Saving Power by Synthesizing Gated Clocks for Sequential Circuits

LUCA BENINI
Stanford University
POLLY SIEGEL
Hewlett-Packard Laboratories
GIOVANNI DE MICHELI
Stanford University

Portable devices demand low power consumption to prolong battery life. Gating the clock is one strategy for saving power. The authors' technique identifies self-loops in an FSM and uses the function described by the self-loops to gate the clock. Applying these techniques to standard benchmarks achieved an average 25% less power dissipation at a cost of only 5% more area.

AS PORTABLE DEVICES proliferate and device sizes continue to shrink, allowing more devices to fit on a chip, power consumption takes on increased importance. Recent work has focused on accurate estimates of power consumption and on its reduction at all levels of abstraction, from high-level synthesis down to physical layout.[1-4]

Most power reduction techniques emphasize reducing the level of activity in some portion of the circuit. We extend this research by reducing the activity level of the clock by selectively stopping it. Because many sequential machines are implementations of reactive systems, they wait for a certain event to occur before changing state. This waiting wastes a lot of power. Clocking latches and computing the next-state function, for example, consume unnecessary power since nothing changes until the requisite event arrives. By stopping the clock during this period, we can realize substantial power savings in many finite state machines (FSMs).

The idea of selectively stopping the clock is not new; designers of large systems commonly use it as a part of dynamic power-management schemes.[5,6] In these schemes, however, the designer manually inserts circuitry to stop the clock for large portions of the system during inactive periods. Our technique works automatically and at a much finer granularity by recognizing wait states within FSMs and synthesizing circuitry to selectively stop the clock during these periods of inactivity.

Some asynchronous design techniques are also based on the idea of selective clocking.[7] These techniques produce circuitry in which the asynchronous FSM is clocked only when there is activity, so power management is built into the system. However, one cannot apply these techniques directly, since they require environmental and output constraints that are not valid for synchronous systems.

Our algorithms automatically synthesize the clock-gating circuitry for a sequential circuit modeled either as an FSM state table or as a synchronous network. Our technique also operates locally within an FSM, requiring no information about the environment. However, designers can use environmental signals in conjunction with our technique to save more power.

Our technique uses the knowledge

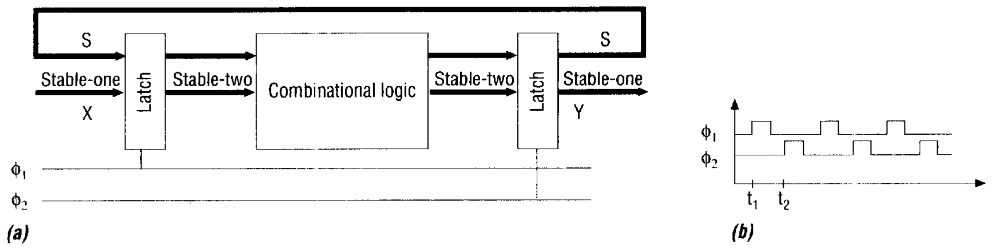


***Figure 1.** Two-phase state machine **(a)**; timing diagram **(b)**.*

of the next-state function to generate a clock activation signal only when the machine needs to perform a state transition. This is done in such a way that the machine is functionally equivalent to the original FSM, with a reduction in power dissipation and a small increase in area and critical-path delay. Applying these algorithms to benchmark circuits show an average 25% less power dissipation accompanied by a 5% increase in area.

Background

Systems with gated clocks are becoming increasingly widespread, but their implementation poses challenges that need to be understood for an automatic synthesis approach to be effective.

Gated clocks. Clocks are often gated with other signals to disable inactive parts of the system thereby saving power.[5,6] The basic idea behind a gated clocking scheme is that the environment around a functional block produces control signals that, when asserted, turn off the clock to the FSM. This reduces the power dissipated on clock lines and eliminates power dissipation through the internal FSM logic during periods of inactivity. Although gated clocks can increase clock skew, causing problems for high-performance designs, many synthesis tools provide effective clocking equalization schemes that eliminate this problem.[8]

With the explosion in portable computing, many recent designs use gated clocks to save a significant amount of power. For example, a version of the Intel Pentium chip[5] uses gated clocking techniques to stop the clock while the processor is idle, resulting in twice the average reduction in power consumption during actual system operation. A superscalar version of the PowerPC chip uses gated clocks on selected portions of the chip to obtain an average savings of 12 to 30%.[6]

Two-phase clocking. Many clocking schemes employed in large VLSI designs allow the use of gated clocks. We initially consider a two-phase nonoverlapping clocking scheme.[9] This method of clocking minimizes clock-skew problems associated with a single-phase scheme at the expense of higher area. With a two-phase clocking scheme, simply increasing the clock period reduces timing problems because there is no bilateral constraint on the clock waveform as there is with a single-phase clocking scheme.

In Figure 1, we show a general sequential circuit implemented using this clocking style. The FSM inputs and state variables enter the first set of latches, which are clocked on first phase of the clock ϕ_1. These stable-one signals must settle to their final value before ϕ_1 becomes active and the corresponding latches store values. The machine latches outputs on the second phase of the clock ϕ_2. The inputs to the second set of latches must settle to their final, stable value before the ϕ_2 clock becomes active, and thus we call them stable-two signals.

For a circuit to operate correctly under this clocking scheme, two conditions must hold. First, the two phases of the clock must not overlap. In other words, they must never be active at the same time. As a result, the designer must carefully control clock skew to preserve this property throughout the chip. Second, critical path length of the combinational logic between the two sets of latches Δt_{cl} must be such that the inputs to the final set of latches reach their final, stable value before ϕ_2 goes high. We express this as $t_1 + \Delta t_{cl} < t_2$.

These two requirements can be satified by equalizing clock distribution delays and partitioning the logic between latch boundaries in such a way that the worst-case critical path is never too long. For the bulk of this article, we assume a two-phase clocking scheme. However, simple extensions allow adapting our technique for use with other clocking schemes, as we will show later.

Self-loops

To save power, our method identifies and extracts the self-loops in a state diagram. We initially assume that the state diagram is implemented as a Moore machine. This assumption is necessary to guarantee that the output value cannot change when the machine is in a self-loop. Assuming this, however, is not overly restrictive, because documented methods exist for transforming a Mealy machine, the most general model used for control synthesis, into a Moore machine.[10]

In addition, we initially assume that the state diagram of the circuit is known. We will relax this second assumption later, however, and show how we can extract the necessary information from a gate-level specification.

We describe a Moore-type FSM by a sextuple $(X, Y, S, s_0, \delta, \lambda)$, where X is the set of inputs, Y is the set of outputs, S is the set of states, and s_0 is the initial (reset) state. Equation $s_{t+1} = \delta(x, s_t)$ gives us next-state function δ. Output function λ is defined by $y_t = \lambda(s_t)$.

The state diagram describes the reactive behavior of a sequential machine by specifying those inputs that cause the machine to transition from one state to another. Typically, we use table-based specifications to specify FSMs, in the format

$$E_{x_t, s_t} : (x_t, s_t) \rightarrow (s_{t+1}, y_{t+1})$$

where x_t and s_t are the inputs and present state, and s_{t+1} and y_{t+1}, are the corresponding next state and outputs.

The self-loop function for given state s of an FSM is a function $Self_s: X \rightarrow \{0,1\}$, which represents the conditions under which the next state remains s. The self-loop function is defined as $Self_s(x) = 1\ \forall\ x \in X$, such that $\delta(x,s) = s$. Overall, we have a set of functions $Self_s$, $s = 1, 2, \ldots, |S|$ that define the set of self-loops for the entire FSM.

For an FSM with a large number of inputs, the table often only specifies those transitions where $s_{t+1} \neq s_t$, and the self-loops are not explicitly specified. Thus, it is more convenient to compute the complement of the self-loop function for a state from its specified next-state transitions, $\overline{Self_s}(x) = 0\ \forall\ x \in X$, such that $\delta(x,s) \neq s$.

Boolean terminology

An incompletely specified Boolean function can take on either value 0,1 or don't care (DC). A DC logic value can be either 1 (true) or 0 (false); it is indicated by – in this article.

For a single output Boolean function, $f:\{0,1\}^n \rightarrow \{0,1,-\}$ and

- **the ON set is a subset of the domain such that f is 1.**
- **the OFF set is a subset of the domain such that f is 0**
- **the DC set is a subset of the domain such that f is –.**

A literal is an input variable or its complement.

A cube is a product of literals.

An implicant is a cube implying the true (1) or DC (–) value of a function.

A minterm is a product of n literals (one for each input variable) implying the true (1) value of a function.

A cover is a set of implicants covering all minterms of a function.

Removing self-loops

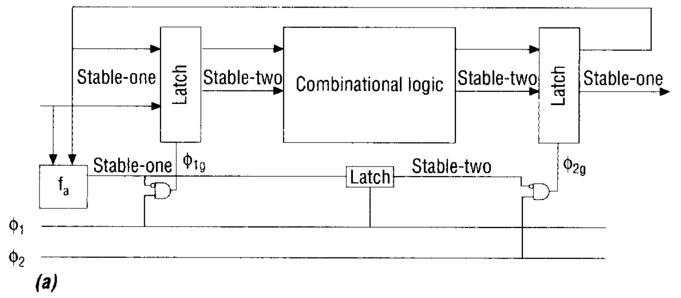
Given these definitions, we can easily identify and extract the self-loops from a state diagram. We can then use the set of self-loops to define a Boolean function that is satisfied only when the machine is in a self-loop. We then define activation function $f_a: X \times S \rightarrow \{0,1\}$ as the union of the self-loops in the FSM:

$$f_a = \left(\bigcup_{s \in S} s \cdot Self_s \right)$$

The activation function is therefore defined as a Boolean function whose inputs are the FSM's primary inputs and state bits. Because the state codes of any unreachable states never appear as inputs to f_a, we can use the unreachable codes as a don't-care (DC) set to reduce the cover size. (For definitions, see the Boolean terminology box).

We then use f_a to selectively gate the clock for power savings, as seen in Figure 2. Because the activation function f_a has inputs that are stable-one signals, f_a is also a stable-one signal and we can use it to enable and disable clock ϕ_1. Function f_a is latched and used to control the second phase of the clock, as shown in the figure.

Because the implementation of the



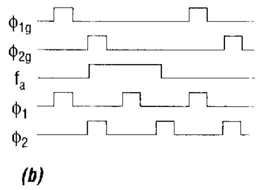


Figure 2. Two-phase state machine with gated clock (a); timing diagram (b).

activation function is often large, we extract a reduced activation function F_a from f_a. The reduced activation function balances the savings from deactivating the clock during self-loops against the area penalty from creating the gating function.

Example 1. Figure 3 shows the state diagram for a simple FSM. The figure also shows entries corresponding to the set of self-loops for the FSM and the DC set formed by the unreachable states.

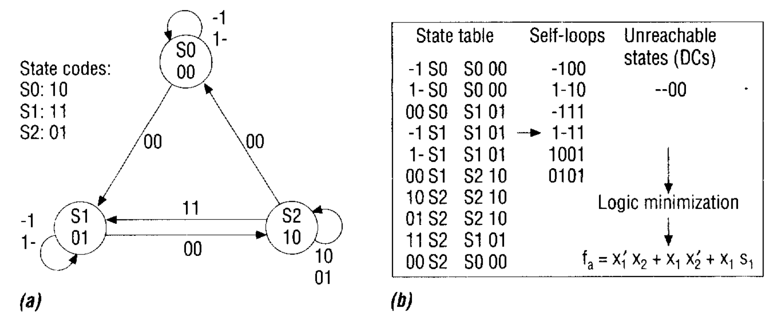
Once we find the activation function, we can implement it with the same methods used to synthesize the combinational part of the FSM.

Example 2. The activation function for example 1 is $f_a = x'_1x_2 + x_1x'_2 + x_1s_1$; it costs 6 literals. The FSM logic, on the other hand, costs 23 literals. (A logic synthesis program automatically generated the multilevel FSM logic; we omit the details for brevity.) In this case, the cost of the activation function is a significant fraction of the cost of the combinational logic in the FSM (greater than 25%). Thus the area overhead may negate some of the power savings.

As example 2 indicates, there is no guarantee that we can implement the activation function efficiently. Since it is possible that the activation function's complexity is of the same order as that of the combinational part of the FSM, we must reduce the size of the activation function to realize the most power savings.

We want to find a function with smaller overhead that covers the self-loops that consume the most power. This will balance the power saved by stopping the clock against the power consumed by the clock-stopping function. We state our problem informally as "given activation function f_a for a specific FSM, find a function $F_a \subseteq f_a$ that has an implementation with small overhead, but still covers most of the self-loops."

This problem is at least as complex as standard two-level logic minimization, a well-known, difficult problem for which a polynomial-time algorithm is not known to exist. As a result, we must devise a heuristic solution.



State table	Self-loops	Unreachable states (DCs)
-1 S0 S0 00	-100	--00
1- S0 S0 00	1-10	
00 S0 S1 01	-111	
-1 S1 S1 01	1-11	
1- S1 S1 01	1001	
00 S1 S2 10	0101	
10 S2 S2 10		
01 S2 S2 10		
11 S2 S1 01		
00 S2 S0 00		

$f_a = x'_1 x_2 + x_1 x'_2 + x_1 s_1$

Figure 3. State diagram (a); corresponding activation function (b).

The main difficulty in solving this problem is selecting a cost function that results in good power savings. This cost function must express the trade-off between the size of F_a and its efficiency in stopping the clock. An F_a that is almost as large as the original f_a will cover many self-loops, but may consume too much power. On the other hand, if F_a is a small subset of f_a, it may not cover enough of the cases where the state machine waits and we again won't realize power savings.

The area overhead of the solution (in terms of literal count) is a rough approximation of the additional power consumption. This approximation yielded good results for many example circuits.

We used the following procedure to compute F_a:

1. extract activation function f_a,
2. initially set F_a equal to f_a, and associate with F_a a (possibly void) DC set related to the unreachable states in the FSM, and
3. apply procedure reduce_cover to F_a to iteratively reduce the size of the activation function.

The best balance of trade-offs comes from using the actual transition probabilities to reduce f_a so that the reduced function covers the highest probability loops. However, the transition probabilities are not always available to the designer, since they require knowledge of the environment that the FSM operates in. Thus, we would like to come up with a good approximation based only on the functional specification of the FSM.

Our approximation involves a constraint on the number of literals in the implementation of F_a. We specify literal threshold *LT* as the upper bound for the number of literals F_a should have. We will restrict F_a to a fraction of the literals in the combinational part of the FSM to ensure that the activation function has a reasonable size and will save power. Using *LT* as a bound, we can specify the constrained optimization problem.

We state an approximation to the original problem as follows:

Given an activation function f_a, we want to find a function F_a such that cover(F_a) $\subseteq$ cover(f_a) contains the maximum number of cubes of cover(f_a) subject to the constraint that *Nlits*(F_a) < *LT*. *Nlits*(F_a) is the number of literals in the Boolean expression for cover(F_a).

In other words, we want to approximate the original activation function by a subset of the original cover. That subset should contain the largest number of self-loops that fit within the constraint on the maximum number of literals.

Either the user can specify *LT* based on the circuit architecture and constraints, or the algorithm can calculate it based on the FSM structure. In either case, selecting an appropriate value for *LT* is difficult. This is because we have no data on the final circuit implementation until after we select *LT*. One simple approach is to initially set *LT* to a percentage of the total number of literals in the combinational part of the original FSM (that is, the FSM without the activation function). A more computationally expensive approach is to start from the complete activation function (set *LT* equal to infinity) and iteratively resynthesize for decreasing values of *LT*. However, because synthesis is time consuming, this approach becomes impractical for large FSMs. Thus, selecting an optimal value for *LT* is still an open problem.

The restated problem can be solved using algorithms of varying degrees of complexity. We used a simple, greedy algorithm to solve it. The procedure shown here eliminates cubes in the original minimized cover of f_a iteratively until it reaches the literal threshold.

```
reduce_cover($F_a$) {
    compute $LT$; /* determine literal
                        threshold */
    while($Nlits(F_a) > LT$) {
        E = select_small_cubes($ONset(F_a)$);
        $c$ = select_less_essential(E,$F_a$);
        $F_a = F_a - c$
    }
}
```

After determining the literal threshold, the inner loop repeats until the number of literals in the cover of F_a falls below the literal threshold. Within the inner loop, select_small_cube selects the subset E of the cubes in F_a that have the highest number of literals (that is, the ones with the smallest cubes). Next, the procedure select_less_essential selects from E the cube c that is the most

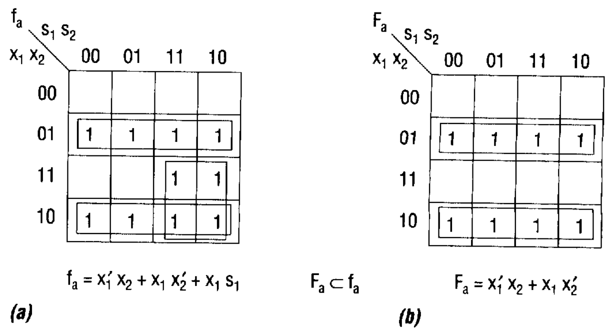


***Figure 4.** Diagram of covers of f_a (a) and F_a (b).*

covered by the other remaining cubes in F_a. In case of a tie, select_less_essential uses a tie-breaking rule based on the number of occurrences of the specified literals. We want to keep the number of occurrences of each literal as uniform as possible for uniform input loading. Finally, the procedure eliminates c from F_a, and the iteration repeats until it satisfies the the condition.

Example 3. Figure 4a gives the complete minimized cover of f_a. We want to reduce its size. Because all cubes in the cover have the same size, E contains the entire cover (the three cubes x'_1x_2, $x_1x'_2$, x_1s_1). Select_less_essential determines that the second and third cubes are partially covered by other cubes in the cover, while the first cube is essential. To choose between the two partially redundant cubes, the algorithm selects the cube that keeps the most uniform input loading. Thus, it sets c equal to third cube x_1s_1, reducing the number of input variables by 1, and eliminates this cube. The size of reduced function F_a is now 4 literals, as shown in Figure 4b.

We used many approximations to formulate this optimization problem. In reality, the implementation of f_a will use multilevel logic, but there is only a weak correlation between the size of a multilevel implementation and its corresponding minimal two-level cover. Additionally, the size of the activation function implementation only weakly correlates to the total size of the modified FSM. Finally, because power dissipation correlates weakly to total area, an activation function with a large number of literals often yields power savings that overcome a large area increase.

There is a trade-off between the number of self-loops covered and the area. In particular, the ability of F_a to reduce power dissipation decreases as we select smaller subsets of the original cover. In the results section, we explore different points of this trade-off curve by iterating with various choices of LT.

Timing issues. We must address two timing issues related to the insertion of the activation function: the effect of glitches within the clock generation circuitry and the way the activation function affects critical-path timing.

A hazard is an unwanted glitch on the output of a gate in response to input changes.[10] Hazards in synchronous systems consume excess power but do not cause the circuit to malfunction, because the signals stabilize before sampling. However, hazards in the clock generation circuitry may cause the circuit to operate incorrectly. As a result, we must examine the hazard behavior of F_a.

Because F_a gates the clock signal, a hazard on F_a may have catastrophic consequences on the internal clock line. However, because we use two-phase clocking, the AND gate feeding the clock will remain low independent of the value of F_a as long as ϕ_1 remains low. So as long as we ensure that F_a has settled before the leading edge of ϕ_1, we are not concerned with hazards in F_a. This important safety property ensures that our design technique is not sensitive to the complex, hazardous behavior of the combinational circuit implementation.

However, note that the presence of the activation function actually modifies the circuit's critical path. In fact, the longest delay through F_a adds to the maximum delay in the stage of the sequential circuit that precedes the FSM under consideration. We can see this in Figure 5. Recall that the activation function has as inputs the FSM inputs, which are sampled on ϕ_1. These input changes must propagate through the activation function logic before ϕ_1 goes high. This reduces the maximum allowable delay in the logic feeding the inputs by T_{Fa}. In other words, the system must obey the timing constraint $T_{in} + T_{Fa} < T_{max}$, where T_{in} is the delay in the logic feeding the inputs, T_{Fa} is the delay in the activation logic, and T_{max} is the timing separation between the leading edges of ϕ_1 and ϕ_2.

As a result, we must take care when applying our ideas to circuits with cycle times tightly matched to the critical-path delay. However, we can slightly increase the cycle time to eliminate any timing violations.

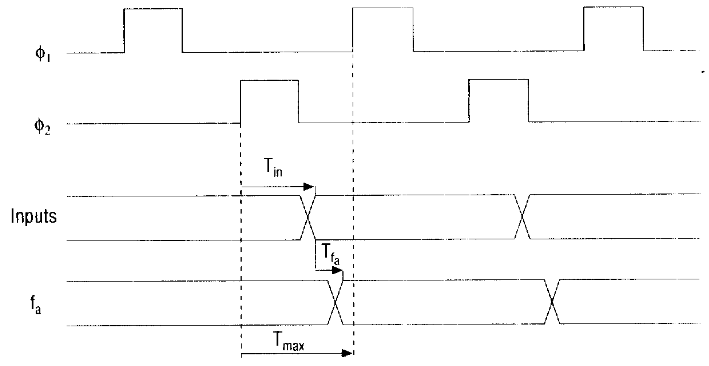


***Figure 5.** Timing requirements for activation function stabilization.*

Reducing FSM logic. Generating F_a increases the area of the FSM implementation. The presence of F_a, however, allows us to use a larger DC set to simplify the combinational part of the FSM. In particular, the DC set of every next state and output function in the FSM can be increased by the ON set of F_a. This is because for each minterm covered by the activation function, F_a will inactivate the machine. Consequently, the inputs and state values satisfying F_a will never be observed at the inputs to the combinational part. We can then use the extended DC set to recover some of the area in the combinational part of the FSM. Thus keeping an F_a with many literals can be advantageous. We can express this mathematically as $DC_{Fa}(\lambda) = DC(\lambda) \cup F_a$ and $DC_{Fa}(\delta) = DC(\delta) \cup F_a$.

Example 4. Continuing with our example circuit, if we use $f_a = x'_1x_2 + x_1x'_2 + x_1s_1$ to gate the clock, we can also use f_a as the DC set to reduce the FSM logic. This reduces the number of literals to 9. If we use reduced activation function $F_a = x'_1x_2 + x_1x'_2$ to gate the clock, then we must also use F_a as the DC set, reducing the FSM logic to 10 literals.

Implicit generation of f_a. In real-life applications, we often generate sequential circuits from specification styles other than a state diagram. Extracting a complete state diagram from a circuit involves a computation that is worst-case exponential in the number of storage elements.[11] So it is useful to find a way to generate f_a directly from a gate-level specification of the circuit.

From the definition of next-state function $\delta(x,s)$, we know that it describes a self-loop in the state diagram when $\delta(x,s)=s$. Equivalently, for each bit i in the state vector:

$$\delta_i(x,s)\overline{\oplus}s_i = 1$$

where the symbol $\overline{\oplus}$ represents the exclusive-NOR operation. Because we want the condition described earlier to be true for all bits in the state vector, our final equation is

$$\prod_i \delta_i(x,s)\overline{\oplus}s_i = 1$$

Thus, we can generate the activation function easily using Boolean decision diagram-based (BDD-based) symbolic manipulation of logic networks,[12] even if the state diagram is too large to be explicitly represented.

Recent research on FSM synthesis and verification[11] indicates that we can calculate the set of unreachable states with various degrees of safe approximation. This means that although we generate only a subset of unreachable states, we will never mark a reachable state as unreachable. We use this information to add to the DC set for the activation function, thus allowing a more efficient implementation in the implicit case as well.

Not needing the complete state diagram to generate f_a widens the applicability of our techniques and makes them suitable for resynthesis and low-power optimization of existing large sequential circuits.

Other clocking schemes. Although we based our initial formulation on a two-phase clocking discipline, we can

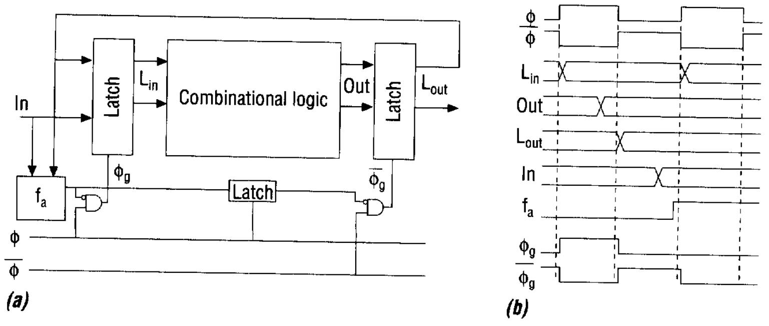


Figure 6. State machine with single-phase gated clock (a); timing diagram (b).

Table 1. Power reduction of gated clocks for MCNC benchmarks.

	Original circuit		Circuit with gated clock		
Circuit	$A_{original}$ (transistor no.)	$P_{original}$ (μW)	$A_{gated\ clock}$ (transistor no.)	$P_{gated\ clock}$ (μW)	Power reduction (%)
Ex	128	51	118	27	48
Bbsse	966	212	1,002	190	11
Bbara	348	328	390	127	61
Bbtas	178	62	188	50	19
Sse	912	175	946	150	15
S386	856	179	882	140	21
Cse	1,320	125	1,406	70	44
Dk14	758	212	852	211	1
S27	146	60	178	54	10
Mc	182	73	225	61	17
Sand1	2,220	265	2,108	180	32

easily extend these techniques to other clocking schemes.

For single-phase clocking schemes using transparent latches, we must add the time delay caused by the activation function to the delay of the functions feeding inputs to the FSM (the delay of the logic generating the previous stage's output). We use this value to determine whether the activation function violates the cycle time constraint (see Figure 6). Moreover, the skew of the gated clocks must be tightly matched to the clock skew of ungated clocks elsewhere in the circuit. This is because single-phase clocks are much more sensitive to clock skews than two-phase clocks.

Pipelined circuits. Pipelined circuits have no state feedback lines because they do not need information about past cycles to perform computations in the present cycle. However, the outputs of any given cycle contain the results of the previous cycle. We use these values as state inputs for the activation function. In this case, the sequential circuit is idle if the result of computation in the present clock cycle is the same as that of the previous cycle. The activation function can therefore be found using the implicit generation methods described in the previous section, simply by substituting the output variables into the equations in place of the state variables.

However, in the majority of cases, only a highly reduced activation function is likely to produce lower power implementations. This is because the combinational logic in the data path of pipelined circuits is typically highly optimized and the additional DC set that is available is not very useful. Moreover, the pipelined circuit's complete activation function is likely to have many inputs. This high number of inputs often leads to complex and power consuming implementations that reduce the advantages gained by stopping the clock.

Results

The described ideas and algorithms are part of Pie, a set of tools for low-power synthesis that is under development at Stanford University.

First, Pie reads the description of the circuit at the state diagram level and extracts information on the self-loops. It then easily extracts the set of unreachable states from the state diagram and uses that as the DC set for the activation function. Pie optimizes the initial cover of f_a using SIS[13] and a standard optimization procedure (script.rugged). Pie compares the size of the implementation with the size of the optimized implementation of the combinational part of the FSM. If f_a's size exceeds *LT*, Pie applies the procedure reduce_cover, and iteratively reduces the size of f_a until it finds the final optimized implementation F_a.

At this point, Pie uses F_a as an additional DC set to optimize the combinational part of the FSM, and that part is passed to SIS for logic minimization. The Ceres technology mapper[12] then maps the combinational portions of the design to obtain the final multilevel implementation. A postprocessing step automatically equalizes clock skew through buffer insertion.

Pie estimates power consumption using an accurate simulation based on a version of the switch-level simulator IRSIM. Pie calculates the average power consumption using a large number of random traces at the circuit inputs. Note that to obtain accurate power estimates, Pie must use transistor level simulations. This is because gate-level simulations do not accurately account for power dissipation in the clock lines.

Example 5. For the example circuit in Figure 3, we synthesized the original implementation along with two implementations (described in example 3) using activation functions f_a and F_a. The original FSM dissipates an average power of 51 μW. The version using the complete f_a dissipates 27 μW, while the version using F_a dissipates 37 μW. In this case, using the complete activation function results in the lowest power consumption.

For this circuit, the area decreases for both low-power implementations, although this is not normally the case. The original FSM uses 128 transistors, the version with f_a uses 118 transistors, and the version with F_a is the smallest, using 110 transistors. Because the state machine in this example does not have many state transitions, the size of the DC set used to optimize the logic is large, resulting in a slight reduction in area after logic optimization.

Benchmarks. Table 1 gives the results of running our tool on some sequential circuits from the MCNC (Microelectronic Center at North Carolina) benchmark suite. We report area overhead, average power dissipation, and power reduction. We measure area overhead ($A_{\text{gated clock}}$ and A_{original}) by the number of transistors in the implementations (including clock circuitry, the activation function, and the FSM implementation). Power reduction compares the average power dissipations of the gated clock and original implementations. It is expressed as a percentage by the following equation: power reduction = (1−power ratio)×100, where power ratio is $P_{\text{gated clock}}/P_{\text{original}}$, and P is average power dissipation.

The quality of the results depend on the selection of LT; the values in the table correspond to the literal threshold that resulted in the most power savings.

The results of applying our tool to these benchmarks depend upon the structure of the initial state machine. For example, there are a number of circuits with few or no self-loops, such as counters. Our techniques will obviously not affect the power of these circuits, since we can never stop the clock. Generally, power reduction gained by applying our techniques depends on how much the machine approximates reactive behavior. If state transitions occur only for a small fraction of the possible input vectors, our techniques yield impressive results. However, for counter-like machines, the advantage is very small or nonexistent.

It is also important to notice how the use of F_a in the DC set allows us to recover some of the area overhead imposed by the activation function. Some of the examples have very small area overhead but show a substantial power reduction.

The algorithm runs in a time-efficient fashion. Consequently, we can easily add this technique to existing FSM synthesis methods. The only bottleneck is the logic minimization of the combinational part of the FSM using F_a as a DC set.

We performed detailed electrical analyses to verify that the gated clock signals did not create incorrect behavior due to hazards, skew, or other unforeseen electrical phenomena. For every circuit we analyzed, the FSM with the gated clock was functionally equivalent to the FSM with an ungated clock and the combinational logic of the FSM established the minimum clock cycle. This fact is deceiving, because in a real circuit the machine will be embedded in a bigger structure, and timing problems can arise if combinational logic blocks belonging to the environment delay the input signals.

As we discussed earlier, the choice of F_a has a strong impact on the power savings. Looking at Figure 7, we can see the trade-offs graphically for three MCNC benchmarks that reflect different typical behaviors. The graphs plot power ratio ($P_{\text{gated clock}}/P_{\text{original}}$) against area overhead ratio ($A_{\text{gated clock}}/A_{\text{original}}$). In the figure, the Max point on each curve corresponds to the use of complete activation function f_a. The Min point corresponds to the use of greatly reduced activation function F_a, and the Med point corresponds to a partially reduced F_a.

For MCNC benchmark s27, the power consumed by f_a completely overwhelms any power saved in the FSM, resulting in no power savings. If we reduce the activation function too much, as shown by the leftmost point on the curve, the gating function incorporates few self-loops and again, we realize little power savings. However, by choosing a slightly reduced activation function, we realize substantial savings.

In contrast, benchmarks mc and sand1 behave monotonically. For mc, the size of the activation function results in an area penalty, while reducing its size increases power dissipation. If the main design concern is reducing power consumption, the Max point would be the right choice, while we obtain an area-constrained low-power implementation using Med.

Finally, for sand1, the complete activation function is the optimal choice for both power and area. This is interesting because it shows that in some cases, a large activation function may allow drastic simplification of the combinational logic of the FSM because of the use of its correspondingly large DC set.

We have described a technique for automatic synthesis of gated clocks for low-power implementations of sequential circuits. We are encouraged by the results obtained using our prototype tool, and want to extending our tech-

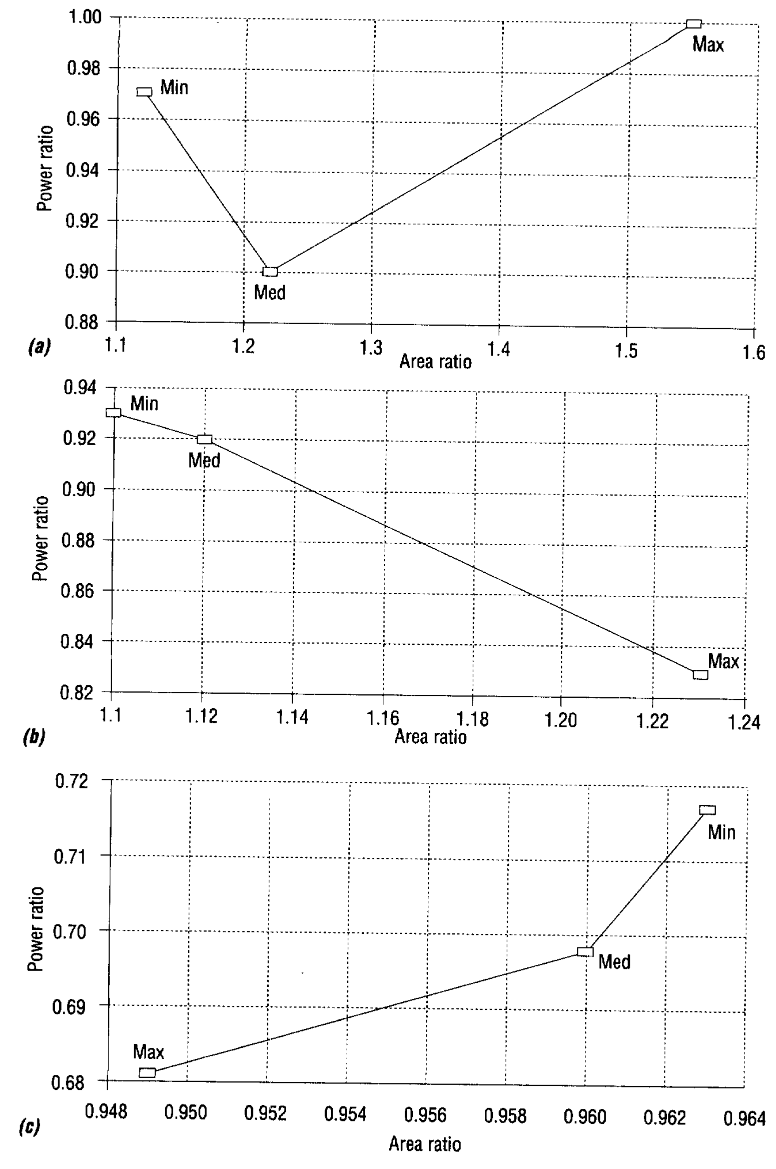


Figure 7. *Power ratio versus area overhead ratio for three MCNC benchmarks: s27 (a), mc (b), and sand1 (c).*

nique to deal with incompletely specified Mealy machines, the most general model used for control synthesis. The applicability of our technique to designs based on edge-triggered flip-flops also warrants further investigation. Finally, we are developing more advanced algorithms for synthesis of the activation function. This effort will define a closer relationship between the choice of the reduced activation function and the attainable power advantage.

Acknowledgments

We thank Steve Nowick of Columbia University for his comments on this article.

ARPA and NSF, under contract MIP 9115432, and SRC, under contract number 92-DJ-205, supported this work.

References

1. L. Benini and G. De Micheli, "State Assignment for Low-Power Dissipation," *Proc. IEEE Custom Integrated Circuits Conf.*, IEEE, Piscataway, N.J., 1994, pp. 137-139.
2. A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-Power CMOS Digital Design," *IEEE J. Solid-State Circuits*, Vol. 27, No. 4, Apr. 1992, pp. 473-484.
3. K. Roy and S. Prasad, "Circuit Activity-Based Logic Synthesis for Low Power Reliable Operations," *IEEE Trans. VLSI Systems*, Vol. 1, No. 4, Dec. 1993, pp. 503-513.
4. C. Tsui, M. Pedram, and A. Despain, "Technology Decomposition and Mapping Targeting Low-Power Dissipation," *Proc. IEEE/ACM Design Automation Conf.*, Assoc. for Computing Machinery, New York, 1993, pp. 68-73.
5. J. Schutz, "A 3.3V 0.6-μm BiCMOS Superscalar Microprocessor," *Proc. IEEE Int'l. Solid-State Circuits Conf.*, IEEE, 1994, pp. 202-203.
6. D Pham et al., "A 3.0W 75SPECINT92 85SPECfp92 Superscalar RISC Microprocessor," *Proc. IEEE Int'l. Solid-State Circuits Conf.*, IEEE, 1994, pp. 212-213.
7. S.M. Nowick and D.L. Dill, "Synthesis of Asynchronous State Machines Using a Local Clock," *Proc. Int'l Conf. Computer Design*, Computer Society Press, Los Alamitos, Calif., 1991, pp. 192-197.
8. T.-H. Chao, Y.-C. Hsu, and J.-M. Ho, "Zero Skew Clock Net Routing," *Proc. IEEE/ACM Design Automation Conf.*, ACM, 1992, pp. 518-523.

9. N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, second ed., Addison-Wesley, Reading, Mass., 1992.
10. E.J. McCluskey, *Logic Design Principles with Emphasis on Testable Semicustom Circuits*, Prentice-Hall, Englewood Cliffs, N.J., 1986.
11. H. Cho and F. Somenzi, "Sequential Logic Optimization Based on State Space Decomposition," *Proc. IEEE/ACM Design Automation Conf.*, ACM, 1993, pp. 200-204.
12. G. DeMicheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, 1994.
13. E. Sentovich et al., "Sequential Circuit Design Using Synthesis and Optimization," *Proc. Int'l Conf. Computer Design*, CS Press, 1992, pp. 328-333.

Luca Benini is a PhD candidate in electrical engineering at Stanford University, where his dissertation is on synthesis for low power. Previously, he was a research assistant at the Department of Electronics and Computer Science, University of Bologna, Italy, working on simulation techniques for power estimation. His current research interests are in computer-aided design and simulation of digital ICs for low-power systems and tools that accurately estimate power dissipation in large digital systems. He is also interested in multilevel logic synthesis, algorithms for optimal state assignment, technology mapping, and probabilistic simulation. Benini received an MS degree in electrical engineering from Stanford University, and a Laurea degree in the same field from the University of Bologna. He is a member of the IEEE and the Computer Society.

Polly Siegel is a project manager at Hewlett-Packard conducting research on information appliances; she did this work as a graduate student at Stanford. In the past she engaged in research and development in a variety of CAD areas including schematic capture, behavioral simulation, IC design systems frameworks, and asynchronous synthesis. Siegel holds a PhD in electrical engineering and a MS in engineering management from Stanford. Her BS and MS degrees in electrical engineering and computer science are from the University of California, Berkeley. She is a member of the IEEE, the Computer Society, and the ACM, and received a Best Paper award at the 1993 Design Automation Conference.

Giovanni De Micheli is associate professor of electrical engineering, and by courtesy, of computer science at Stanford University in California. His research interests include several aspects of the computer-aided design of ICs and systems, especially automated synthesis, optimization, and validation. De Micheli holds a nuclear engineer degree from the Politecnico di Milano and MS and PhD degrees in electrical engineering and computer science from the University of California, Berkeley. He authored *Synthesis and Optimization of Digital Circuits* (McGraw-Hill, 1994). He is a Fellow of the IEEE, recipient of a Presidential Young Investigator award, Editorial Board member of *IEEE Proceedings*, and associate editor of *IEEE Transactions on VLSI Systems and Integration: The VLSI Journal*.

Direct questions concerning this article to Luca Benini, Center for Integrated Systems, Stanford University, Stanford, CA 94305; luca@momus.stanford.edu.

ZAP!
How your computer can hurt you—and what you can do about it

by Don Sellers

Do not let the cute title fool you. This book is not designed to be a scare tactic to make you fear your computer. It raises consciousness and offers fair treatment of computer safety issues, letting you decide their validity. The book examines important subjects for all kinds of users in all types of workplaces. It has been reviewed and recommended by the Practices Editorial Board of the IEEE Computer Society.

160 pages. 1994. Softcover.
ISBN 1-56609-021-0.
Catalog # 6925-21 — $8.95
(Offer good until 12/31/94)

IEEE COMPUTER SOCIETY
Toll-free: 1-800-CS-BOOKS