Symbolic Synthesis of Clock-Gating Logic for Power Optimization of Synchronous Controllers

L. BENINI and G. DE MICHELI
Stanford University
and
E. MACII, M. PONCINO, and R. SCARSI
Politecnico di Torino

Recent results have shown that dynamic power management is effective in reducing the total power consumption of sequential circuits. In this paper, we propose a bottom-up approach for the automatic extraction and synthesis of dynamic power management circuitry starting from structural logic-level specifications. Our techniques leverage the compact BDD-based representation of Boolean and pseudo-Boolean functions to detect idle conditions where the clock can be stopped without compromising functional correctness. Moreover, symbolic techniques allow accurate probabilistic computations; in particular, they enable the use of non-equiprobable primary input distributions, a key step in the construction of models that match the behavior of real hardware devices with a high degree of fidelity. The results are encouraging, since power savings of up to 34% have been obtained on standard benchmark circuits.

Categories and Subject Descriptors: B.6.1 [**Logic Design**]: Design Styles—*Sequential circuits*; B.6.3 [**Logic Design**]: Design Aids—*Automatic synthesis*; *Optimization*

General Terms: Design

1. INTRODUCTION

Dynamic power management is probably the single most successful technique for power minimization in digital systems. A number of examples of handcrafted power-management schemes can be found in the literature [Biggs et al. 1994; Debnath and Debnath 1995] and are actually implemented in well-known commercial devices. Automatic detection of good

Authors' addresses: L. Benini and G. D. Micheli, Computer Systems Laboratory, Stanford University, Stanford, CA 94305; email: luca@azur.stanford.edu; nanni@galileo.stanford.edu; E. Macii, M. Poncino, and R. Scarsi, Dip. di Automatica e Informatica, Politecnico di Torino, Torino, Italy 10129; email: {enrico; poncino; scarsi}@athena.polito.it .

power management opportunities and synthesis of the required circuitry is a relatively new discipline, but several investigations [Alidina et al. 1994; Monteiro et al. 1995; Benini and De Micheli 1996] have shown that CAD tools may be effective in this area as well. In this paper, we propose new techniques for the automatic synthesis of power-managed controllers starting from structural specifications (i.e., sequential logic networks).

The key idea in power management is that unused parts of a complex design can be shut down during system operation. Shut-down can be achieved by zeroing the voltage supply or, in static synchronous CMOS logic, by stopping the clock. From a technology standpoint, supply shut-down has several serious drawbacks, such as current spikes on power and ground lines, loss of information stored in memory cells, and power-up delays. Clock-gating has a reduced technological impact, and can be successfully employed in a wider range of cases. However, it is still a relatively aggressive technique that raises concerns on possible timing violations and testability degradation. An even lower-impact approach to power management requires the use of *enabled* flip-flops, sequential primitives where an additional input signal determines whether the stored values must be updated or held constant.

Supply shut-down, clock-gating, and flip-flop disabling span the trade-off between achievable savings and technology complications. On one extreme, power supply turn-off is the hardest to implement, but it completely eliminates power dissipation during shut-down. On the other extreme, flip-flop disabling is minimally intrusive, but it does not reduce clock power and leakage power. We exploit clock-gating for implementing our power management strategy because we believe that it represents a good trade-off between aggressiveness and achievable power savings. Moreover, our strategy is applicable to register disabling with no modification to the core algorithms, and one minor architectural modification: The same signal used to stop the clock can be used to disable the flip-flops.

From a methodology viewpoint, we adopt a bottom-up approach. The starting point is a synchronous circuit described at the logic level by a netlist containing combinational gate primitives and flip-flops. This is in sharp contrast to top-down techniques, such as those proposed in Benini and De Micheli [1996], where it is assumed that the initial specification of the controller is given as a state transition graph (STG). Given the STG of the circuit to be synthesized, conditions under which the next state and the output signals do not change are identified. In such conditions, the clock is disabled, since no useful computation is performed by the circuit, thus avoiding some node switching that may cause useless power dissipation. In Benini and De Micheli [1996] techniques are described for calculating the conditions under which the clock can be stopped, as well as exact and approximate algorithms for synthesizing the STG description with embedded clock-gating mechanisms.

There are two major limitations in the method we proposed in Benini and De Micheli [1996]. First, the tool can handle only small STGs. Controllers which are automatically synthesized from high-level specifications may

have millions of states; explicitly enumerating all of them, as required by the algorithms of Benini and De Micheli [1996], may thus simply be unacceptable. Second, the computation of the clock-gating conditions is done without considering the controller as a piece of a more complex system but, rather, as a component running in isolation. This implies an inevitable loss of information which could have been successfully exploited to achieve a more effective global optimization.

In this paper, we address both the issues pointed out above. The problem of optimizing larger controllers is tackled in two ways. First, by resorting to symbolic data structures, that is, BDDs [Bryant 1986] and ADDs [Bahar et al. 1997] to simplify the representation of the clock-gating conditions as well as the calculation of the probability of the *activation function*. Second, by employing a new and efficient algorithm for the search of the optimal activation function that is able to dynamically estimate the savings, in terms of power consumption, that different realizations of the clock-gating logic produce on the circuit.

Regarding the calculation of the activation function, in Benini and De Micheli [1996] it has been proposed that it can be computed by looking at the sequential circuit in isolation; that is, as a self-standing computing element, thus implicitly making the assumption of equiprobable input statistics. The control-dominated systems we are considering here are usually interacting with other controllers and/or data-paths; this may pose some constraints on the signals that appear at the circuits I/O interfaces. One way of properly modeling the influence of the environment on the behavior of a design is through non-equiprobable primary input statistics. In addition, even when the circuit's primary inputs are totally independent from the other components, there may be cases in which assuming a 0.5 input transition probability is not realistic (think, for example, of the external reset signal of a microcontroller). We therefore propose to use non-equiprobable primary input probability distributions in the computation of the activation function. Such distributions can be determined from the knowledge of the specific functionalities associated to the various input signals or, alternatively, by performing system-level simulation over a large number of clock periods. Experimental data are provided to show the impact that the accurate knowledge of the primary input statistics can have on both the total probability of the activation function and the power savings obtainable through implementation of the gated-clock architecture.

The applicability of the proposed algorithms strongly depends on the amount of "idleness" in the target specifications. A circuit is idle when either its inputs do not change very often or when input transitions do not propagate to the outputs. The first form of idleness depends on input statistics, while the second depends on both input statistics and functionality of the target circuit. We have performed extensive experimentation on the entire `Iscas'89` sequential benchmarks suite [Brglez 1989] assuming independent and equiprobable inputs. With such input statistics, several benchmarks are almost never idle, and cannot be optimized with our technique. On the other hand, our solution produces sizable power savings

on all benchmarks that are idle for a significant fraction of the operation time. Average power savings of 17% have been obtained at limited area and delay costs (11% and 8%, respectively). Moreover, results show that our approach is capable of exploiting non-uniform input statistics for achieving even higher power reductions.

Other bottom-up power optimization techniques have been proposed in the literature. We investigate analogies and differences between our approach and previous ones, namely, the precomputation-based architectures of Alidina et al. [1994] and Monteiro et al. [1995], and we compare the power optimizations that can be achieved when different classes of circuits are considered.

The paper is organized as follows: Section 2 provides definitions for subsequent usage. In Section 3, we formally state the problem; summarize the gated-clock architecture we adopt; and briefly describe our previous work on the subject, the results of which are reported in Benini and De Micheli [1996]. Section 4 constitutes the core of the paper: It describes the symbolic algorithm for automatically synthesizing the activation function, and discusses most of the issues related to its implementation. In Section 5 we outline extensions to the proposed technique; in addition, we investigate the relationship between our approach and the precomputation architectures proposed by Alidina et al. in [Alidina et al. 1994; Monteiro et al. 1995]. Section 6 is dedicated to experimental results and, finally, Section 7 provides concluding remarks.

2. BACKGROUND

We assume the reader to be familiar with the basic concepts of Boolean functions and with the data structures commonly used for the symbolic manipulation of such functions, that is, the binary decision diagrams (BDDs). Background material on this subject can be found in Bryant [1986]. We review here two Boolean operators essential for our purposes. Given a single-output Boolean function, $f(x_1, \ldots, x_n)$, the *positive* and the *negative cofactors* of f, with respect to variable x_i, are defined as:

$$f_{x_i} = f(x_1, \ldots, x_i = 1, \ldots, x_n) \text{ and } f_{x'_i} = f(x_1, s, x_i = 0, \ldots, x_n)$$

The *existential* and the *universal abstraction* of f with respect to x_i are defined as:

$$\exists_{x_i} f = f_{x_i} + f_{x'_i} \text{ and } \forall_{x_i} f = f_{x_i} \cdot f_{x'_i}$$

2.1 Sequential Circuits and Finite State Machines

We consider synchronous sequential circuits composed of combinational gates and edge-triggered flip-flops. All flip-flops are controlled by the same clock, and we assume that they are all resetable to a given state. Associated with a sequential circuit is an encoded, Mealy-type, finite state machine (FSM) that describes the behavior of the circuit. An FSM, M, is a 6-tuple

$(X, Z, S, s^0, \delta, \lambda)$, where X is the input alphabet, Z is the output alphabet, S is the finite set of states of the machine, s^0 is the reset state, $\delta(x, s)$ is the next state function ($\delta : X \times S \rightarrow S$), and $\lambda(x, s)$ is the output function ($\lambda : X \times S \rightarrow Z$). The sets X, Z, and S are non-empty. Boolean functions δ and λ have multiple outputs: They implicitly define the state transition graph (STG) of the given FSM.

Elements $x \in X$ are encoded by vectors of n Boolean variables, $x_1, \ldots, x_n$, called input variables. Similarly, present states $s \in S$ are encoded by k Boolean variables, $s_1, \ldots, s_k$, called present state variables, elements $z \in Z$ are encoded by vectors of m Boolean variables, $z_1, \ldots, z_m$, called output variables, and next states $t \in S$ are encoded by k Boolean variables, $t_1, \ldots, t_k$, called next state variables.

Mealy-type FSMs produce the output z when the edge labeled x is traversed, while Moore-type FSMs produce the output z when a given state s is reached. Therefore, in Moore-type machines, states, rather than edges, are labeled with output symbols.

2.2 Pseudo-Boolean Functions and ADDs

A n-input pseudo-Boolean function, $f : B^n \rightarrow S$, is a mapping from a n-dimensional Boolean space to a finite set of elements S. Different data-structures have been proposed for storing and manipulating functions of this type. In this work, we use the algebraic decision diagrams (ADDs) [Bahar et al. 1997].

The most important operators for efficient manipulation of the ADDs are: ITE, APPLY, and ABSTRACT.

ITE takes three arguments: f, an ADD restricted to have only 0 or 1 as terminal values, and g and h, generic ADDs. It is defined by:

$$\mathtt{ITE}(f, g, h) = f \cdot g + f' \cdot h$$

APPLY takes one operator, *op* (e.g., $+$, $-$, $\times$), and two operand ADDs as arguments; it applies *op* to all corresponding elements of the two operands and returns the resulting ADD.

ABSTRACT reduces the dimensionality of its argument function through existential arithmetic abstraction of some variables. Let u be the support of a pseudo-Boolean function $f(u)$, and let x and y be two subsets of u such that $x \cup y = u$. The arithmetic existential abstraction of x from $f(u)$ with respect to the arithmetic sum is defined as:

$$\backslash_x^+ f(u) = \sum_x f(u)$$

This definition tells that, instead of taking the Boolean sum of all the cofactors associated with the minterms of the x-variables, as in Boolean existential abstraction, the ABSTRACT operator computes precisely the

arithmetic sum. Similarly, the arithmetic existential abstraction of x with respect to the MAX operator is defined as:

$$\backslash_x^{\mathrm{MAX}} f(u) = \max_x f(u)$$

2.3 Probabilistic Analysis of an FSM

The probabilistic behavior of a finite state machine can be studied by regarding its transition structure as a Markov chain. It is sufficient to label each out-going edge of each state with the probability of the FSM making that particular transition in order to obtain a discrete-parameter Markov chain. On the other hand, studying the behavior of the Markov chain, that is, computing the state occupation probabilities, is related to performing the reachability analysis of an FSM.

Given the transition relation of the FSM representing the sequential circuit, it is possible to compute the vector $\mathbf{p}$ whose entries p_s tell us the steady-state probability of the FSM to be in state s. For mid-sized circuits the calculation can be carried out in an exact fashion using the ADD-based procedures of Hachtel et al. [1996]; for large sequential networks, the approximate techniques of Tsui et al. [1995] must be employed. In both cases, complex primary input probability distributions can be specified in order to have more detailed hardware modeling options. In this work we rely on the performance of these algorithms to overcome some of the limitations which appeared in the implementation of the optimization methods of Benini and De Micheli [1996].

3. REDUCING POWER THROUGH CLOCK-GATING

A gated-clock circuit is obtained by modifying the architecture depicted in Figure 1(a). We define a signal called *activation function* (F_a) that selectively stops the local clock of the circuit, when the machine does not perform state or output transitions. When $F_a = 1$ the clock will be stopped.

The sequential circuit with clock-gating logic is shown in Figure 1(b). The block labeled "L" represents a latch, transparent when the global clock signal CLK is low. The latch is needed for correct behavior, because F_a may have glitches that must not propagate to the AND gate when the global clock is high. Moreover, notice that the delay of the logic for the computation of F_a may be on the critical path of the circuit, and its effect must be taken into account during timing verification.

The activation function is a combinational logic block with the primary inputs and the state lines of the circuit as input variables. No external information is used; the only input data for our algorithm are the gate-level description of the circuit and the probability distributions of the input signals.

3.1 Idle Conditions

Given the gate-level description of the circuit and its probabilistic model, we first want to identify the idle conditions when the clock may be stopped.

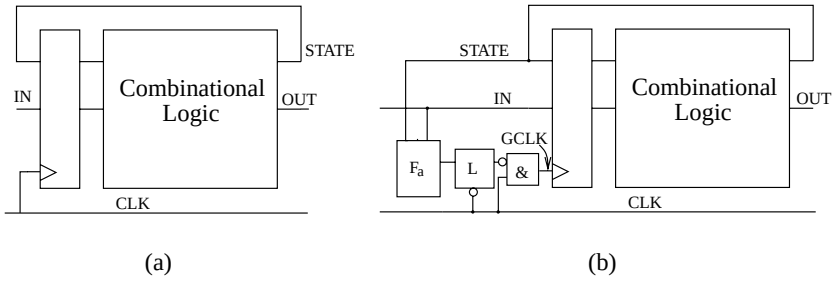

Fig. 1. (a) A sequential circuit. (b) Gated-clock version. .

A gate-level netlist is the implementation of a sequential circuit that can be represented by a finite state machine. In the following, we will refer to the FSM associated to the netlist to clarify some important points.

Determining the idle conditions is a simple task for circuits implementing Moore-type FSMs. When the present state and the inputs are such that the next state does not change, the Moore FSM is idle; in symbols: $\delta(x, s) = s$ (i.e., the self-loops). Unfortunately, this property does not hold for Mealy FSMs.

Consider, for example, the fragment of a Mealy FSM shown in Figure 2. State S_2 has a self-loop, but we cannot stop the clock when we observe the code of S_2 and inputs 11 on the state and input lines. The reason is that the self-loop does not change the next state, but it *changes the output* if the previous transition was $S_1 \rightarrow S_2$. Intuitively, the self-loop on S_2 becomes an idle condition only if it is taken for two consecutive clock cycles. In contrast, the self-loop on S_3 is an idle condition, because every incoming edge of S_3 has the same output, and knowing that the next state is S_3 provides enough information to infer the output value.

This observation has been formalized in Benini and De Micheli [1996] where the states of a Mealy-type FSM have been divided into two classes. States like S_2 where self-loops are not idle conditions (unless taken twice), are called *Mealy-states*, while states like S_3 are called *Moore-states*. For Mealy-states it is not possible to stop the clock of the circuit just by observing the state and input lines. In Benini and De Micheli [1996], an algorithm is described that operates on the STG of the FSM to transform Mealy-states into Moore-states, thus allowing the exploitation of more self-loops as idle conditions where the clock can be stopped.

In this work, we want to extract the idle conditions available in the synchronous network implementing the FSM. It is generally computationally infeasible to explicitly handle the STG representation of large sequential circuits. As a consequence, the transformation from Mealy-states to Moore-states is not applicable and we must restrict ourselves to the Moore-states of the Mealy FSMs.

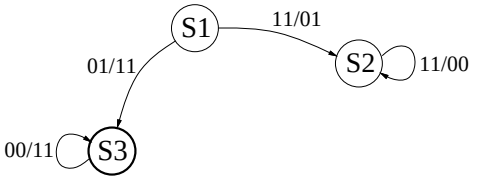

Fig. 2. Fragment of a Mealy FSM.

3.2 Activation Function

Given an FSM implemented by a synchronous logic network, we would like to find the self-loops of the Moore-states. Such self-loops are uniquely identified by the present state and input values and represent the set of idle conditions that may be exploited to stop the clock. For example, for the FSM fragment in Figure 2, the only useful idle condition is the self-loop on S_3 (identified by input value 00 and state value S_3).

The *complete activation function* $F_a(x, s)$ is defined as the union of all the self-loops of the Moore-states (x and s are the input and the state variables, respectively). The set of all self-loops in the FSM includes F_a, because it contains also the self-loops of Mealy-states.

The identification of the Moore-states can be performed implicitly (i.e., without extracting the STG) by a procedure that requires a single *unrolling* of the sequential circuit, that is, duplicating the combinational logic to represent two consecutive time frames, as shown in Figure 3.

There are two cascaded logic blocks: The inputs of the first combinational block are x and s, representing respectively primary and state inputs. The outputs are z. The next state outputs of the first block are fed into the state inputs of the second block (the signals t). The primary input values in the second block are represented by x^+, while the output of the second block are z^+.

With this model, finding the Moore-states is quite simple. For a Moore state t, the following property holds: If in the second combinational logic block the state transition is a self-loop (i.e., $\delta(x^+, t) = t$), for each state transition $s \rightarrow t$ in the first block, the output $z = \lambda(x, s)$ and $z^+ = \lambda(x^+, t)$ are the same. Intuitively, this property expresses the requirement that every incoming edge for state t has the same output value, but we are interested only in states with self-loops, because otherwise no idle conditions are available. Finding all states for which the condition is true is equivalent to finding all Moore-states with self-loops, but no STG extraction is required. This procedure lends itself to an elegant symbolic formulation that will be described in the next section.

4. SYNTHESIS OF THE CLOCK-GATING LOGIC

In this section, we describe a symbolic algorithm to generate the clock-gating circuitry and we discuss the issues related to the global optimiza-

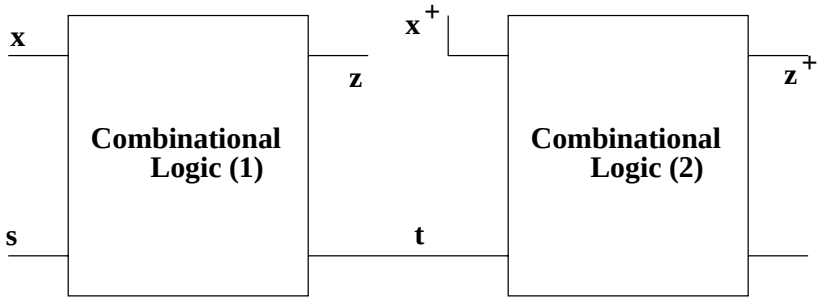

Fig. 3. Unrolling of an FSM.

tions that are enabled by the presence of the new logic into the circuit. The expression giving the activation function F_a in symbolic form is the following:

$$F_a(x^+, t) = A \cdot \forall_{x,s}(B^+C) \tag{1}$$

The term $A = \Pi_{i=1}^{k}(\delta_i(x^+, t) \equiv t_i)$ imposes the condition that, in the second frame of the unrolled circuit, the machine has a self-loop. This is expressed by having each present state variable t_i identical to the next state function $\delta_i(x^+, t)$.

The term $B = \Pi_{i=1}^{m}(\lambda_i(x, s) \equiv \lambda_i(x^+, t))$ describes the constraint on the output values. Since we are detecting Moore-states, we require that the output values of the incoming edge and the self-loop are the same. Notice that the unrolling implies the use of different variables for the two frames of the unrolled circuit.

The term $C = (\Pi_{i=1}^{k}(\delta_i(x, s) \equiv t_i))'$ is OR-ed with the second term to express the fact that the equality of the outputs in two frames does not need to be enforced for transitions not in the next state functions of the FSM.

The resulting activation function F_a is expressed in terms of the auxiliary variables (x^+, t) for convenience, and can be easily re-expressed as a function of the inputs x and present states s by variable renaming.

4.1 Reducing the Activation Function

Direct application of Equation 1 yields, in the general case, functions whose power dissipation may partially mask off the potential power savings. Therefore, it is mandatory to develop a systematic method to reduce the implementation of F_a, while keeping as high as possible the probability of its ON-set.

To reach this objective, we proceed as follows: First, we build a pseudo-Boolean function, P_{F_a}, which implicitly represents the probability of the minterms in the ON-set of F_a. Then, we iteratively remove from F_a some of its ON-set minterms until a given cost criterion breaks the loop. Clearly,

```
procedure Reduce_Fa(F_a, P_Fa) {
    F_a^Best = F_a; P_Best = P_Fa;
    F_a^Curr = F_a; P_Curr = P_Fa;
    Best_Cost = Compute_Cost (F_a^Curr);
    while (not Stop_Test(F_a^Best,P_Best)) {
        F_a^Curr = Prune_Fa (F_a^Curr);
        Curr_Cost = Compute_Cost (F_a^Curr);
        if (Curr_Cost ≤ Best_Cost) {
            F_a^Best = F_a^Curr;
            P_Best = P_Curr;
            Best_Cost = Curr_Cost;
        }
    }
    return (F_a^Best);
}
```

Fig. 4. The Reduce_Fa Algorithm.

both the minterm removal and the stopping condition must be guided by a combination of the size improvement in the implementation of F_a and the probability decrease of the ON-set of F_a. We have devised several heuristics that help in keeping these two requirements together.

4.1.1 *Computing* P_{F_a}. Let us assume the pseudo-Boolean functions of the primary input probabilities, $P_{inputs}(x)$, and of the state occupation probabilities, $P_{states}(s)$, to be known (for the details on how these two functions can be computed implicitly using ADDs the reader may refer to Hachtel et al. [1996]). The probability P_{F_a} can be simply obtained as:

$$P_{F_a}(x, s) = P_{inputs}(x) \cdot P_{states}(s) \cdot F_a(x, s) \tag{2}$$

Obviously, P_{F_a} is stored as an ADD, whose paths from the root to the leaves give the probability of all the minterms in the ON-set of F_a. The total probability of the ON-set of F_a, (i.e., a real number) can then be computed by applying the ABSTRACT operator:

$$PROB(F_a) = \backslash^{+}_{x,s} P_{F_a}(x, s) \tag{3}$$

4.1.2 *Iterative Reduction of* F_a. Given the activation function, F_a, and its probability function, P_{F_a}, the reduction algorithm iteratively prunes some of the minterms of F_a until an acceptable solution is found. The pseudocode of the procedure is shown in Figure 4.

As mentioned earlier, the objective of procedure Reduce_Fa is to determine a new activation function, F_a^{Best}, which is contained into the original F_a, has a high global probability, and is less costly (in terms of both power and area) if compared to F_a. Three main routines are called inside

`Reduce_Fa`: `Prune_Fa`, `Compute_Cost`, and `Stop_Test`. We discuss them in this order.

4.1.3 *Heuristics to Prune Function F_a.* We have experimented with two different heuristics for pruning the activation function.

The first one is based on the idea of removing from the ON-set of F_a the minterms or the cubes whose probability is smaller than a relative, user-selected threshold, $\alpha \in [0,1]$.

Given the probability function $P_{F_a}(x, s)$, we first compute the maximum value of its leaves:

$$Max = \backslash_{x,s}^{Max} P_{F_a}(x, s) \tag{4}$$

Then, we set to 0 all the leaves of the $P_{F_a}(x, s)$ ADD whose values are smaller than $\alpha \cdot Max$, and we set to 1 the remaining leaves. This is accomplished through an ad-hoc ADD operator called `THRESHOLD`; we denote as $\tilde{P}_{F_a}$ the so obtained ADD (which is, actually, a BDD, since it has only 0 and 1 leaves). Finally, the current activation function is computed by application of the `ITE` operator:

$$F_a^{Curr} = \texttt{ITE}(\tilde{P}_{F_a}, F_a, 0) \tag{5}$$

Preserving a high probability for F_a is essential. However, to obtain a minimum power implementation, it is equally important to keep the area of the clock-gating circuitry under control. It is well known that reducing the number of minterms in the ON-set of a function does not guarantee that the size of the corresponding (optimized) circuit decreases. On the other hand, if some minterms in the ON-set are moved to the don't care-set instead of the OFF-set, then the final realization of the circuit is advantageous from the area point of view. To take this aspect into account, we need to generate the don't care function, $DC_{F_a}^{Curr}$, which is associated to the activation function. One way of computing it is the following:

$$DC_{F_a}^{Curr} = F_a \oplus F_a^{Curr} \tag{6}$$

Clearly, $DC_{F_a}^{Curr}$ can be used to optimize F_a^{Curr} at each iteration of the reduction process.

In the rare cases where a large fraction of the minterms of F_a has the same probability, say p, keeping the p leaf in the ADD does not allow enough reduction of F_a; on the other hand, setting it to zero causes an unacceptable decrease in the total probability of the pruned F_a. We propose a solution based on the concept of BDD subsetting [Ravi and Somenzi 1995]. We retain only the "dense" subset of minterms with probability p, in the hope that to a small ADD for the probability function corresponds a compact logic circuit realizing the reduced F_a. Experimental evidence has proved this choice to be reasonably efficient.

The reduction technique outlined above uses as its primary pruning criterion the probability of the minterms to be added to the don't care-set. An alternative heuristics is reminiscent of the strategy presented in Alidina et al. [1994], and it is based on the key observation that reducing the number of variables in the support of F_a may cause a reduction in the size of its implementation, since the number of circuit inputs decreases accordingly. The procedure selects a variable x_i to be eliminated from the support of F_a based on the probability of the universal abstraction $q_i = \forall_{x_i} F_a(x)$. Variables with the highest $P(q_i)$ are eliminated, one at a time, until a user-selected cost requirement (which accounts for both the total probability of the reduced F_a and the size of its implementation) is met. Also in this case, the reduced activation function can be further optimized at each iteration of procedure `Reduce_Fa` by using the don't care information that can be computed using Equation 6. For space reasons, we do not discuss the details of this heuristic.

4.1.4 *Computing the Cost of Function F_a.* The pruning heuristics described in the previous section use, as the driving criterion, the total probability of the reduced activation function as well as the size of its implementation. However, the ultimate objective of the optimization algorithm is the reduction of the dissipated power of the overall design. Therefore, the cost function we employ to decide whether the current solution is acceptable uses power as the primary target. Its expression is the following:

$$Curr\_Cost = POWER(Circ)(1 - PROB(F_a^{Curr})) + POWER(F_a^{Curr}) \quad (7)$$

$POWER(Circ)$ indicates the average power dissipation of the original circuit, computed through Monte-Carlo or symbolic simulation. $POWER(F_a^{Curr})$, on the other hand, is the average power dissipation of an optimized multilevel implementation of F_a^{Curr}. The first term of the summation represents an estimate of the expected power dissipation of the circuit when clock-gating is present. The second contribution is the additional power consumed by the activation function. The biggest source of approximation is in the assumption that the power of the gated-clock circuit (excluding the activation function) scales linearly with the probability of F_a^{Curr}. The advantage of this cost function stands in its limited computational requirements, since $POWER(Circ)$ is calculated once and for all before starting the F_a reduction process. The negative side is that the possibly beneficial effects of simplifying the logic of the overall circuit using F_a^{Curr} as the external don't care set are not accounted for. By contrast, $POWER(F_a^{Curr})$ is clearly recomputed for each new activation function—that is, at each iteration of the `Reduce` F_a algorithm.

Alternative cost functions of increasing accuracy and computational demand are obviously available. We have actually experimented with some of them; for example, we have considered the following:

$$Curr\_Cost = POWER(F_a^{Curr} + Circ) \quad (8)$$

This is clearly more accurate than the one of Equation 7. In fact, each F_a^{Curr} is first synthesized and optimized and then connected to the original circuit as in Figure 1(b). The power dissipation of the overall network is then estimated. The complexity of the computation is increased, because a power estimation of a larger design is required at each iteration. Unfortunately, extensive experimentation has shown that adopting more sophisticated solutions does not pay off in terms of the trade-off between the quality of the synthesized activation function and the time spent in the synthesis process. Therefore, we have chosen to stick to the simplest formulation of the cost function given in Equation 7.

4.1.5 *The Stopping Criterion*. As in any gradient-based refinement procedure (where the iterations continue as long as there are improvements, and stop as soon as the cost function starts increasing again), we reduce the ON-set of F_a at each iteration and we exit the reduction loop the first time the cost function starts increasing, that is, $Curr\_Cost > Best\_Cost$. Experimental evidence in previous work [Benini and De Micheli 1996] has shown that, in most cases, the cost function is either monotone or with a single minimum. This result is intuitive, since the reduction of the activation function is such that the newly generated F_a is contained into the one generated at the previous iteration, and therefore once a minimum is hit, it is difficult to hit another one. This argument is plausible as long as the circuitry implementing F_a and the logic of the original circuit are kept separated. In fact, in this case, a smaller activation function improves the power dissipation only by reducing the consumption in the clock-gating circuitry. A size reduction of F_a that increases the power implies that the power not saved in the circuit is larger than the power saved in the clock-gating circuitry, and using an even smaller activation function will only make this situation worse.

When the cost function is more complex, this line of reasoning may no longer be correct. This is because minterms removed from F_a are actually added to the don't care set of the same function, and since such don't care conditions are used to optimize the overall circuit, a larger don't care set may help in reducing its size, and therefore, possibly, the total power consumption. It is clear then that, due to the complexity of the adopted cost function, finding a direct relationship between such function and the optimality of the computed solution is not an easy task. We are currently investigating more sophisticated stopping criteria which are able to guide a branch-and-bound searching technique.

4.2 Global Circuit Optimization

The result produced by procedure `Reduce_Fa` is a gate-level specification of the activation function, F_a, which is likely to produce some power savings when appropriately connected to the original sequential design.

After the logic is included in the circuit in the way shown in Figure 1(b), some global optimization can be performed. Notice that the activation function is functionally redundant. Since the target is area rather than power minimization, the optimizer may remove the clock-gating logic in its entirety, thus producing a circuit which is very similar to the original one. This is most likely to happen when F_a is used as the external don't care set for each primary and state output and redundancy removal methods are used for the optimization. Clearly, this is something we must avoid.

The solution we have adopted to overcome this problem consists of adding to the circuit an extra output pin to make function F_a directly observable. With this artifact, redundancy removal procedures can be applied to the circuit. This type of optimization has highly beneficial effects on the gated-clock circuits: Not only may it reduce the power dissipation, it also increases the testability of the system, because it eliminates the untestable faults in the combinational logic generated by the insertion of the redundant clock-activation logic [Favalli et al. 1996].

5. EXTENSIONS

If a sequential circuit is an implementation of a Mealy FSM with no Moore-states, the activation function will be empty. In this section, we discuss generalizations of the procedure used to find the initial F_a that allow the exploitation of different kinds of idle conditions. The generalized procedure is applicable to feedback-free sequential circuits as well (i.e., pipeline stages). Consequently, it may be noticed that our approach has the same scope of applicability as precomputation [Alidina et al. 1994; Monteiro et al. 1995], namely, sequential circuits described in a structural fashion at the logic level. In the second part of the section, we analyze the analogies and differences between our clock-gating strategy and precomputation, and we show that the two techniques are *not* equivalent and exploit different types of idle conditions.

5.1 Covering Additional Self-Loops

We have shown in Section 3 that the basic clock-gating technique cannot stop the clock in self-loops of Mealy-states, because for such self-loops output transitions might be required, even if the next state does not change. This problem can be solved if the outputs of the sequential circuit are taken as inputs of the activation function as well as the state and primary inputs. The gated-clock architecture can be modified as shown in Figure 5. If all outputs are taken as inputs of the activation function, *all* self-loops can be exploited to stop the clock.

As an example, consider again Figure 2: If we are allowed to observe the output values, then a state value of S_2, an input value 00, and an output value 11 uniquely identify the self-loop in S_2. Observing these values we can stop the clock because the FSM is in a self-loop, and the output is not going to change in the next clock cycle. Intuitively, the desirable consequence of observing the outputs is that more information is available for

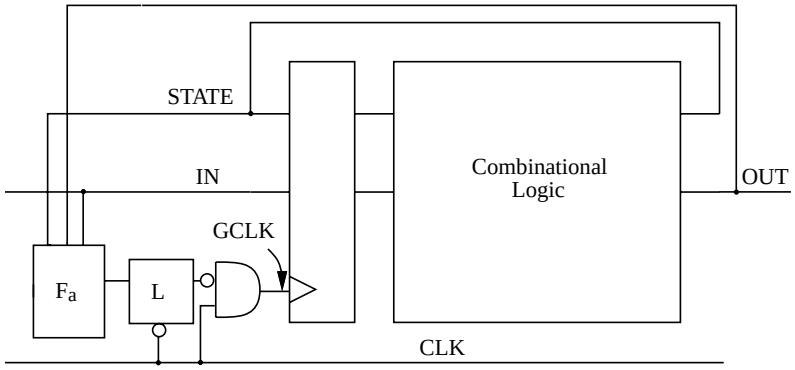

Fig. 5. Modified gated-clock architecture.

detecting idleness. Unfortunately, the negative counterpart is that the activation function has an increased number of inputs and it may become more complex. The symbolic formula for the computation of the activation function including the output values is similar to the one presented in Equation 1:

$$F_a(x^+, t, z^+) = A \cdot (D + C) \tag{9}$$

where terms A and C are the same as in Equation 1, and $D = \Pi_{i=1}^{m}(\lambda_i(x^+, t) \equiv z_i^+)$. Notice that the support of F_a has been extended to include the output variables z^+. Term D is the key difference between Equations 1 and 9; in fact, it expresses the condition that, when the next state and the present state are the same (i.e., $A = 1$), the FSM is idle if the outputs also do not change. Interestingly, the idle conditions captured by Equation 9 are a superset of the "hold conditions" as defined in Theeuwen and Seelen [1997].

It may be observed that, since the number of outputs in a sequential circuit is often very large, the size of the activation logic may increase too much. However, it may be the case that we do not need to use *all outputs* as inputs of F_a. For example, referring to Figure 2, to exploit the self-loop on S_2 it is sufficient to sample the second output, because the first output does not change on all transitions reaching S_2.

Formula 9 can be modified so that only a subset of the outputs becomes part of the support of F_a. We have:

$$F_a(x^+, t, z^+) = A \cdot ((D_1 \cdot D_2) + C) \tag{10}$$

where $D_1 = \Pi_{i=1}^{w}(\lambda_i(x^+, t) \equiv z_i^+)$, $D_2 = \Pi_{i=w+1}^{m}(\lambda_i(x, s) \equiv \lambda_i(x + , t))$, and w is the number of circuit outputs we want to consider.

There is a clear trade-off between the number of additional self-loops that can be considered in the activation function by including one or more outputs to its support and its size (and power dissipation). Ideally, we

would like to include in the support of F_a all outputs whose observation enables the exploitation of many self-loops. The complexity of the implementation of F_a strongly depends on the selection of the subset of outputs to be included in its support.

We have devised a heuristic procedure to perform the selection of an optimal subset of outputs for inclusion in the support of the activation function. For each primary output, z_i^+, of the circuit, we first compute function $F_a(x^+, t, z_i^+)$ and we determine the value of its probability P_i. This step provides information on how beneficial the inclusion of a single output in the support of F_a can be.

Function $F_a(x^+, t, z^+)$ is built iteratively, starting from $F_a^{(0)} = F_a(x^+, t)$ (i.e., the activation function computed by Equation 1, with no output variables in its support). The activation function at step n of the iteration is computed as:

$$F_a^{(n)} = F_a^{(n-1)} + F_a(x^+, t, z^+) \tag{11}$$

where $F_a(x^+, t, z^+)$ is determined using Equation 10. On each iteration, w is incremented by one (i.e., one output is added to the support of F_a). The outputs are selected starting from those with higher P_i.

It is easy to realize that the probability of $F_a^{(n)}$ is non-decreasing for increasing n. However, a point of diminishing return may be reached, when adding outputs to the support of the activation function leads to a marginal increase in probability and to a substantial increase in the complexity of its implementation. For this reason, after each iteration the power savings are estimated with the heuristics described in Section 4. The iteration is stopped when the cost starts increasing.

The iterative algorithm returns an activation function that includes in its support some of the outputs of the sequential circuit. Most of the run time is spent in the evaluation of the power savings for each $F_a^{(n)}$, which involves optimal pruning and cost estimation. The greedy nature of the algorithm guarantees that the worst-case number of iterations is equal to the number of outputs. Hence, the run time is kept under control.

5.2 Feedback-Free Sequential Circuits

Let us consider the case of feedback-free sequential circuits, where all primary inputs are latched and no output is fed back to the inputs. Although such circuits are still sequential (the output at one clock cycle depends on the input in the previous one), they do not have state lines. Therefore, computing the activation function for circuits falling into this category through Equation 1 yields the null function. However, an F_a with nonempty ON-set can be obtained using Equation 9; in fact, the expression for F_a reduces to:

$$F_a = D = \prod_{i=1}^{m} (\lambda_i(x^+, t) \equiv z_i^+) \tag{12}$$

In essence, the meaning of Equation 12 is that there is no need for clocking the flip-flops of the circuit if the new input values are not going to change the current value of the outputs.

Clearly, there is no Boolean function for which $F_a = 0$, but computing the entire F_a is at least as complex as computing the outputs (if not more, because the number of inputs of F_a is actually larger than the number of inputs of the feedback-free circuit). The key observation is again that we do not need to stop the clock whenever F_a is true. We can select a function contained in F_a that stops the clock with maximum efficiency and which has low complexity. The synthesis algorithms described in Section 4 can then be applied with no modification even in the degenerate case of feedback-free circuits.

5.3 Relationship With Precomputation

In this section, we assume the reader to be familiar with the work of Alidina et al. on precomputation techniques for gate-level power management [Alidina et al. 1994; Monteiro et al. 1995]; therefore, a detailed description of the architectures, as well as the algorithms for their automatic generation is omitted.

The precomputation approach has been proposed initially for the optimization of feedback-free sequential designs. However, in Alidina et al. [1994], the authors have claimed that the applicability of their technique is not restricted to these types of circuits, but, instead, it can be easily extended to designs having an arbitrary structure. In particular, they have shown in detail how the precomputation architecture can be adapted to the case of synchronous sequential circuits having the traditional topology illustrated in Figure 1(a).

Unfortunately, in spite of the nice theoretical formulation of the solution, no experimental evidence of its practical usefulness has been presented in the literature. We have thus conducted a deeper investigation on this subject. In Section 6, we report some results on the outcome of our analysis.

Precomputation and gated-clocks are both based on the concept of synthesizing a logic function that stops the clock of a sequential circuit in a subset of all possible input conditions. Although the clock-stopping circuitry for precomputation (i.e., enable signals for the latches) and gated-clocks (i.e., disabling of the clock signal) appear to be different, it is easy to realize that the difference is only a matter of implementation style.

This observation may lead to the conclusion that precomputation and gated-clocks are equivalent. In fact, they are not. Precomputation detects clock-stopping opportunities based on lack of observability. If, for some input conditions, a large number of inputs is not observable at the outputs, there is no need to clock the flip-flops associated to the unobservable

inputs. On the contrary, gated-clocks are based on the intrinsic memory-retaining property of CMOS circuitry. If there are conditions for which we can pre-determine that the outputs are not going to change, there is no need to clock the circuit, because the input activity does not propagate to the outputs. Moreover, we can stop the clock indefinitely, as long as the outputs do not need to be updated. We may thus conclude that there are circuits for which precomputation is completely ineffective, namely, those for which a large fraction of the inputs is always observable. On the other hand, gated-clocks are not effective for systems where at least one output changes every clock cycle.

The nonequivalence of gated-clocks and precomputation, that is, the capability of the two approaches of finding logic conditions for power management with non-empty intersection, may suggest a simultaneously application of the two techniques. Unfortunately, results obtained using both optimizations showed that they are effective on almost disjoint classes of circuits. In other words, for most of the examples we have experimented with, the use of only one of the two methods yielded better power savings than their combination.

6. EXPERIMENTAL RESULTS

The power optimization algorithms proposed in this paper have been implemented within the SIS [Sentovich et al. 1992] environment, and their effectiveness benchmarked onto standard examples taken from the literature. The original descriptions have been optimized for area using the SIS script `script.rugged`, and mapped for speed using the SIS command `map -n 1 -AFG`. These mapped circuits have been used as the starting point for our experiments. The logic for the reduced F_a has been computed through procedure `Reduce_Fa` and connected to the original circuit as indicated in Figure 5. The functional specification of F_a has then been added as external don't care-set for each circuit output, and the circuit optimized for area through `script.rugged`.

The cell library we have used for the experiments contained NAND and NOR gates with up to four inputs, buffers, and inverters. Each cell had 3 different size/drive options. Power values of the initial and final circuit implementations were obtained through transistor-level simulation of 10,000 random vectors using Irsim [Salz and Horowitz 1989]. All the experiments were run on a DEC AXP 1000/300 with 128 MB of main memory.

As mentioned in the introduction, the clock-gating transformation performs best on reactive circuits, i.e., circuits with a large number of idle conditions. We examined the entire `Iscas'89` synchronous sequential benchmarks [Brglez 1989] (a total of 31 examples), assuming independent and equiprobable inputs with 0.5 switching activity. The circuits were simulated for 10,000 clock cycles. Idleness was measured by computing the fraction of clock cycles for which primary and next state outputs did not change. The results of our experiment are reported in Table I. The table

Table I. Idleness of the Iscas'89 Circuits

Circuit	*PI*	*PO*	*FF*	*Idleness*
s27	4	1	3	45.81%
s208.1	10	1	8	45.73%
s298	3	6	14	41.60%
s344	9	11	15	0.68%
s349	9	11	15	0.68%
s382	3	6	21	45.71%
s386	7	7	6	64.08%
s400	3	6	21	45.71%
s420.1	18	1	16	47.83%
s444	3	6	21	45.71%
s510	19	7	6	54.25%
s526	3	6	21	45.42%
s526n	3	6	21	45.42%
s641	35	24	19	0.98%
s713	35	23	19	0.98%
s820	18	19	5	8.75%
s832	18	19	5	8.75%
s838.1	34	1	32	46.41%
s953	16	23	29	6.87%
s1196	14	14	18	0.23%
s1238	14	14	18	0.23%
s1423	17	5	74	0.00%
s1488	8	19	6	34.70%
s1494	8	19	6	34.70%
s5378	35	49	179	0.00%
s9234.1	36	39	211	0.00%
s13207.1	62	152	638	0.00%
s15850.1	77	150	534	0.00%
s35932	35	320	1728	0.00%
s38417	28	106	1636	0.00%
s38584.1	38	304	1426	0.00%

gives the number of inputs, outputs and flip-flops of the benchmarks as well as their idleness. Only circuits with more than 10% idleness have been considered as candidates for testing the clock-gating transformations proposed in the previous sections.

A total of 14 examples has then been selected, and Table II summarizes the obtained results. In particular, columns *Gates*, *Delay*, and *Power* tell the number of gates, the rise and fall delays (in *nsec*), and the power dissipation (in μW), before and after clock-gating is applied. Column *Variation* gives the percentage of gate count and delay increase and power reduction obtained on each example. Finally, columns F_a *PROB*, F_a *Power*, and F_a *Time* report the probability of the simplified F_a, the power (in μW) dissipated by the logic implementing it, and the CPU time (in *sec*) required by procedure Reduce_Fa to determine the simplified activation function.

It is important to note that $PROB(F_a)$ is usually smaller than the corresponding idleness reported in Table I. This is because the on-set of the activation function has been reduced to limit its size and power.

Table II. Results for the Non-Idle Iscas'89 Circuits

Circuit	Before Optimization			After Optimization			Variation			F_a		
	Gates	Delay	Power	Gates	Delay	Power	Gates	Delay	Power	PROB	Power	Time
s27	18	6.60/6.57	33.21	30	8.06/8.04	26.12	+66%	+22%	−21%	0.367	5.06	0.1
s208.1	90	11.00/10.98	75.43	95	11.07/11.05	49.38	+5%	+1%	−34%	0.371	1.76	1.2
s298	131	19.26/19.24	89.18	140	19.90/19.88	72.37	+7%	+3%	−19%	0.241	4.32	1.8
s382	154	19.09/19.07	90.31	166	19.58/19.56	74.42	+8%	+2%	−18%	0.251	6.81	5.1
s386	148	14.94/14.92	63.20	160	15.97/15.95	58.88	+8%	+7%	−8%	0.110	1.71	5.8
s400	168	20.81/20.79	90.36	186	21.24/21.22	69.13	+11%	+2%	−23%	0.249	1.21	7.9
s420.1	171	16.42/16.40	104.05	183	17.40/17.38	68.80	+7%	+6%	−34%	0.382	3.42	14.5
s444	199	20.31/20.29	103.81	217	22.12/22.10	79.21	+9%	+9%	−23%	0.249	2.54	4.6
s510	289	25.62/25.60	95.42	306	27.31/27.29	83.51	+6%	+6%	−13%	0.140	1.79	29.1
s526	206	18.24/18.22	119.36	230	19.83/19.81	114.23	+11%	+9%	−4%	0.244	12.11	11.6
s526n	187	16.12/16.10	114.39	211	17.72/17.70	109.64	+13%	+10%	−4%	0.244	11.09	11.4
s838.1*	342	33.24/33.22	174.83	347	34.30/34.28	114.63	+1%	+3%	−34%	0.398	10.35	26.9
s1488	518	22.72/22.70	177.31	533	26.82/26.80	164.87	+3%	+18%	−7%	0.120	8.20	8.7
s1494	533	24.50/24.48	178.76	549	28.16/28.14	166.17	+3%	+15%	−7%	0.120	8.31	8.5

It should also be observed that the steady-state probabilities of the FSMs associated to the circuits have been computed using the exact methods of Hachtel et al. [1996] in all cases, except for circuit s838.1 (indicated in the table with an *), for which the approximate method of Tsui et al. [1995] has been used.

We have tested both pruning heuristics for the generation of the optimal F_a, but the quality of the results did not change sensibly (for the results in the table we report the best obtained savings).

The size (and the number of flip-flops) of the circuits considered in our experiments is such that the application of the techniques presented in Benini and De Micheli [1996] would not be possible, because of the complexity of the STG extraction procedure. In contrast, our symbolic algorithms easily deal with these examples. For some circuits the power savings are sizable (25%-35%), while for others the advantage given by gating the clock is limited. The area and the delay are kept under control (11% and 8% increase, on average).

The ability of our method to handle relatively large controllers does not imply a loss in accuracy in the computation of the circuits' idle conditions. To support this claim, in Table III we compare the power results achieved by the BDD-based procedure of this paper to those published in Benini and De Micheli [1996] for some of the small, symbolic Mcnc'91 finite state machines [Yang 1991] (the ones for which large savings were obtained by the explicit algorithm). From the data we can conclude that the symbolic approach is at least as powerful as the explicit one.

Since the reactive nature of a controller typically depends on the external environment, it is likely that idle conditions will be exercised when the circuit is interacting with the components in its neighborhood. Such interaction may be modeled through non-equiprobable primary input distributions. Then, given that the computation of the activation function depends on the input probabilities, we expect the size and the probability of F_a to be affected by the use of non-equiprobable input distributions.

Table III. Comparison to the Results of Benini and De Micheli [1996] on the Mcnc'91 FSMs

Circuit	*PI*	*PO*	*States*	*Power Savings*	
				Symbolic	*Explicit*
bbara	4	2	10	45%	49%
bbtas	2	2	6	12%	21%
keyb	7	2	19	26%	11%
lion9	2	1	9	10%	13%
s420.kiss	19	2	18	24%	18%

As an example, let us consider the minmax3 circuit [Coudert et al. 1989]; in Figure 6 we plot the value of $PROB(F_a)$ for varying values of the probability of the enable (active high) and clear (active low) control inputs. For a fixed value of the probability of clear, $PROB(F_a)$ increases as the probability of enable decreases, and it goes up as the probability of clear increases. This is reasonable, since a high probability of both enable and clear to be active drives the circuit into the hold states, corresponding to the traversal of the self-loops of the STG.

To show how the knowledge of the primary input statistics impacts the synthesis and the refinement of F_a, and thus the achievable power savings, in Table IV we present results for the circuits of Table II in which the probability of some of the inputs has been set to values different from 0.5. It is important to note that even though, in principle, assuming non-equiprobable inputs may increase the computational effort required to determine the steady-state occupation probabilities, execution times for the experiments of Table IV have not changed appreciably with respect to those where 0.5 probabilities have been assumed for all the inputs.

Since no information was available for both the circuit functionality and the environment in which the controllers are operating, we have chosen to modify the statistics of the primary inputs belonging to the support of the activation function calculated for the equiprobable case. More specifically, we have set the input probabilities so as to emphasize the reactivity of the benchmarks. As expected, power savings have gone up sensibly.

Obviously, the input vector streams used for simulation have been properly biased so as to match the non-equiprobable input probabilities.

In Section 5, we have shown how gated-clocks can be applied also to feedback-free sequential circuits, a class of circuits for which precomputation-based solutions have shown to be effective.

In Table V, we compare the performance of the two methods on a sample of the Mcnc'91 benchmark suite [Yang 1991]. In particular, we have chosen all the examples used for the experiments in Monteiro et al. [1995]. Since the original descriptions of the benchmarks are inherently combinational, the test cases have been constructed by adding input and output latches to the combinational logic. The data clearly indicate that the effectiveness of the two methods is circuit dependent; therefore, no conclu-

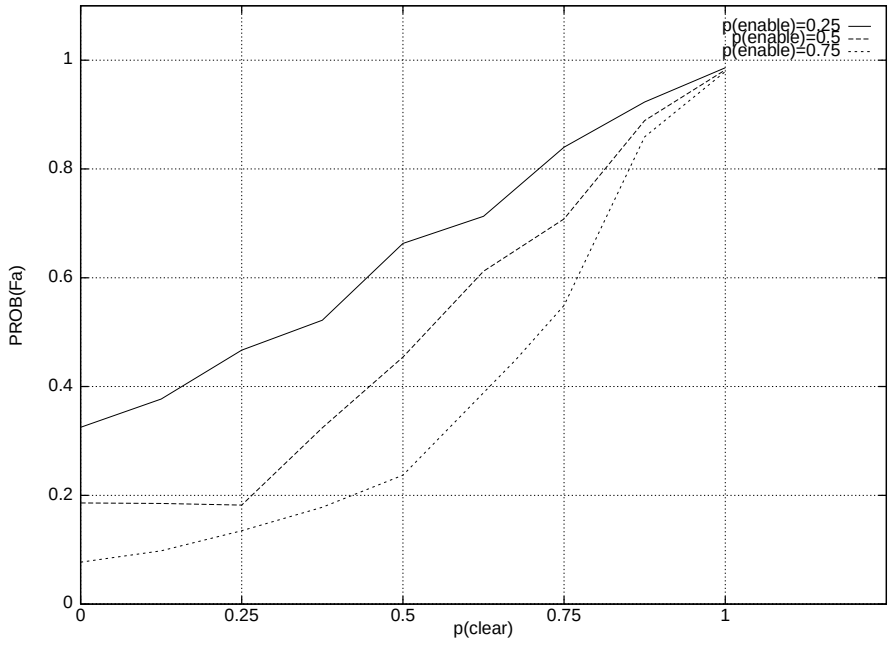

Fig. 6. Case study: The `minmax3` circuit.

Table IV. Results for the `Iscas'89` Circuits with Non-Equiprobable Inputs

Circuit	*PROB(F_a)*	*Power*		
		Orig.	*Opt.*	*Savings*
s27	0.776	27.61	16.23	40%
s208.1	0.831	64.34	17.21	73%
s298	0.902	53.37	10.67	81%
s382	0.814	56.91	27.61	55%
s386	0.696	52.10	18.60	65%
s400	0.809	67.17	15.42	77%
s420.1	0.829	90.76	21.40	76%
s444	0.811	69.43	19.88	72%
s510	0.670	81.19	45.75	44%
s526	0.807	88.13	43.27	51%
s526n	0.806	84.45	41.05	51%
s838.1*	0.894	146.60	86.02	41%
s1488	0.883	131.03	77.86	41%
s1494	0.881	133.89	80.44	40%

sion can be drawn about the superiority of one approach with respect to the other.

It should be observed that the power savings shown in Table V for the precomputation method differ from those reported in Monteiro et al. [1995]; this is due to three factors. First, power estimates in Monteiro et al. [1995] are obtained using the symbolic simulation available in SIS, while in this paper power values have been determined using Irsim. This has allowed us

Table V. Comparison to Precomputation for Feedback-Free Circuits

Circuit	*PI*	*PO*	*Original Power*	*Gated Clocks*		*Precomputation*	
				Power	*Savings*	*Power*	*Savings*
9sym	9	1	123.56	62.33	49%	123.56	0%
z5xp1	7	10	59.41	49.40	17%	51.27	13%
alu2	10	6	108.90	105.09	2%	108.90	0%
apex2	39	3	286.78	286.78	0%	114.18	60%
cm138a[-2	6	8	35.75	13.66	63%	29.81	17%
cm150a[-2	21	1	94.78	60.51	36%	74.78	21%
cm162a[-2	14	5	52.13	52.13	0%	52.13	0%
cmb	16	4	66.49	15.45	77%	45.91	32%
comp	32	3	144.04	116.22	19%	68.69	53%
cordic[-2	23	2	111.30	38.31	66%	81.79	27%
dalu	75	16	588.28	588.28	0%	424.30	28%
mux	21	1	99.67	61.39	38%	70.52	29%
sao2	10	4	65.55	33.05	49%	39.71	40%

Table VI. Comparison to Precomputation for Traditional Sequential Circuits

Circuit	*Original Power*	*Gated Clocks*		*Precomputation*	
		Power	*Savings*	*Power*	*Savings*
s27	33.21	26.12	21%	33.21	0%
s208.1	75.43	49.38	34%	71.56	5%
s298	89.18	72.37	19%	89.18	0%
s382	90.31	74.42	18%	82.50	9%
s386	63.20	58.88	8%	63.20	0%
s400	90.36	69.13	23%	77.26	14%
s420.1	104.05	68.80	34%	97.23	8%
s444	103.81	79.21	23%	86.19	15%
s510	95.42	83.51	13%	95.42	0%
s526	119.36	114.23	4%	119.36	0%
s526n	114.39	109.64	4%	114.39	0%
s838.1*	174.83	114.63	34%	154.12	9%
s1488	177.31	164.87	7%	177.31	0%
s1494	178.76	166.17	7%	178.76	0%

to properly account for the power dissipated by the latches, as well as that consumed by the clock distribution network. Second, the gate library used for technology mapping is different. Third, we employed clock-gating, while, in Monteiro et al. [1995], enabled flip-flops were used.

As discussed in Section 5.3, some theoretical work has been presented in Alidina et al. [1994] on the possibility of extending precomputation-based architectures to the case of sequential circuits having a traditional gate-level structure (see Figure 1(b)), but no experimental evidence was presented. We have thus implemented the various extensions within our framework, and we have compared the results to those obtained by adding to the original circuit the clock-gating logic. The data in Table VI show the ineffectiveness of the precomputation-based approach for this type of designs (savings, in percentage, are computed with respect to the reference

circuits). The reason for this poor behavior lies in that the precomputation function never attempts to stop the present-state inputs, which represent the majority of the inputs to the combinational logic for sequential circuits with a realistic number of memory elements.

7. CONCLUSIONS

We have presented a fully symbolic bottom-up approach to the automatic generation of clock-gating logic for sequential circuits. Our methodology starts from synchronous gate-level networks and does not require the explicit extraction of the STG, a very computationally expensive operation. We leverage the BDD-based representation of Boolean and pseudo-Boolean functions to extend the applicability of clock-gating techniques to classes of sequential systems of size unattainable by previous methods based on explicit enumeration algorithms.

The generality of our formulation enables the application of the synthesis procedure to activation functions with extended support (including some of the circuit outputs). The compactness and expressive power of ADDs allow us to accurately compute the probability of the activation function, and to develop algorithms that control the optimization of the global power dissipation with superior accuracy, compared to previous approaches.

Our optimization strategy also relies on an integrated synthesis methodology that aims at reducing the overhead of the redundant clock-gating logic by effectively exploiting the additional don't care conditions in the combinational logic. The results are promising, since we obtain power reductions as high as 34% with negligible area and performance degradations.

REFERENCES

ALIDINA, M., MONTEIRO, J., DEVADAS, S., GHOSH, A., AND PAPAEFTHYMIOU, M. 1994. Precomputation-based sequential logic optimization for low power. *IEEE Trans. Very Large Scale Integr. Syst. 2*, 4 (Dec. 1994), 426–436.

BAHAR, R. I., FROHM, E. A., GAONA, C. M., HACHTEL, G. D., MACII, E., PARDO, A., AND SOMENZI, F. 1997. Algebraic decision diagrams and their applications. *Formal Methods Syst. Des. 10*, 171–206.

BENINI, L. AND DE MICHELI, G. 1996. Transformation and synthesis of FSMs for low power gated clock implementation. *IEEE Trans. Comput.-Aided Des. Integr. Circuits 15*, 6 (June 1996), 630–643.

BIGGS, T. AND ET AL., 1994. A 1 Watt 68040-compatible microprocessor. In *Proceedings of the IEEE Symposium on Low Power Electronics* IEEE Computer Society Press, Los Alamitos, CA, 12–13.

BRGLEZ, F., BRYANT, D., AND KOZMINSKI, K. 1989. Combinational profiles of sequential benchmark circuits. In *Proceedings of the IEEE International Symposium on Circuits and Systems* (Portland, OR, May 1989) IEEE Press, Piscataway, NJ, 1929–1934.

BRYANT, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput. C-35*, 8 (Aug. 1986), 677–691.

COUDERT, O., BERTHET, C., AND MADRE, J. C. 1989. Verification of sequential machines using Boolean functional vectors. In *Proceedings of the IFIP International Workshop on Applied Formal Methods for Correct VLSI Design* (Leuven, Belgium, Nov.) IFIP, 111–128.

DEBNATH, G., DEBNATH, K., AND FERNANDO, R. 1995. The Pentium processor-90/100, microarchitecture and low-power circuit design. In *Proceedings of the IEEE International Conference on VLSI Design* (Jan. 1995) IEEE Press, Piscataway, NJ, 185–190.

FAVALLI, M., BENINI, L., AND DE MICHELI, G. 1996. Design for testability of gated-clock FSMs. In *Proceedings of the IEEE European Conference on Design and Test* (EDTC '96, Paris, France, Mar. 1996) IEEE Press, Piscataway, NJ, 589–596.

HACHTEL, G. D., MACII, E., PARDO, A., AND SOMENZI, F. 1996. Markovian analysis of large finite state machines. *IEEE Trans. Comput.-Aided Des. Integr. Circuits 15*, 12 (Dec. 1996), 1479–1493.

MONTEIRO, J., RINDERKNECHT, J., DEVADAS, S., AND GHOSH, A. 1995. Optimization of combinational and sequential circuits for low power using precomputation. In *Proceedings of the 1995 Chapel Hill Conference on Advanced Research in VLSI* (Chapel Hill, NC, Mar. 1995) 430–444.

RAVI, K. AND SOMENZI, F. 1995. High-density reachability analysis. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD-95, San Jose, CA, Nov. 5–9), R. Rudell, Ed. IEEE Computer Society Press, Los Alamitos, CA, 154–158.

SALZ, A. AND HOROWITZ, M. 1989. IRSIM: An incremental MOS switch-level simulator. In *Proceedings of the 26th ACM/IEEE Conference on Design Automation* (DAC '89, Las Vegas, NV, June 25–29, 1989), D. E. Thomas, Ed. ACM Press, New York, NY, 173–178.

SENTOVICH, E. M., SINGH, K. J., MOON, C. W., SAVOJ, H., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 1992. Sequential circuits design using synthesis and optimization. In *Proceedings of the IEEE International Conference on Computer Design* (Cambridge, MA, Oct. 1992) IEEE Computer Society Press, Los Alamitos, CA, 328–333.

THEEUWEN, F. AND SEELEN, E. 1997. Power reduction through clock gating by symbolic manipulation. In *Proceedings of the IFIP International Conference on Very Large Scale Integration* (VLSI '97, Gramado, Brazil, Aug. 26-29)

TSUI, C.-Y., MONTEIRO, J., PEDRAM, M., DEVADAS, S., DESPAIN, A. M., AND LIN, B. 1995. Power estimation methods for sequential logic circuits. *IEEE Trans. Very Large Scale Integr. Syst. 3*, 3 (Sept. 1995), 404–416.

YANG, S. 1991. Logic synthesis and optimization benchmarks user guide version 3.0. Microelectronics Center of North Carolina, Research Triangle Park, NC.

Received: June 1997; accepted: January 1998