Symbolic Synthesis of Clock-Gating Logic for Power Optimization of Control-Oriented Synchronous Networks *

L. Benini # G. De Micheli # E. Macii ‡ M. Poncino ‡ R. Scarsi ‡

Stanford University
Computer Systems Laboratory
Stanford, CA 94305

‡ Politecnico di Torino
Dip. di Automatica e Informatica
Torino, ITALY 10129

Abstract

Recent results have shown that clock-gating techniques are effective in reducing the total power consumption of sequential circuits. Unfortunately, such techniques assume the availability of the state transition graph of the target system, and rely on explicit algorithms whose complexity is polynomial in the number of states, that is, exponential in the number of state variables. This assumption poses serious limitations on the size of the circuits for which automatic gated-clock generation is feasible. In this paper we propose fully symbolic algorithms for the automatic extraction and synthesis of the clock-gating circuitry for large control-oriented sequential designs. Our techniques leverage the compact BDD-based representation of Boolean and pseudo-Boolean functions to extend the applicability of gated-clock architectures to designs implemented by synchronous networks. As a result, we can deal with circuits for which the explicit state transition graph is too large to be generated and/or manipulated. Moreover, symbolic manipulation 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 36% have been obtained on controllers containing up to 21 registers.

1 Introduction

The demand of CAD tools that help the power management during the design of digital ICs has increased in the last few years, and much effort has been put by researchers in the development of efficient synthesis methodologies for low-power systems.

In this paper we focus on power optimization of control-oriented sequential circuits. An efficient way for obtaining power savings on a design whose initial specification is given as a state transition graph (STG) has been proposed in [1]. To be more specific, the strategy adopted to save power is based on the concept of *clock-gating*. 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 presence of 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 [2] techniques have been 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 of [2]. First, the tool can handle only small FSMs. 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 [2], may thus be simply 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 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 [3] and ADDs [4], to simplify the representation of the clock-gating conditions as well as the calculation of the probability of the *activation function* [5, 6]. 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.

For what concerns the calculation of the activation function, in [2] it has been proposed to compute it 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, to 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.

We present results that show the feasibility of the proposed techniques. In fact, power savings of up to 36% have been obtained on some of the mid-sized Iscas'89 benchmarks [7]). Moreover, experimental data 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.

*This work was supported, in part, by NSF under contract MIP-9421129.

2 Background

We assume the reader to be familiar with the basic concepts of Boolean functions and with the data structure 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 [3]. 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: $f_{x_i} = f(x_1, \ldots, x_i = 1, \ldots, x_n)$ and $f_{x_i'} = f(x_1, \ldots, x_i = 0, \ldots, x_n)$. The ***existential*** and the ***universal abstraction*** of f with respect to x_i are defined as: $\exists_{x_i} f = f_{x_i} + f_{x_i'}$ 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 ($\delta : X \times S \to S$), and $\lambda(x, s)$ is the output function ($\lambda : X \times S \to Z$). 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, $f : B^n \to S$, 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) [4].

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:

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

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 sub-sets 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 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 a 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 for the FSM to make that particular transition to obtain a discrete-parameter Markov chain. On the other hand, studying the Markov chain, that is, computing the state occupation probabilities, is related to performing the reachability analysis of a FSM.

Given the transition relation of the finite state machine 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.

ADD-based procedures allow the computation of vector $\mathbf{p}$ for very large FSMs [5, 6]. Transition graphs with up to 10^{27} states were successfully handled by the ADD-based probabilistic analysis tool. Complex primary input probability distributions can be specified and efficiently represented with ADDs 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 [1, 2].

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.

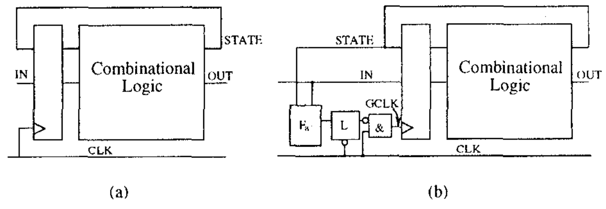

Figure 1: A Sequential Circuit (a). Gated-Clock Version (b).

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. 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.

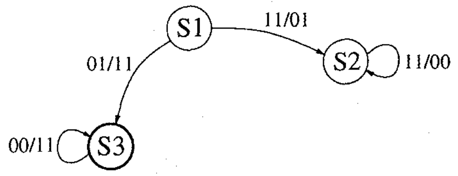

Figure 2: Fragment of a Mealy FSM.

This observation has been formalized in [2] 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 [2] 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 extract the STG representation for 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.

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 self-loops of Moore-states (x and s are respectively the input and state variables). 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, i.e., duplicating the combinational logic to represent two consecutive time frames, as shown in Figure 3.

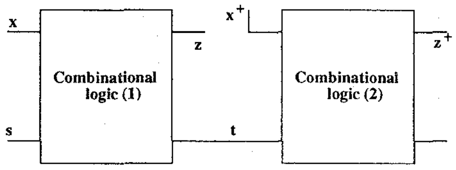

Figure 3: Unrolling of a FSM.

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 optimizations 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:

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

The term $A = \prod_{i=1}^{k}(\delta_i(x^+, t) \equiv t_i)$ 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 $B = \prod_{i=1}^{m}(\lambda_i(x, s) \equiv \lambda_i(x^+, t))$ 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 $C = (\prod_{i=1}^{k}(\delta_i(x, s) \equiv t_i))'$ is ORed 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, 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 together these two requirements.

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 [5, 6]). The probability P_{F_a} can be simply obtained as:

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

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

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 pseudo-code of the procedure is shown in Figure 4.

```
procedure Reduce_Fa($F_a, P_{F_a}$) {
    $F_a^{Best} = F_a$; $P_{Best} = P_{F_a}$;
    $F_a^{Curr} = F_a$; $P_{Curr} = P_{F_a}$;
    $Best\_Cost$ = 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 ($Curr\_Cost \leq Best\_Cost$) {
            $F_a^{Best} = F_a^{Curr}$;
            $P_{Best} = P_{Curr}$;
            $Best\_Cost = Curr\_Cost$;
        }
    }
    return ($F_a^{Best}$);
}
```

Figure 4: The Reduce_Fa Algorithm.

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:

$$Max = \backslash^{MAX}_{x,s} P_{F_a}(x,s)$$

Then, we set to 0 all the leaves of the $P_{F_a}(x,s)$ ADD whose values are smaller than αMax, 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:

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

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:

$$DC_{F_a}^{Curr} = F_a \oplus F_a^{Curr} \qquad (2)$$

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 [8]. 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 primary pruning criterion the probability of the minterms to be added to the don't care-set. An alternative heuristic is reminiscent of the strategy presented in [9], 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 2. 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 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 the power as primary target. Its expression is the following:

$$Curr\_Cost = \text{POW}(Circ)(1 - PROB(F_a^{Curr})) + \text{POW}(F_a^{Curr})$$

POW($Circ$) is the average power dissipation of the original circuit, computed through Monte-Carlo or symbolic simulation. POW(F_a^{Curr}), on the other hand, is the average power dissipation of an optimized multi-level 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 POW($Circ$) is calculated once and for all before starting the F_a reduction process. The negative side is, obviously, that the possibly beneficial effects of simplifying the logic of the overall circuit using F_a^{Curr} as external don't care-set are not accounted for. In contrast, POW(F_a^{Curr}) is clearly recomputed for each new activation function, that is, at each iteration of the Reduce_Fa algorithm.

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, i.e., $Curr\_Cost > Best\_Cost$. Experimental evidence in previous work [1] has shown that, in most of the 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 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 it may 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 [10].

4.3 Covering Additional Self-Loops

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.

We target self-loops on Mealy states. As discussed above, these self-loops are not idle conditions because we cannot guarantee that output transitions will not 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 identifies the self-loop in S_2. Observing these values we can stop the clock because: i) the FSM is in a self-loop, ii) the output is not going to change in the next clock cycle.

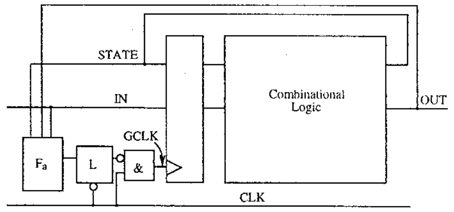

Figure 5: Modified Gated-Clock Architecture.

The expression of the activation function including output values is very similar to the one presented in Equation 1:

$$F_a(x^+, t, z^+) = A \cdot (D + C) \tag{3}$$

where terms A and C are the same as in Equation 1, and $D = \prod_{i=1}^{m}(\lambda_i(x^+, t) \equiv z_i^+)$. Notice that the support of F_a has been extended to include the output variables z^+.

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 transition reaching S_2. Formula 3 can be modified so that only a subset of the outputs becomes part of the support of F_a. We have:

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

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

There is clearly a 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). We have devised the following 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. Then, we sort the outputs according to such values. Finally, we build function $F_a(x^+, t, z^+)$ incrementally by adding to the previously computed activation function the outputs which contribute more to the total probability value of F_a. The construction terminates when a user-defined threshold value is reached by the probability of F_a.

5 Experimental Results

The power optimization algorithms proposed in this paper have been implemented within the SIS [11] environment, and their effectiveness benchmarked onto some examples taken from the literature.

The original synchronous circuits have been optimized for area through 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 1-b. 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 used for the experiments contained NAND and NOR gates with up to four inputs, and buffers and inverters with 3 different size/drive options. Power values of the initial and final circuit implementations were obtained using the Irsim simulator [13]. All the experiments were run on a DEC-Station 5000/240 with 64 MB of main memory.

Table 1 summarizes our results obtained on some `Iscas'89` synchronous networks [7]. In particular, columns *PI*, *PO*, and *FF* show the characteristics of the circuits. Column *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 optimization. Columns *Variation* give the percentage of gate count and delay increase and power reduction obtained on each example. Finally, column F_a *Time* reports the CPU time (in *sec*) required by procedure `Reduce_Fa` to determine the simplified activation function.

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).

Since the reactive nature of a controller typically depends on the external environment, it is likely to happen that idle conditions are 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.

As an example, let us consider the `minmax3` circuit [14]; 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.

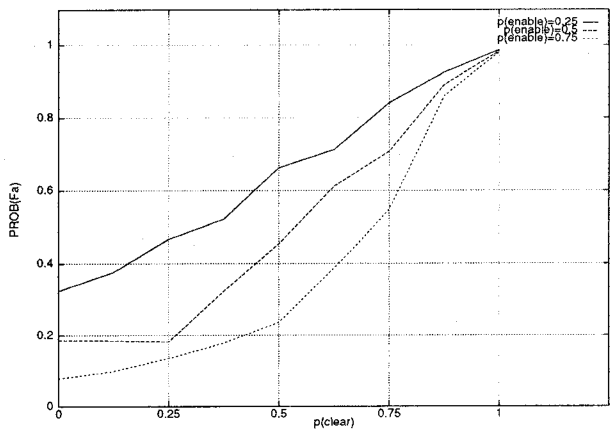

Figure 6: Case Study: The `minmax3` Circuit.

To show how the knowledge of the primary input statistics impacts the synthesis and the refinement of F_a, and thus the power savings achievable with our optimization technique, we present results for the same circuits of Table 1 in which the probability of some of the inputs has been set to values different from 0.5. Since no information was available for both the circuit functionalities 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. 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 [1, 2] would be extremely hard, because of the complexity of the STG extraction procedure. In contrast, our symbolic algorithms easily deal with these examples, even with the limited memory available on our machine. To our knowledge, these are the largest sequential circuits for which gated-clocks have been automatically generated. For some examples the power savings are sizable (25%-35%), while for others almost no advantage is given by gating the clock. The area and the delay are kept under control (8% and 5%, on average).

In Table 3 we compare the power results achieved by our method to those of [2] for some of the small, symbolic `Mcnc'91` FSMs [12] (the ones for which large savings were obtained by the explicit algorithm). From the data we can easily conclude that the symbolic approach is at least as powerful as the explicit one.

Circuit	PI	PO	FF	Before Optimization			After Optimization			Variation			F_a Time
				Gates	Delay	Power	Gates	Delay	Power	Gates	Delay	Power	
s208.1	10	1	8	90	11.00/10.98	75	95	11.07 11.05	49	+5%	+1%	−34%	7.3
s298	3	6	14	131	19.26/19.24	89	140	19.90/19.88	72	+7%	+3%	−19%	17.8
s386	7	7	6	148	14.94/14.92	63	160	15.97/15.95	58	+8%	+7%	−8%	69.9
s400	3	6	21	168	20.81/20.79	90	185	21.14/21.12	63	+10%	+2%	−30%	80.1
s420.1	18	1	16	171	16.42/16.40	106	185	17.61/17.59	67	+8%	+7%	−36%	150.3
s444	3	6	21	199	20.31/20.29	101	217	22.12/22.10	76	+9%	+9%	−25%	51.0
s510	19	7	6	289	25.62/25.60	95	306	27.31/27.29	81	+6%	+6%	−15%	301.3
s526	3	6	21	206	18.24/18.22	119	230	19.83/19.81	114	+11%	+9%	−4%	120.2

Table 1: Results for Some Iscas'89 Circuits.

Circuit	Equiprobable Inputs				Non-Equiprobable Inputs			
	PROB(F_a)	Power			PROB(F_a)	Power		
		Orig.	Opt.	Savings		Orig.	Opt.	Savings
s208.1	0.314	75	49	34%	0.831	64	17	73%
s298	0.241	89	72	19%	0.902	53	10	81%
s386	0.110	63	58	8%	0.642	52	18	65%
s400	0.249	90	63	30%	0.809	67	15	77%
s420.1	0.311	106	67	36%	0.829	90	21	76%
s444	0.249	101	76	25%	0.811	69	19	72%
s510	0.140	95	81	15%	0.670	81	45	44%
s526	0.244	119	114	4%	0.798	88	43	51%

Table 2: Results for Different Input Probability Distributions.

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%

Table 3: Comparison to the Results of [2] on the Mcnc'91 FSMs.

6 Conclusions and Future Work

We have presented a fully symbolic approach to the automatic generation of clock-gating logic for control-oriented sequential circuits. Our methodology starts from synchronous networks and does not require the 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 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 36%.

Future investigation on this subject will focus on several directions. First, approximate algorithms for FSM probabilistic analysis need to be developed to further enhance the applicability of this technique. This is because the real bottleneck of the symbolic approach is the ADD-based calculation of the exact state occupation probabilities, which becomes infeasible when the circuits contain more than a few tens of registers. Constructing and pruning the activation function, on the other hand, is neither computationally intensive nor too memory demanding. Second, we are exploring the relationship between gated-clock and precomputation-based architectures; this with the objective of integrating the two approaches into a more general and effective power management strategy. Finally, ADDs help us in accurately modeling the influence of the external environment on the circuit behavior. In fact, instead of only providing probability distributions for the primary inputs, we can actually feed the optimization tool with the statistical distributions of all the input patterns, since such information, exponential by nature, can be efficiently represented using ADDs. We are currently incorporating this enhanced modeling capability into the code.

References

[1] L. Benini, P. Siegel, G. De Micheli, "Automatic Synthesis of Gated Clocks for Power Reduction in Sequential Circuits," IEEE Design and Test of Computers, pp. 32-40, Dec. 1994.

[2] L. Benini, G. De Micheli, "Transformation and Synthesis of FSMs for Low Power Gated Clock Implementation," IEEE Trans. on CAD, Vol. CAD-15, No. 6, pp. 630-643, Jun. 1996.

[3] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Trans. on Computers, Vol. C-35, No. 8, pp. 79-85, Aug. 1986.

[4] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi, "Algebraic Decision Diagrams and their Applications," ICCAD-93, pp. 188-191, Nov. 1993.

[5] G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi, "Symbolic Algorithms to Calculate Steady-State Probabilities of a Finite State Machine," EDTC-94, pp. 214-218, Feb. 1994.

[6] G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi, "Probabilistic Analysis of Large Finite State Machines," DAC-31, pp. 270-275, Jun. 1994.

[7] F. Brglez, D. Bryan, K. Koźmiński, "Combinational Profiles of Sequential Benchmark Circuits," ISCAS-89, pp. 1929-1934, May 1989.

[8] K. Ravi, F. Somenzi, "High-Density Reachability Analysis," ICCAD-95, pp. 154-158, Nov. 1995.

[9] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, M. Papaefthymiou, "Precomputation-Based Sequential Logic Optimization for Low Power," IEEE Trans. on VLSI Systems, Vol. VLSI-2, No. 4, pp. 426-436, Dec. 1994.

[10] M. Favalli, L. Benini, G. De Micheli, "Design for Testability of Gated-Clock FSMs," EDTC-96, pp. 589-596, Mar. 1996.

[11] E. M. Sentovich, K. J. Singh, C. W. Moon, H. Savoj, R. K. Brayton, A. Sangiovanni-Vincentelli, "Sequential Circuits Design Using Synthesis and Optimization," ICCD-92, pp. 328-333, Oct. 1992.

[12] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," Technical report, Microelectronics Center of North Carolina, Jan. 1991.

[13] A. Salz, M. Horowitz, "IRSIM: An Incremental MOS Switch-Level Simulator," DAC-26, pp. 173-178, Jun. 1989.

[14] O. Coudert, C. Berthet, J. C. Madre, "Verification of Sequential Machines Using Boolean Functional Vectors," IFIP Intl. Workshop on Applied Formal Methods for Correct VLSI Design, pp. 111-128, Nov. 1989.