Short Papers

Static Timing Analysis for Level-Clocked Circuits in the Presence of Crosstalk

Soha Hassoun, Christopher Cromer, and Eduardo Calvillo-Gámez

***Abstract*—Static timing analysis is instrumental in efficiently verifying a design's temporal behavior to ensure correct functionality at the required frequency. This paper addresses static timing analysis in the presence of crosstalk for circuits containing level-sensitive latches, typical in high-performance designs. The paper focuses on two problems. First, coupling in a sequential circuit can occur because of the proximity of a victim's switching input to any *periodic* occurrence of the *aggressor*'s input switching window. This paper shows that only three consecutive periodic occurrences of the aggressor's input switching window must be considered. Second, an arrival time in a sequential circuit is typically computed relative to a specific clock phase. The paper proposes a new phase shift operator to align the aggressor's three relevant switching windows with the victim's input signals. This paper solves the static analysis problem for level-clocked circuits iteratively in polynomial time, and it shows an upper bound on the number of iterations equal to the number of capacitors in the circuit. The contributions of this paper hold for any discrete overlapping coupling model. The experimental results demonstrate that eliminating false coupling allows finding a smaller clock period at which a circuit will run.**

***Index Terms*—Crosstalk, design automation, timing, timing circuits, very large scale integration.**

I. Introduction

Shrinking process geometries have imposed new challenges in both design and verification. One particular problem is the capacitive coupling among two or more signals in the circuit. Coupling exists due to the proximity of a wire to others that are either in the same layer (lateral coupling) or in different layers (interlayer coupling). Coupling creates undesired *noise* and *delay* in the circuit. This phenomenon is commonly referred to as *crosstalk*.

Noise on a signal refers to creating voltage deviation from the nominal supply and ground rails when the signals should otherwise have been stable at a high or low value as dictated by the logic and delay of the circuit [21]. Noise greater than the allowed noise margins causes malfunctions.

Delay variation due to capacitive coupling refers to either speeding or slowing the point in time where a switching net reaches its *receiving* threshold, thus causing receiving gates in the immediate fanouts to switch sooner or later than expected. The delay variation is dependent on the relative arrival times of the *victim* net and the aggressor(s) net(s) that capacitively couple to the victim. If the victim is switching in the same direction as the aggressor(s), then we have *assistive coupling*, and the victim switches sooner than anticipated. Delay improvements could potentially cause race-through or double-clocking conditions, and, thus, circuit failure. With *opposing coupling*, the victim net switches later due to opposing transition on the aggressor(s). Delay degradation causes performance failure; the circuit will not run at the desired frequency. Static timing analysis techniques, which verify a design's temporal behavior to ensure correct functionality at the required frequency, must thus consider the effects of crosstalk.

Several static timing analysis techniques that consider crosstalk have been proposed for combinational circuits. Some are based on iterative techniques [3], [18]; some are based on the propagation of events [5]; others are based on more complex mathematical formulations [10]. The choice of what constitutes coupling (any overlap of the inputs' switching windows v.s. more detailed coupling conditions) affect the complexity of the algorithms. Consideration of the functional correlation of the victim and the aggressors allows further accuracy in analysis [2], [4], [25]. The worst case victim delay can be obtained by driver modeling using reduced order modeling and worst case alignment of the aggressors relative to the victim [7], [9], [22].

This paper addresses crosstalk analysis for circuits with level-sensitive latches. Level-clocked circuits are certainly dominant in high-performance designs because they can operate at faster clock rates than edge-triggered circuits [8]. This is because, unlike edge-triggered registers, latches allow borrowing time across their boundaries. Researchers have efficiently solved the problem of verifying a clock schedule [11], [14], [23]. However, naively assuming worst case crosstalk while running these algorithms yields pessimistic clock periods.

A *clock schedule* specifies the clock period and the relative timing and duration of each of the phases in the schedule. Given a circuit and a clock schedule, we solve the problem of clock schedule verification in the presence of crosstalk. That is, we answer the following question. *Does the circuit run at the specified clock period given the phase waveforms imposed by the clock schedule?*

The difficulty of the clock-schedule verification problem is twofold. First, due to the periodic nature of signals in a sequential circuit, coupling can occur because of the proximity of a victim's switching input to any *periodic* occurrence of the aggressor's input switching window. More than one occurrence of the aggressor waveform must thus must be compared against that of the victim. Second, the arrival times in a level-clocked circuit are typically computed relative to a specific clock phase. Translating the arrival times using a common reference point will be needed to meaningfully compare the switching windows.

This paper addresses both of these problems. We show that only three consecutive switching windows of the aggressor's input must be compared with the victim's input switching window. To determine overlap in switching windows at the inputs of the victim and aggressor, we propose a phase shift operator that can translate values from the aggressor's to the victim's time zones. The paper solves the clock-schedule verification problem in the presence of crosstalk iteratively in polynomial time. Furthermore, it shows an upper bound on the number of iterations equal to the number of capacitors in the circuit.

Several discrete and continuous coupling models are possible for representing the change in delay due to coupling. We choose to use the dynamically bounded delay model [10], an abstract delay model that allows a gate's delay to be assigned one of many values depending on related operating conditions. While more accurate continuous models are possible, e.g., [6], the chosen model is a generalization of discrete coupling models, such as ones that assume a 0 X, 1 X, or 2 X increase in delay, e.g., [18]. While suffering from inaccuracies compared with continuous models, discrete models require less computational complexity. Furthermore, they have proved helpful in understanding the

Manuscript received January 15, 2002; revised August 16, 2002. This work was supported by National Science Foundation POWRE and CAREER grants. This paper was recommended by Associate Editor M. Papaefthymiou.

S. Hassoun and E. Calvillo-Gámez are with the Computer Science Department, Tufts University, Medford, MA 02155 USA (e-mail: soha@cs.tufts.edu).

C. Cromer is with the Infineon Technologies Corporation, San Jose, CA 95512 USA.

Digital Object Identifier 10.1109/TCAD.2003.816209

complex problem of static timing analysis in the presence of crosstalk. Their use in this paper allowed us to achieve an understanding and develop a solution to the coupling problem in level-clocked circuits. The framework and solution proposed here can be easily extended to utilize other discrete coupling models.

The paper is organized as follows. Section II reviews recent advances in timing analysis for combinational circuits in the presence of crosstalk and for level-clocked circuits. Section III introduces the clock-schedule model, the gate-level delay model, and the circuit model. An example is presented in Section IV. Timing equations to model correct circuit operation and coupling conditions are, respectively, derived in Sections V and VI. Then, in Section VII, we present a polynomial algorithm to verify the timing of a level-clocked circuit when given a clock schedule. We conclude with experimental results.

II. Related Work

A. Timing Analysis in the Presence of Crosstalk

Timing analysis techniques for noncyclic combinational circuits are based on traversing an acyclic graph in a time linear in the number of vertices and edges [13]. In the presence of crosstalk, however, such techniques cannot be directly applied because one net can couple to another *anywhere* in the circuit. Mutual dependencies among the signals are created, effectively creating cycles in the underlying timing graph. Iterative techniques have been proposed to solve this problem. An initial solution is first assumed. New solutions are then iteratively computed from previous ones, until the solution converges.

Several researchers have proposed such iterative solutions. Pileggi's group at Carnegie Mellon University model a gate driving an *RC* load as a linear time-varying voltage source in series with a resistance [9]. Their static timing analysis TACO [3] begins by maximizing the switching windows for each signal—the earliest arrival times are set to zero and the latest arrival times are set to infinity. Static timing analysis is then run, computing all arrival times in the circuit assuming worse case alignment of the aggressors. Analyzing the output of this run, some aggressors are found to be nonaligned with the victims. The arrival times for the victims are updated and propagated using a static timing analysis run. The process repeats to tighten the windows until the windows stop shrinking. Sapatnekar also proposes an iterative approach [18]. Whenever switching windows of wires overlap, then the delays are updated. Zhou, Shenoy, and Nicholls establish a theoretical foundation for iterative techniques for timing analysis with crosstalk [26]. They show that different initial solutions lead to different convergent solutions. They also show that the optimal fixpoint (tightest) solution is obtained by starting from the best case solution that assumes no coupling.

B. Verifying Clock Schedules

The biggest challenge in formalizing the verification of clock schedules for level-clocked circuits was creating a general clock-schedule model to reflect borrowing across latch boundaries. Among first-generation timing-analysis tools, such as TA [1], TV [12], Crystal [15], and LEADOUT [24], only the latter correctly verified borrowing across latch boundaries. Second generation timing analysis tools, developed in the early nineties, are based on formalizing the timing constraints and developing efficient algorithms to solve them. Sakallah, Mudge, and Olukoton developed the SMO model [16] which was widely adopted within the timing verification and optimization community. Ishii, Leiserson, and Papaefthymiou also provide a general framework for the timing verification of two-phase level-clocked circuits [11]. Schedule verification algorithms were based on one of two approaches. The Sakallah *et al.* [17] and Szymanski and Shenoy approaches[23] each advocate computing arrival times using iterative approaches based on successive relaxation of arrival and departure times. Szymanski and Shenoy show that clock schedules can be verified using a simple polynomial time algorithm modeled after the Bellman–Ford shortest path algorithm [23]. Lockyear's approach [14] and Ishii *et al.*'s approach [11], however, are based on determining the amount of time in which a computation must complete. This approach also results in efficient polynomial algorithms for verifying schedules.

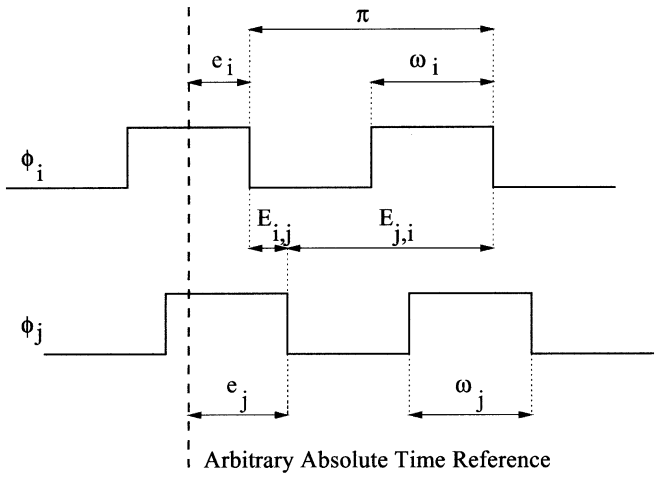

Fig. 1. Example clock schedule that illustrates the SMO clocking model.

III. Preliminaries

A. Clock-Schedule Model

Our clock-schedule model is based on the SMO formulation [16]. An n-phase clock schedule is an ordered collection of n periodic signals, $(\phi_1, \ldots, \phi_n)$, having a common period π. Because phases are periodic, a *local time zone* of width π is associated with each phase. Each phase ϕ_i is characterized by two parameters e_i and w_i. Parameter e_i represents the absolute time when ϕ_i begins (relative to an arbitrary global time reference). Parameter w_i is the length of time that ϕ_i is active (latch is open). To translate one measurement a from the local time zone of ϕ_i into the *next* local time zone of ϕ_j, we subtract from a a phase shift operator $E_{i,j}$, defined as

$$E_{i,j} = \begin{cases} e_j - e_i, & \text{if } i < j \\ \pi + e_j - e_i, & \text{otherwise.} \end{cases}$$

This clocking scheme is demonstrated in Fig. 1. If the clock period π is 10 time units, $w_i = 5$, $w_j = 5$, and $E_{ij} = 2$, then an arrival of 8 in ϕ_i's time zone translates to an arrival of 6 in ϕ_j's time zone.

We assume that the design intention and, thus, the clock schedule specify that a signal departing from a latch k must be captured by the next latching edge (which occurs after the latching edge of k) of the following latch l. The earliest arrival time at the output of a latch k clocked by ϕ_i is $\pi - w_i$, and it must arrive at the input of the following latch l clocked by ϕ_j on or before latch l's closing edge: $\pi + E_{i,j}$ time units after the beginning of ϕ_i. Setup and hold times are ignored to simplify the presentation.

B. Delay Model

The dynamically bounded gate delay model [10], illustrated in Fig. 2, captures most delay variations within the fixed range $[\delta, \Delta]$, while explicitly modeling all other variations. With a narrower fixed range, more explicit variations must be represented. With a wider range, only a few variations must be represented. If all variations are captured with the $[\delta, \Delta]$ range, then our model is essentially the commonly used fixed, or $\min$-$\max$, delay model. Delays associated with crosscoupling

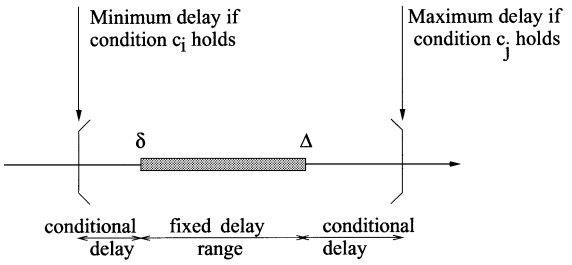

Fig. 2. Dynamically bounded delay model.

are modeled as follows. Assume that the output of a node v capacitively couples to the output a node a. With opposing coupling, v's maximum delay is increased by a $\Delta_{v,a}$. With assistive coupling, v's minimum delay is decreased by a value $\delta_{v,a}$. A predicate indicates when this increase or decrease must hold. To handle additive coupling or more detailed conditions, predicates can conditionally specify when these delays will be used.

C. Circuit Model

A circuit is modeled as a directed graph $G = (V, E, C)$. Each vertex in V represents either a primary input, primary output, a combinational gate, or a latch. The set of all combinational gates is referred to as V_C, and the set of all latches is referred to as V_L. P_v refers to the set of predecessors of node $v \in V$. Each edge in E represents the connectivity between two vertices. C represents the set of capacitors in the circuit. A set C_v is the set of aggressor nodes connected via a capacitor to node v. Each node v has a dynamically bounded delay model consisting of a fixed delay range $[\delta_v, \Delta_v]$. In addition, for each coupling capacitance attached to v and an aggressor node a, four delay values: $\Delta_{v,a}$, $\Delta_{a,v}$, $\delta_{v,a}$, and $\delta_{a,v}$, and a predicate indicates when the conditional delays should be considered.

We designate the latest (earliest) arrival time at a node v as A_v (a_v). The latest (earliest) departure time from a node is denoted by D_v (d_v). The time reference of D_v and d_v is based on associating each node v with a *phase* $p(v)$ which is derived by analyzing the phases of the latches in the combinational fanin and fanout of node v.

IV. Example

To understand how false coupling can produce pessimistic clock schedules, consider the example circuit in Fig. 3(a). A worst case coupling scenario assumes that signals *B* and *F* couple, and signals *D* and *H* couple. The delay of each block is computed based on worst case opposing and assistive coupling. For example, the block generating signal *B* will have a delay of [0,3] (i.e., $[1, 2] + / - 1$), and the arrival window for signal *B* will be [5,9].

The ranges labeled *A*–*H* in Fig. 3(b) indicate the time ranges when these nodes switch for a two-phase, symmetric, nonoverlapping clock schedule with a period of 10 time units. Signal *F* must wait until the opening edge of the ϕ_2 latch before the value is propagated. The smallest possible clock period is forced to be at least 10, to accommodate the critical path, whose worst case delay is 15, from the input of the block generating signal *A* to *D*. Using the schedule in Fig. 3(c), for example, will not work since the period is 9. Other schedules with a period of 10, such as ones with nonsymmetrical phases, will work.

The switching windows of signals *C* and *G* overlap, thus, coupling between *D* and *H* will cause additional delays for both signals. The switching windows of *A* and *E* are, however, far apart. Thus, *B* switches without interference from *F*. Noise might be possible on node *F*, but it will certainly not affect its arrival times. The coupling between *B* and *F* is, thus, false. The delay of the critical path from the block generating *A* through the block generating *D* is 14 instead of 15. The schedule in Fig. 3(c) can be used to clock this circuit. It has a smaller period than the one in Fig. 3(b). Timing analysis that eliminates false coupling, therefore, allows a faster schedule. In this example, the comparison of the overlapping switching windows of the victims and the aggressors was done in absolute time. However, arrival times are computed relative to a specific latch's time zone, and we must translate the time zone of the aggressor to that of the victim (or vice versa) in order to compare them correctly.

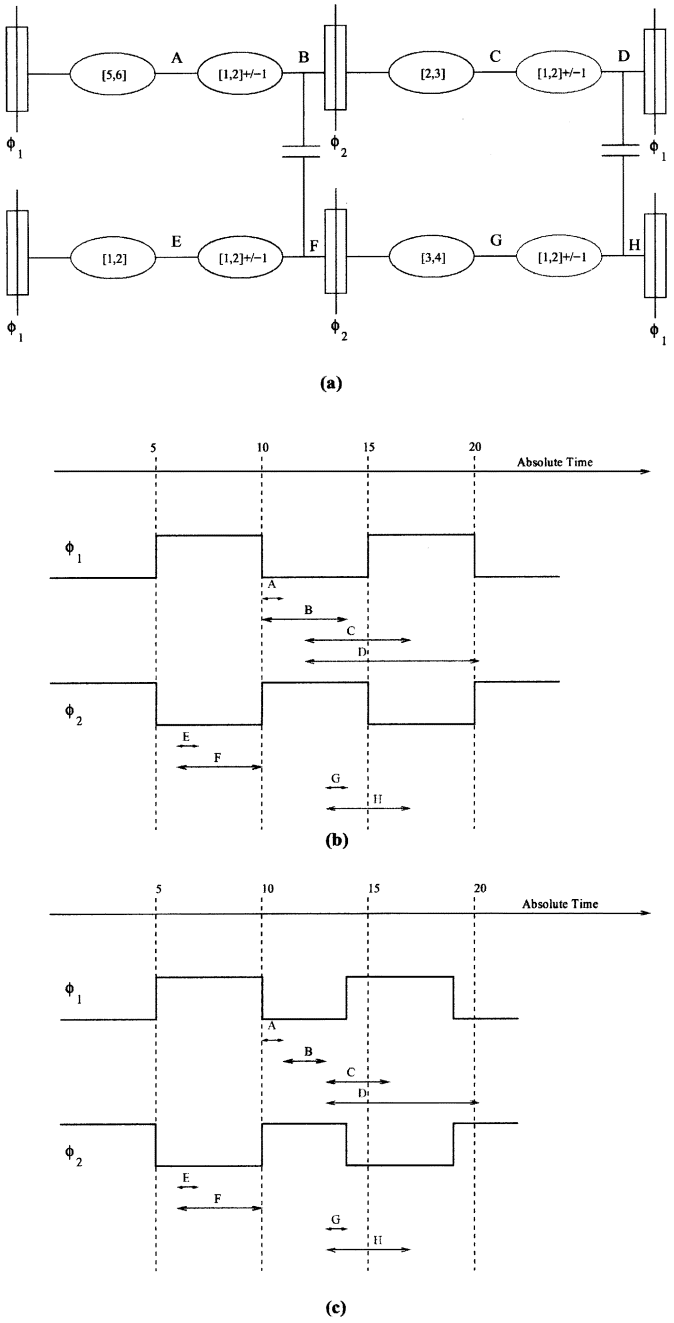

Fig. 3. Example circuit and schedules. (a) Circuit under consideration. Each block has a bounded delay model: Delays are expressed as a range and the conditional delay due to coupling is $+/-1$. (b) A clock schedule with the smallest allowed period of 10 when assuming all coupling causes delays. (c) A clock schedule with a period of nine when false capacitive coupling between *B* and *F* is eliminated.

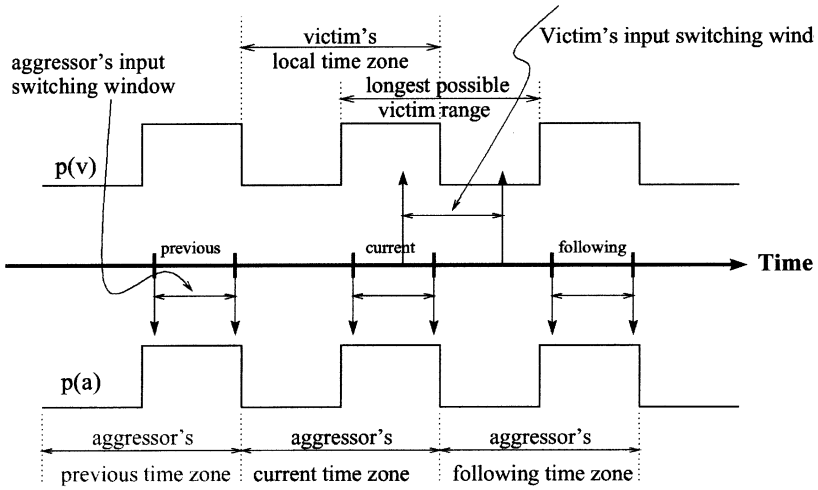

Fig. 4. Aggressor's and victim's time zones are aligned. We must check for overlap between the switching windows of the inputs to the victim and aggressor while considering all switching ranges.

V. Timing Equations

The earliest and latest arriving signals at the inputs of the victim and aggressor must be analyzed to determine if the switching windows overlap. The latest arrival time at a combinational node v

$$A_v = \begin{cases} \max_{\forall k \in P_v}(D_k - E_{p(k),p(v)}), & \text{if } p(k) \neq p(v) \\ \max_{\forall k \in P_v} D_k, & \text{if } p(k) = p(v) \end{cases}. \quad (1)$$

If the phases associated with nodes k and v are different, then the departure time D_k is adjusted by $E_{p(k),p(v)}$ to transfer the departure time of k to v's local time zone.

For a latch v with input k

$$A_v = \max(D_k - E_{p(k),p(v)}, \pi - w_{p(v)}). \quad (2)$$

Here, the latest arrival time at the latch depends on the relative arrival time of the signals at its input D_k and when the latch allows the data through, $\pi - w_{p(v)}$. If the input signal k arrives before the latch is open, then it must wait until the latch opens before k is passed through.

The departure time from a node v, without capacitive coupling on its output, can be specified as follows:

$$D_v = A_v + \Delta_v. \quad (3)$$

To compute D_v, the departure time at v, we augment the latest arriving input to v by an amount Δ_v, the maximum propagation delay through v.

For a node v with capacitive coupling on its output through one or more aggressor in C_v, the maximum departure time is

$$D_v = A_v + \Delta_v + \sum_{\forall a \in C_v} \gamma_{v,a} \Delta_{v,a}. \quad (4)$$

This constraint ensures that the propagation delay of v is augmented by an amount $\Delta_{v,a}$ when a node v (the victim) experiences capacitive coupling through an aggressor a. Worst case opposing coupling between v and a is assumed because we are not considering the functional/logical behavior of the circuit. Variable $\gamma_{v,a}$ is binary indicating if the conditions for capacitive coupling hold. A description of conditions that cause coupling is provided in Section VI.

Similarly, we specify constraints for minimum arrival and departure times. For a combinational node v

$$a_v = \begin{cases} \min_{\forall k \in P_v}(d_k - E_{p(k),p(v)}), & \text{if } p(k) \neq p(v) \\ \min_{\forall k \in P_v} d_k, & \text{if } p(k) = p(v) \end{cases}. \quad (5)$$

For a latch v

$$a_v = \max(d_k - E_{p(k),p(v)}, \pi - w_{p(v)}). \quad (6)$$

The earliest departure time for a node v can be specified as follows assuming worst case assistive coupling between a victim node v and an aggressor a:

$$d_v = a_v + \delta_v - \sum_{\forall a \in C_v} \gamma_{v,a} \delta_{v,a}. \quad (7)$$

VI. Coupling Conditions

Due to the periodicity of signals in a sequential circuit, coupling can occur due to the overlap, or close proximity by an amount of τ, of the switching window at the input of the victim and any *periodic* switching window at the aggressor's input.

Consider the situation depicted in Fig. 4, where the aggressor and the victim have the same phase $p(v) = p(a)$ resulting in aligned time zones. When considering the maximum possible victim range and the need to account for τ, it is apparent that the victim's input switching window can overlap with either one, two, or three of the three possible switching windows of the aggressor's input: the *previous*, the *current*, and the *following* windows.

To determine if coupling exists, we must compare the overlap between the input switching windows with that of the three occurrences of the aggressor. When $p(v) = p(a)$, determining the overlap between the inputs to the victim and the *current* aggressor, is essentially the same as for combinational circuits, namely

$$\max(a_v, a_a) \leq \min(A_v, A_a) + \tau.$$

The comparisons with the previous and following occurrences can also be determined by noting that the previous occurrence of the aggressor can be computed by subtracting π from the range, resulting in $[A_a - \pi, a_a - \pi]$, while computing the *following* occurrence requires adding π.

When $p(v) \neq p(a)$, the arrival times at the input of the aggressor must be translated to the victim's local time zone to perform a meaningful comparison. Consider the case in Fig. 5(a) with the following assumptions: The clock period $\pi = 10$; $e_{p(a)} - e_{p(v)} = 1$; the SMO phase shift operator $E_{a,v} = -9$; $\tau = 1.01$; and 50% duty cycle. If the SMO shift operator is used to translate the aggressor ranges

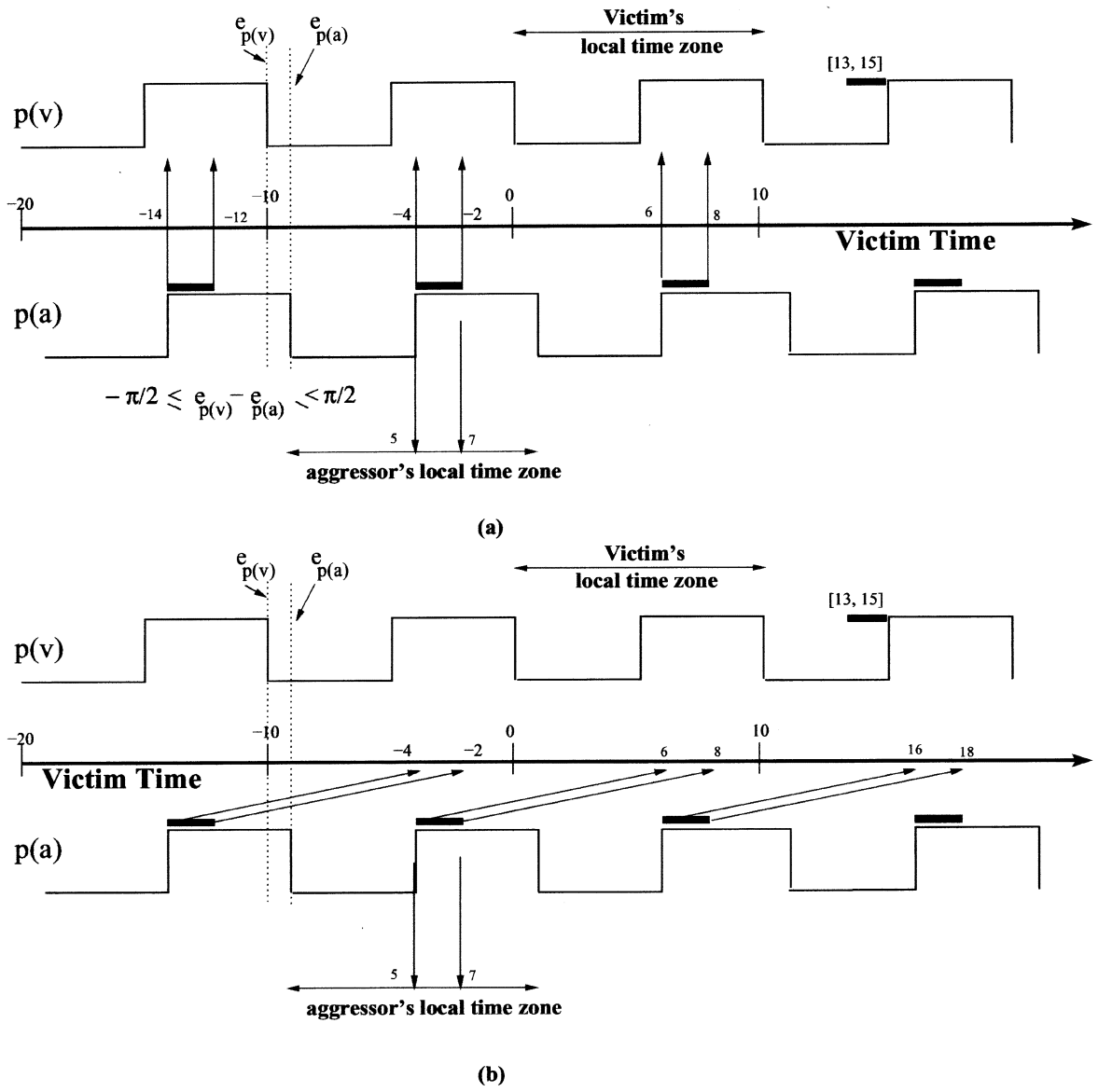

Fig. 5. Comparing overlapping windows. (a) Using the SMO shift operator of nine, coupling is not detected. (b) Using the new phase shift operator of -1, coupling is detected.

$[5-\pi, 7-\pi]$, [5,7], and $[5+\pi, 7+\pi]$, then the ranges, respectively, become $[-14, -12]$, $[-4, -2]$, and [6,8]. If the victim occurrence is [13,15], then comparing the translated aggressor ranges against the victim's will not indicate a coupling problem. However, the fourth occurrence [16,18] was not considered. Because it is within τ of the victim occurrence, coupling should have been detected.

Consider another approach in which we designate the aggressor's *current* time zone as the *closest* in time from the victim's local time zone. The *previous* aggressor's time zone is the one preceding the *current* aggressor's time zone. The *following* aggressor's time zone is the one succeeding the *current* aggressor's time zone.

To determine the *closest* aggressor time zone, we compare the positions of $p(a)$ and the $p(v)$. Recall from Section III-A that the phases are ordered periodic signals and that each is associated with parameters e_i, the time when phase i begins relative to an absolute reference point.

If the victim's time zone leads or lags the aggressor's local time zone by or less than $\pi/2$, (i.e., $-\pi/2 \leq e_{p(v)} - e_{p(a)} \leq +\pi/2$), then the latter time zone is designated as the *current* time zone. If the victim's time zone leads (occurs before) the aggressor's local time zone by more than $\pi/2$ (i.e., $e_{p(v)} - e_{p(a)} < -\pi/2$), then the latter is designated as a *following* time zone. Similarly, if the victim's time zone lags (occurs after) the aggressor's local time zone by more than $\pi/2$ (i.e., $e_{p(v)} - e_{p(a)} > \pi/2$), then the latter time zone is designated as *previous*.

To translate a value local to the aggressor's time zone to the victim's time zone and to have that value appear as a *current* occurrence, we define a new phase shift operator $E'_{i,j}$ as follows:

$$E'_{i,j} = \begin{cases} e_j - e_i + \pi, & \text{if } e_j - e_i < -\frac{\pi}{2} \\ e_j - e_i, & \text{if } -\frac{\pi}{2} \leq e_j - e_i \leq +\frac{\pi}{2} \\ e_j - e_i - \pi, & \text{if } e_j - e_i > +\frac{\pi}{2} \end{cases}.$$

This operator differs from the SMO phase shift operator. Consider again the coupling scenario in Fig. 5. We examine the use of the new phase shift operator which is illustrated in Fig. 5(b). In this case, $E'_{p(a),p(v)} = -1$. Subtracting this phase shift operator, the three aggressor ranges now become $[-4, -2]$, [6,8], and [16,18]. When the range [16,18] is compared against the victim's range of [13,15], then

coupling will be detected because this latter range is within τ from the [16,18] aggressor range.

Based on our analysis and our new operator, we can now define coupling to occur in the following cases:

- coupling with the current occurrence: $\max(a_v, a_a - E'_{a,v}) \leq \min(A_v, A_a - E'_{a,v}) + \tau$;
- coupling with the following occurrence: $A_v + \tau \geq a_a - E'_{a,v} + \pi$ and $A_v > a_a - E_{a,v}$;
- coupling with the previous occurrence: $a_v \leq A_a + \tau - E'_{a,v} - \pi$ and $a_v \leq A_a - E_{a,v}$.

If and only if one of the above coupling conditions holds, then the binary $\gamma_{v,a}$ is set to one.

VII. Algorithm

Our algorithm for verifying that a circuit runs correctly for a given clock schedule is iterative. Initially, all coupling is assumed not to hold; all $\gamma_{v,a}$ variables are set to zero. During each iteration, the steps below are performed. This algorithm is run until no new γ variables are assigned.

Algorithm

1) Compute the latch-to-latch, PI-to-latch, and latch-to-PO minimum and maximum delays as outlined in **[19]**. The run-time is dominated by $O(|V_L| \times (|V| + |E|))$, where $|V_L|$ is the number of latches in the circuit. Because the Szymanski/Shenoy algorithm in the next step utilizes latch-to-latch delays, the computation in this step is needed to ensure the efficiency of the latter algorithm. During each iteration, the latch-to-latch delays are recomputed because new γ variables are assigned and the computed delays will be different.
2) Using the delays computed in step 1, run the Szymanski/Shenoy **[23]** algorithm to compute the arrival and departure times at the latches, PIs, and POs. The run-time of the algorithm is $O(|V_L|^3)$. Because the next step requires the arrival times at the inputs to victims and aggressors, a postprocessing step, linear in the number of circuit nodes and edges, produces these values.
3) Compare the switching windows as outlined in the previous section, and set the appropriate binary γ variables. The run-time is linear in the number of nodes, assuming a small number of aggressors is associated with each victim.

Our algorithm is guaranteed to converge. Once a new γ is assigned, the victim's window is simply stretched (the A_v becomes larger and the a_v becomes smaller). Such a change in the victim's window can only cause other windows to either remain the same or further stretch. The algorithm is guaranteed to converge in $|C|$ iterations because, in the worst case, one γ variable is assigned true through each iteration. Furthermore, once γ is assigned true, it does not change. Once $|C|$ iterations are completed, no switching windows change. The argument of continually shrinking or expanding switching windows was used to prove convergence for timing analysis for combinational circuits [3], [18]. Sapatnekar noted that $|C|$ iterations are needed for convergence [18].

TABLE I
Sequential Circuits From MCNC FSM Benchmarks. We List the Number of Primary Inputs and Outputs, Latches, and Combinational Gates

Circuit	Total Gates	Primary Inputs	Primary Outputs	Latches	Combinational Gates
train4	20	2	1	4	13
bbsse	109	7	7	8	87
ex2	114	2	2	10	100
ex6	119	5	8	17	89
cse	175	7	7	8	153
kirkman	200	12	6	8	174
dk16	205	2	3	11	189
sand	542	11	9	86	436
c1k	1075	11	9	183	872
c2k	2141	11	9	377	1744
c4k	4273	11	9	765	3488

VIII. Experimental Results

Our experiments evaluate the effectiveness of our algorithm in verifying clock schedules in the presence of crosstalk. Our benchmarks are based on a subset of the edge-triggered Microelectronics Center of North Carolina FSM circuits that we convert to circuits with level-clocked latches. Sequential interactive synthesis was first used to perform logic optimization and mapping [20]. We then converted registers to back-to-back $\phi 1/\phi 2$ latches and used sskew, Lockyear's retiming tool [14], to determine an equal, two-phase retiming, and initial clock schedule. The combinational nodes in the circuit were initialized with a maximum random delay within 2.5 and 0.5; the minimum delay was then initialized with a random value that is at most 0.5 less than the maximum delay. We then added random capacitors equal in number to 10% of the total circuit nodes. Each capacitor was assigned a random delay between 0.0 and 1.0. The circuits used are summarized in Table I. We augmented the circuits with three larger ones: *c1k*, *c2k*, and *c4k*. These circuit were obtained by stitching together the mapped *sand* benchmark and then generating delays and capacitors randomly and converting the registers to latches.

We ran sskew to determine the worst and best clock schedules. Table II lists the *maximum period* that assumes worst case capacitive coupling, and the normalized *minimum period*, which assumes no coupling, in column 2 and 3, respectively. To find the best clock period with our algorithm, we search the space starting with a minimum clock period, incrementing this period by 10% of the maximum clock period until we find a period at which the circuit ran. Because the solution space may not be convex we avoided doing a binary search as is possible when trying to determine the minimum clock period when no coupling is considered (e.g., Lockyear's approach [14]). The final period is reported in columns 4 while column 5 lists the reduction achieved with respect to the maximum possible reduction (i.e., the difference between the maximum and minimum clock periods). The final column lists the total run-time.

From our results in Table II, we see that only one circuit operated at the maximum clock period. This circuit has a combinational delay from a primary input to a primary output that sets the clock period. For the others, the circuit ran at a smaller clock period than the maximum one. Some circuits were able to run at the indicated minimum clock period. The number of calculations to reach the minimum clock period was one for all circuits except for circuits dk16, ex2, and ex6, for which

TABLE II
RESULTS TABLE. THE MAXIMUM PERIOD COLUMN PRESENTS THE MAXIMUM CLOCK PERIOD ASSUMING WORST CASE CAPACITIVE COUPLING. THE MINIMUM PERIOD COLUMN REPORTS THE NORMALIZED MINIMUM POSSIBLE PERIOD IGNORING ALL CAPACITIVE COUPLING. THE NORMALIZED FINAL PERIOD IS FOUND BY OUR ITERATIVE ALGORITHM. THE FOLLOWING COLUMN REPORTS THE PERCENTAGE REDUCTION. THE TOTAL TIME IS THE COMBINED RUN TIME FOR ALL ITERATIONS IN SECONDS

Circuit	Maximum Period	Minimum Period	Final Period	Percentage Reduction	Total Time
train4	7.20	0.905	1.000	0.00	0.833
bbsse	13.93	0.903	0.903	100.00	0.108
ex2	14.12	0.892	0.902	90.72	0.376
ex6	13.54	0.947	0.957	81.09	0.405
cse	14.08	0.939	0.939	100.00	0.262
kirkman	14.98	0.875	0.875	100.00	0.261
dk16	14.34	0.890	0.899	90.92	0.874
sand	14.49	0.938	0.938	100.00	3.964
c1k	19.19	0.873	0.873	100.00	13.054
c2k	23.01	0.910	0.911	100.00	62.912
c4k	25.48	0.920	0.920	100.00	307.471

TABLE III
THE NUMBER OF CAPS COLUMNS SHOWS THE NUMBER OF CAPACITORS IN THE CIRCUIT. THE FINAL COLUMN PRESENTS THE CAPACITORS THAT AFFECT THE FINAL ANALYSIS

Circuit	Total Caps	Contributing Caps
train4	2	2
bbsse	10	9
ex2	11	10
ex6	11	11
cse	17	16
kirkman	20	15
dk16	20	17
sand	54	45
c1k	107	63
c2k	214	107
c4k	427	195

the minimum period was obtained during the second calculation, and for train4, where the minimum clock period was obtained during the 11th calculation. This fast convergence is due to the fact that more than one capacitor was effective in contributing to the delay very early in the algorithm. Table III lists the number of capacitors that affect delays in the circuit.

The total run-times are shown in Table II. The 4K circuit c4k ran in less than 6 min. All analyses were performed on a Sun Enterprise-250. Run-times were collected using gethrtime system call which measures user time. This is almost the same as CPU time considering that timing analysis was the only active process running on the machine. Table IV lists the run-times associated with each phase of the algorithm as outlined in the steps in Section VII.

Our results conclude that, for the set of examined benchmarks, it is indeed possible to find a faster clock schedule using more accurate and less pessimistic timing analysis. The implementation seems reasonably fast for the examples presented. The run-time, however, may become prohibitive for larger circuits. From the run-times in Table II, one can see that the run-time grows approximately by a factor of 6 as the circuit size is doubled. Due to the unavailability of realistic public domain larger benchmarks, it is not possible to further assess the implementation.

In light of comments by Szymanski and Shenoy [23], we make the following two observations. First, the SMO equations [17] may have more than one solution when the circuit is running at the optimal

TABLE IV
RUN-TIMES IN MILLISECONDS FOR STEPS 1)–3) OF THE ALGORITHM DURING THE FINAL CALCULATION OF OUR ALGORITHM

Circuit	Step 1	Step 2	Step 3
train4	7.2	1.8	0.7
bbsse	75.6	25.1	2.4
ex2	98.3	23.6	2.7
ex6	98.3	42.3	2.5
cse	185.4	59.6	4.0
kirkman	152.7	85.5	4.9
dk16	210.7	74.6	4.7
sand	1624.4	2216.2	19.4
c1k	4219.5	8376.3	36.9
c2k	16374.4	44660.1	102.6
c4k	53533.1	244434.0	202.2

clock period. The slightest physical perturbation may cause the circuit to switch from one solution to another. Szymanski and Shenoy advise against operating a circuit at such an optimal clock period. Crosstalk could potentially cause timing violations and, thus, errors while switching from one operating point to another. Second, the Szymanski and Shenoy algorithm depicts a simulation of the circuit operation during the first $|V_L|$ cycles once the power is turned on [23]. During such early simulation cycles, hold constraints may be violated but could be corrected later as arrival times monotonically increase to their steady-state values. The authors state that a *reset* operation should persist for as many as $|V_L|$ cycles to ensure proper operation. Additional crosstalk analysis during reset is needed to ensure correct operation.

IX. CONCLUSION

This is the first paper that addresses crosstalk analysis for circuits with level-sensitive latches. The main contributions of this paper are: 1) showing the overlapping conditions necessary to detect changes in delays due to coupling; 2) deriving a new phase shift operator to conveniently translate the aggressor's periodic occurrences to the victim's local time zone; and 3) presenting a polynomial algorithm to solve timing verification for level-sensitive circuits in the presence of crosstalk. These contributions are not specific for the dynamically bounded gate-delay model, but they will hold for any discrete overlapping coupling model. Our experiments demonstrate that eliminating false coupling results in a tighter clock schedule.

ACKNOWLEDGMENT

The authors wish to thank the anonymous reviewers for their suggestions.

REFERENCES

[1] V. Agrawal, "Synchronous path analysis in MOS circuit simulator," in *Proc. 19th Design Automation Conf.*, 1982, pp. 629–635.
[2] R. Arunachalam, R. Blanton, and L. Pileggi, "False coupling interactions in static timing analysis," in *Proc. ACM/IEEE Design Automation Conf.*, 2001, pp. 726–731.
[3] R. Arunachalam, K. Rajagopal, and L. Pileggi, "TACO: Timing analysis with coupling," in *Proc. ACM/IEEE Design Automation Conf.*, 2000, pp. 576–580.
[4] P. Chen and K. Keutzer, "Toward true crosstalk noise analysis," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1999, pp. 132–137.
[5] ——, "Switching window computation for static timing analysis in presence of crosstalk noise," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 2000, pp. 331–337.
[6] P. Chen, Y. Kukimoto, C.-C. Teng, and K. Keutzer, "On convergence of switching windows computation in presence of crosstalk noise," in *Proc. Int. Symp. Physical Design*, 2002, pp. 84–89.

[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