[7] F. Dartu and L. Pileggi, “Calculating worse-case gate delays due to dominant capacitance coupling,” in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 46–51.
[8] C. Ebeling and B. Lockyear, “On the performance of level-clocked circuits,” *Adv. Res. VLSI*, pp. 242–356, 1995.
[9] P. Gross, R. Arunachalam, K. Rajagopal, and L. Pileggi, “Determination of worst-case aggressor alignment for delay calculation,” in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1998, pp. 212–219.
[10] S. Hassoun, “Critical path analysis using a dynamically bounded delay model,” in *Proc. ACM-IEEE Design Automation Conf.*, 2000, pp. 260–265.
[11] A. Ishii, C. Leiserson, and M. Papaefthymiou, “Optimizing two-phase, level-clocked circuitry,” in *Proc. Brown/MIT Conf.: Adv. Res. VLSI Parallel Syst.*, 1992, pp. 245–264.
[12] N. Jouppi, “Timing analysis for nMOS VLSI,” in *Proc. 20th Design Automation Conf.*, 1983, pp. 411–418.
[13] Y. Kukimoto, M. Berkelaar, and K. Sakallah, “Static timing analysis,” in *Logic Synthesis and Verification*, S. Hassoun and T. Sasao, Eds. Norwell, MA: Kluwer, 2002.
[14] B. Lockyear, “Algorithms for Retiming Level-Clocked Circuits and their use in Increasing Circuit Robustness,” Ph.D. dissertation, Univ. Washington, Seattle, WA, 1994.
[15] J. Ousterhout, “Switch-level delay models for digital MOS VLSI,” in *Proc. IEEE 21st Design Automation Conf.*, 1984, pp. 542–548.
[16] K. Sakallah, T. Mudge, and O. Olukotun, “Analysis and design of latch-controlled synchronous circuit,” in *Proc. 27th ACM-IEEE Design Automation Conf.*, 1990, pp. 111–117.
[17] ——, “Analysis and design of latch-controlled synchronous digital circuits,” *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 322–333, Mar. 1992.
[18] S. Sapatnekar, “A timing model incorporating the effect of crosstalk on delay and its application to optimal channel routing,” *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 550–559, May 2000.
[19] S. Sapatnekar and R. Deokar, “Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits,” *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 1237–1248, Oct. 1996.
[20] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, “SIS: A System for Sequential Circuit Synthesis,” Univ. California, Dept. Elect. Eng. and Comput. Sci., Berkeley, CA, UCB/ERL M92/41, 1992.
[21] K. Shepard and V. Narayanan, “Noise in deep submicron digital design,” in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1996, pp. 524–531.
[22] S. Sirichotiyakul, D. Blaauw, C. Oh, R. Levy, V. Zolotov, and J. Zuo, “Driver modeling and alignment for worst-case delay noise,” in *Proc. ACM/IEEE Design Automation Conf.*, 2001, pp. 720–725.
[23] T. Szymanski and N. Shenoy, “Verifying clock schedules,” in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 124–131.
[24] T. G. Szymanski, “LEADOUT: A static timing analyzer for MOS circuits,” in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1986, pp. 130–133.
[25] T. Xiao and M. Marek-Sadowska, “Functional correlation analysis in crosstalk induced critical paths identification,” in *Proc. ACM/IEEE Design Automation Conf.*, 2001, pp. 653–656.
[26] H. Zhou, N. Shenoy, and W. Nicholls, “Timing analysis with crosstalk as fixpoints on complete lattice,” in *Proc. ACM/IEEE Design Automation Conf.*, 2001, pp. 714–719.

Local Watermarks: Methodology and Application to Behavioral Synthesis

Darko Kirovski and Miodrag Potkonjak

***Abstract*—Recently, the electronic design automation industry has adopted the intellectual property (IP) business model as a dominant system-on-chip development platform. Since copyright fraud has been recognized as the most devastating obstruction to this model, a number of techniques for IP protection have been introduced. Most of them rely on a selection of a global solution to a design optimization problem according to a unique user-specific digital signature. Although such techniques provide strong proof of authorship, they fail to provide an effective procedure for watermark detection when a protected core design is augmented into a larger design. To address this fundamental issue, we introduce local watermarks, an IP protection technique which facilitates watermark detection in many realistic design and adversarial scenarios, while satisfying the demand for low overhead and design transparency. We demonstrate the efficiency of the new IP protection paradigm by applying its principles to a set of behavioral synthesis tasks such as operation scheduling and template matching.**

***Index Terms*—Behavioral synthesis, intellectual property protection, operation scheduling, template matching, watermarking.**

I. Introduction

Recently, a number of techniques have been proposed for intellectual property protection (IPP) of designs and tools at various design levels: design partitioning [1], physical layout [2], combinational logic synthesis [3], [4], behavioral synthesis [5], and design-for-test [6]. All of these techniques encode a user’s digital signature as a set of additional design constraints, augment these constraints into the original design specification, and optimize this input specification using an off-the-shelf design tool that retrieves the final optimized design specification. The solution produced by the optimization tool satisfies both the original and user-specific constraints. This property is the key to enabling a low likelihood that another algorithm (or designer) can build such a solution with only the original design specifications as a starting point. Although efficient, these techniques lack support for several important requirements.

- **Effective signature detection.** Since the encoding of a digital signature is dependent upon the structure of the *entire* design specification, detecting an embedded signature requires unique identification of each component of the design [3]. Thus, even a small design alteration by the adversary may negligibly, but significantly alter the identifiers of design components resulting in ineffective watermark detection.
- **Protection of design partitions.** Although current IPP techniques are effective in protecting overall designs, they do not provide protection for design partitions. Namely, in many designs (cores), their parts may have substantial and independent value (for example, a discrete cosign transform filter in an MPEG codec).
- **Watermark detection in systems with embedded IP.** Commonly, a misappropriated design is augmented into a larger system. In order to detect design’s watermark in the suspected

Manuscript received February 8, 2001; revised February 18, 2002 and August 18, 2002. This paper was recommended by Associate Editor R. Gupta.

D. Kirovski is with Microsoft Research, Redmond, WA 98052 USA (e-mail: darkok@microsoft.com).
M. Potkonjak is with the Computer Science Department, University of California, Los Angeles, CA 90095 USA (e-mail: miodrag@cs.ucla.edu).

Digital Object Identifier 10.1109/TCAD.2003.816208

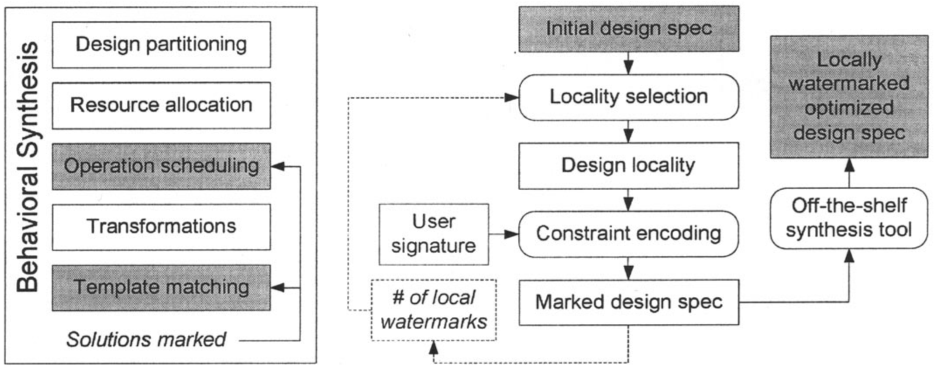


Fig. 1. Global flow of the generic approach for local watermarking behavioral synthesis solutions.

system, existing IPP techniques require an accurate design extraction from the system, a task of substantial computational difficulty in the case when the system is optimized for integration.

We introduce **local watermarks**, a generic IPP technique that hides statistically imperceptible secrets in solutions to numerous combinatorial optimization problems, while providing the aforementioned protection requirements. As in the previous IPP techniques, a local watermark is encoded as a set of design constraints which does not exist in the original specifications. The constraints are uniquely dependent upon the author's signature. Rather than embedding a single error-corrected watermark over the entire design, as in the previous techniques, in local watermarking, a number of "small" watermarks are randomly augmented in the design. "Small," in a sense that the constraints of each watermark are placed in a smaller part (locality) of the design. Each watermark exists and can be detected in its locality independently upon the remainder of the design. Therefore, such watermarks enable protection for parts of the design because the copy detection algorithm does not need to see the entire design in order to decode the added constraints.

Effective detection of watermarks is enabled for two reasons fundamental for design IPP. First, as opposed to global design watermarking techniques, in order to obliterate the copyright protection, an adversary needs to significantly alter the entire design to prevent detection of all augmented local watermarks. Second, design partitions as small as the locality of a watermark are protected and can be identified as embedded in another design.

We have applied the generic IPP methodology of local watermarks on two behavioral synthesis tasks: template matching and operation scheduling. In order to determine the efficacy of developed protection techniques, we have quantified the strength of proof of ownership as well as the overhead induced by local watermarks. The added constraints may result in a synthesis tradeoff. The more constraints, the stronger the proof of authorship, but the higher the overhead on the solution quality.

Watermarking designs at the behavioral synthesis level enables IP commerce of optimized behavioral specifications and register transfer level designs, which is exceptionally important for application-specific systems. Local watermarks can also protect behavioral synthesis tools and designs at levels of abstraction equal or lower than behavioral synthesis. This property is becoming increasingly important because of the progress of reverse engineering technologies (e.g., Take Apart Everything Under The Sun Co., Colorado Springs, CO [7]) which enable precise, fast, and confidential retrieval of a design netlist. In essence, a behavioral specification created by a designer is protected by adding a set of local watermarks to the optimized behavioral specification. The local watermarks remain in the design in the subsequent phases of the design process. Similarly, unlicensed usage of a behavioral synthesis tool or a direct copy of an implementation of a synthesis algorithm into another tool can be detected in the case when the tool/algorithms embed watermarks in their solutions.

II. Preliminaries

With no loss of generality, we selected as our computational model synchronous data flow (SDF) [8]. SDF is a special case of a data flow in which the number of data samples produced or consumed by each node on each invocation is specified *a priori*. Nodes can be scheduled statically at compile time onto application-specific integrated circuit or programmable processors. We restrict our attention to the homogeneous SDF, where each node consumes and produces exactly one sample on every execution. This model is well suited for specification of single task computations in numerous application domains such as a DSP and communications. The syntax of a targeted computation is defined as a hierarchical control-data flow graph (CDFG) [9]. A CDFG represents the computation as a flow graph, with nodes, data edges, and control edges. The semantics underlying the syntax of the CDFG format is that of the SDF flow model.

The are two crucial questions related to localized watermark detection: 1) how to recover the specification of the suspected design from its implementation and 2) how to recover a CDFG from that specification (not necessarily at the behavioral level). There are a number of references [10]–[12] that document a number of standard techniques to reverse engineer even highly complex integrated circuits (ICs). Once the specification is available, one can easily recover its finite state machine (FSM) and, thus, the schedule and assignments used in the IC. Even when the FSM is distributed and uses local decoders, it is straightforward enough to determine which paths are used in the datapath in a particular control step by observing control signals to multiplexers and other control logic associated with datapath components. Note that this is true even if a design is programmable [13].

III. Paradigm of Local Watermarks as an IPP Tool

The generic approach for protecting solutions to behavioral synthesis using local watermarks is shown in Fig. 1. Watermarking of an original behavioral specification is performed in a synthesis preprocessing step. In that step, the specification is augmented with one or several sets of pseudorandomized design constraints which encode the author's signature. Each set is attached to a particular pseudorandomly selected locality within the design specification. For example, while uniquely marking a solution to graph coloring, a local watermark is embedded in a random subgraph.

After the algorithm retrieves a solution to the given problem, the added constraints are removed from the optimized design specification, producing a design which satisfies both user-specific and original constraints. The likelihood that another algorithm applied to the original nonconstrained design specification retrieves a solution which accidentally satisfies also the user-specific constraints (solution coincidence P_c) has to be small in order to have strong proof of authorship $(1 - P_c)$.

During copy detection, the goal is to find at least one local watermark in a particular design. Since each watermark has the property of being detectable within its own locality, the attacker is not safe even if the misappropriated solution is embedded or cut. Hence, a local watermark has to be all of the following:

- hardly recognizable in a given design;
- difficult to find in a nonconstrained design;
- hard to remove by a finite set of solution transformations;
- easy to detect using exhaustive search and knowing the structure of each local watermark.

IV. IPP Protocols for Behavioral Synthesis

In this section, we describe the technical details behind the data hiding, watermark detection, and attacking processes for the developed *local watermarking* techniques for two behavioral synthesis tasks—operation scheduling and template matching.

A. Operation Scheduling

Scheduling is the process of partitioning the set of operations in the CDFG into groups such that the operations in the same group can be executed concurrently in one control step, while taking into consideration possible tradeoffs between total execution time and hardware cost. Scheduling determines the total number of control steps needed to execute all operations in the CDFG, the minimum number of functional modules for design, and the lifetimes of variables. For scheduling, there are two basic approaches—heuristics [14] and integer linear programming (ILP) [15].

The key steps in the local watermarking protocol for operation scheduling are

- **Domain selection,** i.e., selection of a subtree $T \in \mathrm{CDFG}$ which represents the domain where a watermark is embedded;
- **Domain identification,** i.e., assignment of a unique identifier to each operation in T;
- **Constraint encoding,** i.e., creation and addition of user-specific constraints (temporal edges) which, when added to T, enforce additional temporal dependencies among operations that do not exist in the original specification.

Domain Selection and Identification: A simple but ineffective way to perform domain selection is to randomly select a node $n \in \mathrm{CDFG}$ and then select its fanin tree of distance D as the resulting T. In order to strengthen the difficulty of tampering an existing watermark or finding an arbitrary watermark in a solution, we select T of desired cardinality $\tau = |T|$ in the following way. First, we randomly select a node $n_o \in \mathrm{CDFG}$ and identify a subtree T_o as a fanin tree of n_o with max-distance τ from n_o. Next, we assign a unique identifier to each node in T_o using a node ordering routine that sorts nodes in T_o based on an ordered list of sorting criteria. We propose three criteria for sorting nodes in a CDFG.[1] Relation $n_i > n_j$ holds if:

$C1$ $L_i > L_j$, where node n_i, has a level L_i if the longest path in the CDFG from n_o to n_i equals L_i.

$C2$ $K_i(x) > K_j(x)$, where $K_i(x)$ quantifies the number of nodes in the transitive fanin tree of n_i that contains all nodes with maximal distance D_x from n_i.

C3 $\phi(n_i, x) > \phi(n_j, x)$, where $\phi(n_i) = \sum_{\forall n_a \in T_i(x)} f(n_a)$, and $f(n_a)$ returns the unique identifier for the functionality performed by node n_a, in the fanin tree $T_i(x)$ of n_i that consists of all nodes with maximal distance D_x from n_i. All possible distinct operations are uniquely identified (e.g., addition is identified with 1, multiplication with 2, etc.).

[1]An example of similar ordering for netlists is given in [3].

The sorting procedure first determines the order of two nodes n_i and n_j according to the first criterion. If $L_i = L_j$, then the second criterion is consulted, etc. Criteria C2 and C3 are tried for increasing values of D_x until all nodes in the subtree are uniquely identified.

Once all nodes are uniquely identified, subtree $T \in T_o$ is selected using an author-specific pseudorandom sequence of bits. For example, the sequence can be generated using the RC4 stream cipher by iteratively encrypting a certain standard seed number keyed with the author's digital signature D[16]. In order to determine T, the watermarking procedure traverses the subtree T_o in a top-down (in reverse direction of edges) breadth-first fashion. At each traversed node, the author-unique bit sequence determines 1) at least one input to include in the next level of breadth-first search and 2) whether each of the remaining inputs should be included or excluded from the list of succeeding nodes to be visited during the breadth-first search. In general, the exclusion of inputs can be done with a given probability. The selection process cannot be misinterpreted because of the unique identification of each node input.

Constraint Encoding: The pseudocode for constraint encoding is presented in Fig. 2. In this step, the selected subtree T is augmented with edges which indicate temporal dependencies between operations. Such edges are standard nomenclatures for behavioral descriptions (e.g., HYPER [9]). A temporal edge enforces that its source operation is scheduled before its destination operation. The temporal edges are augmented according to the author's digital signature on a subset T' of nodes of the subtree T. For each node $n_i \in T'$, there exists at least one more node $n_j \in T'$ with an overlapping scheduling period, i.e., $\mathrm{asap}(n_j) + 1 > \mathrm{alap}(n_i)$ or $\mathrm{asap}(n_i) + 1 < \mathrm{alap}(n_j)$. Functions $\mathrm{alap}(\cdot)$ and $\mathrm{asap}(\cdot)$ return the "as late as possible" and "as soon as possible" control steps, respectively, for the argument operation.

In addition, each node $n_i \in T'$ must have a laxity of $C \cdot (1 - \epsilon)$, where C is the length of the CDFG's critical path and $\epsilon > 0$ is a user-specified parameter. A node n_i has a *laxity* of x if the longest path that contains n_i traverses the CDFG and has a length of x. The restriction with respect to the node's overlapping "asap-alap" lifetimes and laxity is imposed to avoid significant timing overhead and to increase the scheduling freedom for the operations in the domain which results in strengthened authorship proof. If cardinality T' is less than some predetermined τ', the entire process of subtree selection is repeated.

Temporal edges are added to an ordered set of nodes T'' in the following way. The author-specific pseudorandomly generated bitstream is used to identify a pseudorandomly ordered selection $T'' \in T'$ of K of nodes from T', where K is a user-defined parameter. Different authors would have additional constraints imposed on two different sets of K-node selections. In the order of appearance, for each node $n_i \in T''$, we identify a set $g \in T''$ of nodes where each node $n_j \in g$ has overlapping life periods with n_i, $\mathrm{asap}(n_i) + 1 > \mathrm{alap}(n_j)$, or $\mathrm{asap}(n_j) + 1 < \mathrm{alap}(n_i)$. Using the author-specific bitstream, the watermarking procedure selects one node n_k from g and draws a temporal edge e_i between $e_i(n_i \to n_k)$. The watermarking process is terminated when all K temporal edges are drawn. The marking process

```
Given a $CDFG = \{N, E\}$ with a set of nodes $N$, a set of directed edges $E$,
a central node $n_o$ and a subtree $T_o$ that encompasses all nodes with max-distance $\tau$ from $n_o$.
1 Determine the critical path $C$ of the CDFG. $T' = \emptyset$.
2 For each node $n_i \in T$
3   If $\text{laxity}(n_i) > |C|(1-\epsilon)$ and $(\exists n_j \in T_o | \text{asap}(n_j) + 1 < \text{alap}(n_i)$ or $\text{asap}(n_i) + 1 < \text{alap}(n_j))$
4     Add $n_i$ to $T'$.
5 Using the author-specific bit sequence pseudo-randomly select an ordered selection $T''$ of $K$ nodes from $T'$.
6 For each $n_i \in T''$
7   Find a subset of nodes $g \in T'' | \forall n_j \in g, j > i$ and $(\text{asap}(n_j) + 1 < \text{alap}(n_i)$ or $\text{asap}(n_i) + 1 < \text{alap}(n_j))$.
8   Using the author-specific bit sequence pseudo-randomly select a node $n_k \in g$.
9   Draw a temporal edge between $e_i(n_i \rightarrow n_k)$.
```

Fig. 2. Pseudocode of the proposed protocol for constraint encoding of local watermarks for marking solutions to operation scheduling.

is transparent with respect to the design tool. After constraint addition, the designer runs the scheduler to determine an optimized scheduling which satisfies the constraints imposed by the original design specification as well as the additional constraints. During the detection process, the marking process is repeated with a modification that constraints are only verified. The detection procedure visits each node in the *CDFG* and checks whether it represents a root n_o of the memorized subtree T and whether the scheduling of nodes in T corresponds to the additional constraints that a marked solution would have.

Discussion: The key to the efficiency of such watermarking approach lies in the following three facts.

First, for each CDFG, the approximate likelihood of coincidence of finding an arbitrary watermark in a solution is approximately equal to $P_c \approx \prod_{i=1}^{K} \psi_W(e_i)/\psi_N(e_i)$, where functions $\psi_W(e_i)$ and $\psi_N(e_i)$ for an edge $e_i(n_s \rightarrow n_d)$ return the number of possible different schedules in which operation n_s is scheduled after n_d with and without the added temporal edges, respectively. According to published results, we have assumed the Poisson distribution of the operation's "asap-alap" times as well as that second order effects have negligible influence on the actual scheduling probabilities [14]. Obviously, for large selected subtrees, the strength of the approximate proof of authorship $\{1 - P_c\}$ can be made very strong. A small example of determining P_c for an exemplary design is presented in Fig. 3. For example, two operations $O[i]$ and $O[j]$ can be scheduled in 77 different ways. However, there are only ten possible schedulings how $O[j]$ can be scheduled before $O[i]$. For an edge $e(O[i] \rightarrow O[j])$, the corresponding cardinalities are $\psi_W(e) = 10$ and $\psi_N(e) = 77$. Since the exhaustive enumeration of solutions, in general, results in exponential runtimes, we have used a trivial exhaustive enumeration technique to calculate these probabilities only for small examples.

Second, the technique has solid resistance against tampering. The attacker may try to modify the output locally in such a way that the watermark disappears or the proof of authorship is lowered below a predetermined standard. Thus, the watermarking scheme has to be such that, to delete the watermark and still preserve solution quality, the attacker has to perturb a great deal of the obtained solution, forcing him/her to repeat the design process. For example, consider a design that has a total of 100 000 operations which satisfy the laxity requirement with 100 additional temporal edges imposed for watermarking purposes. Consider that the attacker aims to reduce the likelihood of authorship by doing local changes to the design. To reduce the proof of authorship to one in a million, under the assumption of average $E[\psi_W(e_i)/\psi_N(e_i)] = 1/2$, the attacker has to alter the execution order of at least 31 729 pairs of nodes, i.e., alter 63% of the final solution.

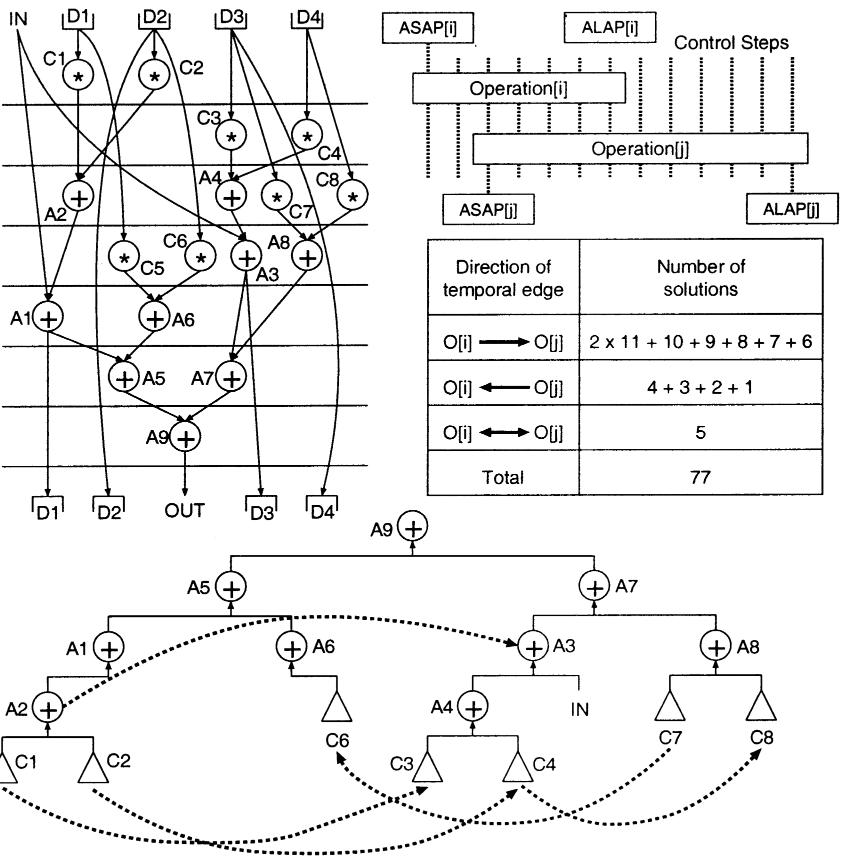
Third, the one-way property of the pseudorandom bitstream generator prohibits the attacker to locally modify the design in order to augment her/his signature. Namely, this finite set of modifications would require the knowledge of the inverse to the bitstream generator. Such an easy-to-compute inverse function is not known for several encryption schemes including RC4 [16].

Motivational Example: We demonstrate the developed protocol for local watermarking of scheduling solutions using a simple example—fourth order parallel IIR filter. The unscheduled *CDFG* for this filter structure is illustrated in Fig. 4. An example subtree $T \in \text{CDFG}$ is presented at the bottom of Fig. 3. Assuming that $T' = T$, the ordered set of temporal edge sources is *C*1, *C*2, *C*4, *C*7, *A*2, and the set of destination nodes is *C*3, *C*4, *C*8, *C*6, *A*3. For example, the temporal edge $e_1(C1 \rightarrow C3)$ imposes that operation *C*1 should be executed before *C*3. The total number of scheduling solutions of the original T subtree is 166, while only 15 solutions can be obtained when the additional constraints are imposed. Thus, for this small example, the likelihood of solution coincidence is equal to $P_c = 15/166$. Obviously, P_c is in exponential correspondence with respect to the *CDFG* cardinalities. An example of a final solution to the additionally constrained scheduling problem is shown in the upper left corner of Fig. 3.

B. Template Matching

In template mapping at the behavioral level, groups of primitive operations are replaced with more complex and specialized hardware units which are designed to implement common operations and are optimized for low area, power, or delay [17]–[19]. The template mapping step involves template matching, template selection, and clock selection. In this paper, we address only the problem of local watermarking template matching solutions.

A protocol for *global* watermarking of a variant of such a problem has been introduced by Kirovski *et al.* [3]. Their constraint encoding technique assigns a signature-specific subset of circuit's internal nodes to become pseudoprimary outputs (PPOs), thus inducing the optimization algorithm to preserve these nodes as visible in the technology mapping solution. We introduce a novel approach for constraint encoding of mapping problems. The key idea guiding the new constraint encoding protocol is enforcement of node-to-module matching by constraint manipulation in accordance with the user's digital signature. Particular matchings are enforced to appear by assigning the nodes neighboring the matched module to become PPOs. In the remainder of this section, the watermarking protocol is explained in more detail and the approach is demonstrated using an explanatory example.



Direction of temporal edge	Number of solutions
O[i] ⟶ O[j]	2 x 11 + 10 + 9 + 8 + 7 + 6
O[i] ⟵ O[j]	4 + 3 + 2 + 1
O[i] ⟷ O[j]	5
Total	77

Fig. 3. Example of local watermarking scheduling solutions: fourth order parallel IIR filter.

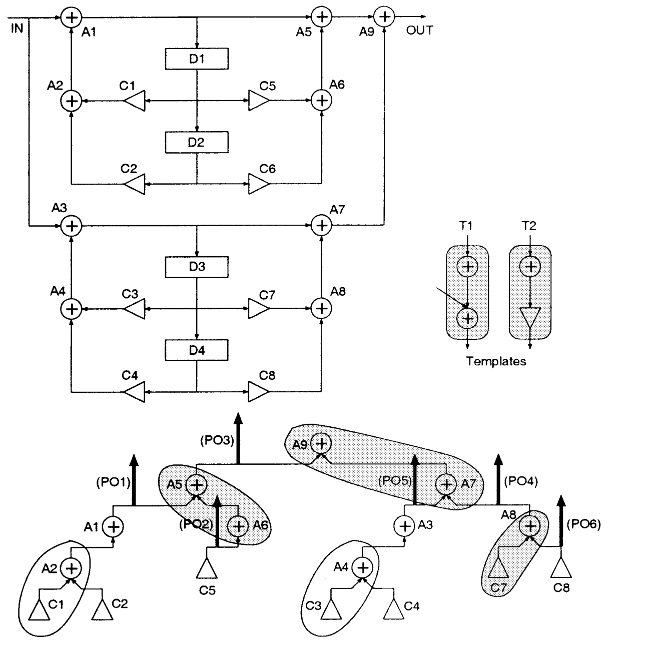


Fig. 4. Example of local watermarking template matching solutions: fourth order parallel IIR filter.

The protocol for local watermarking of template matching solutions relies on the process of identifying a uniquely enumerated subtree T of the original *CDFG* which is augmented with user-specific constraints. The process of identifying the subtree T is signature dependent, i.e., for a single signature only one subtree T, starting from a random root node n_o in the *CDFG* can be determined. We have adopted the same sequence of steps for DOMAIN SELECTION AND IDENTIFICATION as implemented in the equivalent protocol for watermarking operation scheduling solutions. Therefore, we introduce only the constraint encoding protocol formally presented using the pseudocode in Fig. 5.

Constraint Encoding: The constraint encoding procedure imposes additional constraints on the selected subtree T with the goal to isolate particular groups of operations that can be matched to the templates available from a given library. An example of such matching is illustrated in Fig. 4. In order to isolate the two-adder template matched with *A*5 and *A*6, variables *PO*1, *PO*2, and *PO*3 in the neighborhood are assigned to become PPOs. Since one of the inputs to *A*6 is a primary input, it is not additionally constrained. An important fact is that any variable in the *CDFG* (not only the ones in the subtree T) can be a part of the subset of variables that has to be promoted to PPOs. Addition of these constraints may affect the matchings along the critical path resulting in decreased solution quality. Therefore, from the selected subtree T, we exclude all nodes that are on the critical path (of length C operations) or paths of laxity greater than $C \cdot (1 - \epsilon)$ operations, where ϵ is a user-defined parameter. This exclusion creates a new subset of nodes denoted as $T' \in T$.

The encoding procedure embeds the watermark iteratively in a loop which contains two steps.

In the first step, given the subset of nodes T' and a library of modules L, all possible nodes-to-modules matchings are exhaustively enumerated. The complexity of this task is at most $O(\tau' \lambda)$, where λ is the number of modules in the library. A module is defined as a set of operation trees. Each operation in each module is uniquely identified. The result of the enumeration is an ordered list $\mathcal{M}$ of matchings. A matching $m = \{(n \bowtie O)^{|m|}\}$ denotes a set of $|m|$ pairs of nodes n with their corresponding operations O in the module selected for

```
Given a $CDFG$, a uniquely enumerated subtree $T$, and a library of modules $L$.
01 Repeat $Z$ times
02   Compute the critical path $C$ of the $CDFG$.
03   Create $T'$ by removing all nodes from $T$ with laxity greater than $C \cdot (1-\epsilon)$.
04   List of all nodes-to-module matchings $\mathcal{M} = \emptyset$.
05   For each non-processed node $n_i \in T'$
06     For each operation $O_j$ in each module $l_k \in L$ that can be mapped into $n_i$
07       If (find exhaustively whether operations in $l_k$ other than $O_j$ can be mapped to nodes in $T'$)
08         $\mathcal{M} = \mathcal{M} \cup m_k, m_k = \{(n_i \bowtie O_j), \ldots\}$.
09   Using the author-specific bit sequence pseudo-randomly select a matching $m_i$ from $\mathcal{M}$.
10   For each input and output node $n_j$ of the selected matching $m_i$
11     Make $n_j$ a pseudo-primary output (PPO) and mark $n_j$ as processed.
```

Fig. 5. Pseudocode of the proposed protocol for constraint encoding during local watermarking of template matching solutions.

matching. For example, in the explanatory example, operation *A*9 can be matched in five different ways: as first addition in T_1, as second addition in T_1 with no mapping for the first addition, or as *A*5 or *A*7 as first additions, and as an addition in T_2. The enumeration procedure assigns a unique identifier for each matching. This procedure is described using the pseudocode in Fig. 5.

In the second step, the author-specific pseudorandomly generated bitstream is used to point to one of the matchings m_i from $\mathcal{M}$. Then, within the entire *CDFG*, all variables that are used as inputs/outputs to/from the operations covered by the module in m_i are assigned to become PPOs. Next, all operations $n_i \in m_i$ are marked as "processed." In the next iteration, nodes that are marked as "processed" are not considered during enumeration of matchings (steps $4-8$). The constraint encoding loop is repeated Z times where Z is a user-specified parameter with an important tradeoff. The higher the number of iterations, the stronger the proof of authorship. Conversely, more preselected matchings constrain the optimization abilities of the design tool applied after the local watermarking procedure. The security implications of local watermarking applied to template matching are equivalent to the ones described for operation scheduling.

Motivational Example: The presented watermarking protocol is demonstrated using the fourth order parallel IIR filter. The CDFG of the filter and the available library of templates are presented in Fig. 4. The watermarking process has isolated the following matchings {(*A*5, *A*6), (*A*9, *A*7), (*A*8, *C*7)} (shaded in Fig. 4). The nonshaded matchings indicate a possible solution to this instance of the template matching problem.

The likelihood of solution coincidence for this protocol equals the number of nodes in the original *CDFG* from which one can find the subtree T times the number of solutions of quality Q for the watermarked design specification divided by the number of solutions of quality Q for the nonwatermarked (nonconstrained) design specification. For an optimization goal of minimizing the number of modules that cover a given CDFG, a solution of quality Q implicates that the CDFG is covered with Q modules. Since this approach for computing P_c requires explicit enumeration of all possible solutions, which can be exponentially dependent upon the *CDFG* cardinalities, we opt to use an approximate technique for determining P_c: $P_c \approx \prod_{i=1}^{Z} \mathrm{Solutions}(m_i)^{-1}$. Function $\mathrm{Solutions}(m_i)$ returns the number of different matchings for all nodes covered by the enforced template m_i. In the example in Fig. 4, a pair of nodes (*A*5, *A*6) can be covered in the following six ways (see the equation at the bottom of the page).

V. Experimental Results

We have conducted a set of experiments in order to evaluate the efficacy of local watermarking on the operation scheduling and template matching tasks. We have applied local watermarks to operation scheduling using a set of benchmark programs specified in C—the MediaBench set of benchmarks [20]. Temporal edges were induced using additional operations with unit operators (e.g., additions with variables assigned to zero at runtime). Note that in the actual implementation the added instructions must be extracted from binaries for security and performance reasons. All programs were compiled for a four-issue very long instruction word machine with four arithmetic-logic units, two branch and two memory units, and 8-KB cache [21]. The code was compiled for the described machine using the retargetable IMPACT C compiler [22].

The obtained results for operation scheduling are presented in Table I. The first two columns present the name of the application and its number of operations N. For each application, we have augmented local watermarks within a subtree of cardinality $\tau = 10 \cdot \alpha \cdot N$, $a = 0.2$, 0.5. In columns three and five, we present the likelihood of solution coincidence P_c for $K = 0.2 \cdot \tau$. Columns four and six demonstrate the percentage of increase of execution time induced by the augmented code due to watermarks. As presented, all IPP properties enabled by local watermarking were provided with negligible performance overhead.

The results of experiments conducted to test the template matching algorithm are presented in Table II. The local watermarking

$A5$	$A6$	$A5$	$A6$	$A5$	$A6$
$A5, A9$	$A6$	$A5, A9$	$A6, C5$	$A1, A5$	$A6$
$A1, A5$	$A6, C5$	$A1$	$A6, C5$	$A5, A6$	$A5, A6$

TABLE I
EXPERIMENTAL RESULTS DESCRIBING THE EFFICIENCY OF APPLIED LOCAL WATERMARKING PROTOCOLS TO OPERATION SCHEDULING

Application		Operation Scheduling			
Descri-ption	Oper-ations	2% nodes constrd		5% nodes constrd	
		$P_c^{\sim}$	Perf. OH	$P_c^{\sim}$	Perf. OH
D/A Cnv.	528	10^{-26}	0.5%	10^{-53}	1.5%
G721	758	10^{-27}	0.7%	10^{-67}	1.7%
epic	872	10^{-39}	0.6%	10^{-91}	2.4%
PEGWIT	658	10^{-27}	0.2%	10^{-73}	1.1%
PGP	1755	10^{-89}	0.1%	10^{-283}	0.5%
GSM	802	10^{-34}	0.3%	10^{-87}	1.4%
JPEG.c	1422	10^{-65}	0%	10^{-212}	0.2%
MPEG2.d	1372	10^{-58}	0.2%	10^{-185}	0.4%

TABLE II
EXPERIMENTAL RESULTS DESCRIBING THE EFFICIENCY OF APPLIED LOCAL WATERMARKING PROTOCOLS TO TEMPLATE MATCHING

Design				Template Matching	
Description	Available control steps	Critical path	Vari-ables	% mod. enf.	Overhead module count
8th Order	18	18	35	3%	8.2%
CF IIR	36	18	35	3%	3.3%
Linear GE	12	12	48	5%	11.1%
Cntrlr	24	12	48	5%	5%
Wavelet	16	16	31	4%	10%
Filter	32	16	31	4%	3.3%
Modem	10	10	33	5%	8.7%
Filter	20	10	33	5%	2.5%
Volterra	12	12	28	5%	8.7%
2nd ord.	12	24	28	5%	6%
Volterra	20	20	50	3%	9%
3rd non-lin.	20	40	50	3%	5.2%
D/A	132	132	354	4%	3%
Converter	132	264	354	4%	0.4%
Long Echo	2566	2566	1082	2%	1%
Canceler	5132	2566	1082	2%	0.1%

techniques for template matching were tested on a set of small real-life designs [9]. We used HYPER as a behavioral synthesis tool [9]. Columns 1–4 present the design's description, number of available control steps, critical path, and number of variables. Column 5 quantifies the percentage β of templates that were enforced $Z = 0.07 \cdot \tau$, $T = \mathrm{CDFG}$. Finally, column 6 presents the percentage of increase of the count of used modules to cover the entire design with respect to two design strategies: nonwatermarked and watermarked. Knowing the simplicity of target benchmark designs, the order of the likelihood of design coincidence ranged from 10^{-5} to 10^{-27} for all specified designs. Therefore, note that in both the operation scheduling and template matching task, local watermarking has been applied as an effective IPP methodology providing partial protection with low overhead and high confidence and reliability.

VI. CONCLUSION

We have introduced *local watermarking*, an IPP technique which enables protection of design partitions, provides an easy procedure for watermark detection, and enables detection of watermarks when the misappropriated design or its part is augmented into another larger design. We have applied the new IPP technology to a subset of behavioral synthesis tasks: operation scheduling and template matching. We have demonstrated that the difficulty of erasing author's signature or finding another signature in the synthesized design can arbitrarily be made computationally difficult. The watermarking method has been experimented on a set of benchmarks, where high likelihood of authorship has been achieved with negligible overhead in solution quality.

REFERENCES

[1] A. Kahng, J. Lach, W. Mangione-Smith, S. Mantik, I. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Watermarking techniques for intellectual property protection," in *Proc. Design Automation Conf.*, 1998, pp. 776–781.

[2] A. Kahng, S. Mantik, I. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Robust IP watermarking methodologies for physical design," in *Proc. Design Automation Conf.*, 1998, pp. 782–787.

[3] D. Kirovski, Y.-Y. Hwang, M. Potkonjak, and J. Cong, "Intellectual property protection of combinational logic synthesis solutions," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 194–198.

[4] J. Lach, W. Mangione-Smith, and M. Potkonjak, "Fingerprinting digital circuits on programmable hardware," in *Proc. Workshop Inform. Hiding*, 1998, pp. 16–31.

[5] G. Qu and M. Potkonjak, "Analysis of watermarking techniques for graph coloring problem," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 190–193.

[6] D. Kirovski and M. Potkonjak, "Intellectual property protection using watermarking partial scan chains for sequential logic test generation," in *Proc. High Level Design, Test Verification*, 1998.

[7] (1998). Take Apart Everything Under the Sun, Inc., Colorado Springs, CO. [Online]. Available: http://www.taeus.com

[8] E. Lee and D. Messerschmitt, "Synchronous dataflow," *Proc. IEEE*, vol. 75, pp. 1235–1245, 1987.

[9] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast prototyping of data path intensive architectures," *IEEE Design Test Comput.*, vol. 8, pp. 40–51, June 1991.

[10] R. Anderson and M. Kuhn, "Tamper resistance – A cautionary note," in *Proc. Usenix Workshop Electron. Commerce*, 1996, pp. 1–11.

[11] ——, "Low cost attacks on tamper resistant devices," in *Proc. Security Protocols Workshop*, 1997, pp. 7–9.

[12] J. Kumagai, "Chip detectives [reverse engineering]," *IEEE Spectrum*, vol. 37, pp. 43–48, Nov. 2000.

[13] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory," in *Proc. USENIX Security Symp.*, 1996, pp. 77–89.

[14] P. Paulin and J. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 661–679, June 1989.

[15] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 464–475, Apr. 1991.

[16] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography* Boca Raton, FL, 1997.

[17] M. Corazao, M. Khalaf, L. Guerra, M. Potkonjak, and J. Rabaey, "Performance optimization using template mapping for datapath-intensive high-level synthesis," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 877–888, Aug. 1996.

[18] W. Geurts, F. Catthoor, and H. De Man, "Time constrained allocation and assignment techniques for high throughput signal processing," in *Proc. Design Automation Conf.*, 1992, pp. 124–127.

[19] D. Rao and F. Kurdahi, "Partitioning by regularity extraction," in *Proc. Design Automation Conf.*, 1992, pp. 235–238.

[20] C. Lee, M. Potkonjak, and W. Mangione-Smith, “Mediabench: A tool for evaluating and synthesizing multimedia and communications systems,” in *Proc. Int. Symp. Microarchitect.*, 1997, pp. 330–335.
[21] C. Lee, J. Kin, M. Potkonjak, and W. Mangione-Smith, “Media architecture: General purpose vs. multiple application specific programmable processor,” in *Proc. Deisgn Automation Conf.*, 1998, pp. 321–326.
[22] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W. W. Hwu, “Impact: An architectural framework for multiple-instruction-issue processors,” *Comput. Architect. News*, vol. 19, no. 3, pp. 266–275, 1991.

A Simulation Framework for Energy-Consumption Analysis of OS-Driven Embedded Applications

T. K. Tan, A. Raghunathan, and N. K. Jha

***Abstract*—Energy consumption has become a major focus in the design of embedded systems (e.g., mobile computing and wireless communication devices). In particular, a shift of emphasis from hardware-oriented low-energy design techniques to energy-efficient embedded software design has occurred progressively in the past few years. To that end, various techniques have been developed for the design of energy-efficient embedded software. In operating system (OS)-driven embedded systems, the OS has a significant impact on the system's energy consumption directly (energy consumption associated with the execution of the OS functions and services), as well as indirectly (interaction of the OS with the application software).**

As a first step toward designing energy-efficient OS-based embedded systems, it is important to analyze the energy consumption of embedded software by taking the OS energy characteristics into account. To facilitate such studies, we present, in this work, an energy simulation framework that can be used to analyze the energy consumption characteristics of an embedded system featuring the embedded Linux OS running on the StrongARM processor. The framework allows software designers to study the energy consumption of the system software in relation to the application software, identify the energy hot spots, and perform design changes based on the knowledge of the OS energy consumption characteristics as well as application-OS interactions.

***Index Terms*—Embedded system, energy analysis, energy simulation, operating system.**

I. Introduction

The complexity of embedded system software and the underlying hardware, tight performance and power budgets, and aggressive time-to-market schedules, usually necessitate the use of sophisticated runtime software support. In embedded systems where a single processor executes many different system tasks (each system task may be further divided into communicating processes), the use of an embedded operating system (OS) for runtime execution support is quite common. The use of an embedded OS significantly impacts both performance and energy consumption. Representative investigations of performance issues related to the use of an OS can be found in [1]–[4]. However, in modern microprocessors, where idle or sleep modes are usually exploited by software, performance optimization does not necessarily lead to energy optimization. Not much literature deals with energy consumption issues related to an embedded OS. The effect of a real-time OS on the energy consumption of embedded software was illustrated in [5] using $\mu\mathrm{C}$/OS and SPARClite processor, where it was shown that the OS could consume a significant portion of the total energy consumption. However, this was just a first step. Much remains to be done to fully appreciate and model the impact of an embedded OS on software energy consumption.

In this paper, we present an energy simulation framework that enables simulation of an embedded Linux OS on a typical hardware platform based on the Intel StrongARM processor. We have also made this energy simulation framework available for public use (please visit http://www.ee.princeton.edu/~tktan/emsim to download our simulator, EMSIM). As described in Section III, such a simulation imposes special requirements in terms of modeling the functionalities of the hardware platform. Function and process-based energy accounting has been implemented to provide energy analysis capability, enabling users to identify specific energy hot spots in embedded software programs, and better guide code optimizations.

The remainder of the paper is organized as follows. In the next section, we present some previous work and highlight our contributions. Section III presents the design and validation of our simulator. Section IV describes a case study that demonstrates the usefulness of our energy simulation framework. Section V discusses several limitations and design issues pertaining to our simulator. Section VI presents the conclusion and future work.

II. Related Work and Our Contributions

In this section, we discuss related work and highlight our contributions.

A. Related Work

Many instruction-level energy simulators based on the instruction-level power modeling approach have emerged since its introduction in [6]. Applications of the instruction-level modeling technique include [5], [7]–[15], and enhancement of the basic ideas leading to either more accurate or efficient modeling techniques are presented in [7], [16]–[18]. Mehta *et al.* [10] described an energy simulator based on the DLX architecture which they used to evaluate several compiler optimization techniques for low-energy software. Li *et al.* [8] presented an energy simulation framework for an embedded hardware-software system based on Fujitsu's SPARClite processor. Simunic *et al.* [7] enhanced cycle-accurate models of an embedded system based on the StrongARM SA-1100 processor to estimate energy consumption and battery life. Such embedded system simulators could be used to perform design space exploration for low power. *JouleTrack* [11], a web-based energy simulation tool, performs energy estimation using models that separate the energy consumption into a first-order component that depends only on the operating frequency and voltage, and a second-order component that considers instruction statistics. They also presented techniques to separate out the energy consumption due to leakage from the energy consumption due to switching.

A complementary approach to instruction-level power modeling is through the use of structure-based power modeling. In this approach, the power consumption of the microprocessor in each execution cycle is estimated by summing up the power consumption of all the active functional blocks in that cycle. The total energy consumption of a program is obtained by accumulation of processor power consumption over the

Manuscript received March 6, 2002; revised October 17, 2002. This work was supported by DARPA under Contract DAAB07-00-C-L516. This paper was recommended by Associate Editor R. Gupta.

T. K. Tan and N. K. Jha are with the Department of Electrical Engineering, Princeton University, NJ 08544 USA (e-mail: tktan@ee.princeton.edu; jha@ee.princeton.edu).

A. Raghunathan is with NEC Laboratories America, Princeton, NJ 08540 USA (e-mail: anand@nec-labs.com).

Digital Object Identifier 10.1109/TCAD.2003.816207