Transient Power Management Through High Level Synthesis

Vijay Raghunathan[†], Srivaths Ravi[‡], Anand Raghunathan[‡], and Ganesh Lakshminarayana[‡]
[†]Dept. of Electrical Engineering, University of California, Los Angeles, CA 90095
[‡]NEC USA, C&C Research Laboratories, Princeton, NJ 08540

Abstract

The use of nanometer technologies is making it increasingly important to consider transient characteristics of a circuit's power dissipation (*e.g.*, peak power, and power gradient or differential) in addition to its average power consumption. Current transient power analysis and reduction approaches are mostly at the transistor- and logic-levels. We argue that, as was the case with average power minimization, architectural solutions to transient power problems can complement and significantly extend the scope of lower-level techniques.

In this work, we present a high-level synthesis approach to transient power management. We demonstrate how high-level synthesis can impact the cycle-by-cycle peak power and peak power differential for the synthesized implementation. Further, we demonstrate that it is necessary to consider transient power metrics judiciously in order to minimize or avoid area and performance overheads. In order to alleviate the limits on parallelism imposed by peak power constraints, we propose a novel technique based on the selective insertion of data monitor operations in the behavioral description. We present enhanced scheduling algorithms that can accept constraints on transient power characteristics (in addition to the conventional resource and performance constraints). Experimental results on several example designs obtained using a state-of-the-art commercial design flow and technology library indicate that high-level synthesis with transient power management results in significant benefits — peak power reductions of up to 32% (average of 25%), and peak power differential reductions of up to 58% (average of 42%) — with minimal performance overheads.

1 Introduction

Power dissipation issues are being made increasingly important and mainstream in the design of deep sub-micron system chips due to electronic system and circuit technology trends. The widespread demand for ubiquitous (wireless) communications and information access devices implies that a significant and growing fraction of chips are designed with battery considerations. Equally important are the challenges ushered in by the use of nanometer technologies for system integration. Ensuring efficient and reliable power delivery and signal integrity are daunting tasks for deep sub-micron system-on-chip designs. These problems are further complicated by the use of low-power design techniques such as supply voltage reduction, power management, and variable voltage design [1, 2, 3, 4, 5]. Thus, in nanometer technologies, analyzing and managing a circuit's transient power characteristics (*e.g.* peak power, power or current gradient, *etc.*) is equally if not more important than minimizing average power or total energy consumption. Peak power dissipation is directly related to packaging and cooling requirements, and determines I-R drops and electromigration in the on-chip power supply network. The current (power) differential determines the noise introduced due to inductive ground bounce. In this paper, we address the issue of managing these transient power consumption characteristics through the choice of appropriate architectures during high-level synthesis (HLS).

1.1 Related Work

It is well known that architectural decisions can have a significant impact on a circuit's power consumption. Hence, a significant body of work has been devoted to minimizing average power consumption or total energy consumption during high-level design [1, 2, 3, 4, 5]. However, relatively little work has been targeted at designing circuits with improved transient power characteristics.

Analysis and design techniques to identify and alleviate transient power related problems have been proposed at the lower levels of abstraction. Techniques to estimate the cycle-by-cycle peak power consumption of circuits at the gate level were proposed in [6, 7]. Design analysis and validation techniques for ground bounce were proposed in [8]. Techniques for timing analysis of digital circuits considering the effect of power supply noise on the delays of gates in the circuit were proposed in [9].

The above work mostly addresses analysis of the effects of power transients. Current commercial tools mostly fall into the category of accurate simulation and design of the power supply network itself [10, 11]. While the use of such accurate analysis techniques is eventually necessary for full-chip validation before sign-off, their use late in the design flow implies that it may be very expensive (in terms of design effort and design overhead), if not impossible, to re-design the circuit in order to address any problems. The use of high-level strategies, such as those presented in this paper, to manage transient power, has a potential to significantly ease the burden on power supply network design and avoid re-designs late in the design flow. As shown in this paper, although only coarse timing and power information is available at the high-level, it is still possible to take decisions that result in designs with more desirable transient power characteristics.

In [12], the authors propose the use of staged shut down and wake up for circuit blocks to alleviate inductive noise problems introduced due to clock gating. The idea is to gradually "freeze" and "un-freeze" the unit's inputs over multiple clock cycles, resulting in reduced power transients in each clock cycle. The application of this idea to VLIW and array processor architectures is explored in [13]. Low power HLS tools for data flow dominated behaviors which considered peak power were presented in [14]. At the system-level, power transients can have undesirable effects on voltage regulator and battery efficiency. Work on controlling power transients at the system level through communications protocols and scheduling to improve battery life were presented in [15, 16]. We believe that, similar to average power or energy reduction, it is important to develop techniques to address transient power issues at every stage in the design process (including the system level, hardware architecture level, and logic/circuit level). System-level transient power management should aim at controlling coarse-grain transients, while hardware architecture design techniques (such as our work) should aim to manage power transients on a cycle-by-cycle basis, and circuit-level tools can be used for fine-grained analysis and optimization of the circuit and power supply network. Thus, the techniques presented in this paper are complementary to previous work on managing transient power at the system and circuit levels.

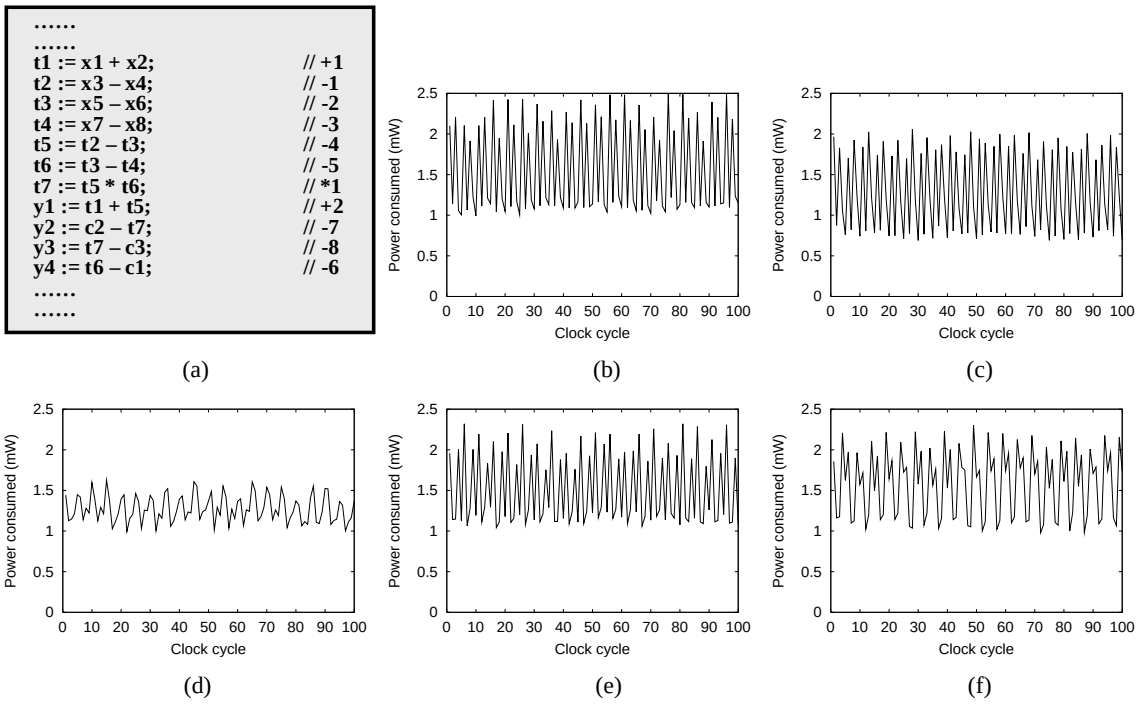

Figure 1: (a) Example behavior `test1`, and (b)-(f) power profiles for five distinct implementations of the `test1` behavior

1.2 Paper Overview and Contributions

This paper presents techniques for transient power management during high-level synthesis. We demonstrate that the manner in which high-level synthesis is performed can significantly impact the transient power profile of the resulting circuit implementation. Further, we demonstrate that it is necessary to consider transient power metrics judiciously in order to minimize or avoid area and performance overheads. In order to alleviate performance overheads that result from the limits on parallelism imposed by peak power constraints, we propose a novel technique based on the selective insertion of *data monitor* operations in the behavioral description.

We show how to modify generic high-level synthesis algorithms in order to produce designs with desirable transient power profiles while incurring minimal overheads. Finally, we demonstrate through experimental results that the use of our techniques can lead to implementations that display significantly superior transient power characteristics compared to implementations generated by a state-of-the-art high-level synthesis tool.

2 Issues and Illustrations

In this section, we demonstrate through examples that decisions made during high-level synthesis can have a significant impact on the transient power profile of the resulting implementation. We then illustrate the issues involved in generating transient power managed architectures and show that transient power management techniques need to be integrated carefully into the high-level synthesis flow to avoid or minimize the attendant overheads. Finally, we illustrate a novel technique based on the use of data monitors to alleviate the performance overheads arising from the limits on parallelism imposed by peak power constraints.

2.1 Impact of high-level synthesis on transient power

Example 1: Consider the example behavior `test1` that is shown in Figure 1(a). In order to study the effects of high-level synthesis on the transient power profile of the implementation, we generated different RTL implementations of the behavior `test1` by varying the resource constraints, scheduling, and resource sharing used during the synthesis process. In order to consider reasonably power optimized designs, we generated RTL implementations that employed a combination of clock gating for inactive registers and operand isolation for functional units based on information from the high-level synthesis tool [1].

The resulting RTL implementations were synthesized using Synopsys Design Compiler [17], and mapped to NEC's 0.35 micron gate array technology [18]. Gate level power estimation was performed using an in-house tool that is used commercially for sign-off [19] (the power estimation process includes estimated interconnect and clock network parasitics). Figures 1(b)-(f) present the cycle-by-cycle power consumption profiles of five distinct designs that implement the behavior `test1`. Each profile represents the cycle-by-cycle power variation over 100 clock cycles for pseudo-random input traces. Inspection of Figure 1 clearly supports our hypothesis that high-level synthesis decisions significantly impact the power profile of the synthesized implementation. Even if we restrict our comparison to designs with similar area and performance, there is still a significant variation in the transient power characteristics of the implementation. For example, the power profile shown in Figure 1(c) is quite different from the power profile shown in Figure 1(d), although the designs have similar area and performance. Thus, we can conclude that high-level transient power management has the potential to result in significant benefits. ∎

Having established the motivation for transient power management through high-level synthesis, we next demonstrate that it is possible to quantitatively analyze and influence the power profiles of the synthesized implementation during high-level synthesis, using the techniques proposed in this paper.

Example 2: Consider again the `test1` behavior of Figure 1(a). We focus our attention on two different schedules for the behavior, that were derived under identical resource constraints (1 `add_rc_16`, 3 `sub_rc_16`, 1 `mul_wal_12`).

[1] However, this is not a necessary condition for applying the techniques presented in this paper.

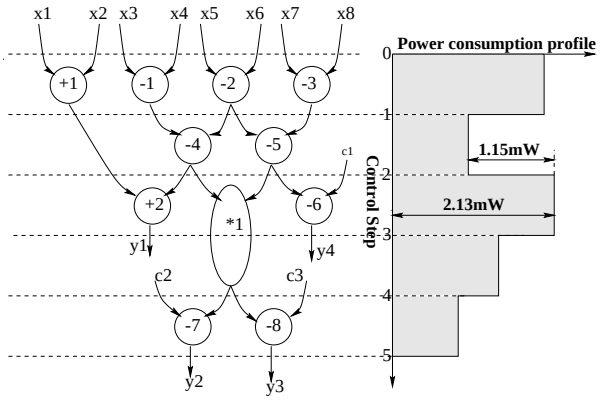

Figure 2: An example scheduled behavior used to illustrate high-level transient power management

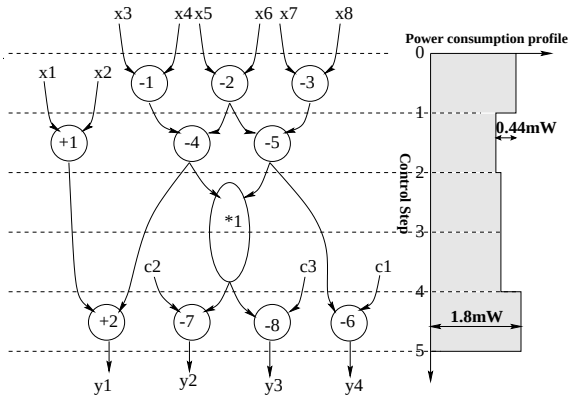

Figure 3: An alternative schedule for the example of Figure 1 with improved transient power characteristics

The first schedule, shown in Figure 2, was derived using the original high-level synthesis tool without any consideration for transient power characteristics. In order to evaluate transient power characteristics during high-level synthesis, we have developed a high-level technique to estimate the cycle-by-cycle power consumption profile (details of the procedure are provided in Section 3). This procedure uses the peak power consumption for each component in the RTL library (derived once per technology, during library characterization), together with the set of components that are active during each control step (derived through analysis of the schedule and module selection information). Peak power estimation for individual components can be performed using known techniques for peak power estimation at the logic and transistor levels [6, 7]. The peak power consumption values for a representative sub-set of components from the RTL library used in our work are shown in Table 1.

Component	Peak Power consumption
`add_rc_16`	0.260 mW
`add_cla_16`	0.475 mW
`sub_rc_16`	0.358 mW
`sub_cla_16`	0.549 mW
`mul_wal_12`	1.238 mW
`reg_16`	0.068 mW
`lt_cmp_16`	0.113 mW
`gt_cmp_16`	0.107 mW
`eq_cmp_16`	0.037 mW

Table 1: Peak power consumption values for some components in our RTL library

The schedule shown in Figure 2 is annotated with its power consumption profile, derived using the above mentioned procedure. The estimated peak power consumption associated with the schedule of Figure 2 is 2.13 mW, while the peak power differential is 1.15 mW, as indicated in the figure. Analysis of the figure indicates that the peak power and peak power differential are caused in the third control step, due to the scheduling of operations $+2$ and -6 in parallel with multiplication operation $*1$. Note that both operations $+2$ and -6 have some slack, hence it is not necessary to schedule them in the third control step. However, the high-level synthesis tool, being unaware of transient power considerations, chose the schedule shown in Figure 2.

Let us now consider an alternative schedule for the `test1` behavior that was derived using the high-level synthesis tool after the incorporation of the transient power management techniques proposed in this paper. Figure 3 shows the schedule, along with the corresponding cycle-by-cycle power consumption profile (again, derived using our high-level estimation procedure). Note that, the high-level synthesis tool has now avoided scheduling any other operations in parallel with $*1$, in order to improve the design's transient power characteristics. We synthesized the RTL implementations resulting from the schedules of Figures 2 and 3, and, derived the power consumption profiles at the gate-level, as explained earlier in this section. The resulting power consumption profiles were presented earlier in Figures 1(c) and (d), respectively. The transient power managed implementation corresponding to Figure 3 results in reductions of around 22% and 49%, in the peak power and peak power differential respectively, compared to the original implementation corresponding to Figure 2.

■

The following points demonstrated by the above example are worth noting:

- Incorporation of transient power management into high-level synthesis results in significant improvements in the transient power characteristics of the synthesized implementations.
- The high-level power profiles (Figures 2 and 3) derived using the procedure proposed in Section 3 are quite well correlated with the actual power consumption profiles derived at the gate-level (Figures 1(c) and 1(d)).

2.2 Performance Issues in Transient Power Management

The next example demonstrates that naive incorporation of transient power management techniques into high-level synthesis may negatively impact other design quality metrics (performance, area). This motivates the need for algorithms (such as the ones described in Section 3) that perform transient power management while judiciously evaluating and trading off its impact on performance and area. Further, the performance bottlenecks that result from the limits on parallelism imposed by peak power constraints motivate the need for our novel technique based on *data monitor* operations (presented later in this section).

Example 3: Figure 4(a) shows an example behavior fragment represented as a control data flow graph (CDFG). The behavior contains a data-dependent loop that performs computations and stores the results in two arrays (through operations M1 and M2). In Figure 4(a), dotted lines represent control dependencies, while solid lines represent data dependencies. Suppose that it is required to synthesize the behavior of Figure 4(a) under the resource constraint of 2 multipliers, 1 adder, 1 subtractor, and 1 comparator. In addition, suppose that it is necessary to satisfy a constraint of 2 mW on the cycle-by-cycle peak power.

By analyzing the peak power constraint and the peak power consumption of individual components presented in Table 1, we can conclude that, although we have two multiplier resources available, we

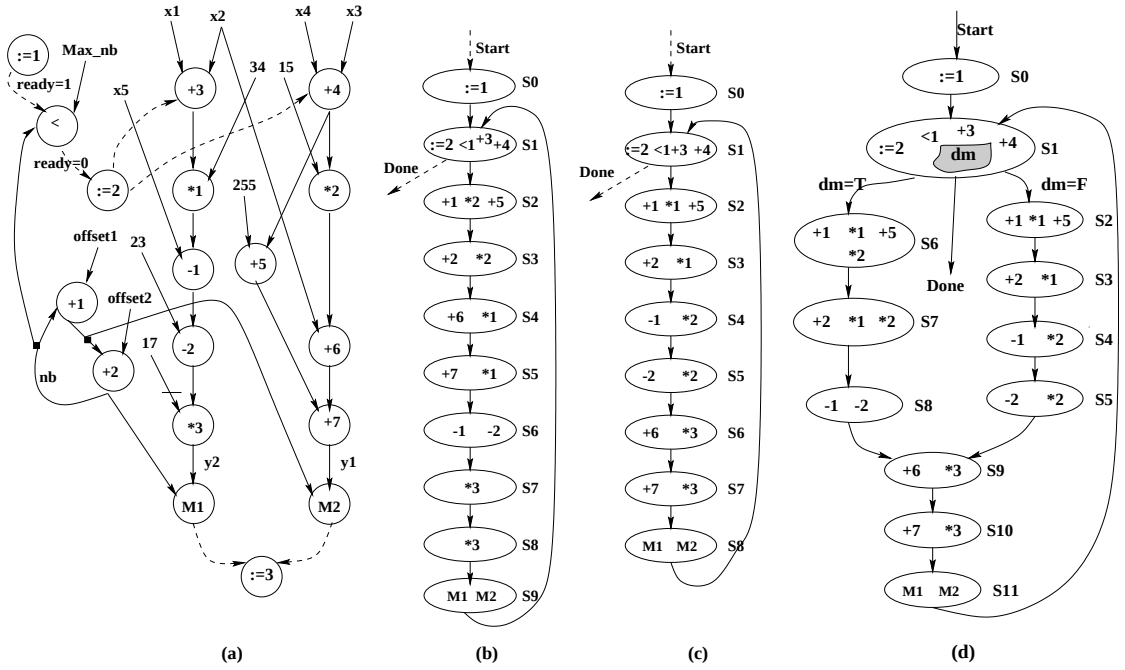

Figure 4: (a) Behavior used in Example 2 (b) A schedule for the behavior (c) An alternate schedule for the same behavior (d) A schedule with data monitors

cannot schedule two multiplication operations concurrently. In order to enforce such sequential execution, we can introduce "implicit dependencies" between every pair of multiplication operations in the behavior that can potentially be executed in parallel (*e.g.* between $*1$ and $*2$). However, the manner in which these dependencies are introduced can significantly impact performance, as shown next. Let us focus on operations $*1$ and $*2$ in the CDFG of Figure 4(a). When introducing a dependency to enforce sequential execution of $*1$ and $*2$, two natural possibilities arise for their execution order. We performed high-level synthesis for both these cases, and the resulting schedules are presented as state transition graphs (STGs) in Figures 4(b) and 4(c), respectively. The schedule in Figure 4(b) is obtained by introducing a dependency that forces execution of $*2$ before $*1$, while the schedule in Figure 4(c) is obtained by introducing a dependency from $*1$ to $*2$.

Upon examining the two alternative schedule STGs, it can be seen that, in the schedule of Figure 4(b), each iteration of the loop requires 9 cycles, while in Figure 4(c), each loop iteration requires 8 cycles. In order to compare the performance of the two designs, we calculated the *expected number of clock cycles* (ENC) [20] for both the schedules [2]. The schedule of Figure 4(b) has an ENC of 176.86 cycles, while the schedule of Figure 4(c) has an ENC of 157.32 cycles. That is because, in the schedule of Figure 4(b), the introduced dependency $*2 \rightarrow *1$ delays the execution of operations on the critical path ($+3 \rightarrow *1 \rightarrow -1 \rightarrow -2 \rightarrow *3 \rightarrow M1$). Finally, a performance optimized schedule derived using the same resource constraints, but without any transient power management, requires only *seven* cycles per loop iteration, while the best schedule possible with transient power management requires *eight* cycles per iteration. ∎

The above example illustrates that naive use of transient power management techniques can lead to significant performance overheads. It also demonstrates that the performance impact can be alleviated to some extent by exploiting relevant information that is available during the HLS process, including resource constraints and criticality of various operations in the behavior. However, it is not always possible to avoid performance overheads, and the use of transient power management can lead to a deterioration in performance. We next propose a technique to alleviate this limitation.

[2] ENC is a performance metric used for schedules of behavioral descriptions that contain significant control constructs in the form of conditionals and data-dependent loops.

2.3 Using Data Monitors to Alleviate Performance Overheads

In general, peak power constraints effectively impose a limit on the parallelism that is available to the synthesis tool. We propose a novel technique, based on the selective insertion of *data monitor* operations, to overcome these bottlenecks on parallelism to a large extent. From our experiments, we observed that, depending on the values that appear at the inputs of an embedded RTL component, its actual power consumption is frequently significantly lower than the worst-case value assumed during high-level synthesis. The peak power consumption values for RTL components that were presented in Table 1 were derived assuming no knowledge about the input values (*i.e.*, for the worst case input data). The percentile distribution of power consumption for the component `add_rc_16` is plotted in Figure 5. The plot indicates that very few input vectors exercise the worst case peak power consumption in the component. For example, consider the point P indicated in Figure 5. Point P indicates that, for 87.5% of the input vectors, the power dissipation in the component is bounded by $0.11mW$, which is less than half the worst case of $0.260mW$. Further, suppose that we divide the input space I for the component `add_rc_16` into two parts - P_1, which contains all input values for which the 8 MSBs of the two variables being added are 0, and $\bar{P_1}$, which contains all other input values. It turns out that, for all input vectors in P_1, the power consumption of the component `add_rc_16` is bounded by $0.11mW$, *i.e.*, all points in P_1 are contained to the left of point P in Figure 5. That is because, for the adder implementation we considered, for all vectors in P_1, there is no switching activity in a significant part of the circuit. Suppose that we are synthesizing a design under a peak power constraint that only allows one addition operation to be performed per cycle based on worst case peak power consumption of component `add_rc_16`. The above discussion implies that, most of the time, we could have actually allowed two addition operations to be scheduled in parallel.

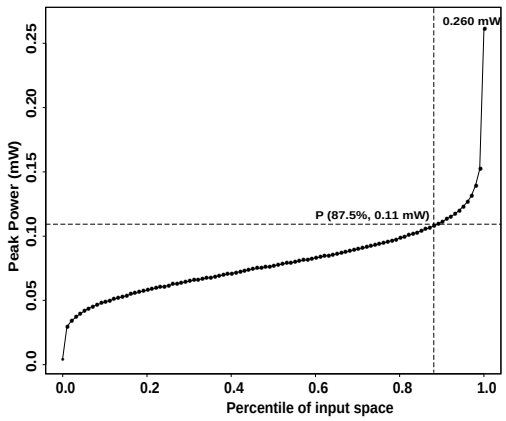

Figure 5: Percentile power distribution for component `add_rc_16` motivating the use of power modes

We alleviate the performance degradation that results from the pessimism in peak power consumption values of RTL components by (i) partitioning the input space for an RTL component (set of all possible input vectors) into two or more distinct sub-spaces, or *power modes*, and deriving a different bound on power consumption for each power mode [3], and (ii) modifying the behavior during high-level synthesis by selectively inserting *data monitor* operations that classify, at run-time, the values appearing at the component's inputs into the appropriate power mode. The concept that RTL components display different power behavior for different parts of the input space (*i.e.*, power modes) was originally proposed for improving the accuracy of average power estimation at the register-transfer and behavioral levels [21, 22]. Here, we are adapting the concept for use in a different context, and we focus on peak power of the component in each power mode as opposed to the average power consumption.

Example 4: Consider again the example behavior of Figure 4(a). Recall that the schedules of Figures 4(b) and 4(c) were derived using a peak power constraint of 2 mW, which resulted in sequential execution of multiplication operations $*1$ and $*2$ despite the availability of two multiplier resources. During the process of characterizing the multiplier for peak power consumption, we derived two separate power modes (based on the values at the higher order half of the input bits). The peak power consumption of the multiplier in the two power modes P_1 and $\bar{P_1}$ was found to be 0.562 mW and 1.238 mW, respectively. Based on the procedure described in Section 3, we inserted two data monitor operations at the inputs of operations $*1$ and $*2$. Based on the outputs of the data monitor operations, we introduce a condition dm in the behavior that evaluates to $True$ if and only if the multipliers implementing $*1$ and $*2$ would both be in power mode P_1. When the condition dm is $True$, we can schedule operations $*1$ and $*2$ in parallel. Otherwise, we need to enforce sequential execution as in Figures 4(b) and 4(c). The STG that implements this optimization is shown in Figure 4(d). There are two distinct paths that can be taken through the STG for each iteration of the loop - one represents the case when the condition dm evaluates to $True$ and requires 7 cycles, while the other requires 8 cycles. The expected number of clock cycles for the schedule of Figure 4(d) was computed to be 140.22 cycles, which is only 1.8% higher than the ENC of the performance optimized schedule for this behavior without any transient power management. ∎

The above example demonstrated that the use of data monitor conditions to exploit power modes of RTL components can eliminate or significantly reduce performance bottlenecks. Note that, the monitor operations themselves result in power, performance, and area overheads. Hence, it is critical that they be used in a judicious manner, as is done by the algorithms presented in Section 3.

The algorithms proposed in Section 3 for integrating transient power management into HLS consider the above issues, resulting in implementations with maximal reduction in power transients at minimum performance and area overheads. In addition, the enhancements made were kept independent of the underlying high-level synthesis algorithms in order to ensure easy integration into the framework of any HLS tool.

[3] In theory, it is possible to have any number of power modes. However, in practice, we found that the use of two or three power modes was the most beneficial since the overhead of identifying power modes at run-time based on input data increases, while the difference in the peak power for the different power modes decreases, as we increase the number of power modes considered.

3 Methodology and Algorithms

In this section, we describe our algorithm for synthesizing a circuit that incorporates transient power management capabilities while minimizing performance overheads (if any). Section 3.1 presents an overview of this framework, while Section 3.2 details the constituent steps.

3.1 Overview

Figure 6 outlines the basic features of our algorithm. Conventional design methodologies use a high-level synthesis flow to synthesize an RTL controller/datapath from a given behavior, design constraints, and optimization objectives. The design flow typically involves scheduling (Step **1**) and resource sharing (Step **3**) in conjunction with many optimizations targeting design metrics like area, performance, *etc.* As shown in the figure, our techniques (shaded boxes in gray) can be incorporated into such a flow as a two-phased plug-in (Steps **1a** and **2**). The added steps then judiciously resolve the interplay of synthesis choices and transient power issues.

The *transient power check* (Step **1a**) simply examines the schedulability of an operation op in the current state $state$ based on the given transient power constraints P_{max} and δP_{max} (see Section 3.2.1). If the constraints are satisfied, a value of *True* is returned on the Boolean variable SCHEDULABLE to the scheduler. When a *False* is returned to the scheduler, the scheduler removes op from its list of schedulable operations for the current state. The operation op can then be considered for scheduling in future states, where the aforementioned transient power constraints may not be violated.

The scheduled behavior description available at the end of Step **1** now obeys the user-specified transient power constraints. However, this schedule is pessimistic in the sense that it has been derived assuming that the execution of each operation in each state consumes maximum power under all conditions. This is clearly not true of all input data, and as seen earlier, such a hard constraint forms a bottleneck in the derivation of high-performance schedules. We, therefore, use *data monitor insertion* (Step **2**) to judiciously insert data monitor operations in the schedule and aggressively perform performance recovery (recall that the data monitor operations sample the data at the inputs of a functional unit and dynamically determine its power mode. During this rescheduling process, we again ensure that the transient power constraints are not violated.

The expanded flow of Step **2** is also shown in Figure 6. Step **2a** begins the loop that chooses the best k operations in the schedule for which the data monitors are to be inserted. This is done as follows. First, a short-list of k operations that are likely to yield the maximum performance improvements, if implemented with data monitors, is drawn up. The following criteria are used to assess an operation for inclusion in the short-list:

- The sensitivity of the ENC of the schedule to the operation: if an increase in the delay of the operation increases the total schedule length, the operation is deemed critical, and is a good candidate for data monitor insertion.
- The probability that the operation will execute in a power mode with a reduced peak power value.

Step **2b** (see Section 3.2.2) finds the k operations *best* satisfying these two characteristics. Set S_{mtr} represents a collection of these operations.

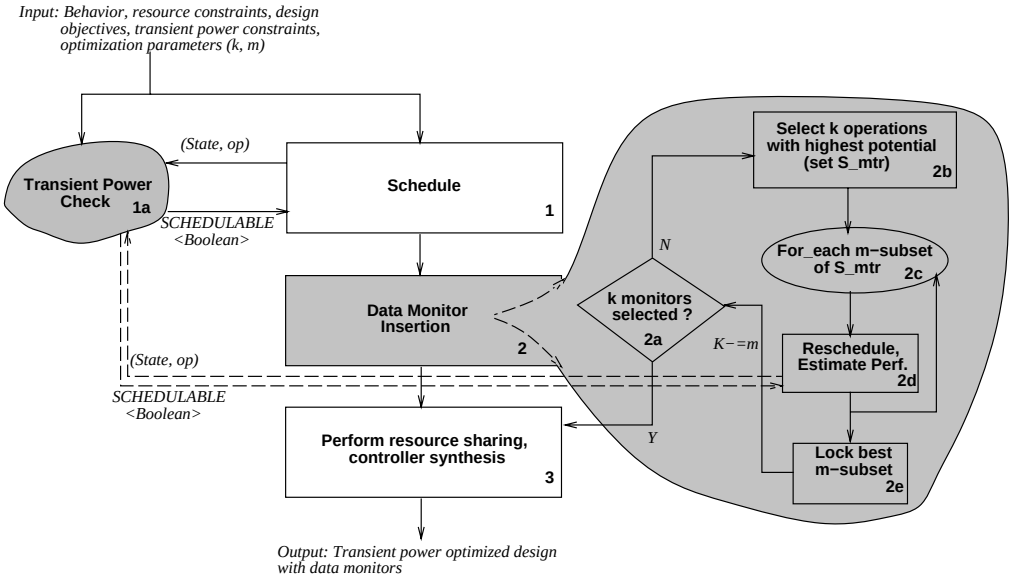

Figure 6: High-level synthesis flow enhanced with transient power management techniques

Steps **2c - 2e** then choose the "best" design that can be obtained by implementing m of these k operations with data monitors. Note that, the best design is not necessarily obtained by choosing the m operations with best individual impact. This is because, the performance impact of implementing multiple operations with data monitors is not a simple function of the performance impacts of individual operations. The overall performance of the design depends, in addition to the delay of individual operations, on the resource constraints, schedule, and the topology of the behavior.

In order to consider the cumulative impact of implementing multiple operations using data monitors, we need to consider all subsets, Sub, of S_{mtr}, of cardinality m. This involves the evaluation of kC_m subsets. Clearly, the complexity of this evaluation increases significantly with m, and so does the probability of actually determining the best set of operations to be implemented using data monitors. The parameter m, which is specified by the user, therefore, trades off the CPU time with the quality of the output design.

Once the "best" subset has been determined, we lock the operations implemented with data monitors and go back to Step **2a** to determine whether the user specified limit (k) on the number of data monitors has been reached. If the limit is reached, resource sharing and controller synthesis follow, resulting in a design optimized for transient power management.

3.2 Details

In this section, we detail selected aspects of our algorithm. Section 3.2.1 describes the transient power check (Step **1a**), and Section 3.2.2 describes the selection phase (Step **2b**) during the course of data monitor insertion. Finally, Section 3.2.3 details the rescheduling phase (Step **2d**).

3.2.1 Enforcing transient power constraints

In this section, we describe the transient power check performed during the course of scheduling. The transient power check examines a candidate operation op to be scheduled in a state S with respect to the following criteria:

- Does the scheduling of op in S violate the peak power threshold (P_{max}) in a cycle?
- For all states in the STG with transitions into state S (predecessor states), does this scheduling violate the specified cycle-to-cycle peak power differential (δP_{max})?

In order to perform this check, we need to estimate the power consumption profile of state S, considering all operations that have already been scheduled in S and the candidate operation op. As mentioned in Section 2, we use the following information in this phase:

1. The peak power consumption for each component in the RTL library. This information is derived once per technology, during library characterization. Known techniques for peak power estimation at the logic and transistor levels [6, 7] can be used for this purpose.
2. The set of RTL components that are active in state S. This is derived using the operation-to-component mapping (module selection) information for all the operations in S, and the number of register writes performed in state S.
3. The clock network capacitance, including the interconnect and clock buffer parasitics, estimated using high-level techniques such as those presented in [23].

Suppose that the set of RTL components active during state S is $Active\_Comps(S)$, and the peak power consumption for a library component c is given by $Peak\_Power(c)$. Also, suppose that the estimated clock network capacitance for the entire clock network is C_{clock}, and the clock frequency is f_{clock}. The peak power consumption of the state S is estimated using the formula:

$$\sum_{comp_i \in Active\_Comps(S)} Peak\_Power(comp_i) + C_{clock} \times V_{dd}^2 \times Clock\_Frac(S) \times f_{clock}$$

The term Clock_Frac(S) represents the estimated fraction of the clock network that switches in state S, and its value depends on the clock gating strategy employed. When no clock gating is employed, $Clock\_Frac(S) = 1, \forall S$. When the clock gating strategy suppresses the clocking of registers that do not need to load new values in a clock cycle, Clock_Frac(S) can be approximated as the ratio of the number of registered variables generated in S to the total number of registers (this information can be easily derived from the variable lifetime analysis performed in any high-level synthesis tool).

3.2.2 Selection

In this section, we present a simple technique for determining an initial set of k candidate operations for possible implementation with data monitors. We base our formulation on the following general observations.

- An operation that frequently sees low switching activity at its inputs (and, hence, consumes lower power) is a good candidate for implementation with data monitors. As mentioned earlier, this frequency depends on the statistics of the input values.
- An operation whose speedup has a higher impact on the ENC of the schedule must have a greater chance of selection (*i.e.*, operations on "critical paths" should be given higher priority).

Given an operation op, $E(\lambda(op, out))$, the expected length of the longest path connecting op to the primary output out, forms a good measure of operation criticality [24]. This is because, operations which have longer paths to primary outputs, when re-scheduled, are likely to influence the computation times of the primary output (and hence the ENC of the entire schedule). The probability of low switching activity of an operation, denoted by α, is computed through a simulation of the schedule by monitoring the variable values.

We multiply $E(\lambda(op, out))$ by α to obtain the measure given below.

$$potential(op) \quad = \quad \alpha * E(\lambda(op, out)) \tag{1}$$

The k operations with the highest potential values are selected as candidates for further analysis.

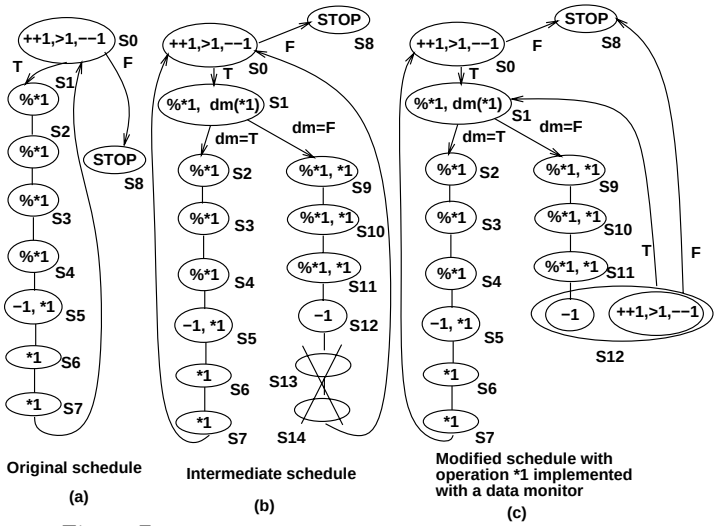

Figure 7: Example schedules shown before and rescheduling

3.2.3 Rescheduling

When an operation in the behavior is implemented with data monitors, its timing behavior is altered. The schedule, therefore, needs to be modified to reflect this change. It is important that the scheduling procedure be incremental, *i.e.*, start with the existing schedule information and modify the schedule so as to realize the maximal performance gains. Our procedure operates as a set of transformations to the existing schedule. These transformations include moving of operations across states, deletion of empty states, *etc.*, as outlined in [25]. The following example illustrates our scheduling technique.

Example 5: Consider the original schedule shown in Figure 7(a) derived (with transient power management) for the following input specification.

- An initial allocation constraint that does not include any functional units with data monitors.
- In any given iteration, the dependencies are as follows. -1 is schedulable if and only if $\%*1$ (modular multiplication operation) has been scheduled. Likewise, $*1$ and $\%*1$ are schedulable if and only if > 1 and $++1$ have been scheduled. $++1$ and > 1 are schedulable if and only if $*1$ of the previous iteration terminates.
- Operations $*1$ and $\% * 1$ cannot be scheduled in the same cycle since the sum of the peak powers associated with their respective functional units violate the peak power constraint.

Observe that operation $*1$ is schedulable in state $S1$ but for the transient power constraint. Now, assume that operation $*1$ is now deemed a suitable candidate for implementation with a data monitor. Then, a scheduler has to perform the basic actions enumerated below to incorporate this solution.

1. The addition of data monitor operation $mtr(*1)$ creates an alternative path ($S1 \rightarrow S9 \rightarrow S10 \rightarrow S11 \rightarrow S12 \rightarrow S13 \rightarrow S14 \rightarrow S0$ in Figure 7(b)) that the schedule can take. Since this represents the power mode for *1 with reduced peak power, *1 can be scheduled in parallel with $\% * 1$ in states $S9, S10,$ and $S11$. Consequently, $*1$ is absent from states $S12 \rightarrow S14$.
2. Empty states $S13$ and $S14$ can be deleted since the functionality of the schedule will still be preserved.
3. Since $*1$ finishes earlier in the $S1 \rightarrow S9 \rightarrow ... \rightarrow S12$ path, operations dependent on its output become immediately schedulable subject to the input allocation constraint. For example, we can advance the operations in state $S0$ to state $S12$. The composite state $S12$ shown in Figure 7(c) captures this scenario. This move also adds the additional edges from $S12$ to $S1$ and $S8$.

The above re-scheduling steps are no different from a set of simple rearrangements across dependencies for a piece of code. They extract whatever parallelism remains in the schedule without affecting the execution semantics. These tasks map directly to the transformations employed in scheduling algorithms like Percolation Scheduling [25]. When the scheduler is called, these transformations operate on the original schedule to derive intermediate representations leading to the final output as shown in Figure 7(c). ∎

4 Experimental Results

We applied the proposed transient power management techniques to several example benchmarks and evaluated them using a commercial design flow. We generated structural RTL implementations (controller and datapath) for each benchmark, without and with the use of transient power management, using identical resource and clock period constraints for both cases. We refer to these as the *original* and *transient power optimized* RTL implementations, respectively. For each design, the expected number of clock cycles (ENC) metric, computed as described in [20], was used to compare the performance of the synthesized RTL implementations. Logic synthesis was performed using Synopsys Design Compiler [17] and the design was technology mapped to the NEC CB-C9VX 0.35 micron technology library [18]. We performed experiments to evaluate the transient power reduction obtained through the use of the proposed techniques. Power estimation was performed through simulation using input sequences that were specified for functional verification (*i.e.*, they provide high behavioral and RTL code and value coverage). NEC's in-house gate-level power estimation tool [19] was used to generate a cycle-by-cycle report, which was post-processed to compute the peak power as well as the peak power differential.

We evaluated the proposed technique using ten example benchmarks. Example *Poly* represents the computation of a polynomial. *PPsum* is a parallel prefix sum routine used in address calculations. *SeqDev* is a sequential implementation of the standard integer division algorithm, while *Findmin* finds the minimum of a set of given values. *Quad* is a subroutine used in the computation of Gauss-Jacobi abscissas and weights, and *SpHarm* computes spherical harmonics for solving wave equations. *Matrix* is an algorithm for matrix multiplication and *Wavelet*, *FIR* and *DCT* are well known signal processing algorithms.

Table 2 summarizes the results of our experiments. Major columns **Peak Power, Peak Power Differential, ENC, Avg. P.O.** and **A.O.** report the expected number of clock cycles (in tens of cycles), peak power consumed in any cycle (in milliwatts), the peak cycle-to-cycle power differential (in milliwatts), the average power consumption overhead that is incurred due to transient power management and the area overhead, respectively. Minor columns **Orig** and **TP-Opt** represent the original and transient power optimized RTL implementations for each example.

Circuit	Peak Power (mW)		Peak Power Differential (mW)		ENC (tens of cycles)		Avg. P.O.	A.O.
	Orig	TP-Opt	Orig	TP-Opt	Orig	TP-Opt		
Poly	5.1	4.1 (-19.61%)	1.1	0.7 (-36.36%)	289	295 (2.07%)	0.74%	3.72%
PPsum	32.6	25.8 (-20.85%)	9.3	4.4 (-52.69%)	256	262 (2.34%)	1.09%	4.14%
Seqdiv	11.3	8.0 (-29.20%)	2.9	1.6 (-44.83%)	155	159 (2.58%)	0.67%	2.98%
Findmin	7.7	5.2 (-32.46%)	3.1	1.3 (-58.06%)	409	422 (3.18%)	0.55%	5.03%
Quad	40.6	30.5 (-24.87%)	8.4	4.9 (-41.67%)	783	799 (2.04%)	0.97%	6.81%
SpHarm	13.3	9.6 (-27.82%)	3.6	2.7 (-25.0%)	927	941 (1.51%)	1.14%	3.16%
Matrix	13.2	10.9 (-17.42%)	4.4	2.9 (-34.09%)	312	319 (2.24%)	1.06%	2.52%
Wavelet	8.2	6.1 (-25.61%)	3.1	1.7 (-45.16%)	1564	1595 (1.98%)	0.87%	3.47%
FIR	5.2	4.0 (-23.07%)	2.1	1.3 (-38.10%)	902	910 (0.88%)	0.91%	5.66%
DCT	13.4	9.7 (-27.61%)	4.7	2.6 (-44.68%)	484	493 (1.86%)	1.21%	7.13%

Table 2: Transient power, performance, and area results

The results indicate that the proposed transient power management technique achieves up to 32% (average of 25%) reduction in peak power consumption in the synthesized circuits over a state-of-the-art high level synthesis tool. The peak cycle-to-cycle power differential was also reduced by up to 58% (average of 42%). These benefits come with minimal performance, area and average power dissipation overheads (average of 2.07%, 4.5% and 0.92% respectively). These results reflect the overheads for a complete RTL implementation including data monitors, additional control logic, and multiplexers.

Since our transient power management technique consists of two optimizations (*i.e.*, transient power check and data monitor insertion) applied together, we investigated the effect of these optimizations when applied individually. We performed an additional experiment with examples *Seqdiv* and *Quad*. We generated and compared implementations of these benchmarks without and with the use of data monitor units. While the peak power and the power differential remained almost the same for the two implementations, we found that data monitor insertion decreased the performance overhead (in terms of ENC) from 9.6% and 8.5% to 2.58% and 2.04%, respectively, for the two benchmarks, compared to designs without any transient power management.

In our experiments, we found that data monitor insertion does not cause any overhead in terms of average power consumption. While monitor units dissipate some power, the use of these units leads to a decrease in overall average power consumption due to the following reason. Data monitor insertion leads to a reduction in the number of clock cycles to complete the design's computation, which translates into reduced switched capacitance in the clock network and registers. This reduction compensates for the power consumed by the monitor units themselves, thus eliminating any average power consumption overhead. Monitor units also cause a negligible overhead in terms of implementation area. For the above two examples, the area overhead due to monitor insertion was found to be 1.98% and 1.81%, respectively.

These results clearly suggest that incorporating transient power management techniques into high-level design results in architectures that have significantly improved power transient profiles with negligible performance, area, and average power consumption overheads.

5 Conclusions

We have presented techniques to manage power transients through high level synthesis. We have shown that significant impact can be made on the transient power profile of the resulting implementation by considering power transients during generation of the architecture. We have demonstrated how to incorporate our technique into a high-level design flow in order to minimize the associated performance overheads. Our technique does not assume any specific high-level synthesis tools/algorithms and can be plugged into any high-level synthesis system. Results show that RTL implementations synthesized through the use of our technique have significantly lower peak power consumption and cycle-to-cycle power differential.

References

[1] A. P. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design*. Kluwer Academic Publishers, Norwell, MA, 1995.

[2] J. Rabaey and M. Pedram (Editors), *Low Power Design Methodologies*. Kluwer Academic Publishers, Norwell, MA, 1996.

[3] A. Raghunathan, N. K. Jha, and S. Dey, *High-level Power Analysis and Optimization*. Kluwer Academic Publishers, Norwell, MA, 1998.

[4] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Academic Publishers, Norwell, MA, 1997.

[5] E. Macii, M. Pedram, and F. Somenzi, "High-level power modeling, estimation, and optimization," in *Proc. Design Automation Conf.*, pp. 504–511, June 1997.

[6] C. Y. Wang and K. Roy, "Maximum power estimation for CMOS circuits using deterministic and statistical techniques," *IEEE Trans. VLSI Systems*, pp. 134–140, Mar. 1998.

[7] Y. M. Jiang, A. Krstic, and K. T. Cheng, "Estimation of maximum instantaneous current through supply lines for CMOS circuits," *IEEE Trans. VLSI Systems*, vol. 8, pp. 61–73, Feb. 2000.

[8] Y. S. Chang, S. K. Gupta, and M. A. Breuer, "Analysis of Ground Bounce in Deep Sub-Micron Circuits," in *Proc. VLSI Test Symp.*, pp. 110–116, Apr. 1997.

[9] Y. M. Jiang, A. Krstic, and K. T. Cheng, "Dynamic timing analysis considering power supply noise effects," in *Proc. Int. Symp. Quality of Electronic Design*, pp. 137–143, Mar. 2000.

[10] RailMill, Synopsys Inc. (http://www.synopsys.com).

[11] VoltageStorm SoC, Simplex Solutions Inc. (http://www.simplex.com).

[12] M. D. Pant, P. Pant, D. S. Wills, and V. Tiwari, "Architectural Solution for the Inductive Noise Problem due to Clock-Gating," in *Proc. Int. Symp. Low Power Electronics & Design*, pp. 255–257, Aug. 1999.

[13] M. D. Pant, P. Pant, D. S. Wills, and V. Tiwari, "Inductive noise reduction at the architectural level," in *Proc. Int. Conf. VLSI Design*, pp. 162–167, Jan. 2000.

[14] R. San Martin, and J. P. Knight, "Power Profiler: Optimizing ASICs power consumption at the behavioral level", in *Proc. Design Automation Conf.*, pp. 42–47, June 1995.

[15] C. F. Chiasserini and R. R. Rao, "Energy efficient battery management," in *Proc. IEEE Infocomm*, pp. 396–403, Mar. 2000.

[16] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi, "Extending lifetime of portable systems by battery scheduling," in *Proc. Design Automation & Test Europe (DATE) Conf.*, pp. 197–201, Mar. 2001.

[17] Design Compiler, Synopsys Inc. (http://www.synopsys.com)

[18] *CB-C9 Family VX/VM Type 0.35um CMOS CBIC Users Manual*. NEC Electronics, Inc., Sept. 1998.

[19] *OpenCAD V 5 Users Manual*. NEC Electronics, Inc., Sep. 1997.

[20] S. Bhattacharya, S. Dey, and F. Brglez, "Performance analysis and optimization of schedules for conditional and loop-intensive specifications," in *Proc. Design Automation Conf.*, pp. 491–496, June 1994.

[21] L. Benini, A. Bogliolo, M. Favalli, and G. De Micheli, "Regression models for behavioral power estimation," in *Proc. Int. Wkshp. Power & Timing Modeling, Optimization, and Simulation*, 1996.

[22] N. R. Potlapally, A. Raghunathan, G. Lakshminarayana, M. S. Hsiao, and S. T. Chakradhar, "Accurate macro-modeling techniques for complex RTL components," in *Proc. Int. Conf. VLSI Design*, pp. 235–241, Jan. 2001.

[23] R. Mehra and J. Rabaey, "Behavioral level power estimation and exploration," in *Proc. Int. Wkshp. Low Power Design*, pp. 197–202, Apr. 1994.

[24] G. Lakshminarayana, K. S. Khouri and N. K. Jha, "WAVESCHED: A novel scheduling technique for control-flow intensive behavioral descriptions" in *Proc. Intl. Conf. Computer Aided Design*, pp. 244–250, Nov. 1997.

[25] R. Potasman, J. Lis, A. Nicolau, and D. Gajski, "Percolation based synthesis," in *Proc. Design Automation Conf.*, pp. 444–449, June 1990.