Transformation and synthesis of FSMs for low-power gated-clock implementation

Luca Benini and Giovanni De Micheli
Center for Integrated Systems
Stanford University
Stanford, CA, 94305

Abstract

We present a technique that automatically synthesizes finite state machines with gated clocks to reduce the power dissipation of the final implementation.

We describe a new transformation for general incompletely specified Mealy-type machines that makes them suitable for gated clock implementation. The transformation is probabilistic-driven, and leads to the synthesis of an optimized combinational logic block that stops the clock with high probability.

A prototype tool has been implemented and its performance, although strongly influenced by the initial structure of the finite state machine, shows that sizable power reductions can be obtained with our technique.

1 Introduction

The majority of the currently published work in the area of automatic synthesis for low power focuses on the reduction of the level of activity in some portion of the circuit [3, 4, 5, 6], since in the dominant CMOS technology the most important fraction of the power is dissipated during switching events.

In synchronous circuits, a very promising technique is based on selectively stopping the clock in portions of the circuit where active computation is not being performed. Local clocks that are conditionally enabled are called *gated clocks*, because a signal from the environment is used to qualify (gate) the global clock signal. Gated clocks are commonly used by designers of complex power-constrained systems [10, 7]. It should be noticed, however, that it is usually responsibility of the designer to find the conditions that disable the clock.

Some attempts have been made to automate the generation of signals that can be used to gate the global clock. In [1] a *Precomputation-based approach* has been described that focuses mainly on data-path circuits, while in [2] the authors have described a method to generate gated clocks for systems described as finite state machines.

Our previous work [2] exploits the concept of *self-loop*, an idle condition for a Moore machine. If the machine is in a self-loop, the next state and the output do not change, therefore clocking the FSM only wastes power. Obviously, detecting self-loop conditions requires some computation to be performed by additional circuitry. This computation dissipates power, and sometimes it will be too expensive to detect all self-loop conditions. It is therefore very important to select a subset of all self-loops that are taken with high probability during the operation of the FSM.

We have improved in several directions and extended the validity of the techniques discussed in our previous work [2]. First, applicability of our techniques has been increased, and the limitation to Moore finite state machines has been removed. Our new method deals with a very general model of sequential circuit, the *incompletely specified Mealy machine*. Second, we adopt a novel probabilistic approach, that can selectively individuate and exploit the idle conditions that occur with high probability. Moreover, our new algorithms for the synthesis of the clock-stopping logic are more accurate an powerful, and are able to find exact and heuristic solutions as well.

A tool has been implemented and applied to a number of benchmark circuits. We have embedded our tool in a complete path from high-level specification to transistor-level implementation, and we have verified our results using accurate switch-level simulation.

For some circuits, more than 100% improvement in average power dissipation have been obtained. Notice that the quality of the results is strongly dependent on the type of finite state machine we start with. In particular, our method is well suited for FSMs that behave as *reactive systems*: they wait for some input event to occur and they produce a response, but for a large fraction of the total time they are idle. These FSMs are common in portable devices where low power consumption is important.

2 Background

In this work we will assume a single clock scheme with edge triggered flip-flops, as shown in Figure 1 (a). This is not a limiting assumption. We have discussed the applicability of our methods to different clocking schemes in [2]

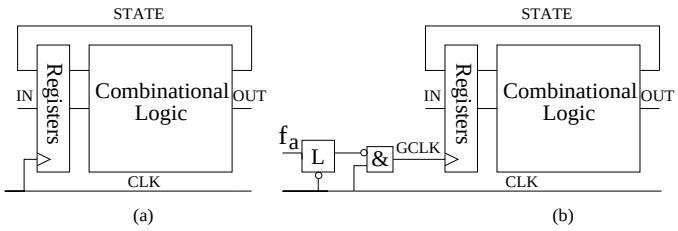

Figure 1: (a) Single clock, flip-flop based finite state machine. (b) Gated clock version.

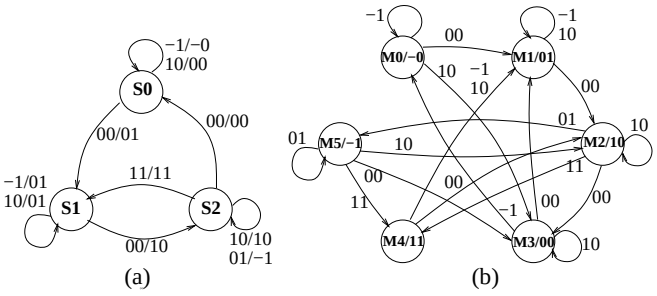

Figure 2: (a) STG of a Mealy machine. (b) STG of the equivalent Moore machine.

where we used transparent latches and multiphase clocks.

With a gated clock implementation, we need to slightly modify the structure in Figure 1 (a). We define a new signal called *activation function* (f_a) whose purpose is to selectively stop the local clock of the FSM, when the machine is idle and do not perform state transitions. When $f_a = 1$ the clock will be stopped. The modified structure is shown in Figure 1 (b). The block labeled "L" represents level-sensitive register, transparent when the global clock signal CLK is low. Notice that the presence of L is needed for a correct behavior, because possible glitches on f_a must be filtered out when the global clock signal is high.

We assume that the activation function f_a becomes valid before the raising edge of the global clock. At this time the clock signal is low and L is transparent. If the f_a signal becomes high, the upcoming edge of the global clock will not filter trough the AND gate, therefore the FSM will not be clocked and GCLK will remain low. Note that when the global clock is high, L is not transparent, therefore the GCLK signal is forced low at least up to the next falling edge of the global clock.

When the local clock is stopped, no power is consumed in the FSM combinational logic, on the clock line and in the sequential elements (differently from the scheme proposed in [1] where enabling signals are used). Notice however that the delay of the logic for the computation of f_a is on the critical path of the circuit, and its effect must be taken into account during timing verification.

Our technique automatically generates the activation function in form of a combinational logic block that uses as its inputs the primary input IN and the state lines STATE of the FSM. The input data for our algorithm is the behavioral description of the FSM and the probability distribution of the input signals.

In the following subsections we will describe some basic concepts from automata and probability theory that will be useful for the understanding of our algorithms. Refer to [14, 8] for a more detailed treatment.

2.1 Models of finite state systems

A Mealy-type FSM can be described by a six-tuple $(X, Y, S, s_0, \delta, \lambda)$ 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. The next state function δ is given by:

$$s_{t+1} = \delta(X, s_t) \tag{1}$$

The output function λ is defined as:

$$y_t = \lambda(X, s_t) \tag{2}$$

If the machine is incompletely specified δ and λ are partial functions. For a Moore FSM the output *does not* depend on the input, therefore λ_M is defined as:

$$y_t = \lambda_M(s_t) \tag{3}$$

Conceptually, Mealy and Moore machines are equivalent, in the sense that it is always possible to specify a Moore machine whose input-output behavior is equal to a given Mealy machine behavior, and viceversa [11]. Practically, however, there is an important difference. The Mealy model is usually more compact than the Moore model, in fact the transformation from Mealy to Moore involves a state splitting procedure that may significantly increase the number of states and state transitions [11].

Example 1 *In Figure 2 (a), a Mealy machine is represented in form of state transition graph (STG). If it is transformed in the equivalent Moore machine (using the procedure outlined in [11]), the new STG is shown in Figure 2 (b). The higher complexity in terms of states and edges of the Moore representation is evident. Notice that both FSMs are incompletely specified.*

2.2 Probabilistic models of FSMs

We model the probabilistic behavior of a general FSM using a Markov chain [8], as done in [9, 6, 16]. This model can be described by a weighted directed graph with a structure isomorphic to the STG of the machine. For a transition from state s_i to state s_j, the weight $p_{i,j}$ on the corresponding edge represents the *conditional probability* of the transition (i.e., the probability of a transition to state s_j *given* that the machine was in state s_i). Symbolically this can be expressed as:

$$p_{i,j} = Prob(Next = s_j | Present = s_i) \tag{4}$$

The $p_{i,j}$ are collected in a matrix $\mathbf{P}$ and depend on the probability distribution of the inputs, that is initially known. However, using the conditional probability as an estimate of the total transition probability can lead to large errors, because the probability of a transition strongly depends on the probability for the machine to be in the state tail of the transition.

In order to find the probability of a transition without any condition, we need to know the *state probabilities* q_i, that represent the probability for the machine to be in a

given state i. Namely, the *total transition probabilities* we are looking for are

$$r_{i,j} = p_{i,j} q_i \tag{5}$$

Many methods have been proposed to calculate the state probabilities [8, 9]. In this work we have used the *Power Method*. Using this method, the state probability vector $\mathbf{q} = [q_1, q_2, ..., q_{|S|}]^T$ can be computed using the iteration:

$$\mathbf{q}_{n+1}^T = \mathbf{q}_n^T \, \mathbf{P} \tag{6}$$

with the normalization condition $\sum_{i=1}^{|S|} q_i = 1$ until convergence is reached. The convergence properties of this method are discussed in [9]. The power method has been chosen because of its simplicity and its applicability (if sparse matrix manipulation or symbolic formulation are used [9]) to FSMs with a very large number of states. In the following sections we assume that the state probability vector and the total transition probabilities have already been computed.

If we now consider a Boolean function f with inputs the state and input variables of the machine, we can compute its probability (the probability for the function to be 1) in an exact fashion. If f is specified in cover form (a list of cubes that cover the ON-set of the function), the probability of f can be calculated using the following steps.

- Make the cover disjoint.
- Compute the probability of the disjoint cubes.
- Sum the disjoint cube probabilities.

Notice that this calculation is of vital importance in our algorithm, that performs a search based on the probability of the activation function.

3 Problem formulation

Given the knowledge of the FSM structure and its probabilistic model, we first want to identify the idle conditions where the clock may be stopped. If the machine is a Moore one, this is a simple task. For each state s we identify all the input conditions such that $\delta(x, s) = s$. We therefore define a set of *self-loop state function* $Self_s : X \rightarrow \{0, 1\}$ such that $Self_s = 1 \;\; \forall x \in X \; where \;\; \delta(x, s) = s$.

We then encode the machine. After the encoding step every state will have a unique code: $s_i \leftrightarrow e_i$ and $e_i = (e_{i,1}, e_{i,2}, ..., e_{i,|V|})$, where V is the set of the state variables used in the encoding.

Finally, the *activation function* is defined as $f_a : X \times V \rightarrow \{0, 1\}$:

$$f_a = \sum_{i=1,2,...,|S|} Self_{s_i} \cdot e_i \tag{7}$$

These definitions can be clarified using an example.

Example 2 *For the Moore machine in Example 1, the self-loop state function for state M5 is* $Self_{M5} = in_0' in_1$ *Similarly, all the other self-loop state functions can be obtained. We encode the states using three state variables,* v_1, v_2, v_3. *The encodings are:* $M0 \rightarrow v_1' v_2' v_3'$, $M1 \rightarrow v_1' v_2 v_3'$, $M2 \rightarrow v_1 v_2 v_3'$, $M3 \rightarrow v_1 v_2 v_3$, $M4 \rightarrow v_1' v_2 v_3$, $M5 \rightarrow v_1' v_2' v_3$. *The activation function is therefore:* $f_a = in_2 v_1' v_2' v_3' + in_2 v_1' v_2 v_3' + in_1 in_2' v_1' v_2 v_3' + in_1 in_2' v_1 v_2 v_3' + in_1 in_2' v_1 v_2 v_3 + in_1' in_2 v_1' v_2 v_3$.

If the machine is Mealy-type, the problem is substantially more complex. The knowledge of the state and the input is not sufficient to individuate the conditions when the clock can be stopped. If only the next state lines and the inputs are available for the computation of the activation function, we do not have a way to determine what was the output. This is a direct consequence of the Mealy model: since the outputs are on the edges of the STG, we may have the same next state for many different outputs. The important consequence is that, even if we know that the state in not going to change, we cannot guarantee that the output too will remain constant, therefore we cannot safely stop the clock.

There are two ways to solve this problem. The simpler way is to use the outputs of the FSM as additional inputs for the activation function. The other approach is to transform the STG in such a way that the FSM will be functionally compatible with the original one, but only the input and state lines will be sufficient to compute the activation function.

We decided to investigate the second method for two main reasons. First, since for many FSMs the number of output signals is large, it is likely that adding all the output signals to the inputs of the activation function will produce poor results because of the high complexity of the activation function itself. Second, in the present implementation our tool operates using state transition tables as input, therefore we still have the freedom to modify the number of states and the STG structure (this is not the case if we start from a synchronous network that is an implementation of the STG).

The simplest transformation that enables us to use only input and state signals as inputs of the activation function f_a, is a Mealy to Moore transformation. The algorithm that performs this conversion is well known [11] and its implementation is simple, but it may sensibly increase the number of states and edges (correlated with the complexity of the FSM implementation).

3.1 Locally-Moore machines

We now define and study a new kind of FSM transformation that enables us to use a Moore-like activation function without a large penalty in increased complexity of the FSM. A *Moore state* is a state such that all incoming transitions have the same output field. Formally: $s \in S \mid \forall x \in X, r \in S, \delta(x, r) = s \;\Rightarrow\; \lambda(x, r) = const.$

Proposition 1 *A Mealy-state s with k different values of the output fields on the edges that have s as a destination can be transformed in k Moore-states. No other state splitting is required.*

We could transform the FSM simply applying the Mealy to Moore transformations locally to states that have self loops. The local Moore transformation has the advantage

that it allows us to concentrate only on states with self-loops, avoiding the useless state splitting on the states without self-lops. Still, this is not enough, because for many examples all the states will have self-loops and the local transformation will produce the complete Moore equivalent machine.

Our next step is to further localize the transformation. Consider an incompletely specified Mealy-machine. In general we may have many different outputs for different inputs, even if the next state is always the same. Intuitively we want split the Mealy state with self-loops simply in a couple of states. One of the two states will be Moore-type with a self-loop that has maximum probability.

We define the *maximum probability state self-loop function* $MPself_s : X \rightarrow \{0,1\}$. Its ON-set represents the set of input conditions for a state that are on self-loops and produce compatible outputs (two outputs fields are compatible if they differ only in entries where at least one of the two is don't care) and are taken with maximum probability.

To find $MPself_s$ we group the self-loops from state s in (possibly overlapping) compatibility classes, we then compute the probability of each compatible class and we choose the class with maximum probability as the ON set of $MPself_s$.

Example 3 *In the Mealy machine of Example 1, if we consider state S2, we have two self loops: $in_1 in_2'$ with output* 10 *and $in_1' in_2$ with output* -1*. The two output fields are not compatible, therefore we have two compatible classes (the same two functions). We will choose the class that is more probable. In this particular example, we assumed equiprobable and independent inputs and both functions have the same probability, therefore one of the two is randomly chosen.*

Once the $MPself_s$ functions have been found for all the states with self-loops, the second step of our transformation algorithms is performed. If a state s is not a Moore-state, it is split in two states s_o and s_l. The first state (s_o) is the original one, but its edges corresponding to the self-loops included in $MPself_s$ become transitions from s_o to s_l. The second state (s_l) is reached only from s_o and has a self-loop corresponding to $MPself_s(x)$. All the outgoing edges that leave s_o are replicated for s_l. The s_l state is now Moore-type, because all the edges that have s_l as tail have the same output field.

This procedure is advantageous for many reasons. First, the increase in the number of states is tightly controlled (in the worst case, if all the states are Mealy-type and have self-loops, we can have a twofold increase in the number of states). Second, the self-loops with maximum probability are selected. Third, if we really want to limit the increase in the number of states, we may define a threshold: only the first k states in a list ordered for decreasing *total probability* of $MPself_s$ are duplicated.

We call the FSM obtained after the application of this procedure *locally-Moore* FSM, because in general only a subset of the states is Moore-type.

Example 4 *The application of our procedure on the Mealy machine of Example 1 produces the locally-Moore FSM shown in Figure 3. The shaded areas enclose states that have been split. The Moore-states with self-loops are drawn with bold lines. The number of states and edges of the locally Moore machine is smaller than those that we obtained with the complete Mealy to Moore transformation.*

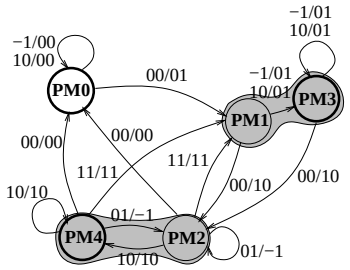

Figure 3: STG of the locally-Moore FSM

If we restrict our consideration to Moore-states, we can generate an activation function that uses as inputs only the state and primary input of the FSM. The next step is to generate an activation function that produces a final implementation with minimum power dissipation.

3.2 Optimal activation function

If we call $S' \subseteq S$ the set of Moore-states in a FSM, the complete activation function is given by

$$f_a = \sum_{i=1,2,\ldots,|S'|} Self_{s_i} \cdot e_i \qquad (8)$$

where e_i is the encoding of the states in S'. The simplest approach is to try to use the complete f_a as activation function. This is seldom the best solution, because the complexity if f_a can be too high, and the power dissipated by its implementation may reduce or nullify the power reduction that we obtain stopping the clock. It is therefore necessary to be able to choose a simpler function contained in f_a whose implementation dissipates minimum power, but whose efficiency in stopping the clock is maximum.

In [2] we proposed a simple greedy algorithm that will be shortly outlined. First, the f_a is two-level minimized, and a minimum cover is obtained. Then, the larger cubes in the cover are greedily selected until the number of literals in the partial cover exceed a user-specified literal threshold. The rationale of this approach is that generally large cubes have high probability and the primes that compose a minimum cover are as large as possible.

The solution proposed in [2] is highly heuristic and can be improved. We have devised a new strategy for the synthesis of reduced activation functions that exploits the knowledge of the probability of the self-loops. Moreover, we have studied and solved the new combinational synthesis problem that arises when we want to find a minimum complexity function that is active with a pre-fixed probability.

We do not describe in detail our new algorithms because of space limitations. Intuitively, we find the optimal activation function using a branch-and-bound algorithm that select a minimum-literal-count cover of a function $F_a \subseteq f_a$ that is guaranteed to be active (one) with

probability $P(F_a) \geq \alpha P(f_a)$. The parameter $\alpha \leq 1$ is user-defined.

We can now briefly summarize the full procedure used for the synthesis of our low-power gated clock FSMs.

- The Mealy machine is transformed in an equivalent locally-Moore machine.
- The complete activation function f_a is extracted from the Moore-states of the locally-Moore machine.
- The probability of the complete f_a is computed.
- The branch-and-bound algorithm finds the minimum literal count solution F_a whose probability is a pre-specified fraction α of the probability of f_a
- F_a is used as additional DC set for the combinational logic of the FSM.

The last step (more thoroughly described in [2]) can sensibly improve the quality of the results, in particular if F_a is large. Unfortunately its effect is to greatly increase the theoretic complexity of the problem, because it is very hard to foresee the effect of F_a used as DC set. Sometimes it may be convenient to choose a F_a that is not minimal in the sense discussed above, if it allows a large simplification in the combinational part of the FSM. Our heuristic approach is to try different F_a of decreasing size (and complexity), in an attempt to explore the trade-off curve. We generate a set of solutions using different values of α, in such a way that the possible range of solutions is uniformly sampled. The details of this approach will not be discussed here for space reasons.

4 Experimental results

We implemented the described algorithms as a part of a tool-set for low-power synthesis that we are developing. The tool reads the state transition table of the FSM. The first step is the transformation of the Mealy machine to a Locally-Moore machine and the extraction of the self-loops from the Moore-states. We then apply the power method to compute the exact state probabilities given an input probability distribution

Once the state codes have been assigned (using JEDI [17]), our probabilistic-driven procedure for the selection of the activation function can start. First, all the primes of the activation function are generated using symbolic methods [15], then the probability of the complete activation function f_a is computed starting from a minimized cover (obtained with ESPRESSO [12])

The user specifies the number of activation functions that the procedure should generate, and the branch-and-bound algorithms finds reduced activation functions as many times as it is requested. Surprisingly, for all the MCNC benchmarks this step has never been the bottleneck. This is certainly due to the fact that the majority of the FSM MCNC benchmarks do not have a large number of self-loops (in particular the larger ones).

The combinational logic of the Locally-Moore FSM is then optimized in SIS [12] using the additional DC set given by the activation function. The DC-based minimization of the combinational logic using the activation functions is the main bottleneck of our procedure. In our tool the user has the possibility to specify a CPU-time limit for each minimization attempt. This of course limits the possible improvements obtainable on large FSMs.

	Original		Locally-M.		Gated		
Circuit	Size	P	Size	P	Size	P	%
bbara	330	67	422	72	408	34	97
bbsse	640	121	742	137	736	119	2
bbtas	142	56	138	57	164	44	27
keyb	721	128	754	132	820	114	12
lion9	188	60	226	60	248	52	15
s298	7492	899	7496	900	7502	810	11
s420	544	132	544	132	602	108	22
scf	3222	437	3222	437	3169	400	9
styr	1474	159	2468	230	2534	208	0
test	348	73	442	76	374	32	128

Table 1: Results of our procedure applied to MCNC benchmarks. Power is in μW.

The activation functions are also optimized using SIS, then the alternative solutions are mapped with CERES [13], and the gated clocking circuitry is generated. Finally the alternative gated clock implementations and the implementation of the original Mealy FSM are simulated with a large number of test patterns using a switch level simulator (IRSIM [18]) modified for power estimation.

The quality of the results strongly depends on two factors. First, how much state splitting has been needed to transform the machine in a locally-Moore one. Second, for what percentage of the total operation time the FSM is in a self-loop condition (this depends on the FSM structure and on the input probability distribution). For machines with a very small number of self-loops or a very low-probability complete activation function, the area of improvement is almost null. This is the case for many MCNC benchmarks for which the final improvement is negligible. As for the first problem, it may be worthy to investigate if, in case the state duplication is too high, using an activation function with the outputs of the FSM as additional inputs may lead to better results.

Table 1 reports the performance of our tools on a subset of the MCNC benchmarks. The first six columns show the area (number of transistors) and the power dissipation of the normal Mealy FSM, the locally-Moore FSM without gated clock and the locally-Moore machine with gated clock. The last column shows the power improvement, computed as $(100(P_{mealy}/P_{gated} - 1))$. Notice that, if there is no power improvement this number is set to 0.

The tool is able to process all the benchmarks, but in the table we list examples representative of various classes of possible results. The benchmarks `bbara` and `test` are reactive FSMs. The high number and probability of the self-loops allow an impressive reduction of the total power dissipation, even if the area penalty can be not negligible.

In contrast, for `bbsse` and `styr` there is no power reduction or even a power increase. The `bbsse` benchmark is

representative of a class of machines where the number and probability of the self-loops is too small for our procedure to obtain substantial power savings. The `styr` benchmark has many self-loops, but they all have very low probability. Moreover, the transformation to locally-Moore machine is paid in this case with a too large area overhead.

For the other examples in the table the power savings vary between 10% and 30%. For some of these machines (`s420` and `scf`), the self-loops are only on Moore states and there is no area overhead for the locally-Moore transformation. We included some of the large examples in the benchmark suite (`s298` and `scf`) to show the applicability of our method to large FSMs.

From the observation of the results, it is quite clear that several complex trade-offs are involved. First, the transformation to locally-Moore machine can sometimes be very expensive in terms of area overhead. Second, the choice of the best possible activation function is paramount for good results. In fact, for many examples, the complete activation function was too large, and reduced activation functions gave better results.

5 Conclusions and future work

We have described a technique for the automatic synthesis of gated clocks for FSMs of a very general class. We want to emphasize that our method is a complete procedure, from the FSMs high-level specification to the fully mapped network, and it has been tested with accurate power estimation tools. The quality of our results depends on the initial structure of the FSM, but we obtain substantial power savings for a large class of finite state machines.

We have presented a new transformation for Mealy FSMs that makes them suitable for gated-clock implementation, and outlined a complete procedure for the synthesis of an optimized combinational logic function whose purpose is to stop the clock with high probability during the operation of the machine.

Future research will concentrate on the implementation of fully symbolic algorithm for the synthesis of the activation function and on the application of our techniques to large synchronous networks.

6 Acknowledgements

This research is supported by NSF and ARPA under contract number 9115432.

References

[1] M. Alidina, et al., "Precomputation-based sequential logic optimization for low power," in *ICCAD, Proceedings of the International Conference on Computer-Aided Design*, pp. 74–80, Nov. 1994.

[2] L. Benini, P. Siegel and G. De Micheli, "Automatic synthesis of gated clocks for power reduction in sequential circuits" *IEEE Design and Test of Computers*, pp. 32–40, Dic. 1994.

[3] A. Shen, A. Ghosh, S. Devadas, and K. Keutzer, "On average power dissipation and random pattern testability of CMOS combinational logic networks," in *ICCAD, Proceedings of the International Conference on Computer-Aided Design*, pp. 402–407, Nov. 1992.

[4] C. Tsui, M. Pedram, and A. Despain, "Technology decomposition and mapping targeting low power dissipation," in *DAC, Proceedings of the Design Automation Conference*, pp. 68–73, June 1993.

[5] K. Roy and S. Prasad, "Circuit activity based logic synthesis for low power reliable operations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 4, pp. 503–513, Dec. 1993.

[6] L. Benini and G. De Micheli, "State assignment for low power dissipation," in *CICC, Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 136–139, May 1994.

[7] B. Suessmith and G. Paap III, "PowerPC 603 microprocessor power management," *Communications of the ACM*, no. 6, pp. 43–46, June 1994.

[8] K. Trivedi. *Probability and statistics with reliability, queuing and computer science applications.* Prentice-Hall, 1982.

[9] G. Hachtel, E. Macii, A. Pardo and F. Somenzi "Symbolic algorithms to calculate Steady-State probabilities of a finite state machine," in *Proc. of IEEE European Design and Test Conf.*, pp. 214–218, Feb. 1994.

[10] J. Schutz, "A 3.3V 0.6μm BiCMOS superscalar microprocessor," in *IEEE International Solid-State Circuits Conference*, pp. 202–203, Feb. 1994.

[11] J. Hartmanis and H. Stearns, *Algebraic Structure Theory of Sequential Machines.* Prentice-Hall, 1966.

[12] E. Sentovich, et al., "Sequential circuit design using synthesis and optimization," in *ICCD, Proceedings of the International Conference on Computer Design*, pp. 328–333, Oct. 1992.

[13] F. Mailhot and G. De Micheli, "Algorithms for technology mapping based on binary decision diagrams and on Boolean operations," *IEEE Transactions on CAD/ICAS*, pp. 599–620, May 1993.

[14] G. De Micheli. *Synthesis and optimization of digital circuits.* McGraw-Hill, 1994.

[15] O. Coudert and C. Madre, "Implicit and incremental computation of primes and essential primes of Boolean functions," in *DAC, Proceedings of the Design Automation Conference*, pp. 36–39, June 1992.

[16] R. Marculescu, D. Marculescu and M. Pedram, "Switching activity analysis considering spatiotemporal correlations," in *ICCAD, Proceedings of the International Conference on Computer-Aided Design*, pp. 294–299, Nov. 1994

[17] B. Lin and A. R. Newton, "Synthesis of multiple-level logic from symbolic high-level description languages," in *Proc. of IEEE Int. Conf. On Computer Design*, pp. 187–196, Aug. 1989.

[18] A. Salz and M. Horowitz, "IRSIM: an incremental MOS switch-level simulator," in *DAC, Proceedings of the Design Automation Conference*, pp. 173–178, June 1989.