Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2006, Article ID 98045, Pages 1–11
DOI 10.1155/ES/2006/98045

Rapid Energy Estimation for Hardware-Software Codesign Using FPGAs

Jingzhao Ou[1] and Viktor K. Prasanna[2]

[1] *DSP Design Tools and Methodologies Group, Xilinx, Inc., San Jose, CA 95124, USA*
[2] *Veterbi School of Engineering, University of Southern California, Los Angeles, CA 90089, USA*

Received 1 January 2006; Revised 25 May 2006; Accepted 19 June 2006

By allowing parts of the applications to be executed either on soft processors (as software programs) or on customized hardware peripherals attached to the processors, FPGAs have made traditional energy estimation techniques inefficient for evaluating various design tradeoffs. In this paper, we propose a high-level simulation-based two-step rapid energy estimation technique for hardware-software codesign using FPGAs. In the first step, a high-level hardware-software cosimulation technique is applied to simulate both the hardware and software components of the target application. High-level simulation results of both software programs running on the processors and the customized hardware peripherals are gathered during the cosimulation process. In the second step, the high-level simulation results of the customized hardware peripherals are used to estimate the switching activities of their corresponding register-transfer/gate level ("low-level") implementations. We use this information to employ an instruction-level energy estimation technique and a domain-specific energy performance modeling technique to estimate the energy dissipation of the complete application. A Matlab/Simulink-based implementation of our approach and two numerical computation applications show that the proposed energy estimation technique can achieve more than 6000x speedup over low-level simulation-based techniques while sacrificing less than 10% estimation accuracy. Compared with the measured results, our experimental results show that the proposed technique achieves an average estimation error of less than 12%.

1. INTRODUCTION

The integration of multimillion gate configurable logic and various heterogeneous hardware components, such as embedded multipliers and memory blocks, offers FPGAs exceptional computational capabilities. Soft processors, which are RISC processors realized using configurable resources available on FPGA devices, have become popular for embedded system development. Examples of such soft processors include Nios from Altera [1], a SPARC architecture-based LEON3 from Gaisler [2], an ARM7 architecture-based CoreMP7 from Actel [3], and MicroBlaze from Xilinx [4]. As shown in Figure 1, for the development of FPGA-based embedded systems, parts of the application can be executed either on soft processors as programs or on customized hardware peripherals attached to the processors. Customized hardware peripherals are efficient for executing many data intensive computations. On the other hand, processors are efficient for executing many control and management functions, and computations with tight data dependency between steps (e.g., recursive algorithms). The use of soft processors leads to more compact designs and thus requires a much smaller amount of hardware resources than that of customized hardware peripherals. Having a compact design that fits into a small FPGA device can effectively reduce static energy dissipation [5]. The ability to make hardware and software design tradeoffs has made FPGAs an attractive choice for implementing a wide range of embedded systems.

Energy efficiency is an important performance metric for many embedded systems, such as software-defined radio (SDR) systems. In SDR systems, dissimilar and complex wireless standards (e.g., GSM, IS-95) are processed in a single adaptive base station, where a large amount of data from the mobile terminals present high computational requirements. State-of-the-art RISC processors and DSPs are unable to meet the signal processing requirements of these base stations. Power consumption minimization has become a critical issue for base stations, due to the high computational requirement that leads to high energy dissipation in inaccessible and distributed base station locations. FPGAs stand out as an attractive choice for implementing various SDR functions due to their high performance, low power

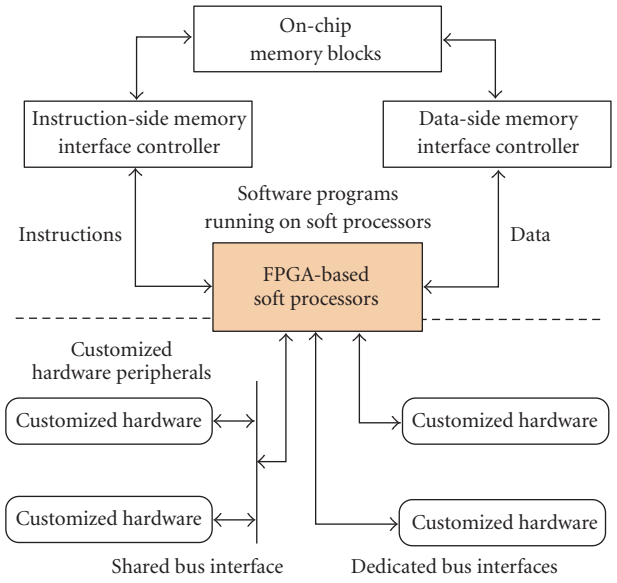

Figure 1: FPGA-based hardware-software codesign.

dissipation per computation, and reconfigurability [6]. Many hardware-software mappings and application implementations are possible on modern FPGA devices. The various hardware-software mappings and implementations can result in a significant variation in energy dissipation. Therefore, being able to obtain the energy dissipation of these different mappings and to evaluate implementations of the applications rapidly is crucial to energy efficient application development using FPGAs.

In this paper, we consider an FPGA device configured with a soft processor and several customized hardware peripherals attached to it. The processor and the hardware peripherals communicate with each other through specific bus protocols. The target application is decomposed into a set of tasks. Each task can be mapped onto either a soft processor (i.e., software), or a specific customized hardware peripheral (i.e., hardware), for execution. A specific mapping and execution schedule of the tasks are given. For tasks executed on customized hardware peripherals, their implementations are described using high-level modeling environments (e.g., MILAN [7], Matlab/Simulink [8], and Ptolemy [9]). For tasks executed on the soft processor, the software implementations are described as C code and compiled using the appropriate C compiler. One or more sets of sample input data are also given. Under these assumptions, our objective is *to rapidly and accurately (within about 10%) obtain the energy dissipation of the complete application.*

There are two major challenges for rapid and accurate energy estimation for hardware-software codesigns using FPGAs. One challenge is that state-of-the-art energy estimation tools are based on low-level (register transfer level and gate level) simulation results. While these low-level energy estimation techniques can be accurate, they are time-consuming and would be intractable when used to evaluate the energy performance of the different FPGA implementations. This is especially true for software programs running on soft processors. Considering the designs described in Section 5, the simulation of ~ 2.78 milliseconds execution time of a matrix multiplication application using post place-and-route simulation models takes about 3 hours in ModelSim [10]. Using XPower [4] to analyze the simulation file that records the switching activities of low-level hardware components and to calculate the overall energy dissipation requires an additional hour. The other challenge is that high-level energy performance modeling, which is crucial for rapid energy estimation, is difficult for FPGA designs. Lookup tables connected through programmable interconnect, the basic elements of FPGAs, can realize a wide range of different hardware architectures. They lack a single high-level model found in general purpose processors, which can capture the energy dissipation behavior of the various possible architectures.

As discussed in Section 2, while instruction-level energy estimation techniques can provide rapid energy estimates of processor cores with satisfactory accuracy, they are unable to account for the energy dissipation of customized instructions and tightly coupled hardware peripherals. More detailed energy performance models are required to capture the energy behavior of the customized instructions and hardware peripherals.

We propose a high-level simulation-based two-step rapid energy estimation technique for hardware-software codesign using FPGAs. In the first step, a high-level modeling environment is created to combine the corresponding high-level abstractions that are suitable for describing the hardware and software execution platforms. Within this high-level modeling environment, hardware-software cosimulation is performed to evaluate a cycle-accurate high-level behavior of the complete system. Instruction profiling information of the software execution platform and high-level activity information of the customized hardware peripherals are gathered during the cycle-accurate cosimulation process. The switching activities of the corresponding low-level implementations of the customized hardware peripherals are then estimated. In the second step, by utilizing the instruction profiling information, an instruction-level energy estimation technique is employed to estimate the energy dissipation of software execution. Also, by utilizing the estimated low-level switching activity information, a domain-specific modeling technique is employed to estimate the energy dissipation of hardware execution. The energy dissipation of the complete system is obtained by summing the energy dissipation of hardware and software execution.

A Matlab/Simulink-based implementation of the proposed energy estimation technique and two widely used numerical computation applications are used to demonstrate the effectiveness of our approach. For various implementations of these two applications, our high-level cosimulation technique achieves more than a 6000x speedup versus techniques based on low-level simulations. Such speedups can directly lead to a significant speedup in energy estimation. Compared with low-level techniques, our high-level simulation approach achieves an average estimation error of less than 10%. Compared with experimentally measured results,

our approach achieves an average estimation error of less than 12%.

The paper is organized as follows. Section 2 discusses related work. Section 3 describes our two-step rapid energy estimation technique. An implementation of our technique based on a state-of-the-art high-level modeling environment is presented in Section 4. The design of two numerical computation applications is described in Section 5. We conclude in Section 6.

2. RELATED WORK

Energy estimation techniques for FPGA designs can roughly be divided into two categories. One category is based on low-level simulation, which is employed by tools such as Quartus II [1], XPower [4], and the tool developed by Poon et al. [11]. In low-level simulation-based energy estimation techniques, the user generates low-level implementations of the FPGA designs. Simulation is performed based on the low-level implementations to obtain the switching activity of the low-level hardware components used in the FPGA design (e.g., basic configurable units and programmable wires). Each of the low-level hardware components is associated with an energy function that captures its energy behavior with different switching activities. Using the low-level simulation results and the low-level energy functions, the user can estimate the energy dissipation of all low-level components. The energy dissipation of the complete application is calculated as the sum of the energy dissipation of the low-level hardware components. Low-level estimation techniques are inefficient for FPGA-based hardware-software codesign. The creation of a low-level implementation includes synthesis, placement, and routing. This sequence forms a lengthy process. Simulations based on low-level implementations are very time consuming. This is especially true for the simulation of software.

The other category of energy estimation techniques is based on high-level energy models. The FPGA design is represented as a few high-level models interacting with each other. The high-level models accept parameters that have a significant impact on energy dissipation. These parameters are predefined or provided by the application designer. This technique is used by tools such as the RHinO tool [12] and the web power analysis tools from Xilinx [13]. While energy estimation using this technique can be fast, as they avoid time-consuming low-level simulation, its estimation accuracy varies among applications and application designers. One reason is that different applications demonstrate different energy dissipation behaviors. We show in [14] that using predefined parameters for energy estimation results in energy estimation errors as high as 32% for input data with different statistical characteristics. The other reason is that requiring the application designer to provide these important parameters would demand a deep understanding of the energy behavior of the target devices and applications, which can prove to be very difficult in practice. This approach is not suitable for estimating the energy estimation of software execution as instructions with different energy dissipations are executed on soft processors.

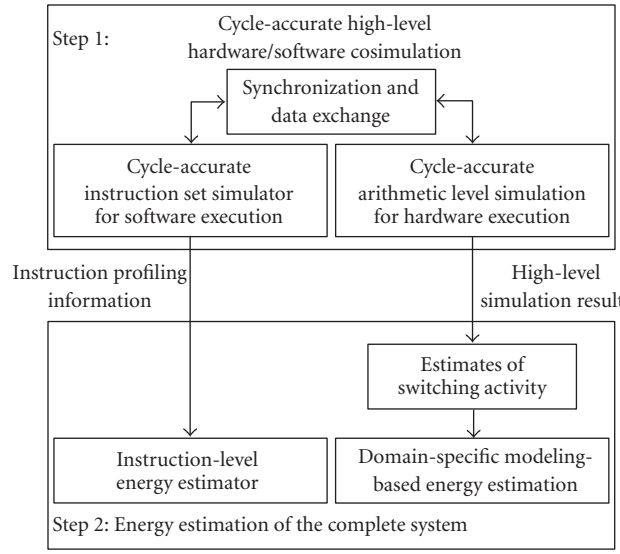

Figure 2: The two-step energy estimation approach.

For software execution on processors, instruction-level energy estimation is an effective technique for obtaining energy dissipation. This technique is used by several popular commercial and academic processors, such as *Wattch* [15], *JouleTrack* [16], and *SimplePower* [17]. *JouleTrack* estimates the energy dissipation of software programs on StrongARM SA-1100 and Hitachi SH-4 processors. *Wattch* and *SimplePower* estimate the energy dissipation of an academic *SimpleScalar* processor. We proposed an instruction-level energy estimation technique in [18], which can provide rapid and accurate energy estimation for FPGA-based soft processors. These energy estimation frameworks and tools target processors with fixed architectures. They do not account for the energy dissipated by customized hardware peripherals and communication interfaces. Thus, they are unable to provide energy estimation of combined hardware-software designs targeted to FPGA platforms. Low-level energy models are required for customized hardware peripherals.

3. OUR APPROACH

Our two-step approach for the rapid energy estimation of the hardware-software designs using FPGAs is illustrated in Figure 2. The two energy estimation steps are discussed in detail in the following sections.

3.1. Step 1: high-level cosimulation

In the first step, a high-level cosimulation is performed to simultaneously simulate hardware and software execution on a cycle-accurate basis. Note that we use "cycle-accurate" to denote that on both positive and negative edges of the simulation clock, the behavior of the high-level simulation models matches the corresponding low-level implementations. Other timing information between the clock edges (e.g., the glitches), as well as the logic and path delays between the

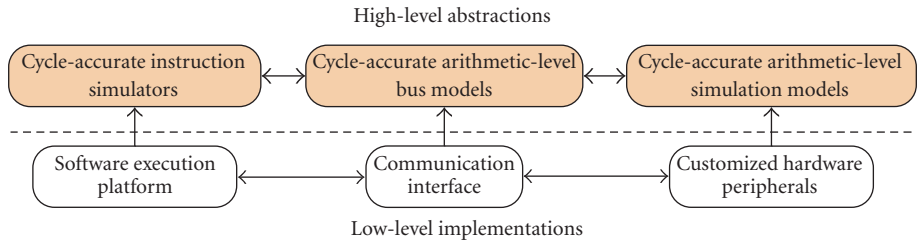

Figure 3: Architecture of the cycle-accurate high-level cosimulation environment.

hardware components, is not accounted for in the high-level simulation. There are two major advantages of maintaining cycle accuracy during cosimulation. One advantage is that by ignoring the low-level implementation and sacrificing some timing information, the high-level cosimulation framework can greatly speed up the simulation. This greatly speeds up the energy estimation process. Most importantly, the simulation results gathered during the high-level cosimulation process can be used to estimate the switching activities of the corresponding low-level implementations, and can be used in the second step of the energy estimation process to derive rapid and accurate energy estimates of the complete system.

It can be argued that urging cycle accuracy early, the design process prevents efficient design space exploration as cycle accuracy is usually not required in early hardware-software partitioning and in the development of software drivers. Our cosimulation framework only maintains cycle accuracy at the instruction level for software execution and arithmetic level for hardware execution. The cosimulation environment presents a view similar to the combination of the *architects view* and *programmers view* in transaction level modeling (TLM). Kogel et al. points out in [19] that "*there is usually no need for 100% timing accuracy since the impact of an architecture change is on a much bigger scope than a single clock cycle. Still an accuracy of 70–80% needs to be maintained to ensure the quality of the analysis results.*" Many state-of-the-art high-level modeling environments for digital signal processing systems, control systems, and so forth, enforce such cycle accuracy in their modeling process. Examples include the concept of high-level simulation clocks within the Matlab/Simulink and Ptolemy modeling environments. Compared with System C implementations of the transaction-level models, our design and cosimulation framework is based on visual data-flow modeling environments and thus is more suitable for describing embedded systems.

The architecture of the cosimulation environment is illustrated in Figure 3. The low-level implementation of the FPGA execution platform consists of three major components: the *soft processor* (for executing programs), *customized hardware peripherals* (hardware accelerators for parallel execution of some specific computations), and *communication interfaces* (for exchanging data and control signals between the processor and the customized hardware components). High-level abstractions are created for each of the three major components. The high-level abstractions are simulated using their corresponding simulators. The hardware and software simulators are tightly integrated into our cosimulation environment and concurrently simulate the high-level behavior of the hardware-software execution platform. Most importantly, the simulation among the integrated simulators is synchronized at each clock cycle and provides cycle-accurate simulation results for the complete hardware-software execution platform. Once the high-level design process is completed, the application designer specifies the required low-level hardware bindings for the high-level operations (e.g., binding the embedded multipliers to multiplication arithmetic operations). Finally, register-transfer/gate level ("low-level") implementations of the complete platform with corresponding high-level behavior can be automatically generated based on the high-level abstraction of the hardware-software execution platforms.

3.1.1. *Cycle-accurate instruction-level simulation of programs running on the processor*

We employ cycle-accurate instruction-level simulation models to simulate the execution of the instructions on a soft processor. These simulation models provide cycle-accurate simulation information regarding the execution of the instructions of the target program. With MicroBlaze [4], for example, the cycle-accurate instruction-set simulator records the number of times that an instruction passes the multiple execution stages, as well as the status of the soft processor, on a cycle-accurate basis. Most importantly, as we show in Section 4.2.1, such cycle-accurate instruction-level information can be used to derive rapid and accurate energy estimation.

3.1.2. *Cycle-accurate arithmetic level simulation of customized hardware peripherals*

Arithmetic level simulation is performed to simulate the customized hardware peripherals attached to the processors. By "arithmetic level," we mean that only the arithmetic aspects of the hardware-software execution are captured by the coimulation environment. For example, low-level implementations of multiplication on Xilinx Virtex-II FPGAs can be realized using either slice-based multipliers or embedded multipliers.

3.1.3. Maintenance of cycle accuracy throughout the cosimulation process

For each simulation clock cycle, the high-level behavior of the complete FPGA hardware platform predicted by the cycle-accurate cosimulation environment should match with the behavior of the corresponding low-level implementation. When simulating the execution of a program on a soft processor, cycle-accurate cosimulation should take into account the number of clock cycles required for completing a specific instruction (e.g., the multiplication instruction of the MicroBlaze processor takes three clock cycles to finish) and the processing pipeline of the processor. Also, when simulating the execution of customized hardware peripherals, cycle-accurate simulation should take into account delays in the number of clock cycles caused by the processing pipelines within the customized hardware peripherals. Our high-level simulation environment ignores low-level implementation details, and only focuses on the arithmetic behavior of the designs. By doing so, the hardware-software cosimulation process can be greatly sped up. In addition, cycle accuracy is maintained between the hardware and software simulators during the cosimulation process. Thus, the instruction profiling information and the low-level switching activity information, which are used in the second step for energy estimation, can be accurately estimated from the high-level cosimulation process.

3.2. Step 2: rapid energy estimation

In the second step, the information gathered during the high-level cosimulation process is used for rapid energy estimation. The types and the numbers of instructions executed on soft processors are obtained from the cycle-accurate instruction simulation process. The instruction execution information is used to estimate the energy dissipation of the programs running on the soft processor. For customized hardware implementations, the switching activities of the low-level implementations are estimated by analyzing the switching activities of the arithmetic level simulation results. Then, with the estimated switching activity information, energy dissipation of the hardware peripherals is estimated by utilizing a domain-specific energy performance modeling technique proposed in [20]. Energy dissipation of the complete system is calculated as the sum of the energy dissipation of the software and hardware implementations.

3.2.1. Instruction-level energy estimation for software execution

An instruction-level energy estimation technique is employed to estimate the energy dissipation of the software execution on the soft processor. A per-instruction energy lookup table is created, which stores the energy dissipation of each type of instruction for the specific soft processor. The types and the number of instructions executed when the program is running on the soft processor are obtained during the high-level hardware-software cosimulation process. By querying the instruction energy lookup table, the energy dissipation of these instructions is obtained. The energy dissipation of the program is calculated as the sum of the energy dissipations of all of the instructions.

3.2.2. Domain-specific modeling-based energy estimation for hardware execution

The energy dissipation of the customized hardware peripherals is estimated through domain-specific energy performance modeling presented in [20]. Domain-specific modeling is proposed to address the challenge of high-level FPGA energy performance modeling. FPGAs allow for implementing designs using a variety of architectures and algorithms. These architectures and algorithms use a different amount of logic components and interconnect. While these tradeoffs offer a great design flexibility, they prevent energy performance modeling using a single high-level model. For example, matrix multiplication on an FPGA can employ a single processor or a systolic architecture. An FFT on an FPGA can adopt a radix-2-based or a radix-4-based algorithm. Each architecture and algorithm would have different energy dissipation.

Domain-specific modeling (DSM) is a hybrid (top-down followed by bottom-up) modeling approach. It starts with a top-down analysis of the algorithms and the architectures for implementing a kernel. Through top-down analysis, the various possible low-level implementations of the kernel are grouped into *domains*, depending on the architectures and algorithms used. This DSM technique enforce a high-level architecture for the implementations belonging to the same domain. With such enforcement, high-level modeling within the domain becomes possible. Analytical formulation of energy functions are derived within each domain to capture the energy behavior of the corresponding implementations. Then, a bottom-up approach is used to estimate the constants of these analytical energy functions for the identified domains through low-level sample implementations. This includes profiling individual system components through low-level simulations, hardware experiments, and so forth. These domain-specific energy functions are platform-specific. That is, the constants in the energy functions would have different values for different FPGA platforms. During the application development process, these energy functions are used for rapid energy estimation of hardware implementations belonging to a particular domain.

The domain-specific models can be hierarchical. The energy functions of a kernel can contain the energy functions of the subkernels that constitute the kernel. Characteristics of the input data (e.g., switching activities) can have considerable impact on energy dissipation and are also inputs to the energy functions. This characteristic information is obtained through low-level simulation, or through high-level cosimulation described in Section 4.1. See [20] for more details regarding the domain-specific modeling technique.

4. AN IMPLEMENTATION

To illustrate our approach, an implementation of our rapid energy estimation technique based on Matlab/Simulink is described in the following sections.

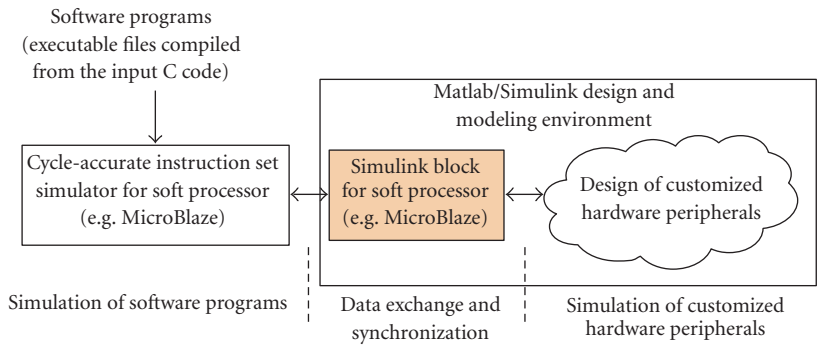

Figure 4: An implementation of the hardware-software cosimulation environment based on Matlab/Simulink.

4.1. Step 1: cycle-accurate high-level cosimulation

An implementation of the high-level cosimulation framework presented in Section 3.1 is shown in Figure 4. The four major functionalities of our Matlab/Simulink-based cosimulation environment are described as follows.

4.1.1. Cycle-accurate simulation of the programs

The input C programs are compiled using the compiler for the specific processor (e.g., the GNU C compiler *mb-gcc* for MicroBlaze) and translated into binary executable files (e.g., *.ELF* files for MicroBlaze). These binary executable files are then simulated using a cycle-accurate instruction set simulator for the specific processor. Taking the MicroBlaze processor as an example, the executable *.ELF* files are loaded into *mb-gdb*, the GNU C debugger for MicroBlaze. A cycle-accurate instruction set simulator for the MicroBlaze processor is provided by Xilinx. The *mb-gdb* debugger sends instructions of the loaded executable files to the Micro Blaze instruction set simulator and performs cycle-accurate simulation of the execution of the programs. *mb-gdb* also sends/receives commands and data to/from Matlab/Simulink through the Simulink block for the soft processor and interactively simulates the execution of the programs in concurrence with the simulation of the hardware designs within Matlab/Simulink.

4.1.2. Simulation of customized hardware peripherals

The customized hardware peripherals are described using the Matlab/Simulink-based FPGA design tools. For example, *System Generator* supplies a set of dedicated Simulink blocks for describing parallel hardware designs using FPGAs. These Simulink blocks provide arithmetic-level abstractions of the low-level hardware components. There are blocks that represent the basic hardware resources (e.g., flip-flop-based registers, multiplexers), control logic, mathematical functions, memory, and proprietary (intellectual property IP) cores (e.g., the IP cores for fast Fourier transform and finite impulse filters). For example, the *Mult* Simulink block for multiplication provided by *System Generator* captures the arithmetic behavior of multiplication by presenting at its output port the product of the values presented at its two input ports. The low-level design tradeoff of using either embedded or slice-based multipliers is not captured in its arithmetic level abstraction. The application designer assembles the customized hardware peripherals by dragging and dropping the blocks from the block set to his/her designs and connecting them via the Simulink graphic interface. Simulation of the customized hardware peripherals is performed within Matlab/Simulink. Matlab/Simulink maintains a simulation timer to keep track of the simulation process. Each unit of simulation time counted by the simulation timer equals one clock cycle experienced by the corresponding low-level implementations. Finally, once the design process in Matlab/Simulink completes, the low-level implementations of the customized hardware peripherals are automatically generated by the Matlab/Simulink-based design tools.

4.1.3. Data exchange and synchronization among the simulators

The soft processor Simulink block is responsible for exchanging simulation data between the software and hardware simulators during the cosimulation process. Matlab/Simulink provides *Gateway In* and *Gateway Out* Simulink blocks for separating the simulation of the hardware designs described by *System Generator* from the simulation of other Simulink blocks (including the MicroBlaze Simulink blocks). These *Gateway In* and *Gateway Out* blocks identify the input/output communication interfaces of the customized hardware peripherals. For the MicroBlaze processor, the Simulink MicroBlaze block sends the values of the processor registers stored in the MicroBlaze instruction set simulator to the *Gateway In* blocks as input data to the hardware peripherals. Vice versa, the Simulink MicroBlaze block collects the simulation output of the hardware peripherals from *Gateway Out* blocks and use the output data to update the values of the processor registers stored in the MicroBlaze instruction set simulator. The Simulink block for the soft processor also simulates the communication interfaces between the soft processor and the customized hardware peripherals described in Matlab/Simulink. For example, the Simulink MicroBlaze block simulates the communication protocol and the FIFO buffers for communication through Xilinx dedicated (fast simplex link FSL) interfaces [4].

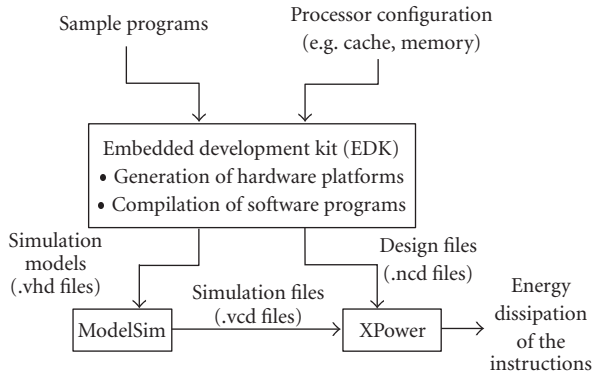

Figure 5: Flow of generating the instruction energy lookup table.

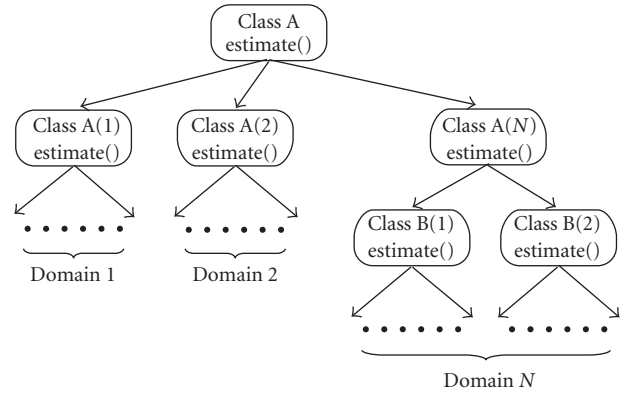

Figure 6: Python classes organized as domains.

The Simulink soft processor block maintains a global simulation timer which keeps track of the simulation time experienced by the hardware and software simulators. When exchanging the simulation data between the simulators, the Simulink soft processor block takes the number of clock cycles required by the processor and the customized hardware peripherals into account. This process considers both the input data and the delays caused by transmitting the data between them. Then, the Simulink block increases the global simulation timer accordingly. By doing so, the hardware and software simulations are synchronized on a cycle-accurate basis.

4.2. Step 2: rapid energy estimation

The energy dissipation of the complete system is obtained by summing up energy dissipation of the software and the hardware. These values are estimated separately by utilizing the activity information gathered during the high-level cosimulation process.

4.2.1. Instruction-level energy estimation for software execution

We use the MicroBlaze processor to illustrate the creation of the instruction energy lookup table. The overall flow for generating the lookup table is illustrated in Figure 5. We developed sample programs that target each instruction in the MicroBlaze processor instruction set by embedding assembly code into the sample C programs. In the embedded assembly code, we repeatedly execute the instruction of interest for a certain amount of time with more than 100 different sets of input data and under various execution contexts. ModelSim was used to perform low-level simulation for executing the sample programs. The gate-level switching activities of the device during the execution of the sample programs are recorded by ModelSim as simulation record files (.vcd files). Finally, a low-level energy estimator such as XPower was used to analyze these simulation record files and estimate energy dissipation of the instructions of interest. See [18] for more details on the construction of instruction-level energy estimators for FPGA configured soft processors.

4.2.2. Domain-specific modeling-based energy estimation for hardware execution

The energy dissipation of the customized hardware peripherals is estimated using the domain-specific energy modeling technique discussed in Section 3.2.2. In order to support this modeling technique, the application designer must be able to group different designs of the kernels into domains and associate the performance models identified through domain-specific modeling with the domains. Since the organization of the Matlab/Simulink block set is inflexible and is difficult to reorganize and extend, we map the blocks in the Simulink block set into classes in the object-oriented Python scripting language [21] by following some naming rules. For example, block *xbsBasic_r3/Mux*, which represents hardware multiplexers, is mapped to a Python class *CxlMul*. All the design parameters of this block, such as *inputs* (number of inputs) and *precision* (precision), are mapped to the data attributes of its corresponding class and are accessible as *CxlMul.inputs* and *CxlMul.precision*. Information on the input and output ports of the blocks is stored in data attributes *ips* and *ops*. By doing so, hardware implementations are described using Python language and are automatically translated into corresponding designs in Matlab/Simulink. For example, for two Python objects A and B, A.ips $[0:2]$ = B.ops $[2:4]$ has the same effect as connecting the third and fourth output ports of the Simulink block represented by B to the first two input ports of the Simulink block represented by A.

After mapping the block set to the flexible class library in Python, reorganization of the class hierarchy according to the architectures and algorithms represented by the classes becomes possible. Considering the example shown in Figure 6, Python class A represents various implementations of a kernel. It contains a number of subclasses $A(1), A(2), \ldots, A(N)$. Each of the subclasses represents one implementation of the kernel that belongs to the same *domain*. Energy performance models identified through domain-specific modeling (i.e., energy functions shown in Figure 7) are associated with these classes. Input to these energy functions is determined by the attributes of Python classes when they are instantiated. When invoked, the *estimate*() method associated with the Python

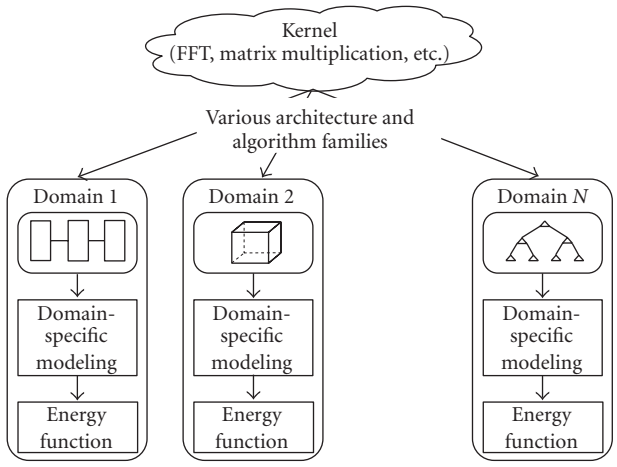

Figure 7: Domain-specific modeling.

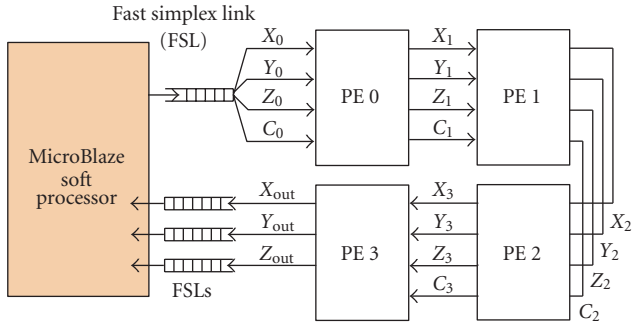

Figure 8: CORDIC processor for division ($P = 4$).

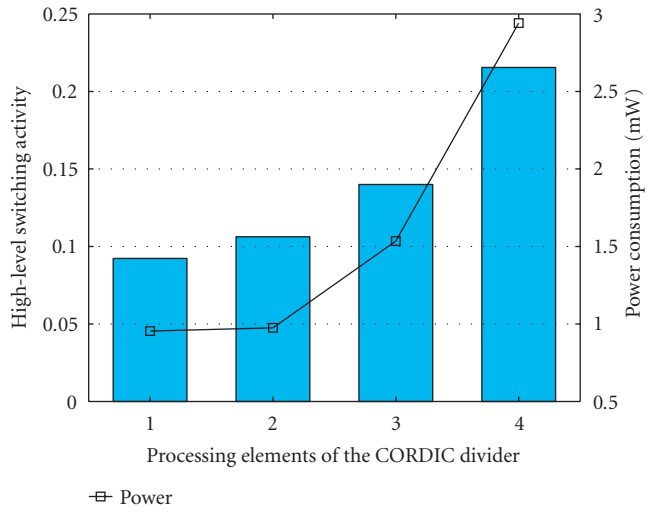

Figure 9: High-level switching activities and power consumption of the PEs shown in Figure 8.

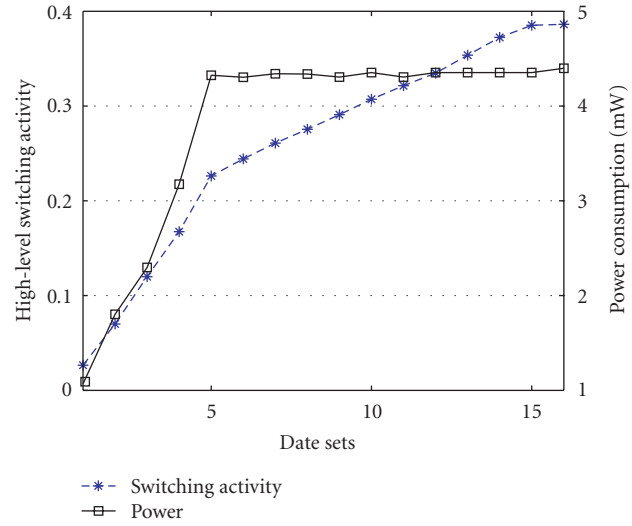

Figure 10: High-level switching activities and power consumption of slice-based multipliers.

classes returns the energy dissipation of the Simulink blocks calculated using the energy functions.

As a key factor that affects energy dissipation, switching activity information is required before these energy functions can accurately estimate energy dissipation of a design. The switching activity of the low-level implementations is estimated using the information obtained from the high-level cosimulation described in Section 4.1. For example, the switching activity of the Simulink block for addition is estimated as the average switching activity of the two input data and the output data. The switching activity of the processing elements (PEs) of the (coordinate rotation digital computer CORDIC) design [22] shown in Figure 8 is calculated as the average switching activity of all the wires that connect the Simulink blocks contained by the PEs. As shown in Figure 9, high-level switching activities of the processing elements (PEs) shown in Figure 8 obtained within Matlab/Simulink coincide with their power consumption obtained through low-level simulation. Therefore, using such high-level switching activity estimates can greatly improve the accuracy of our energy estimates. Note that for some Simulink blocks, their high-level switching activities may not coincide with their power consumption under some circumstances. For example, Figure 10 illustrates the power consumption of slice-based multipliers for input data sets with different switching activities. These multipliers demonstrate "ceiling effects" when switching activities of the input data are larger than 0.23. Such "ceiling effects" are captured when deriving energy functions for these Simulink blocks in order to ensure the accuracy of our rapid energy estimates.

5. ILLUSTRATIVE EXAMPLES

To demonstrate the effectiveness of our approach, we evaluate the design of a CORDIC processor for division and a block matrix multiplication algorithm. These designs are widely used in systems such as software-defined radio, where energy is an important performance metric [6]. We focus on MicroBlaze and *System Generator* in our illustrative examples

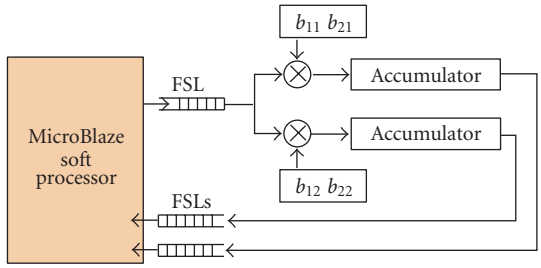

Figure 11: Matrix multiplication with customized hardware for multiplying 2×2 matrix blocks.

due to their easy availability. Our approach is also applicable to other soft processors and other design tools.

(i) CORDIC processor for division

The architecture of the CORDIC processor is shown in Figure 8. The customized hardware peripheral is implemented as a linear pipeline of P processing elements (PEs). Each of the PEs performs one CORDIC iteration. The software program controls the data flowing through the PEs and ensures that the data are processed repeatedly until the required number of iterations is completed. Communication between the processor and the hardware implementation is through the FSL interfaces. It is simulated using our MicroBlaze Simulink block. Our implementation uses 32-bit data precision.

(ii) Block matrix multiplication

Smaller matrix blocks of matrices A and B are multiplied using a customized hardware peripheral. As shown in Figure 11, data elements of a matrix block from matrix B (e.g., b_{11}, b_{21}, b_{12} and b_{22}) are fed into the hardware peripheral, followed by data elements of a matrix block from matrix A. The software program running on MicroBlaze controls the data to be sent to and retrieved from the attached customized hardware peripheral, performs part of the computation (e.g., accumulating the multiplication results from the hardware peripheral), and generates the result matrix.

In our experiments, the MicroBlaze processor is configured on a Xilinx Spartan-3 xc3s400 FPGA [4]. The processor, the two (local memory bus LMB) interface controllers and the customized hardware peripherals operate at 50 MHz. (embedded development kit EDK) 6.3.02 [4] is used to describe the software execution platform and for compiling software programs. *System Generator* 6.3 is used to describe the customized hardware peripherals. ISE (integrated software environment) 6.3.02 [4] is used for synthesizing and implementing (placing and routing) the complete applications.

Power measurement is performed using a Spartan-3 FPGA board from Nu Horizons [23] and a SourceMeter 2400 instrument (a programmable power source with the measurement functions of a digital multimeter) from Keithley [24]. Except for the Spartan-3 FPGA device, all the other components on the prototyping board (e.g., the power supply indicator, the SRAM chip) are kept in the same state during measurement. We assume that the changes in power consumption of the board are mainly caused by the FPGA device. We fix the input voltage and measure the changes in input current to the FPGA board. The dynamic power consumption of the designs is calculated based on the changes in input current. Note that *static power* (power consumption of the device when there is no switching activity) is ignored in our experimental results, since it is fixed in the experiments.

The simulation time and energy estimation for implementations of the two numerical computation applications are shown in Table 1. Our high-level cosimulation environment achieves simulation speedups between 5.6x and 88.5x compared with low-level timing simulation using ModelSim. The low-level timing simulation is required for low-level energy estimation using XPower. The speed of the cycle-accurate high-level cosimulation is the major factor that determines the estimation time and varies depending on the hardware-software mapping and scheduling of the tasks that constitute the application. This is due to two main reasons. One reason is the difference in simulation speeds of the hardware simulator and the software simulator. Table 2 shows the simulation speeds of the cycle-accurate MicroBlaze instruction set simulator, the Matlab/Simulink simulation environment for simulating the customized hardware peripherals, and ModelSim for timing-based low-level simulation. Cycle-accurate simulation of software executions is more than 4 times faster than cycle-accurate arithmetic level simulation of hardware execution using Matlab/Simulink. If more tasks are mapped to execute on the customized hardware peripherals, the overall simulation speed of the proposed high-level cosimulation approach is further slowed down. Compared with low-level simulation using ModelSim, our Matlab/Simulink-based implementation of the cosimulation approach can potentially achieve simulation speedups from 29x to more than 114x for the chosen applications. A reason for the variance is the frequency of data exchanges between the software program and the hardware peripherals. Every time the simulation data is exchanged between the hardware simulator and the software simulator, the simulation performed within the simulators is stalled and later resumed. This adds quite some extra overhead to the cosimulation process. There are close interactions between the hardware and software execution for the two numerical computation applications considered in the paper. Thus, the speedups achieved for the two applications are smaller than the maximum speedups that can be achieved in principal.

If we consider the implementation time (including synthesizing, placing-and-routing), the complete system, and generating the post place-and-route simulation models (required by the low-level energy estimation approaches) our high-level cosimulation approach would lead to even greater simulation speedups. For the two numerical applications, the time required to implement the complete system and generate the post place-and-route simulation models is about 3

Table 1: High-level/low-level simulation time and measured/estimated energy performance of the CORDIC-based division application and the block matrix multiplication application.

Designs	Simulation time		Energy estimation		
	High-level	Low-level*	High-level	Low-level	Measured
CORDIC with $N = 24$, $P = 2$	6.3 sec	35.5 sec	1.15 μJ (9.7%)	1.19 μJ (6.8%)	1.28 μJ
CORDIC with $N = 24$, $P = 4$	3.1 sec	34.0 sec	0.69 μJ (9.5%)	0.71 μJ (6.8%)	0.76 μJ
CORDIC with $N = 24$, $P = 6$	2.2 sec	33.5 sec	0.55 μJ (10.1%)	0.57 μJ (7.0%)	0.61 μJ
CORDIC with $N = 24$, $P = 8$	1.7 sec	33.0 sec	0.48 μJ (9.8%)	0.50 μJ (6.5%)	0.53 μJ
12×12 matrix mult. (2×2 blocks)	99.4 sec	8803 sec	595.9 μJ (18.2%)	675.3 μJ (7.3%)	728.5 μJ
12×12 matrix mult. (4×4 blocks)	51.0 sec	3603 sec	327.5 μJ (12.2%)	349.5 μJ (6.3%)	373.0 μJ

Note: * timing-based post place-and-route simulation. The times for placing-and-routing and generating simulation models are not included.

Table 2: Simulation speeds of the hardware-software simulators considered in this paper.

	Instruction set simulator	Simulink(1)	ModelSim(2)
Simulated clock cycles per second	>10000	254.0	8.7

Note: (1) only considers simulation of the customized hardware peripherals; (2) timing-based post place-and-route simulation. The time for generating the simulation models of the low-level implementations is not accounted for.

hours. Thus, our high-level simulation-based energy estimation technique can be about 200x to 6500x faster than those based on low-level simulation for these two numerical computation applications.

For the hardware peripheral used in the CORDIC division application, our energy estimation is based on the energy functions of the processing elements shown in Figure 8. For the hardware peripheral used in the matrix multiplication application, energy estimation is based on the energy functions of the multipliers and the accumulators. As one input to these energy functions, we calculate the average switching activity of all the input/output ports of the Simulink blocks during arithmetic level simulation. Table 1 shows the energy estimates obtained using our high-level simulation-based energy estimation technique. Energy estimation errors ranging from 9.5% to 18.2% and 11.6% on average are achieved for these two numerical computation applications compared with measured results. Low-level simulation-based energy estimation using XPower achieves an average estimation error of 6.8% compared with measured results.

6. CONCLUSIONS

A two-step rapid energy estimation technique for hardware-software codesign using FPGAs was proposed in this paper. An implementation of the proposed energy estimation technique based on Matlab/Simulink and the design of two numerical computation applications were provided to demonstrate its effectiveness. One major approximation that affects the energy estimation accuracy of the proposed technique is a failure to consider glitches in high-level simulation. This limitation creates two scenarios that causes our technique to fail to give energy estimates with satisfactory errors. One scenario occurs when an application runs close to its maximum operating frequency. The other scenario occurs when an application has long combinational circuit paths. In both scenarios, numerous glitches can occur in the circuits, causing high energy estimation errors for the proposed technique. The integration of high-level glitch power estimation techniques is an important extension of the proposed technique. Another important extension of our work is to provide confidence level information of the energy estimates. Providing such information is desired in the development of many practical systems.

ACKNOWLEDGMENTS

This work is supported by the United States National Science Foundation (NSF) under Award No. CCR-0311823. The authors would like to thank Brent Milne, Haibing Ma, Shay P. Seng, and Jim Hwang from Xilinx, Inc. for their help and discussions on creating the Matlab/Simulink-based high-level cosimulation environment.

REFERENCES

[1] Altera Inc., http://www.altera.com.
[2] Gaisler Research Inc., "LEON3 User Manual," http://www.gaisler.com.
[3] Actel Inc., http://www.actel.com.
[4] Xilinx Inc., http://www.xilinx.com.
[5] T. Tuan and B. Lai, "Leakage power analysis of a 90nm FPGA," in *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC '03)*, pp. 57–60, San Jose, Calif, USA, September 2003.

[6] C. Dick, "The platform FPGA: enabling the software radio," in *Proceedings of the Software Defined Radio Technical Conference and Product Exposition (SDR '02)*, San Diego, Calif, USA, November 2002.

[7] A. Bakshi, V. K. Prasanna, and Á. Lédeczi, "MILAN: a model based integrated simulation framework for design of embedded systems," in *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems*, pp. 82–93, Snowbird, Utah, USA, June 2001.

[8] MathWorks Inc., http://www.mathworks.com.

[9] "The Ptolemy Project," http://ptolemy.eecs.berkeley.edu.

[10] Mentor Graphics Inc., http://www.mentor.com.

[11] K. K. W. Poon, S. J. E. Wilton, and A. Yan, "A detailed power model for field-programmable gate arrays," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 2, pp. 279–302, 2005.

[12] "Reconfigurable Hardware in Orbit (RHinO)," Information Sciences Institute, http://rhino.east.isi.edu.

[13] "Web Power Analysis Tools," Xilinx, http://www.xilinx.com/power.

[14] J. Ou and V. K. Prasanna, "PyGen: a MATLAB/Simulink based tool for synthesizing parameterized and energy efficient designs using FPGAs," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04)*, pp. 47–56, Napa, Calif, USA, April 2004.

[15] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA '00)*, pp. 83–94, Vancouver, BC, Canada, June 2000.

[16] A. Sinha and A. Chandrakasan, "JouleTrack: a web based tool for software energy profiling," in *Proceedings of the 38th Design Automation Conference (DAC '01)*, pp. 220–225, Las Vegas, Nev, USA, June 2001.

[17] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of simplepower: a cycle-accurate energy estimation tool," in *Proceedings of the 37th Design Automation Conference (DAC '00)*, pp. 340–345, Los Angeles, Calif, USA, June 2000.

[18] J. Ou and V. K. Prasanna, "Rapid energy estimation of computations on FPGA based soft processors," in *Proceedings of the IEEE International System-on-Chip Conference (SoCC '04)*, pp. 285–288, Santa Clara, Calif, USA, September 2004.

[19] T. Kogel, A. Haverinen, and J. Aldis, "OCP TLM for Architectural Modeling (white paper)," OCP-IP, 2005, http://www.ocpip.org.

[20] S. Choi, J.-W. Jang, S. Mohanty, and V. K. Prasanna, "Domain-specific modeling for rapid energy estimation of reconfigurable architectures," *Journal of Supercomputing*, vol. 26, no. 3, pp. 259–281, 2003.

[21] Python, http://www.python.org.

[22] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Proceedings of the ACM/SIGDA 6th International Symposium on Field Programmable Gate Arrays (FPGA '98)*, pp. 191–200, Monterey, Calif, USA, February 1998.

[23] Nu Horizons Electronics Inc., http://www.nuhorizons.com.

[24] Keithley Instruments Inc., http://www.keithley.com.

Jingzhao Ou received his B.S. and M.S. degrees in electrical engineering from the South China University of Technology, and his M.S. and Ph.D. degrees in computer engineering from the University of Southern California. He is now working for the DSP Design Tools and Methodologies Group at Xilinx, Inc. His main research interests include hardware-software codesign and energy efficient application development using reconfigurable hardware.

Viktor K. Prasanna received his B.S. degree in electronics engineering from the Bangalore University, his M.S. degree from the School of Automation, Indian Institute of Science, and his Ph.D. degree in computer science from the Pennsylvania State University. He is now a Professor of electrical engineering and Professor of computer science at the University of Southern California (USC). He is also a Member of the NSF Supported Integrated Media Systems Center (IMSC), an Associate Member of the Center for Applied Mathematical Sciences (CAMS), and a Member of USC-ChevronTexaco Center of Excellence for Research and Academic Training on Interactive Smart Oilfield Technologies (CiSoft) at USC. His research interests include high-performance computing, parallel and distributed systems, network computing, and embedded systems.