processes blocks of 8-by-8 pixels of image data. The encoding process proceeds through a series of image-transform steps, which range from pure signal processing to control-oriented run-length coding. The output of the encoder is a bitstream, captured within a string of bytes. The JPEG file format (JFIF) is obtained by appending a JPEG header onto the JPEG data.

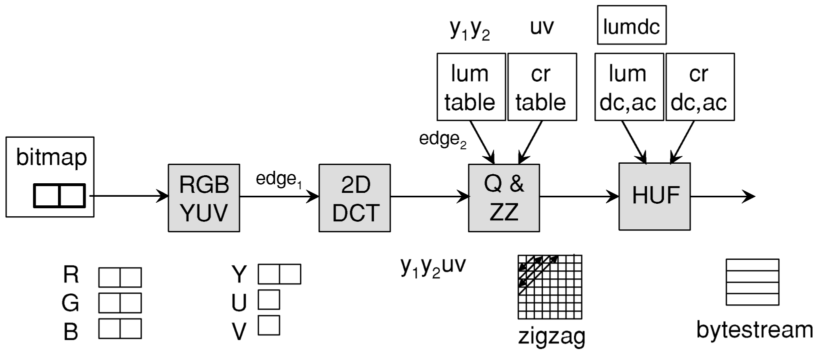

Fig. 6. Data flow in the JPEG encoder.

Although it is out of the scope of this article to discuss the detailed operation of a JPEG encoder, it is still useful to point out the educational value of the different parts of a JPEG encoding chain. Looking at the system dataflow in Figure 6, we identify four different stages in a JPEG encoder. Each of those has different design aspects.

- RGBYUV is a two-phase downsampling filter. It reads in 2 pixels of red–blue–green color data (6 bytes) and produces 2 pixels of luminance/chrominance data. Doing such a design requires students to get an understanding of color spaces and of multirate filter structures. In addition, the filter design is a nice target for fixed-point refinement because of the limited 1-byte resolution on input and output.
- 2D-DCT is a two-dimensional discrete cosine transform (DCT). It processes blocks of 8-by-8 pixels (luminance or chrominance data). There are two aspects in such a design. First, students must understand how a 2D-DCT can be decomposed into a series of 1D-DCTs using a transpose operation on 2D data. Second, they must also find an optimal implementation for the DCT in terms of precision and operations. Such optimal designs are available from literature [Chen et al. 1977], but students must still locate the correct references.
- Q & ZZ is a quantization/zigzag-scan block, which reduces the precision of the DCT-transformed data. Because we used a fixed quantization spec, this block reduces to a computationally simple architecture. The design of the block is interesting because the system dataflow changes radically just after it—from a 2D to a 1D representation using so-called zigzag scan.
- HUF is a Huffman encoder, which implements a run-length encoding algorithm. The encoder has different modes of operation for luminance and chrominance components and, in addition, the zero-frequency component of each block (the pixel at coordinates 0,0 of each 64-pixel block) is coded differentially, while the higher-frequency components are not. A Huffman coder, being a run-length encoding algorithm, exhibits data-dependent control flow. In contrast to previous blocks, where all operations can be scheduled statically (and optimized using static-scheduling principles), the students

Table II. Platforms and Tools for the JPEG Encoder Project

Platform	Language	Development Toolkit	Prototyping Platform
TI C54	C	Code Composer	TI development kit
AD BlackFin	C	Visual DSP++ (2.0)	AD development kit
ART Designer	C	ART Designer (ARM OptimoDE)	Insight FPGA PCB
Celoxica DK1	HandleC	DK1	Insight FPGA PCB

encounter a situation where efficient implementation of control flow becomes crucial.

- SYSTEM. Even when the four functions mentioned above are understood and designed correctly, there is still a system integration task. System-level aspects to address are system-level pipelining, image row-scan to block-scan conversion, and JPEG image file formats.

A broad range of skills and know-how is required in order to complete the JPEG encoder design. In our experience, it is exactly this broad range that makes it a challenging course project. The organizational effort went into structuring the design flow of the JPEG encoder into a series of lab projects, and to stripping out a number of typical time-wasters associated with system design.

First and foremost, there was the issue of a good reference implementation. The standard reference implementation from the Independent JPEG group [IJG 2004] provides an open-source implementation. Being a standard, however, this implementation comprises over 30,000 lines of C code in 72 files and is an obvious overkill for our goals. We ended up developing our own reference JPEG encoder of 868 lines of C++ code in 5 files. This implementation was based on the encoder available in the vic project [McCanne and Jacobson 1995].

A second issue was the availability of good target platforms including development tools. Table II lists the four platforms that we selected for our project. All of the tools and platforms were made available by the vendors free of cost. In return, we offered these companies first insight into our project results and, in some cases, provided detailed feedback. A point of concern was that platforms with combined development of application and architecture (such ASIP and dedicated hardware) require more effort than platforms with a predefined architecture (such as DSP). We, therefore, used a flexible design plan and included fall-back options for the more complicated targets.

3.2 Design Flow of the JPEG Encoder Project

Figure 7 illustrates the steps in the design flow of the JPEG encoder and their mapping to lab projects. There were 26 students enrolled in the reference class. They were asked to form two-person teams, which resulted in 13 design teams for the whole class. We felt that the concept of teamwork is an essential element. Grades were assigned to the teams, however, outside of the project students were still graded individually. The teams were preassigned to a particular target to ensure a balanced outcome. Even though it was a 10-week course, we had only five assignments in the project. For some of the steps more than 1 week was allocated.

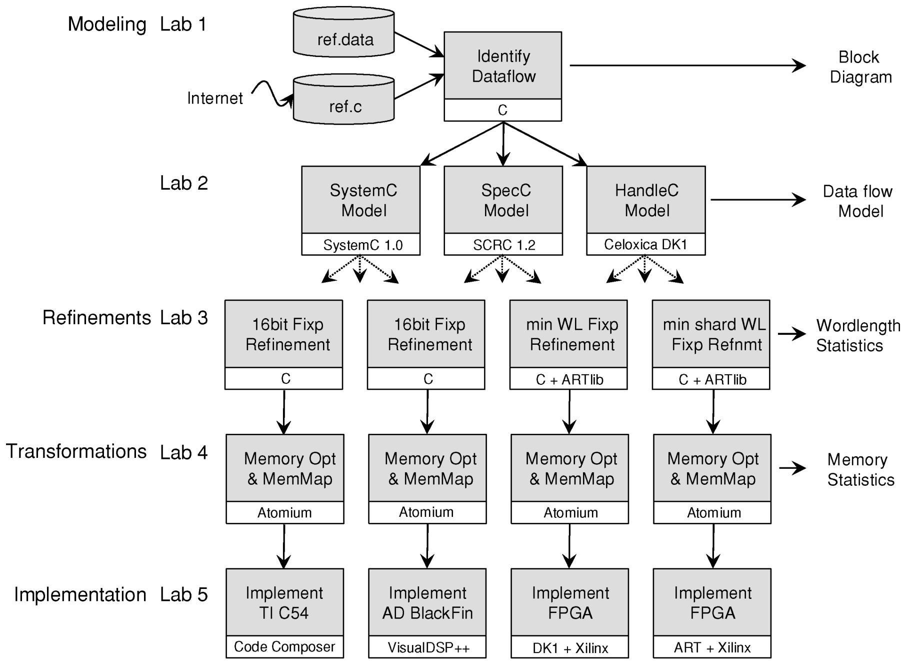

Fig. 7. Design flow and lab projects for the JPEG encoder.

Identify Data Flow. The very first task for the students was to acquire an understanding of the JPEG encoding algorithm. They received a reference implementation of a JPEG encoder and were asked to identify the data flow in the algorithm. The data flow can be expressed in terms of the production/consumption pattern of data for each of the edges in Figure 6. The result of this analysis can be expressed in tabular form as follows:

Edge	Contents	Read	Write
edge1	2 × 64 pixels chrom	1..64 U 1..64 V	1..64 U,V
edge2	64 pixels Q table	1..64	—

For example, in edge 1, chrominance data is provided per 64 pixels and with interleaved U and V components. The data is read out by the DCT block per 64 pixels in a noninterleaved fashion. Edge 2 shows a read-only sequence of quantization coefficients. The design of such a table requires the students to walk through the C code and, using a block diagram of a JPEG encoder, identify the data flow at system level.

System-Level Model. In the second lab of the project, students created an actor-based data flow model out of the sequential C code. The system-level data flow was known from the first lab. In addition, data flow semantics had been covered during the course [Lee and Messerschmitt 1987], in particular, the concept of actors and the associated mechanism of firing rule. Students also

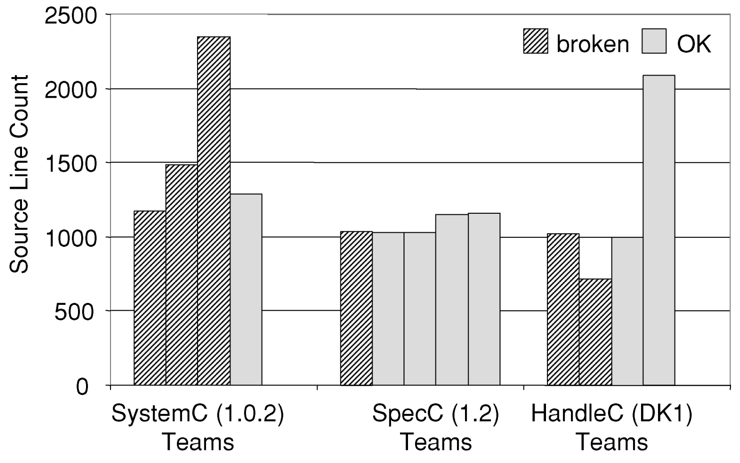

Fig. 8. Data flow model construction in system design languages.

received a data flow model of the encoder in C++ in which the different encoder components are captured as data flow actors with first-in first-out (FIFO) data queues in between them. We instructed the students to use one of the C-based system design languages, including SystemC, SpecC, and HandleC.

To our surprise the outcome of this lab demonstrated a strong correlation between result and the system-level language used. Figure 8 shows the size of the solutions (in lines of code) for each of the teams per system language. Some environments show large variations in program size (and complexity). Many models were also nonoperational, mostly because the students had not figured out how to map data flow semantics into the concurrency model offered by the language. Our conclusion from this experiment was that documentation and methodology is an essential aspect for a system design language. At the time of this course project, only SpecC had elaborate documentation on system-level communication semantics freely available on the web [Gerstlauer 2004]. The graph presented in Figure 8 should be treated with caution because it contains timely information; newer versions of SystemC (TLM in 2.0) and HandleC (DK3) are better documented and have a better orientation towards system-level modeling problems. The goal of the lab project was to investigate the quality and usefulness of these system-level languages, at a time when "the language war" between C-based environments was still in full effect.

Fixed-Point Refinement. In a third assignment, the students performed fixed-point data-type refinement for their platform. The methodology for fixed-point refinement was based on measuring the difference of an image with that of a JPEG-encoded and decoded image. Because a JPEG encoder by itself is a lossy compressor, it also introduces quantization errors as part of the algorithm. Hence, the fixed-point refinement criterion is to minimize additional degradation apart from the JPEG encoding algorithm. Fixed-point refinement is a nonlinear optimization and, in general, we felt that this poses significant difficulties for the students.

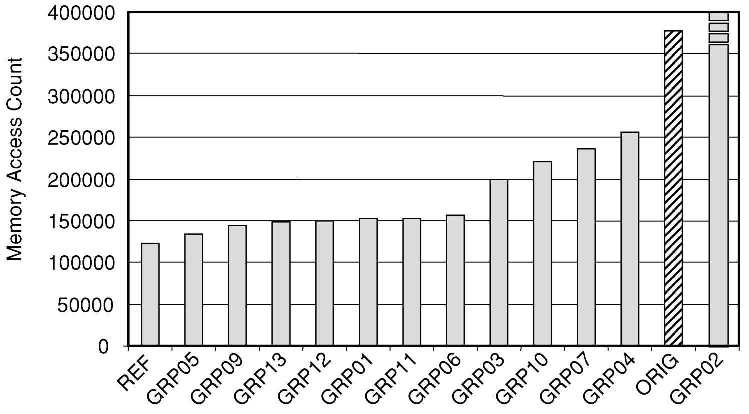

Fig. 9. Memory access count of optimized JPEG data flow model as obtained by different groups.

Fixed-point refinement also poses specific simulation requirements depending on the target platform. For the 16-bit DSP architectures (TI C54 and AD Blackfin), good programming environments are available. Specific features such as multiply-accumulate on increased wordlength can be expressed with pragma's in a C program.

For ASIP and dedicated hardware design, fixed-point simulation requires the use of a fixed-point precision data-type library. At the time of the project, we made use of the C++ library provided by the AR|T simulation environment [ARM OptimoDE 2004], but other options could be considered.

Memory Optimization. A fourth step was to optimize memory consumption for the JPEG encoder on the different target platforms. The goal was to minimize the amount of memory required to run the code on a particular platform. The starting specification was the data flow model of the JPEG encoder that the students had used before. The goal was to rewrite this model, gradually removing FIFO buffers and reintroducing control flow so that the overall storage in the system was minimized. We made use of the Atomium memory analysis tools [Imec 2004] . These tools parse in a C-program, instrument it with memory-access profiling code, and write out an instrumented C-program. The traces that are created as a result of running this program can be analyzed and formatted into tables by postprocessing tools. The results of these optimizations are shown in Figure 9. Almost all teams were able to reduce the amount of memory accesses with a factor of 2 to 3. We were also very pleased with the "point-tool" concept of Atomium. The input and output is plain C code (with a number of acceptable restrictions) and all of the analysis results are easily accessible.

Implementation. The final step of the project was the implementation step. Students had to use all the results obtained previously (data flow, fixed-point behavior, and memory organization) to find the fastest implementation for their platform. The results for the project are illustrated in Table III. All of the teams that targeted a DSP platform could successfully demonstrate their encoder running on a development PCB. In the case of FPGA-based platforms, however, compiling the architecture was quite difficult.

Table III. Implementation Results for the JPEG Encoder Project

Platform	Cycles 64 × 64 Image	Average Code Length (Lines)	Performance (JPEG blocks/sec)
AD BlackFin	1,524K	879	12,602 (@300MHz)
TI C54	1,499K	707	4,270 (@100MHz)
ART Designer	677K	1,015	—
Celoxica DK1	700K	1,312	1,357 (@15MHz)

3.3 Experience from the Trenches

Running a project according to a design flow demands a significant investment of time and preparation. However, we feel that the results in our projects show that such effort is worthwhile. Indeed, there is no other place than an academic curriculum that has the opportunity to simultaneously sample the quality of many different state-of-the-art architectures and tools. From a practical viewpoint there are some aspects that can simplify the organization of such a project.

A complete design flow is, for a large part, a software-development process. Tools are run from a command line prompt and their results are available in files. A student who is not familiar with basic computer skills (file management, command-line operations, data filtering, and so on) in a Unixlike environment is at a significant disadvantage. One might argue that these are practical matters that should not burden the student. However, we feel that students should be able to experience design in the same way as do practicing engineers.

For the same reason, we prefer to use point tools over closed environments. A point tool performs a well-defined task in between well-defined design representation formats (examples are Atomium, a C compiler, or GEZEL). In closed environments, design activities are shielded by a GUI that often also forces a certain manner of operation. We would not want all our design teams to run the same design files in the same design environment. Rather, we want them to use their creativity and outsmart their peers with a better design.

From an organizational viewpoint, software license management quickly becomes a burden. This is especially true in a contemporary academic computing environment. Most of the students have their own laptop, in addition to their own machine at home. They run many different operating system configurations. They prefer to work at odd hours, from arbitrary locations connected to the University through Internet. Because of this, we found node-locked licenses to be virtually useless and floating licenses useful only if they can freely float. Our own experience with license-less open-source software has been very positive because of this. Also, the availability of free (capability-restricted) FPGA design and simulation software by some vendors is a very positive evolution. At least these vendors see that design automation tools do not have the same value for students as mp3 files.

4. CONCLUSIONS

The field of embedded system design and the state of contemporary technology makes it possible to create complete systems with students in a course. The course brings together application, design methods, and implementation alternatives. A good understanding of the interaction between these concepts, will lead to successful designs. In this way, an embedded systems course does not differ from an older VLSI design course.

What has changed is the complexity and the granularity of the problem, for which adapted models of computation, associated design methods, and architecture platforms are presented. As the field of embedded systems evolves, so will these topics. However, the basic concepts will remain and are part of any engineering project.

The project makes it possible to create a realistic simulation of actual industrial practice within the fail-safe context of a University. Besides a thorough understanding of a design problem, such a simulation includes design work, team work, and reporting. It brings a broad range of skills to student teams, and shows them that there is more than one way to measure success.

REFERENCES

ARM OptimoDE. 2004. (http://www.arm.com/products/CPUs/families/OptimoDE.html), OptimoDE Data Engine Family.

Carey, D. 2004. (http://www.eet.com/sys/uth/), Under the hood. *Electronic Engineering Times*, Article series.

Catthoor, F., Wuytack, S., De Greef, E., Balasa, F., Nachtergaele, L., and Vandecappelle, A. 1998. Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design. Kluwer Academic Publ., Boston, MA.

Chen, W., Smith, C., and Fralick, S. 1977. A fast computational algorithm for discrete cosine transform. *IEEE Trans. Commun. Com. 25* (Sept.), 1004–1009.

Cmar, R., Rijnders, L., Schaumont, P., Vernalde, S., and Bolsens, I. 1999. A methodology and design environment for DSP ASIC fixed point refinement. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'99)*. 271–276.

De Man, H. 2002. Nanoscale system design challenges: Business as usual? In *Proceedings European Solid-State Circuits Conference (ESSCIRC'02)*. 3–10.

Gajski, D. and Kuhn, R. H. 1983. Guest editor's introduction: new VLSI tools. *IEEE Computer 16*, 12, 11–14.

Gajski, D., Zhu, J., Domer, R., Gerstlauer, A., and Zhao, S. 2000. SpecC Specification language and design methodology. Kluwer Academic Publishers, Boston, MA.

Gerstlauer, A. 2004. (http://www.ics.uci.edu/~specc) The SpecC System.

Gezel Homepage. 2004. (http://www.ee.ucla.edu/~schaum/gezel).

Gupta, R. 2003. Driving research in system-chip technology. In *IEEE Computer Magazine* (July). 95–97.

Hoffmann, A., Leupers, R., and Meyr, H. 2002. Architecture exploration for embedded processors with LISA. Kluwer Academic Publishers, Boston, MA.

Harel, D. 1987. Statecharts: A visual formalism for complex systems. *Science of Computer Programming 8*, 3 (June), 231–274.

Imec 2004. (http://www.imec.be/design/atomium) Welcome to the Atomium project.

ITRS 2003. (http://public.itrs.net/Files/2003ITRS/Home2003.htm) International Technology Roadmap for Semiconductors.

Keding, H., Willems, M., Coors, M., and Meyr, H. 1998. FRIDGE: a fixed-point design and simulation environment. In *Proceedings of the Design, Automation and Test in Europe*. 429–435.

Lee, E. and Sangiovanni-Vincentelli, A. 1996. The tagged signal model—a preliminary version of a denotational framework for comparing models of computation. *Memorandum UCB/ERL M96/33*, EECS, UC Berkeley, CA.

Lee, E. and Messerschmitt, D. 1987. Synchronous Data Flow. *Proceedings of the IEEE 75*, 9, 1235–1245.

McCanne, S. and Jacobson, V. 1995. vic: A Flexible framework for packet video. *ACM Multimedia* 1995.

PANDA, P. R., CATTHOOR, F., AND DUTT, N. D. 2001. Data and memory optimization techniques for embedded systems. *ACM Transactions on Design Automation of Electronic Systems 6*, 2 (Apr.), 149–206.

PARHI, K. 1999. VLSI Digital Signal Processing Systems: Design and Implementation. Wiley, New York, 1999.

PENNEBAKER, W. AND MITCHELL, J. 1992. JPEG Still Image Data Compression Standard. Kluwer Academic Publishers, Boston, MA.

ROWEN, C. 2004. Engineering the Complex SoC. Prentice-Hall PTR, Upper Saddle River, NJ 07458.

SAKIYAMA, K., SCHAUMONT, P., AND VERBAUWHEDE, I. 2003a. Finding the best system design flow for a high-speed JPEG encoder. *Asia and South Pacific Design Automation Conference* (ASP-DAC'03), (Jan.).

SAKIYAMA, K., SCHAUMONT, P., HWANG, D., AND VERBAUWHEDE, I. 2003b. Teaching trade-offs in system-level design methodologies. In *Proceedings IEEE International Conference on Microelectronics Systems Education (MSE'03)*. 62–63.

SCHAUMONT, P., VERBAUWHEDE, I., KEUTZER, K., AND SARRAFZADEH, M. 2001. A quick safari through the reconfiguration jungle. In *38th IEEE/ACM Design Automation Conference (DAC'01)*. 172–177.

STRAUSS, W. 2000. Digital signal processing, the new semiconductor industry technology driver. *IEEE Signal Processing Magazine* (Mar.), 52–56.

VERBAUWHEDE, I., SCHEERS, C., AND RABAEY, J. 1996. Analysis of Multi-dimensional DSP Specifications. *IEEE Transactions on Signal Processing 44*, 12, 3169–3174.

VERBAUWHEDE, I. AND NICOL, C. 2000. Low power DSP's for wireless communications. In *2000 IEEE/ACM international symposium on low power electronics and design (ISLPED'00)*. 303–310.

VERHAEGH, W., AARTS, E., VAN GORP, P., AND LIPPENS, P. 2001. A Two-Stage approach to multidimensional periodic scheduling. *IEEE Transactions on CAD 20*, 10 (Oct.).

Received September 2004; revised February 2005; accepted May 2005