

Cut-Based Functional Debugging for Programmable Systems-on-Chip

Darko Kirovski, Miodrag Potkonjak, and Lisa M. Guerra

Abstract—Due to the growth of both design complexity and the number of gates per pin, functional debugging has emerged as a critical step in the development of a system-on-chip (SOC). Traditional approaches, such as system emulation and simulation, are becoming increasingly inadequate to address the system debugging needs. Design simulation is two to ten orders of magnitude slower than emulation and, thus, is used primarily for short, focused test sequences. Emulation has the required speed but imposes strict limitations on signal observability and controllability.

We introduce a new debugging approach for programmable SOC's that leverages the complementary advantages of emulation and simulation. We propose a set of tools, transparent to both the design and debugging process, that enables the user to run long test sequences in emulation and, upon error detection, roll back to an arbitrary instance in execution time and switch over to simulation-based debugging for full design visibility and controllability.

The efficacy of the developed approach is dependent upon the method for transferring the computation from one execution domain to another. Although the approach can be applied to any computational model, we have developed a suite of optimization techniques that enable computation transfer in a mixed synchronous data flow semi-infinite stream random-access machine computation model. This computation model is frequently used in many communications and multimedia SOC's. The effectiveness of the developed debugging methodology has been demonstrated on a set of multicore designs where combined emulation-simulation has been enabled with low hardware and performance overhead.

Index Terms—Core-based design, debugging, emulation, error diagnosis, simulation, system-on-chip.

I. INTRODUCTION

AS THE complexity of designs increases, verification emerges as a dominant step with respect to time and cost in the development of a system-on-chip (SOC). For example, the UltraSPARC-I design team reported that debugging efforts took twice as long as their design activities [29]. The difficulty of verifying designs is likely to worsen in the future. The Intel development strategy team foresees that a major design concern for year 2006 microprocessor designs will be the need to exhaustively test all possible compatibility combinations [30]. The same team also states that the circuitry in their future designs devoted to debugging purposes is estimated to increase sharply to 6% from the current 3% of the total die area.

The two most important components for efficient functional and timing verification are speed of functional execution and design controllability and observability. Traditional approaches, such as design emulation and simulation, are becoming increasingly inefficient to address system debugging needs. Design emulation—implemented on arrays of rapid prototyping modules [field-programmable gate arrays (FPGA's)] or specialized hardware—is fast, but due to strict pin limitations provides limited and cumbersome design controllability and observability. Simulation—a software model of the design at an arbitrary level of accuracy—has the required controllability and observability but is, depending on the modeling accuracy, two to ten orders of magnitude slower than emulation [31]. For example, the functional verification team for the new HP PA8000 processor reported eight orders of magnitude difference in speed between the register-transfer level (RTL) simulated (0.5 Hz on a workstation) and FPGA-emulated (300 KHz) functional execution of their PA8000-based 200-MHz workstation system [18].

To combine the strengths of both verification domains, Kirovski and Potkonjak recently introduced a cut-based functional verification method that enables the verifier to seamlessly migrate the execution back and forth between design simulation and emulation [13]. Long test sequences are run in emulation. Upon error detection, the computation is migrated to the simulation tool for full design visibility and controllability. The functional execution is switched from one domain to another by transferring the *complete cut* of the computation. A complete cut of a computation is a set of variables that fully determines the design state at an arbitrary time instance. The running design (simulation or emulation) periodically outputs its cuts. The cuts are saved by a monitoring workstation. When a transition to the alternate domain is desired, any one of the previously saved cuts can be used to initialize and then continue execution with preserved functional and timing accuracy. However, this debugging technique, as proposed by Kirovski and Potkonjak, is restricted only to single-core statically scheduled ASIC designs [13].

Since current trends in the semiconductor industry show that programmable SOC's are becoming the dominant design paradigm, providing adequate verification tools for such systems is a premier engineering task. We have developed a generalized cut-based methodology for coordinated simulation and emulation of SOC's consisting of a system of programmable and application-specific cores. The methodology introduces a number of optimization problems and a need for efficient implementation mechanisms. We provide a set of tools that solve these problems for a mixed synchronous data flow (SDF) semi-infinite stream random-access machine (SISRAM) model of computation. This computation model is frequently used in many com-

Manuscript received July 31, 1998; revised April 7, 1999.

D. Kirovski and M. Potkonjak are with the Computer Science Department, University of California, Los Angeles, CA 90095 USA (e-mail: miodrag@cs.ucla.edu).

L. M. Guerra was with Conexant Systems, Newport Beach, CA 92660 USA. She is now with Bevoval, Mountain View, CA 94043 USA.

Publisher Item Identifier S 1063-8210(00)00771-X.

munications, multimedia, and digital signal-processing (DSP) applications. We propose a suite of algorithms that effectively identifies the minimal computation state (cut) and postprocesses the system components to enable I/O of cut variables. The experiments, conducted on a set of standard multicore benchmarks and industry-strength designs, quantify the overhead induced to enable the developed debugging paradigm. In all cases, no or negligible hardware and performance overhead was incurred while providing both fast functional execution and full design controllability and observability.

A. Motivational Example

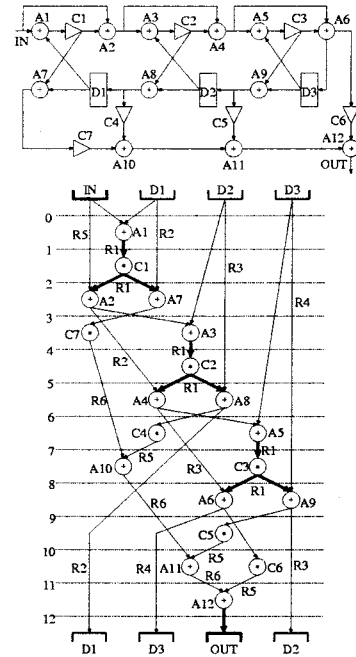
In this section, using a simple multicore application-specific system, we provide an overview of the design-for-debugging techniques used to enable cut-based functional debugging. The goal of design-for-debugging is to add minimal hardware resources into the component cores, such that during idle system bus cycles, both cores can output or input their states in the shortest possible time. The optimum solution to this problem often requires interleaving of the transfers of minimal states of component cores. In order to formalize the debugging approach, we introduce the generic definition of a cut (minimal computation state) and its application to the synchronous data flow computation model. A complete cut at time T is generically defined as a subset of variables from which any other variable computed after T can be computed. We introduce two alternative definitions of a cut of an SDF computation.

1) *The First Definition of a Complete Cut:* A complete cut is a set of variables generated within one computation iteration that bisects all possible paths in the computation.

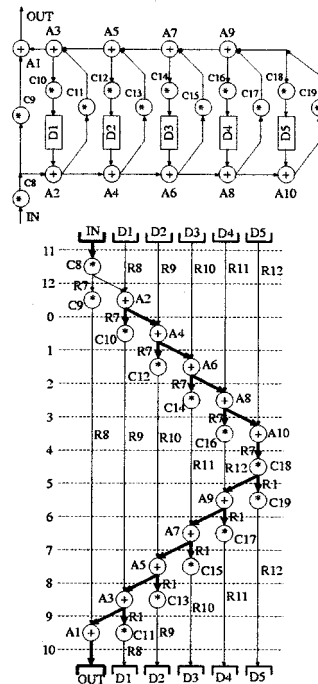
2) *The Second Definition of a Complete Cut:* A complete cut is a subset of variables that bisects all cyclic paths in the control data flow graph of a computation.

These two definitions enable exploration of tradeoffs in the cut-selection process. The first definition of a cut imposes a limitation that all contained variables must be selected from a single computation iteration. The second definition relaxes this requirement by enabling the search for a cut to be conducted among variables in several consecutive computation iterations. While cuts, which obey the first definition, require smaller trace capturing devices and induce lower computation initiation startup times, the cuts formed according to the second definition frequently require less hardware resources. In the remainder of this section, we demonstrate how a cut can be used to restart a computation and what are the involved tradeoffs in cut selection of multicore designs.

Consider an SOC consisting of a programmable core and two application-specific cores, a fifth-order CF infinite impulse response (IIR) filter [6], and a third-order Gray–Markel ladder IIR filter [10]. The programmable core places the inputs for both filters on a shared system bus. The filters periodically read the data, process it, and place the results of the computation back onto the shared bus. A single iteration of a computation process is finished when the programmable core reads the output data from the bus. The computations running on the ASIC’s are statically scheduled. The corresponding control data flow graphs (CDFG’s), operation scheduling, and register allocation



(a) The 3rd order Gray–Markel ladder filter.



(b) The 5th order CF IIR filter.

Fig. 1. Motivational example: system component core scheduled, allocated, and assigned CDFG’s.

are shown in Fig. 1. A single iteration of computation on each core requires 13 control steps. The Gray–Markel ladder filter inputs and outputs data in control steps 0 and 12, respectively, while the CF IIR filter inputs and outputs data in control steps 11 and 10, respectively. The architecture is shown in Fig. 2.

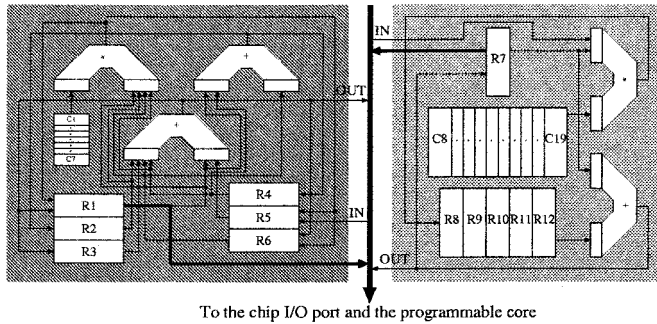


Fig. 2. Motivational example: ASIC architectures for the fifth-order CF IIR and third-order Gray-Markel ladder filter.

For example, consider two different cuts of the Gray-Markel ladder filter: 1) variables $D1$, $D2$, and $D3$ and 2) variables at the outputs of adders $A1$, $A3$, and $A5$, all stored in register $R1$. Both variable subsets bisect all cyclic paths in the CDFG—by deleting the edges in the CDFG that represent these variables, all cyclic paths are removed. The computation at iteration (i) can be restarted by injecting the values of the cut (a) in the first three control steps of the iteration (i), respectively, and applying the appropriate input sequence $In(i)$. In order to enable such data injection, registers $R2$, $R3$, and $R4$ have to be connected to the system I/O pins.

In order to use cut (b) to restart the computation at iteration ($i + 3$), cut values from three consecutive iterations (i), ($i + 1$), and ($i + 2$) are required before the machine state is correctly restored. Fig. 3 illustrates how variables $D1(i + 2)$, $D2(i + 2)$, and $D3(i + 2)$, and therefore the machine state at control step 0, are restored using the system primary inputs and cut variables (outputs from adders $A1$, $A3$, and $A5$) from the specified consecutive iterations. We demonstrate how the computation in that case will be executed correctly using an unfolded CDFG in Fig. 3. The bold lines in the unfolded CDFG represent all the intermediate variables required for computation of $D1$, $D2$, and $D3$. The design is initialized [prior to control step 0 of iteration (i)] with arbitrary variable values. To output cut (b), its variables have to be output through the I/O port at control steps 1, 4, and 7, respectively, in three consecutive iterations. Note that to enable I/O of cut (b), only one register $R1$ has to be connected to the I/O port.

In order to facilitate efficient SOC synthesis, the system integrator has to be provided with a certain level of cut scheduling programmability. The goal is to find for each core a subset of registers of minimal cardinality such that all the variables held by these registers constitute at least one complete cut. The additional constraint for register selection is that the variables of both cuts can be output and input through the available set of system pins. According to the register assignment in Fig. 1, one register in both cores is sufficient to provide a large set of complete cuts. In the case of the Gray-Markel filter, complete cuts are enabled by connecting register $R1$ to the output port. The cuts are formed by dispensing the output of $A1$ or $C1$, $A3$ or $C2$, and $A5$ or $C3$ at control steps 1 or 2, 4 or 5, and 7 or 8, respectively. These three pairs of variables, which exclusively have to be output, define eight different cuts. Similarly, by connecting the register $R7$ to the I/O pins, 32 complete cuts are en-

abled for the CF IIR filter. The variables computed at operations $A2$ or $A3$, $A4$ or $A5$, $A6$ or $A7$, $A8$ or $A9$, and $A10$ or $C12$ can be output at control steps 0 or 9, 1 or 8, 2 or 7, 3 or 6, and 4 or 5, respectively. Note that if the cut output schedule for the CF IIR filter is fixed at control steps 5–9, either a buffer needs to be allocated to one of the filters to hold the unscheduled variables or the cuts have to be dispensed sequentially.

II. RELATED WORK

State-of-the-art tools for system debugging have concentrated on enhancing the performance of simulation models as well as design visibility of hardware emulation blocks, rather than trying to provide methods for their synergy. State-of-the-art RTL simulators are equipped with debuggers capable of performing error tracing and timing analysis¹ and simulation backtracking.² Although instruction and cycle-accurate programmable processor simulators provide full system visibility, their speed corresponds directly to the achieved accuracy and is often insufficient to debug complex systems [31], [21]. To partially overcome these problems, a great deal of attention has been paid to providing kernels for mixed-model cosimulation of systems integrated using building blocks from different parties (Mentor Graphics Seamless [13]).

Hardware emulators were developed as early as 1979 [3] and have been under further development ever since [23], [25], [22]. Typical custom in-circuit emulator circuitry comprises capture logic, which monitors the contents of the program address register, the internal data bus, and control lines of the processor; trace circuitry comprising a first-in–first-out buffer, which puts data from the capture logic to the output pins of the chip; and a content addressable memory and a software programmable logic array with emulation counters that together function as a finite-state machine, which performs the desired predetermined testing. FPGA-based emulation systems have been developed by a number of companies, including Quickturn,³ Ikos,⁴ Aptix,⁵ and Axis.⁶

The observability and controllability of variables in such systems is a great challenge for emulator developers. The developed approaches are inefficient, expensive, or both. A common approach uses the expensive, low-bandwidth, and intrusive JTAG boundary scan methodology [20]. The most advanced application of JTAG circuitry has been introduced in the industry's first solution for run-time target application-host data exchange (RTDX) by Texas Instruments.⁷ Software developers use C or DSP assembly code to address an internal data exchange library, which in turn makes use of a scan-based emulator to move data on and off chip via the JTAG serial test bus.

Design controllability and observability can be obtained also by addressing user-customized SRAM memory cells (Quickturn Cobalt, Synopsys Eagle) or by probing nets into the FPGA

¹<http://www.interrainc.com/picasso.html>.

²http://www.synopsys.com/products/simulation/cyclone_cs.html.

³<http://www.quickturn.com/prod/hdlice/hdlice.htm>.

⁴<http://www.ikos.com>.

⁵<http://www.aptix.com:80/Products/mp4.html>.

⁶<http://www.axiscorp.com/>.

⁷<http://www.ti.com/sc/docs/dsps/tools/c5000/c54x/spr012.pdf>.

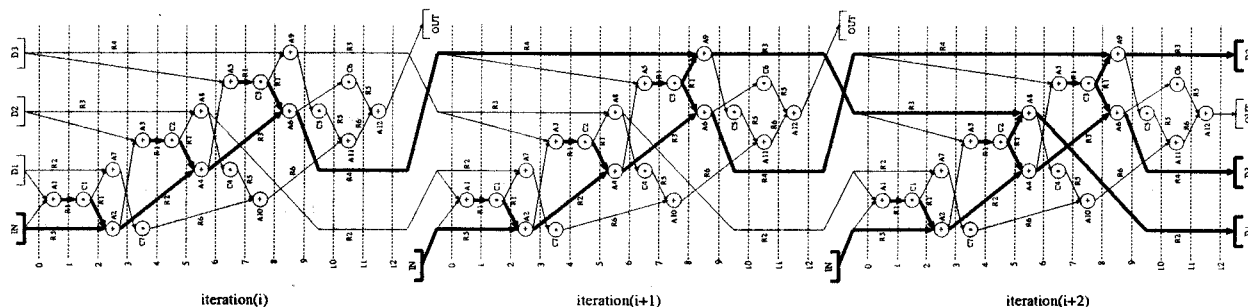


Fig. 3. Motivational example: unfolded CDFG over three consecutive iterations shows how variables $D1$, $D2$, and $D3$ are computed.

testbed (Quickturn System Realizer, Mentor Graphics SimExpress,⁸ etc.). While the former case raises expenses, the latter reduces visibility (1024 signals can be traced during four million cycles in Mentor Graphic Celaro, 1152-6912 probes—each 128 K deep—available in System Realizer).

Another approach to system debugging involves partitioning the system execution in simulation and emulation subsystems (Quickturn Q/Bridge, Synopsys Eagle, Axis Corporation). For example, Eagle uses emulation for the programmable components and simulation for the ASIC components. Novel challenges in system debugging are streamlined toward verification of emulation hardware with respect to the targeted functionality and timing [17], efficient tracing of a subset of signals from the emulator [14], and signal reconstruction for increased visibility and reduced emulation bandwidth demands [19].

We also present a brief overview of checkpointing in distributed and parallel systems. Checkpointing is used in fault-tolerant computing systems to prevent data loss and recomputation. In the domain of parallel systems, the usage of local data checkpoints has been explored for construction of consistent global checkpoints [27], [28]. Checkpoints have been used for synchronization of redundant task executions in fault-tolerant real-time systems [2].

III. PRELIMINARIES

The architecture template used to evaluate the developed debugging method is depicted in Fig. 4. The architecture is typical for most modern consumer electronics, multimedia, and telecommunications devices. It consists of a master programmable core (MPC) and a set of application-specific and slave programmable cores (SPC's), all connected to a shared bus. As shown in Fig. 4, an example ASIC could be a datapath unit with registers. Alternatively, an ASIC could be a background memory.

Two main, often contradictory, criteria for evaluation of system and behavioral synthesis models of computations are expressiveness [7] and suitability for optimization. While high expressiveness implies wider application domain, suitability for optimization often implies efficient implementation. For this target system, the following heterogeneous model of computation is assumed. The backbone of the model is the

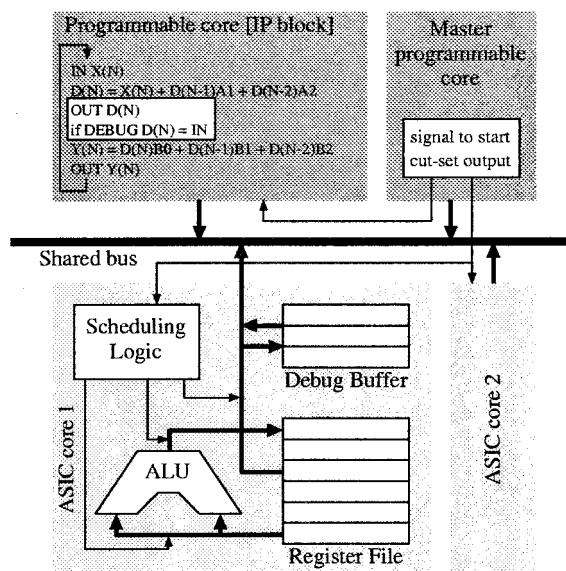


Fig. 4. The targeted system: individual core architecture, embedded software, and core intercommunication.

SISRAM model. The standard RAM model [1] is relaxed by removing a requirement for algorithm termination. The SISRAM model provides high flexibility with well-tested and widely used semantics and syntax (C and Java). The second component of the heterogeneous model is synchronous data flow [15]. This component facilitates optimization-intensive compilation for both programmable and ASIC platforms. Using this heterogeneous model of computation, we simultaneously address the needs of both control- and data-intensive applications. The programmable cores use a mixture of the two models, while the ASIC's use solely the SDF model. We have used C as the specification language for programmable cores and Silage [9] for the ASIC's. The intermediate representation used for algorithms is the control data flow graph.

The cut-based debugging approach is not limited to a specific computation model. For each computation model, though, a cut definition has to be established to satisfy the generic concept of a cut. In this paper, we describe cut selection for ASIC's synthesized using the SDF computation model. This simplification is assumed because of three reasons: brevity, availability of synthesis tools, and the fact that the SDF computation model corresponds to many data-intensive applications.

⁸<http://www.mentorg.com/codesign/main-f/index.htm>.

to modify the design upon request at integration time and 2) the system integrator can benefit from multiple cuts if hard-to-solve scheduling instances are encountered.

The I/O of variables of a particular computation cut is enabled by explicit connection of registers that store these variables to the I/O ports of the ASIC (if these registers are not already connected). On the other hand, note that one subset of registers may be used for I/O of a number of different cuts. This property of the register selection for cut I/O can be used to enable selection of the particular cut to be output at integration time. The goal is to achieve more flexible cut I/O in the cases when multiple cores are outputting their cuts on the same bus or the developed core is used as a subblock in a larger core. The integrator controls the selection of a particular cut using, for example, different control microcode.

B. Code Compilation-for-Debugging

In general, each programmable core has two components in its cut: instruction-accessible states (e.g., general-purpose registers) and states nonaccessible using machine code (e.g., branch prediction hardware, caches, and pipeline latches). The part of the cut accessible to instructions is transferred using debug instructions that are instrumented into the original code. The portion of the cut that is not accessible by instructions can be exported in several ways. Many state-of-the-art processors provide built-in debug ports that control breakpoint, pipeline and general-purpose registers, and memory access logic [30], [12]. Alternately, flushing of caches, pipelines, and/or branch predictors can be used as means of state invalidation. However, this approach may decrease the performance of the debugging system. Such deficiency may be unacceptable for debugging real-time systems. An alternate approach is to shadow the invisible states in such a way that at the moment of cut I/O the programmable core continues processing using one copy, while the shadowed copy is transferred to the monitoring workstation. Upon transfer, the shadowed copy is updated with the latest changes.

The debug instructions can be added either before or after compilation. While the precompilation choice requires in-source-code embedding of cut I/O instructions (for example, used in the MIPS Pixie [26]), the postprocessing step encompasses object code instrumentation similar to that implemented in Purify (Purify uses this technique to locate memory access errors) [11]. The precompilation approach has a significant advantage due to independence with respect to the hardware platform. Debugging platforms that can provide real-time JTAG support for such an approach have been already developed and marketed.

An example of instrumented code is given in Fig. 4 where the programmable core executes a second-order direct-form IIR filter. The instruction `OUT(D[N]);` is used for observability and if `Debug D[N] := IN;` for controllability. Controllability in emulation is beneficial because of the following application. The designer can change easily the state of the computation in simulation. Next, as a part of the debugging process, using emulation controllability, the designer can transfer the computation state from simulation to emulation, and hence restart the computation in emulation with an arbitrary computation state as a starting point.

The instrumentation process is performed in four phases. In the first phase, the minimal-size cut for each statically scheduled user-defined SDF computation island running on each programmable core is identified. An SDF computation island in a SISRAM computation is a set of instructions that process input data following the SDF computation model. The SDF computation islands are triggered by interrupts and executed pseudoperiodically according to the input data rate. In the second phase, the code is augmented with debug instructions that perform cut I/O. In the third phase, we identify the cut variables outside the SDF islands. Last, we instrument the code with instructions that initiate and call the function performing the system state I/O. Usually, the call to this function is placed in the main loop of the program. Using profiling tools, the designer determines the best location and frequency of calling of this function. The fourth phase of instrumentization for debugging is currently not automated.

C. Integration-for-Debugging

The ASIC developer provides the system integrator with information about the set of cuts that can be enabled. For each ASIC, the variables and control steps at which they can be dispensed through the virtual pins of the ASIC are given. The system integrator faces three design problems. First, for each ASIC, a single cut has to be selected. Second, the selected cuts, jointly with the primary inputs and outputs, are scheduled for I/O over the available set of pins. We integrated these two phases into a tight optimization loop, which searches for a feasible scheduling. Last, if no scheduling is found, the ASIC cuts are transferred sequentially in such a way that no two scheduled cut variables are displayed on the system bus in the same control step.

V. DESIGN-FOR-DEBUGGING: ALGORITHMS

In this section, the optimization problems related to design-for-debugging are identified, their computation complexity is established, and efficient algorithms are developed. There are three main optimization problems related to the support for debugging: 1) finding a minimal cut of a program being executed on a PC; 2) finding a set of register-to-output interconnects that enables a large number of nonoverlapping cuts; and 3) selection of a cut for each ASIC such that the I/O of all cuts is interleaved and conducted in minimal number of control cycles.

A. Code Instrumentation for Cut I/O

The export and import of cut variables of computations executed on programmable cores is performed by executing debug instructions embedded into the original code by a compilation postprocessing tool. The embedded instructions impose an overhead on the program performance and storage. The code size overhead is directly proportional to the cardinality of the selected cuts. The number of embedded instructions also affects the number of required cycles, and therefore the time overhead to output the programmable core cut. These two problems are not identical since different code segments can be executed with different dynamic frequencies in the SISRAM computation model. Since in Section VI the timing overhead is shown to be minimal on a variety of applications, we focus our

attention on the first optimization problem. Solutions for both optimization problems are strongly positively correlated in the sense that a good solution to one of the problems implies high likelihood for a good solution to the other.

PROBLEM: PC Cut Selection.

INSTANCE: Given a computation presented as directed cyclic graph and an integer M .

QUESTION: Is there a subset of edges that correspond to node outputs $O_i, i = 1, \dots, N$ such that when deleted leaves no directed cycles in the graph, and that $N < M$?

The PC Cut Selection is an NP-complete problem since there is one-to-one mapping between the special case of this problem, when all operations in the computation are executed exactly the same number of times, and the problem is finding the minimal feedback arc set [8].

To address this problem, we have developed a heuristic summarized using pseudocode in Fig. 7. The heuristic initially partitions the graph into a set of strongly connected components (SCC's) using the breadth-first search algorithm [5]. This algorithm has complexity $O(V + E)$, where V is the number of vertices and E is the number of edges in a graph. All trivial SCC's, which contain exactly one vertex, are deleted from the resulting set since they do not form cycles. The algorithm then iteratively performs several processing steps on each of the non-trivial SCC's. At the beginning of each iteration, to reduce the solution search space, a graph compaction step is performed. In this step, each path $P : A \rightarrow B$, which contains only vertices $V \in P, V \neq A$ with exactly one variable input, is replaced with a new edge $E_{A,B}$ that connects the source A and destination B and represents an arbitrary selected edge (variable) of the same path.

In the next step, an objective function decides which node (variable) in the current set of SCC's is to be deleted. The function analyzes, for the deletion of each vertex, the cardinality and the vertex cardinalities of the new set of SCC's. The vertex that results in the smallest objective function is deleted from the set of nodes as well as all adjacent edges. The deleted vertex is added to the resulting cut. The process of graph compaction, candidate node deletion evaluation, node deletion, and graph updating is repeated while the set of nontrivial SCC's in the graph is not empty. The set of nodes deleted from the computation represents the final cut selection.

Consider the example shown in Fig. 8. The CDFG of the third-order Gray–Markel ladder IIR filter has only one non-trivial SCC. The graph compaction step is explained in Fig. 8(a) where vertex B is merged with vertex A as well as variable W merged with variable V . In Fig. 8(b), an example of node deletion is described. The deleted node creates two smaller SCC's.

B. Cut Selection and Register-to-Port Interconnection

The goal of the ASIC design-for-debugging process is to assign a minimal number of register-to-output interconnects such that large number of complete cuts can be output from the core. An additional constraint is set on the timing occurrence of these cuts. Since the core developer does not know in advance the multicore system configuration, i.e., the future scheduling constraints of the cut variables, its search for a set of register-to-output interconnects is targeted for a large number

```

Create a set  $SCC = Scc(CDFG(V, E))$  of SCCs [Cor90]
For each  $SCC_i \in SCC$ 
  If ( $|SCC_i| = 1$ ) Delete  $SCC_i$  from  $SCC$ 


---


Repeat  $LOOPS$  times
   $CUT = null$ 
  While  $SCC \neq empty$ 
    For each  $SCC_i \in SCC$ 
       $GraphCompaction(SCC_i)$ 
      For each node  $V_{i,j}$ 
        Compute  $scc = Scc(SCC_i - V_{i,j})$ 
         $OF(scc) = (1 + \alpha) \sum_{i=1}^{|scc|} (|scc_i| \cdot NEdges(scc_i))$ ,
        where  $\alpha$  is random number  $\alpha \in \{0, \frac{1}{|SCC|^2}\}$ 
        Select  $V_{i,j} \rightarrow OF(scc(SCC_i, V_{i,j}))$  is minimal
        Delete  $V_{i,j}$  from  $SCC_i$ 
         $SCC = scc(SCC_i, V_{i,j})$ 
      For each  $SCC_i \in SCC$ 
        If ( $|SCC_i| = 1$ ) Delete  $SCC_i$  from  $SCC$ 
         $CUT = CUT \cup E_{i,j}$ 
  If ( $|CUT| < |BESTCUT|$ )  $BESTCUT = CUT$ 


---


Return  $BESTCUT$ 


---


Procedure  $GraphCompaction(SCC_i)$ 
  For each vertex  $V_i \in SCC_i$ 
    If  $V_i$  has one input edge  $E_{j,i}$  with a source in  $V_j$ 
      Foreach edge  $E_{i,k}$ 
        Create edge  $E_{j,k}$ 
        Delete  $E_{i,k}$ 
      Delete  $E_{j,i}$  and  $V_i$ 

```

Fig. 7. Finding the cut of the third-order Gray–Markel ladder IIR filter. The original CDFG for this filter structure is presented in Fig. 1(a).

of time nonoverlapping small cuts. Such a subset of registers gives the system integrator more flexibility to find a scheduling solution for interleaved cut I/O. We introduce a heuristic search objective and algorithm for such subset of registers.

1) *Heuristic Definition 1:* A “debugging prospective register subset” is a subset of registers $R_i, i = 1, \dots, R_M$ that defines a set of M distinct cuts $C_i, i = 1, \dots, M$ that satisfies the following two statements: $R_M < Max$ and

$$OF(C_i, i = 1, \dots, M) = \frac{\sum_{i=1}^M \sum_{V \in C_i} LifeTime^2(V)}{(\sum_{i=1}^M |C_i|) \cdot (\sum_{V \in ControlStepCS} LiveVariables^2(CS))} < K$$

where function $LiveVariables(CS)$ returns the number of variables alive at control step CS , K is a given real number, and M, Max, R_M are given integers.

The definition of a debugging prospective register subset forces selection of registers that define a set of cuts with large cardinality, small cardinality of containing cuts, long lifetimes of containing cut variables, and nonoverlapping lifetimes of variables in the set of cuts. These three properties of the selected subset of registers provide flexibility at various decision levels for multicore cut scheduling. The core developer faces an optimization problem to find the debugging register subset with the smallest possible constants Max and K where constant Max has priority over K .

PROBLEM: Debugging-Prosppective Register Selection.

INSTANCE: Given scheduled and assigned CDFG, real number $MinOF$, and integer $MinCard$.

QUESTION: Is there a subset of registers $R_i, i = 1, \dots, R_M$ that determines a set of M distinct CDFG cuts $C_i, i = 1, \dots, M$, and has $OF(C_i, i = 1, \dots, M) < Min$ and

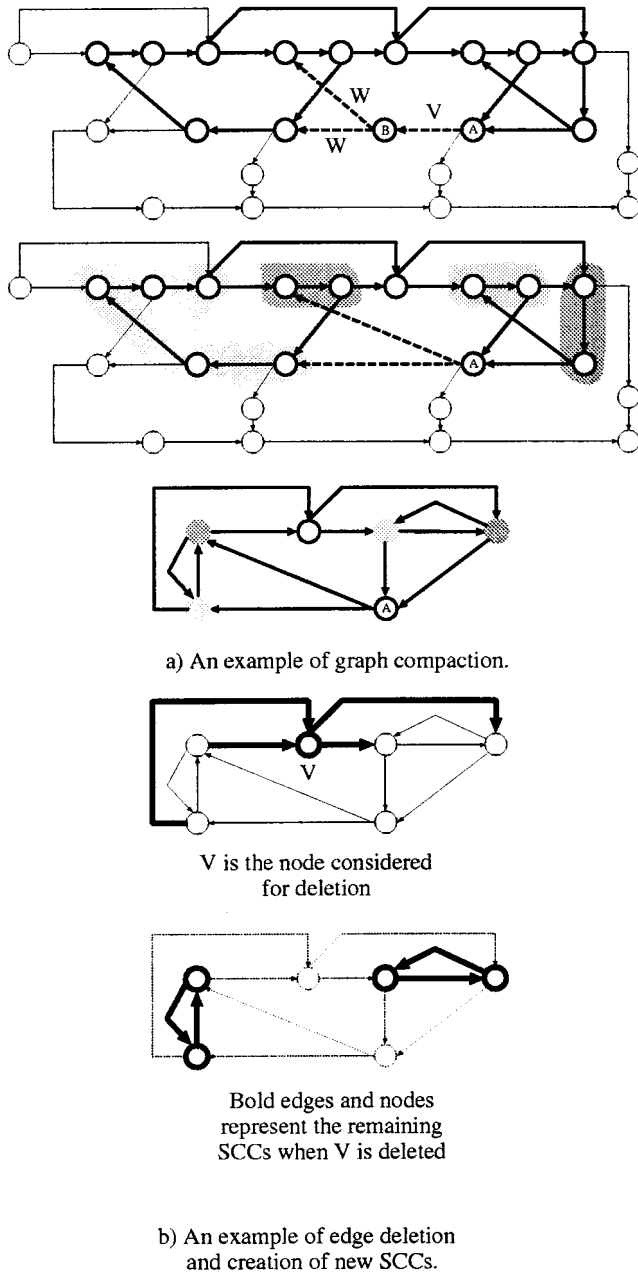


Fig. 8. Finding the cut of the third-order Gray-Markel ladder IIR filter. The original CDFG for this filter structure is presented in Fig. 1(a).

$R_M < \text{MinCard}$? A special case of the debugging register selection problem, with no register sharing among CDFG computation variables and no additional heuristic requirements, is equivalent to the problem of finding the minimal feedback arc set [8]. Therefore, debugging-prospective register selection is computationally an intractable problem.

We developed a heuristic to search for a debugging prospective register subset in a scheduled and assigned CDFG. The algorithm is formally explained using the pseudocode in Fig. 9. First, the algorithm partitions the CDFG into a set SCC of SCC_i 's. Then for each register R_i , the heuristic, using the objective function described below, evaluates the set of strongly connected subgraphs $scc_j \in scc$ that are a result of deletion of all variables held by register R_i . The objective

```

Create a set  $SCC = scc(CDFG(V, E))$  of SSC [Cor90].
For each  $SCC_i \in SCC$ 
  If ( $|SCC_i| = 1$ ) Delete  $SCC_i$  from  $SCC$ 
Repeat  $LOOPS$  times
  Starting set of registers  $SR = \text{null}$ 
  While  $SCC \neq \text{empty}$ 
    For each register  $R_i \ni SR$ 
      Compute  $scc = scc(CDFG - E_{i,j} | E_{i,j} \in R_i)$ 
      Select the register  $R_k$  which results in minimum
       $OF(R_k, SR, CDFG)$  and delete all variables
      held by  $R_k$  from  $SCC_i$ 
       $SR = SR \cup R_k$ 
       $SCC = scc(SCC_i, E_{i,j} | E_{i,j} \in R_k)$ 
    For each  $SCC_i \in SCC$ 
      If ( $|SCC_i| = 1$ ) Delete  $SCC_i$  from  $SCC$ 
  If ( $OF(SR) > OF(BESTSR)$ )  $BESTSR = SR$ 
Return  $BESTSR$ 
    
```

Fig. 9. Pseudocode for the debugging prospective register subset search.

function used to quantify the register selection importance is shown in the equation at the bottom of the next page, where $SCCcar(R, CDFG)$ returns the sum of squares of cardinalities of strongly connected components scc_i when all variables held by register R are deleted from the CDFG. $\text{LiveVariables}^2(\text{CS}, SR)$ returns for control step CS a sum of squares of number of variables alive at CS and held by the currently selected subset of registers SR . The register with the highest objective function is selected, added to the currently selected subset of registers SR , and all its variables (edges) are deleted from the original CDFG. The process of register selection is recursively repeated while the set of nontrivial SCC 's is not empty.

In order to provide probabilistic register selection, and therefore improve the search engine, in our implementation we multiply the original objective function value with a random number within a prespecified offset. The search for the best register selection is iterated S times (in our experiments S was equal to the number of operations in the CDFG).

C. Cut Scheduling of Multiple Cores

In this section, we introduce an algorithmic solution that enables efficient I/O of cut variables from multiple statically scheduled ASIC's. This design step is done during system integration. The optimization problem has three modular subtasks. In the first subtask, at least one cut is selected for each ASIC. The second phase encompasses a search within the common multiple (CM) of periods of all ASIC's in the system for a subset of cuts that can be scheduled in the fewest successive control steps. The third task encompasses scheduling of the system cut.

As an input to the algorithm, the system integrator is provided in a table where each row represents a list of variables that constitute an ASIC cut and a range of control steps when each variable can be read or written. In order to reduce the size of the table, we treat a list of consecutively alive variables in a single cut that are stored in the same register and that are dependent only upon the variable previously stored in that register (except the first variable in the list) as a single "compacted" variable. An example of such a cut table, which corresponds to the CDFG of the third-order Gray-Markel ladder filter presented in Fig. 1(a), is depicted in Table I. Since the output variables of operations $A1$ and $C1$ fulfill the above requirements and

TABLE I
PSEUDOCODE FOR PC CUT SELECTION
SEARCH

Variables and their life-times		
A1-C1 (1,2)	A3-C2 (4,5)	A5-C3 (7,8)

are stored in the same register $R1$, we represent these variables using a single variable (denoted as “compacted” in the remainder of the text). Similarly, output variables of operations (A3 and C2) and (A5 and C3) are compacted as two distinct variables with lifetimes that span over the lifetimes of the original variables.

In general, each ASIC can have a different period of its ongoing computation. Therefore, we use the CM of all, as the system ASIC debugging period. Within this period the algorithm tries to find a feasible schedule of variables of all ASIC cuts such that the range of control steps is minimal between the moments when the first and last cut variable in the ASIC subsystem are output. The problem is formally stated in the following way.

PROBLEM: Cut Selection and Scheduling.

INSTANCE: Given a set of cores ASIC, a set of cuts CUT_{Core} for each core $CORE \in ASIC$, a set of variables V for each cut $C \in CUT_{Core}$, a set of control steps CS_v for which each variable $v \in V$ is alive, a set of CS control steps at which chip ports are idle, and integer MaxRange.

QUESTION: Is there a selection f of a cut CUT_{Core}^f for each core, such that for each variable $v \in CUT_{Core}^f$ exists distinct control steps $CS_v^f \in CS$ at which the chip port is idle, no two variables $v \in CUT_{Core1}^f$, $w \in CUT_{Core2}^f$ are scheduled for transfer $CS_v^f \neq CS_w^f$ through the chip port at the same idle control step, and that the $\max(CS_v^f - CS_w^f) < MaxRange$?

The problem of scheduling a subset of variables in a CDFG [13] is a special case of the cut selection and scheduling problem. Since the former problem is NP-complete [13], the cut selection and scheduling problem is also computationally intractable. We developed a most constrained, least constraining heuristic in order to provide a competitive solution to this problem. The heuristic is explained using the pseudocode in Fig. 10.

The developed algorithm addresses simultaneously the cut selection and scheduling by integrating heuristics for their solution in a tight search loop. Initially, for each ASIC, the available cuts, all of equal cardinality, are sorted in decreasing order with respect to the average lifetime of contained variables. The selection and scheduling search loop starts by selecting one cut for each ASIC from its list of available cuts. In order to provide search randomization and at the same time give priority to cuts with lower indexes, i.e., cuts that contain variables with longer average lifetimes, the probability of selecting a particular cut is made proportional to the square of its average variable lifetime.

Cut Selection Preprocessing:

For each ASIC_i

Create a list L_i of cuts $CS_{i,j}$ in decreasing order of average life-time of contained variables.

Cut Selection:

Repeat $LOOPS$ times

For each ASIC_i

Select cut $CS_{i,j} \in L_i$ where j is an index selected among all other indexes with probability proportional to the square of average life-time of variables in $CS_{i,j}$

$Range = \sum_{i=1}^{|ASIC|} |CS_{i,j}|.$

Repeat

Find the set $TIME$ of $Range$ idle consecutive control steps in the CM of periods of all ASICs that contain the cuts of all $CS_{i,j}$.

For each $TIME_p \in TIME$

For each subset of $|ASIC|$ distinct cuts of each ASIC encompassed with $TIME_p$

Schedule $CS_{i,j}$ in $TIME_p$.

If schedule found

and in shorter time than the best schedule **then** best = current schedule.

$Range = Range + 1.$

until $Range > BestRange$ or $Range == |LCM|.$

Cut Scheduling: [Schedule $CS_{i,j}$ in $TIME_p$]

Repeat until all variables scheduled

For each control step C_i

Compute its constraint $C_i.constraint$ as sum of $\frac{1}{v.lifetime}$ for each variable V alive at C_i .

For each variable V_i

Compute its constraint $V_i.constraint$ as sum of constraints of control steps at which V_i is alive.

Select N most-constrained tasks and exactly schedule them at control steps with the smallest sum of constraints.

Fig. 10. Pseudocode for the cut selection and scheduling algorithm.

Once cuts for all ASIC's are selected, the next step is to find, within CM consecutive control steps, the smallest subset of M consecutive control steps in which the variables of the selected cuts can be scheduled. The search is initiated by determining the lower bound on the range of control steps $M = M_{\min}$ for which all cuts can be dispensed. This bound is equal to the sum of the cardinalities of all ASIC cuts. Next, within the frame of CM consecutive control steps, a set $TIME$ is found where each element $TIME_p \in TIME$ represents a particular subset of N_p consecutive control steps that contains at least M_{\min} idle control steps, and for each variable of all ASIC cuts there must be at least one of these idle steps in which it is alive. One frame of control steps $TIME_p$ may “contain” more than one of the available cuts for one ASIC. Therefore, for each combination of cuts within $TIME_p$, a scheduling heuristic is performed.

The scheduling heuristic iteratively constructs the solution by selecting N most constrained cut variables and scheduling them exactly at the N least constraining control steps. The functions used to quantify the constraint are given in the pseudocode in Fig. 10. If feasible scheduling is found, the range of the solution N_p is compared to the best current solution and if more

$$OFR(R, SR, CDFG) = \frac{SCCcar(R, CDFG) \cdot \sum_{V \in ControlStep} CS_{LiveVariables}^2(CS, SR)}{\sum_{V \in R} LifeTime^2(V)}$$

TABLE II
PSEUDOCODE FOR THE DEBUGGING PROSPECTIVE REGISTER SUBSET SEARCH

ASIC	Architecture						Debug information		
	Period	Variables	Area (mm^2)			cut cardinality	Area overhead		
			ALU	Reg	Mux		Total	(mm^2)	%
Cascade	14	51	2.79	0.73	0.36	3.88	2 + 4	0	0
Continued Fraction	19	53	4.28	1.09	0.31	5.69	2 + 8	0	0
Direct Form II	10	53	10.65	1.45	0.57	12.68	2 + 1	0	0
Parallel	10	57	3.56	0.70	0.41	4.67	2 + 4	0	0
Parallel	9	57	4.70	0.71	0.38	5.79	2 + 4	0	0
Modem	20	50	1.83	0.71	0.25	2.79	2 + 1	0.01	0.3
Lin3	10	86	13.11	1.51	0.78	15.40	5 + 0	0	0
Lin3	15	86	6.63	1.48	0.58	8.70	6 + 0	0	0
Mat	15	29	2.21	0.39	0.04	2.65	4 + 0	0	0
Ellip	15	50	4.42	0.81	0.22	5.46	5 + 0	0	0
Volterra	15	40	1.41	0.35	0.13	1.88	2 + 1	0.06	3

TABLE III
TABLE OF CUTS FOR THE THIRD-ORDER GRAY-MARKEL LADDER FILTER

Application-specific core mix	System period	Number of system output variables	Control steps required to output the system cut
Cascade, Modem, Direct Form II, Volterra	20	15	18
Continued Fraction, Lin3 (T=15)	19	16	17
Parallel (T=10), Mat, Ellip	15	15	15
Lin3 (T=10), Lin3 (T=15), Continued Fraction	22	21	22
Cascade, Direct Form II, Parallel (T=10), Mat, Ellip	31	24	27
Continued Fraction, Mat, Direct Form II, Modem	21	20	20
Parallel (T=10), Modem, Direct Form II, Cascade	19	18	18
Parallel (T=9), Ellip, Lin3 (T=10), Volterra	21	19	20
Parallel (T=9), Lin3 (T=10), Lin3 (T=15), Volterra	25	20	23
All cores in Table 2	61	57	59

competitive is memorized as the best. If feasible scheduling is not found, the control step range M is increased and the search procedure is repeated until for a given set of cuts scheduling is found or the current range exceeds the scheduling range of the current best solution. The best scheduling solution is the one that a minimal number of consecutive control steps inputs or outputs the variables of each ASIC in the system.

VI. EXPERIMENTAL RESULTS

We have conducted a set of experiments to evaluate the effectiveness of our system debugging paradigm. Table II shows the set of application-specific cores that were used, and the area overhead introduced by the design-for-debugging post-processing step that introduces hardware to enable cut I/O. The application-specific cores include a number of Avenhaus IIR filters, several linear controllers, a nonlinear volterra filter, and a modem. The designs were synthesized using HYPER [24]. In the first column, the name of the core is presented followed in the next six columns with core architecture data, such as the length of one iteration in control steps, number of variables in the application-specific computation, core area dedicated to the execution units, registers, multiplexers, and total area. In column eight, the number of variables in the smallest cut is presented. The first number in the sum is the number of control steps at

which functional I/O is performed while the second number is the number of control steps at which the cut variables are I/O. Finally, the last two columns show the final total area and the percentage area overhead. For all designs, the cut selection algorithm succeeded to find competitive solutions in less than five seconds on a Sun UltraSPARC-II.

The application-specific cores were integrated into a number of system configurations to test the efficiency of our cut-selection and scheduling technique. The results are illustrated in Table III. While in the first column the core mix is specified, the second column presents the common system period. The last two columns present the number of variables of all cuts in the system that have to be transferred in one system period and the range of control steps in which this transfer is accomplished. By comparing the last two columns, it is self-evident that we efficiently utilize the idle control steps in order to transfer cut variables. For all design integrations, the cut selection and scheduling algorithm succeeded to find competitive solutions in less than one second on a Sun UltraSPARC-II.

Last, in order to evaluate the feasibility and overhead of embedding instructions for programmable core cut I/O, we instrumented the code from the MediaBench benchmark suite [16] with instructions that dispense the program cut out of a general-purpose processor. The experimental results are presented using two subtables in Table IV. The first column of each subtable presents the name of the multimedia application. The next

TABLE IV
PSEUDOCODE FOR THE CUT SELECTION AND SCHEDULING ALGORITHM

Application	Variables	cut cardinality	% of Vars for I/O	Application	Variables	cut cardinality	% of Vars for I/O
ADPCM.enc	22	6	27%	PEGWIT	101	14	14%
ADPCM.dec	13	5	38%	PGP	970	33	3.4%
D/A Converter	213	3	1.4%	GSM.enc.dec	140	12	8.6%
G721.enc.dec	28	2	7%	JPEG.enc	513	17	3%
epic/unepic	298	32	11%	MPEG2.dec	432	24	5.5%

two columns quantify the total number of variables in the program and the cardinality of the program cut, respectively. Finally, the last column presents the ratio of cut versus total variables. For all programs, the cut selection algorithm succeeded to find competitive solutions in less than a minute on a Sun UltraSPARC-II. Importantly, as a proof of concept, we outline the fact that on the average only 10.8% of all variables in the targeted benchmark suite were found to constitute cuts, and therefore, present a direct transfer cost.

VII. CONCLUSION

This paper has introduced the first approach to functional verification of statically or dynamically scheduled programmable SOC's that coordinates design emulation and simulation. We have established the complexity of all optimization tasks and developed efficient heuristics to provide competitive solutions. The effectiveness of the new approach and accompanying algorithms has been demonstrated on a set of programmable and application-specific multicore designs where full system observability and controllability have been enabled with low hardware and performance overhead.

REFERENCES

- [1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*. Reading, MA: Addison-Wesley, 1983.
- [2] A. A. Bertossi, M. Bonometto, and L. V. Mancini, "Increasing processor utilization in hard-real-time systems with checkpoints," *Real-Time Syst.*, vol. 9, no. 1, pp. 5–29, 1995.
- [3] J. Cocke, R. L. Malm, and J. J. Shedletsky, "Logic simulation machine," U.S. Patent 430 628 6, 1981.
- [4] A. J. Colmenarez and T. S. Huang, "Pattern detection with information-based maximum discrimination and error bootstrapping," in *Proc. Int. Conf. Pattern Recognition*, 1998, pp. 222–224.
- [5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.
- [6] R. E. Crochiere and A. V. Oppenheim, "Analysis of linear networks," *Proc. IEEE*, vol. 63, no. 4, pp. 581–595, 1975.
- [7] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of embedded systems: Formal models, validation, and synthesis," *Proc. IEEE*, vol. 85, no. 3, pp. 366–390, 1997.
- [8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.
- [9] D. Genin, P. Hilfinger, J. Rabaey, and C. Scheers *et al.*, "DSP specification using the silage language," in *Proc. Int. Conf. Acoustics, Speech and Signal Processing*, vol. 2, 1990, pp. 1056–1060.
- [10] A. H. Gray and J. D. Markel, "Digital lattice and ladder filter synthesis," *Trans. Audio Electroacoust.*, vol. 21, no. 6, pp. 491–500, 1973.
- [11] R. Hastings and B. Joyce, "Purify: Fast detection of memory leaks and access errors," *USENIX*, pp. 125–136, 1992.
- [12] P. Keller and R. Eads, "Integration requires SOS imagination," *EETimes*, no. 973, p. 102, Sept. 1997.
- [13] D. Kirovski, M. Potkonjak, and L. M. Guerra, "Improving the observability and controllability of datapaths for emulation-based debugging," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 1529–1541, Nov. 1999.
- [14] H. Kuijsten, "Method and apparatus for a trace buffer in an emulation system," U.S. Patent 568 058 3, 1997.
- [15] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. C-36, no. 1, pp. 24–35, 1987.
- [16] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems," *IEEE Micro*, vol. 30, 1997.
- [17] D. L. Liu, J.-T. Li, T. B. Huang, and K. S. K. Choi, "Method and apparatus for debugging reconfigurable emulation systems," U.S. Patent 542 503 6, 1995.
- [18] S. T. Mangelsdorf *et al.*, "Functional Verification of the HP PA 8000 Processor," *Hewlett Packard J.*, Aug. 1997.
- [19] J. Marantz, "Enhanced visibility and performance in functional verification by reconstruction," in *Proc. Design Automation Conf.*, 1998.
- [20] C. Maunder, "JTAG, the joint test action group," in *Proc. Inst. Elect. Eng. Colloquium New Ideas in Testing*, 1986, pp. 6/1–6/4.
- [21] Y. Morley, "Software emulation system with dynamic translation of emulated instructions for increased processing speed," U.S. Patent 575 198 2, 1998.
- [22] C. Patel, "Method and apparatus to emulate VLSI circuits within a logic simulator," U.S. Patent 554 656 562, 1996.
- [23] M. Poret and J. McKinley, "In-circuit emulator," U.S. Patent 467 408 9, 1987.
- [24] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast prototyping of datapath-intensive architectures," *Design Test Comput. Mag.*, vol. 8, no. 2, pp. 40–51, 1991.
- [25] S. P. Sample, M. R. D'Amour, and T. S. Payne, "Apparatus for emulation of electronic hardware system," U.S. Patent 510 935 3, 1992.
- [26] M. D. Smith, "Tracing with pixie," Stanford Univ., Stanford, CA, Tech. Rep. CSL-TR-91-497, Nov. 1991.
- [27] J. Tsai, S.-Y. Kuo, and Y.-M. Wang, "Theoretical analysis for communication-induced checkpointing protocols with rollback-dependency trackability," *IEEE Trans. Parallel Distribut. Syst.*, vol. 9, no. 10, pp. 963–71, 1998.
- [28] Y.-M. Wang, "Consistent global checkpoints that contain a given set of local checkpoints," *IEEE Trans. Comput.*, vol. 46, pp. 456–68, 1997.
- [29] L. Yang *et al.*, "System design methodology of UltraSPARC-I," in *Design Automation Conf.*, 1995, pp. 7–12.
- [30] A. Yu, "The future of microprocessors," *IEEE Micro*, vol. 16, no. 6, pp. 46–53, 1996.
- [31] V. Zivojnovic and H. Meyr, "Compiled HW/SW co-simulation," in *Design Automation Conf.*, 1996, pp. 690–695.

Darko Kirovski received the M.Sc. degree from the University of California, Los Angeles, in 1997. He is currently working toward the Ph.D. degree at the University of California, Los Angeles.

His interests include various aspects of design and debugging of systems-on-chip such as combine simulation and emulation, engineering change, intellectual property protection, content protection systems, etc. He is collaborating with Microsoft Research, Redmond, WA, and Conexant Systems, Newport Beach, CA.

Mr. Kirovski received a Microsoft Research Graduate Fellowship and a DAC Graduate Fellowship.

Miodrag Potkonjak received the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1991.

He has been with the Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles (UCLA), since 1995, where he is an Associate Professor. In 1991, he joined C&C Research Laboratories, NEC USA, Princeton, NJ. He has published more than 150 papers in leading CAD, real-time, and signal-processing journals and conferences. He has received five patents. His research interests include system design, embedded systems, computational security, and intellectual property protection.

Dr. Potkonjak received an Okawa Foundation Grant, an NSF CAREER award, and a number of best paper awards. He also received the TRW/School of Engineering and Applied Science at UCLA Excellence in Teaching Award in 1998.

Lisa M. Guerra received the B.S. degree in electrical engineering from Stanford University, Stanford, CA, in 1990 and the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1996.

She was an AT&T Bell Labs and Office of Naval Research Scholar. She worked for several years on system-on-chip verification at Conexant Systems, Newport Beach, CA, and is currently with Bevoval, Mountain View, CA.