# Optimal synthesis of gated clocks for low-power Finite-State Machines

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

## Abstract

*The automatic synthesis of low power FSMs with gated clocks relies on efficient algorithms for the synthesis and optimization of dedicated clock-stopping circuitry. In a previous paper [3] we have described a framework for the transformation of FSMs and the extraction of input and state conditions where the clock can be safely stopped without modifying the external behavior.*

*In this paper we concentrate on the optimal synthesis of the logic block that controls the local clock of the FSM. We formulate and solve a new logic optimization problem, namely, the synthesis of a sub-function of a Boolean function that is minimal in size under a constraint on its probability to be one. We describe the relevance of this problem for the optimal synthesis of gated clocks.*

*A prototype tool has been implemented and its performance, although 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 [5, 6, 7, 8], since in CMOS technology the largest fraction of the power is dissipated during switching events.

In synchronous circuits, it is possible to selectively stop 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 large power-constrained systems [11, 17]. Notice however that it is usually responsibility of the designer to find the conditions that disable the clock.

Two different approaches to the automatic synthesis of logic circuits that can be conditionally disabled by environmental signals have been reported so far. In [1] Alidina et. al have described a *precomputation-based approach* that focuses mainly on data-path circuits, while the authors have described in [2] a method to generate gated clocks for systems described as finite state machines.

Our work is based on the observation that during the operation of a FSM there are conditions such that the next state and the output do not change. Therefore, clocking the FSM only wastes power in the combinational logic and in the registers. If we are able to detect when the machine is idle, we can stop the clock until a useful transition must be performed and the clocking must resume. The presence of a gated clock has a two-fold advantage. First, when the clock is stopped, no power is consumed in the FSM combinational logic, because its inputs remain constant. Second, no power is consumed in the sequential elements (flip-flops) and the gated clock line (differently from the scheme proposed in [1] where enabling signals are used).

Obviously, detecting idle conditions requires some computation to be performed by additional circuitry. This computation dissipates power, and sometimes it will be too expensive to detect all idle conditions. It is therefore very important to select a subset of all idle conditions that are taken with high probability during the operation of the FSM. We have shown in [2] that idle conditions correspond to self-loops of Moore FSMs, therefore it is relatively easy to detect them. Idle conditions in Mealy FSMs can also be detected, but with more effort.

In [2] two problems were left open. First, our method was applicable only to Moore-type FSMs, second, the synthesis of the clock-stopping circuitry was based on a simple and fast heuristic. In [3], we removed the limitation to Moore-type FSMs, extending the applicability of our approach to the more general class of incompletely specified Mealy-type FSMs. In this work we address the synthesis of the clock-stopping logic. More in detail, we formulate and solve a new logic synthesis problem, namely the choice of a minimum-complexity sub-function $F_a$ of a given Boolean function $f_a$, such that its probability of being one is larger than a predefined fraction of the total probability of $f_a$. From a practical point of view, this means choosing a suitable subset all idle conditions such that the clock-stopping circuitry dissipates minimum power but stops the clock with high efficiency.

In order to verify our results, we embedded our tool in a complete path from high-level specification to transistor-level implementation and we employed accurate switch-level simulation, because gate level power estimation has limited accuracy. For some circuits more than 100% improvement (computed as $100(P_{orig}/P_{gated} - 1)$) in average power dissipation has been obtained, but quality of the

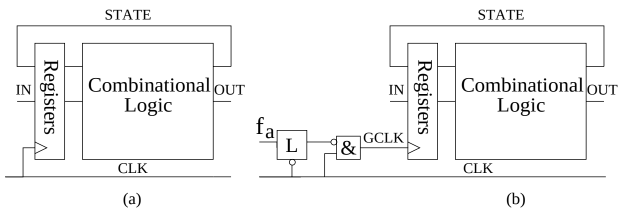

Figure 1: *(a) Single clock, flip-flop based finite state machine. (b) Gated clock version.*

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.

## 2 Background

In this work we will assume a single clock scheme with edge-triggered flip-flops, shown in Figure 1 (a). This is not a limiting assumption. We have indeed applied our methods to different clocking schemes in [2] (where we used transparent latches and multiphase clocks).

A gated clock FSM is obtained modifying 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 does not perform state or output transitions. When $f_a = 1$ the clock will be stopped. The modified structure is shown in Figure 1 (b). The block labeled "L" represents a latch, transparent when the global clock signal CLK is low. Notice that the presence of the latch is needed for a 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 $f_a$ is on the critical path of the circuit, and its effect must be taken into account during timing verification.

The modified circuit operates as follows. 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 the latch L is transparent. If the $f_a$ signal becomes high, the upcoming edge of the global clock will not filter through the AND gate and therefore the FSM will not be clocked and GCLK will remain low. Note that when the global clock is high, the latch is not transparent and the negated input of the AND gate cannot change at least up to the next falling edge of the global clock.

The activation function is a combinational logic block that uses as its inputs the primary input IN and the state lines STATE of the FSM. No external information is used, the only 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 [22, 14] for a more detailed treatment.

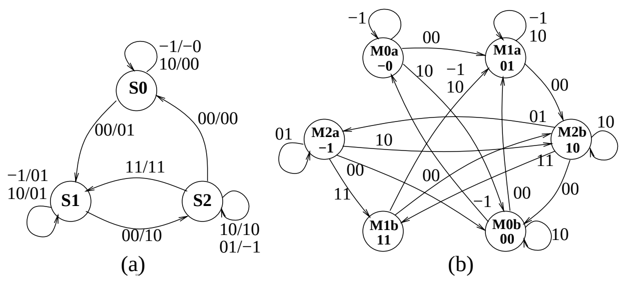

Figure 2: *(a) STG of a Mealy machine. (b) STG of the equivalent Moore machine.*

### 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 $\delta$ is given by:

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

The output function $\lambda$ is defined as:

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

The definition of Moore-type FSM is similar, with the only exception of the output function. For a Moore FSM the output *does not* depend on the input. Therefore we define $\lambda_M$ 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 [18]. Practically, however, there is an important difference. The Mealy model is usually more compact than the Moore model. Indeed the transformation from Mealy to Moore involves a state splitting procedure that may significantly increase the number of states and state transitions [18]. When *don't care* conditions ($DC$) are present, the FSM becomes *incompletely specified*, $\lambda$ and $\delta$ become *partial functions.*

**Example 1** *In Figure 2 (a), a Mealy machine is represented in form of state transition graph (STG). If it is transformed to the equivalent Moore machine (using the procedure outlined in [18]), the new STG and is shown in Figure 2 (b). The STG of the Moore machine has the output associated with the states, while in the Mealy model the outputs are associated with the edges. The higher complexity in terms of states and edges of the Moore representation is evident. Notice that both FSMs are incompletely specified, because of some output don't cares, represented with "–" in the output fields.*

### 2.2 Probabilistic models of FSMs

We model the probabilistic behavior of a general FSM using a Markov chain [14] (as done in [15, 8, 24, 25]), 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 [14, 15]. 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 [16]. The power method has been chosen because of its simplicity and its applicability (if sparse matrix manipulation or symbolic formulation are used [15]) 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 using the power method and equation (5).

Knowing input and state probability distribution allows us to compute the probability of a Boolean function $f$ with inputs the state and input variables of the machine in an exact fashion. Notice that this calculation is of vital importance in our algorithm, that performs a search based on the probability of the activation function.

## 3 Machine transformation

In a Moore machine, the detection of idle conditions is straightforward. For each state $s$ we find all input conditions such that $\delta(x, s) = s$. If the next state is equal to the present state, the output cannot change either and therefore the machine is completely idle.

For a Mealy machine the problem complicates. Even if the next state and present state are the same the output may change, because a state may have incoming edges with different output fields. A self-loop on a state is an idle condition only if all incoming edges have the same output field. We call such a state a *Moore-state*; and states of a Mealy machine are not generally Moore-states. Apparently, this observation leads to the conclusion that, in order to detect the idle conditions on a Mealy-type FSM, one must observe input, output and state lines. Unfortunately, the number of outputs can be quite high and the complexity of the combinational logic for the detection of idle conditions (i.e. the activation function $f_a$) will consequently increase.

In order to overcome this difficulty we have devised a transformation that operates on the STG of a Mealy machine and produces a *locally-Moore machine*, an equivalent FSM with the following properties.

- The number of states in the transformed STG increases by a factor that is guaranteed to be less than or equal to two. This is an important property, because the simple Mealy to Moore transformation [18] may produce Moore FSMs that are much more complex than the corresponding Mealy FSMs (in terms of number of states and edges), and there is a strong correlation between the complexity of the STG and the power dissipation of the final implementation.
- The idle conditions that occur with maximum probability can be detected observing only the next state and output lines. In other words, all states with high probability self-loops are guaranteed to be Moore-type in the locally-Moore FSM.

A byproduct of the transformation is a *maximum probability activation function*, whose ON-set includes the input and state conditions corresponding to self-loops that occur with maximum probability. The support of the activation function includes only the state and input variables. The transformation of a Mealy FSM into a locally-Moore FSM has been thoroughly described in [3]. After the transformation we have a new STG of a machine that is compatible with the original Mealy FSM, with activation function:

$$f_a = \sum_{s_i \in S'} Self_{s_i} \cdot e_i \tag{7}$$

where $S' \subseteq S$ the set of Moore-states in a FSM, $e_i$ is the encoding state $s_i \in S'$ and $Self_{s_i}$ groups all input cubes corresponding to the maximum compatible set of self loops leaving state $s_i$.

**Example 2** *The transformation of 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 (state S0 has not been split).*

*The inputs are $in_1$ and $in_2$. Assume that we use three state variables for the encoding: $v_1$, $v_2$ and $v_3$. The state codes are LM0 $\rightarrow v_1' v_2' v_3'$, LM1a $\rightarrow v_1' v_2' v_3$, LM2a $\rightarrow v_1' v_2 v_3'$, LM1b $\rightarrow v_1' v_2 v_3$ and LM2b $\rightarrow v_1 v_2' v_3'$. The activation function includes all self-loops leaving Moore-states: $f_a = in_2 v_1' v_2' v_3' + in_1 in_2' v_1' v_2' v_3' + in_2 v_1' v_2 v_3 + in_1 in_2' v_1' v_2 v_3 + in_1 in_2' v_1 v_2' v_3'$.*

Once the activation function has been found, we still need to solve the problem of synthesizing the clock-stopping logic in an optimal way. This problem will be addressed in the next section and it is the major contribution of this paper.

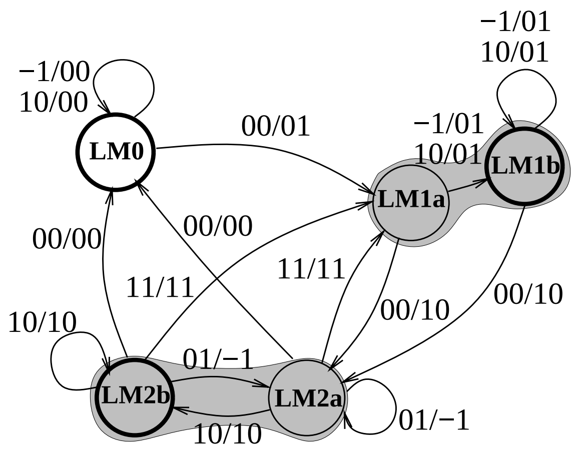

Figure 3: *STG of the locally-Moore FSM*

## 4 Optimal activation function

The simplest approach is to try to use the complete $f_a$ as activation function. This is seldom the best solution, because the size of the implementation of $f_a$ can be too large, and the corresponding power dissipation may reduce or nullify the power reduction that we obtain stopping the clock. Roughly speaking, it is 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, a minimum cover of $f_a$ is obtained with a two-level minimizer. Then, the largest cubes in the cover are greedily selected until the number of literals in the partial cover exceeds 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.

There are several weak points in this approach.

- There is no guarantee that choosing the largest cubes in a cover will maximize the probability of the cover, because the probability of a cube in general depends on the input and state probability distribution. Even if we assume uniform input probability distribution, the state probability distribution is in general not uniform.
- The subfunction sought may not be found by looking only at a subset of cubes of the minimum cover of the original activation function. The number of possible subfunctions of $f_a$ is much larger than the subfunctions that we can generate using subsets of the cubes of the minimum cover.
- Even if we restrict our attention to the list of cubes in the minimum cover of $f_a$ assuming uniform distribution for input and states, the cubes of the cover are in general overlapping (the minimum cover is not guaranteed to be disjoint). Finding the minimum-literal, maximum-probability subset of cubes becomes a set covering problem that certainly is not solved exactly by a greedy algorithm.
- The relation between the number of literals in a two-level cover of the activation function and the power dissipation of a multilevel implementation is not guaranteed to be monotonic.

In the next section, we will propose a new algorithm that overcomes the first three limitations listed above. As for the last issue, we will assume that there is correlation between the number of literals of a two-level cover and the power dissipated in the final implementation, as suggested by experimental results presented in [4]. We now formulate the problem that we want to solve in a more rigorous way.

**Problem 1** *Given the activation function $f_a$, find $F_a \subseteq f_a$ such that its probability $P(F_a)$ is $P(F_a) \geq MinProb = \alpha P(f_a)$, (with $0 \leq \alpha \leq 1$) and the number of literals in a two level implementation of $F_a$ is minimum.*

We call this problem *constrained-probability minimum literal-count covering* (CPML). Notice that we could as well formulate the dual problem, constrained literal count cover with maximum probability. The two problems can be solved using the same strategy, and are equivalent for our purposes. With the assumption of a good correlation between number of literals and power dissipation, we propose an exact solution to CPML and, by consequence to the problem of finding the best reduced activation function given a complete $f_a$ to start with.

### 4.1 Finding a minimum power implementation

The first source of difficulty comes from the fact that we are not constrained to completely cover $f_a$, therefore, the "algorithmic machinery" developed in the area of two-level minimization seems not useful. We first show that this is not true. Consider the set of primes of $f_a$, called $Primes(f_a)$. Consider the set $Sub_{f_a}$ of all possible subfunctions of $f_a$. The set of primes of a generic subfunction $F_a \in Sub_{f_a}$ is called $Primes(F_a)$. We state the following theorem.

**Theorem 1** *For every prime $p \in Primes(F_a)$ only two alternative are possible.*

- *$p \in Primes(f_a)$.*
- *$p$ is contained in at least one element $q$ of $Primes(f_a)$ (consequently its literal count is larger than the literal count of $q$).*

**Proof**. Assume that $p \in Primes(F_a)$, $(F_a \subseteq f_a)$. Two alternatives are possible: i) $p \in Primes(f_a)$, ii) $p \notin Primes(f_a)$. We will prove by contradiction that, if (ii) is true, there is always at least a prime $q \in Primes(f_a)$ such that $p \cdot q = p$ (in other words, $p$ is contained in at least a prime of $f_a$). Assume that the assert is not true, therefore, $p$ is not contained in any prime of $f_a$. Notice that $p$ is an implicant of $F_a$ therefore it is an implicant of $f_a$ because $F_a \subseteq f_a$. By consequence, $p$ is an implicant of $f_a$ not contained in any prime of $f_a$. Therefore $p$ is a prime of $f_a$ by definition. This is not possible, because we assumed (ii) to start with. $\square$

The important consequence of this proposition is that we do not need to generate all possible subfunctions of $f_a$. We can restrict our search to subfunctions that are formed by subsets of $Primes(f_a)$ if we want to find a minimum literal subfunction. Functions that belong to this class have all primes in the first category of Theorem 1.

Now that we have defined our search space ($Primes(f_a)$), we must find a search strategy that guarantees an optimum solution. The choice of a subset of $Primes(f_a)$ with minimum literal count satisfying the probability constraint cannot be done using a greedy strategy, because the primes are in general overlapping and a choice done in one step affects the following choices. An example will help to clarify this statement.

**Example 3** *Suppose that $f_a$ is a function of four variables: $a, b, c, d$. The set of primes is $Primes(f_a) = \{a'b', a'c', bc', ab\}$. Assume for simplicity that all minterms are equi-probable ($Prob = 1/16$) and our probability constraint $MinProb$ (the minimum allowed probability of the subfunction) is $MinProb = 1/2$. All primes in this example are equi-probable (they have the same size). If we choose $a'b'$ first, the following choices are biased. Since $a'c'$ is partially covered by $a'b'$, it will not be the right next choice because we want to cover the largest number of minterms (remember that we are assuming equi-probable minterms). Consequently either $bc'$ or $ab$ must be chosen.*

CPML complexity is at least the same as two-level logic minimization, because CPML becomes two-level logic minimization for the particular case $\alpha = 1$. We describe here a branch-and-bound algorithm that has been shown to work efficiently on the benchmarks, even if its worst case behavior is exponential. Furthermore, the branch-and-bound can be modified to provide heuristic minimal solutions when the exact minimum is not attainable in the allowable computation time.

## 4.2 Branch-and-bound solution

Our algorithm operates in two phases. In the first phase, an heuristic minimal solution is found in polynomial time (in the number of primes $N_P$). The second phase finds the global minimum cost solution using a branch-and-bound approach.

We exploit the similarity of this problem with *knapsack* [19]. We need to find the set of *items* (primes) with total *size* (probability) larger or equal to the *knapsack capacity* ($MinProb$) minimizing the total *value* (number of literals). This formulation differs from knapsack in two important details. First, knapsack targets the maximization of value given a constraint of the maximum allowed size (we face the opposite situation). Second, and most important, in knapsack the *size* of an item is a constant, while in our case the probability of a prime varies when other primes are selected. To clarify this statement, observe that primes may overlap and they contribute to the total probability of the reduced activation function only with minterms that are not covered by other already selected primes.

As a consequence, CPML is not solved in *pseudo-polynomial* time [19] by dynamic programming. Notice however that CPML reduces to knapsack if all primes are disjoint. In the first phase of our algorithm, we employ a greedy procedure that would produce a solution within a factor of two from the optimum in the case of disjoint primes [19]. Since this is not guaranteed in CPML, the solution is only a reasonable starting point for the second phase of the algorithm, that will provide an exact solution.

We define *value density* $\mathcal{D}$ for a prime $p$ the ratio $P_{ME}(p)/N_{lits}(p)$, and we greedily select the primes with maximum $\mathcal{D}$ until the constraint on $MinProb$ is satisfied. $P_{ME}$ is the total probability of minterms in $p$ which are not covered by already chosen primes.

When the first phase of the algorithm terminates, it returns a feasible solution that will be useful for increasing the efficiency of the branch-and-bound algorithm employed in the second phase. At the beginning of the second phase, we order the primes for decreasing ratio $P(p)/N_{lits}(p)$ (the probability $P(p)$ of a prime is computed multiplying the probability of the input part by the probability of the state part).

At the starting point of the branch-and-bound we have a *search list* corresponding to all primes (that generates $N_P$ branches in the decision tree). Moreover, we have a *current best* solution from the first phase. The *current partial* solution is initially empty. In order to explore all possible choices, we iteratively choose one prime following the order in the list, and we agument the current partial solution with the selected prime. We then call recursively the branch-and-bound procedure passing the list of all primes following the chosen one as search list and the primes chosen so far as current partial solution. The pseudo-code of the algorithm follows.

```
FindFa (PrimeList, CurBest, CurPartial, MinProb)
{
  if(Bound(PrimeList, CurBest, CurPartial, MinProb))
    return; /* Bounding step */
  DoneList = $\emptyset$;
  foreach (Prime $\in$ PrimeList) { /* Branching step */
    DoneList = DoneList $\cup$ Prime;
    NewPartial = CurPartial $\cup$ Prime;
    if ( $N_{lits}$(NewPartial) < $N_{lits}$(CurBest) ) {
      if ( Prob(NewPartial) $\geq$ MinProb )
        CurBest = NewPartial; /* New Best solution */
      else
        FindFa (PrimeList $-$ DoneList, CurBest,
          NewPartial, MinProb); /* Recursion */
    }
  }
}
```

Note that the backtracking involved in the branch and bound is implicitly obtained in the pseudo-code. For each iteration of the inner loop, we generate a new partial solution adding to the original partial solution a single prime from the search list. In this way, each new iteration backtracks on the choice of the prime in the previous iteration.

The bounding procedure (`Bound`) works on the search list. If the search list (`Primelist`) is empty, obviously the return value is 1. If the current partial solution (`CurPartial`) is empty, the return value is 0. In the general case, we select primes from the top of the search list until the sum of their literal count becomes larger than $N_{lits}$(`CurBest`) $-$ $N_{lits}$(`CurPartial`). We compute the sum of the probability of all selected primes (excluding the last selected one) $P_{tot}$. We choose the maximum $P_{max}$

between $P_{tot}$ and $P_{one}$, where $P_{one}$ is the largest probability value of a single prime in the search list whose literal count is less than $N_{lits}(\texttt{CurBest}) - N_{lits}(\texttt{CurPartial})$. If $P_{max} < (MinProb - \texttt{Prob(CurPartial)})/2$ we can discard the partial solution and prune the search tree.

The bound is based on the approximation algorithm for the solution of knapsack mentioned above. The greedy procedure guarantees a solution to knapsack within a factor of 2 from the optimum. The optimum knapsack solution itself is an upper bound to the solution of our problem (it becomes the exact solution if all primes are disjoint). Intuitively, the bound eliminates the partial solutions that could not improve the current best solution even if all primes in the search list were mutually disjoint and not overlapping with primes in the current partial solution.

**Example 4** *Assume that we have a current best solution that satisfies the constraint on the probability ($MinProb = .4$) with a cost of $Nlits = 40$. Assume that the current partial solution has cost $Nlits' = 36$ and probability* $\texttt{Prob(CurPartial)} = .35$. *Suppose that the first two element of the unselected prime list are $a'b'$ with probability .01 and $a'c'$ with probability .012. The maximum probability prime with at most 4 literals has probability .015. $P_{max}$ is therefore $P_{max} = max\{.015, .012 + .01\} = .022$. This branch of the search tree is pruned, because $P_{max} <$ $(MinProb -$ $\texttt{Prob(CurPartial)})/2 = .025$. Notice that the two primes are partially overlapping, therefore the actual increase in probability for the current solution if we select the two primes would be smaller than the estimated one.*

The bound can be made even tighter if after selecting a new prime in a partial solution, the probabilities of the remaining primes are reduced accordingly to the overlap with the chosen prime. Notice that the computation of this second bound requires the re-calculation of all probabilities of the currently unselected primes (and the reordering of the search list). By consequence the second bound should be computed only after the first has been unsuccessful in pruning the search tree.

Whenever the choice of a prime leads to a solution that satisfies $MinProb$ with a number of literals smaller that the total literal count of the current best solution, the current best solution is replaced with the new solution found. The algorithm terminates when all choices in the search list of the upper level of the recursion have been tried.

We want to point out that there are two possible sources of complexity explosion in our algorithm. First, the number of primes for a Boolean function is worst case exponential in the number of the function inputs. Second the branch-and-bound algorithm has a worst case exponential complexity in the number of primes that form the candidate list.

The double source of exponential behavior may seem worrisome. Nevertheless, the structure of our algorithm is flexible enough to generate fast heuristic solutions if the execution time exceeds some user-defined limit. The problem of the large number of primes can be avoided if we apply the algorithm to a reduced set of primes. The most natural candidate is obviously a prime and irredundant cover of the function, obtainable using two-level minimizers that can provide optimum or near-optimum covers for single output functions with a large number of inputs [9].

Moreover, if the time required by the branch-and-bound algorithm to find the exact solution becomes too large, we can interrupt the search when a user specified CPU-limit has been reached. The exact minimality of the last solution found is not guaranteed, but we will have in general a good approximation to a minimal solution. Notice that the first phase of our algorithm finds a feasible heuristic solution in polynomial time, and we could even completely skip the branch-and-bound if we consider it too expensive.

Finally, an efficient implementations of the algorithm can be achieved using symbolic BDD-based techniques. Many parts of the current implementation already use symbolic techniques (for example, the prime generation is fully symbolic [23], and the cube probability calculation is also done in a symbolic fashion), but still the prime list is manipulated by the branch and bound algorithm in a traditional way.

## 4.3 The overall procedure

We can now briefly outline the full procedure used for the synthesis of our low-power gated clock FSMs. Our starting point is a FSM specified with a transition table or a compatible format. The synthesis flow is the following.

- The Mealy machine is transformed to an equivalent locally-Moore machine (full details on this step are in [3]).
- 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 prime set $Primes(f_a)$ is generated.
- The branch-and-bound algorithm finds the minimum literal count solution $F_a$ whose probability is a prespecified fraction $\alpha$ of the probability of $f_a$
- $F_a$ is used as additional $DC$ set for optimizing the combinational logic of the FSM.

The last step can sensibly improve the quality of the results, in particular if $F_a$ is large [2]. Unfortunately, it is 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 $F_a$ that range from the complete $f_a$ to a much smaller subfunction, in an attempt to explore the trade-off curve.

This iterative search strategy raises the problems of choosing appropriate values of the parameter $\alpha \leq 1$ used to scale down the probability of $f_a$ when the reduced activation functions are generated. The approach that we adopted is to generate a set of solutions using different values of $\alpha$, in such a way that the possible range of solutions is uniformly sampled.

We have devised a heuristic algorithm that generates suitable $\alpha$ values and we briefly outline it. The rationale behind our approach is to split the range of probability available for $F_a$ in $N_{cand}$ equal intervals and to generate a set $Cand_F$ generated using the following values of $\alpha$:

$$\alpha_i = i/N_{cand} \quad i = 1, 2, ..., N_{cand} \tag{8}$$

If empty or duplicated solutions are generated during this process, our algorithm adaptively select new values of $\alpha$ such that the new candidates will have a probability between those of solutions generated with two consecutive values of $\alpha$ that have maximum literal cost difference.

Obviously, if large $N_{cand}$ are used, the computational time required to generated $Cand_F$ increases. Notice however that only the last two steps in the overall procedure describe before need to be iterated, and usually a small number of different values of $\alpha$ is sufficient to find a satisfying solution.

## 5 Implementation and experimental results

We implemented the 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 (we assumed uniform input probability distribution, but this assumption is not restrictive). Notice that this step can be modified to use the exact and approximate methods described in [16, 25, 24] that have been demonstrated to run on very large sequential circuits. Presently, our procedure employs sparse matrix techniques and it has been able to process all MCNC benchmarks provided in state transition table format in a small time (less than 1sec for the largest example s298).

We then state assign both the original machine and the locally-Moore FSM using JEDI [26]. Once the state codes have been assigned, our probabilistic-driven procedure for the selection of the activation function can start. First, all primes of the activation function are generated using symbolic methods [23], then the probability of the minimized cover (obtained with ESPRESSO [20]) of the complete activation function $f_a$ is computed. The number of literals of the complete minimized cover is used as initial literal cost limit in the branch-and-bound algorithm.

The user specifies the number of activation functions that the procedure should generate, and the branch-and-bound algorithms solves the CPML as many times as it is requested. Surprisingly, for all MCNC benchmarks this step has never been the bottleneck, the CPU time being in the order of 30 seconds maximum. 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). Nevertheless, even if difficult cases are found, our algorithm stops the search when a user specified CPU time limit has been reached. The solution becomes then suboptimal, but there are other sources of inexactness in the overall procedure. Therefore the search for an exact optimum solution of CPML is not of primary practical importance.

The combinational logic of the locally-Moore FSM is then optimized in SIS [20] using the additional $DC$ set given by the activation function. This step is repeated for all activation functions generated in the preceding step, and alternative solution are generated. 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. More work has to be done in this area, in order to be able to rapidly estimate if there is available space for improvements or if the additional $DC$ set given by the activation function is too small to be useful, and the $DC$-based optimization step can be skipped.

The activation functions are also optimized using SIS, then the alternative solutions are mapped with CERES [21], and the gated clocking circuitry is generated. Again the same optimization and library binding programs are used for both the original Mealy machine and the locally-Moore gated clock machines. 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 [27]) 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 to 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 limited or null. This is the case for many MCNC benchmarks for which the final improvement is negligible. As for the first problem, it may be worth 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.

**Example 5** *The Mealy machine of Example 1 has been synthesized without any gated clock. The number of states is 3, the mapped implementation has 124 transistor and a total nodal capacitance of 2.32 pF. The average power dissipation is 52 $\mu W$.*

*Using our algorithm, the minimum power implementation (obtained with the complete activation function in this case) of the equivalent locally-Moore gated clock machine has 178 transistors and a total nodal capacitance of 3.14 pF. The average power dissipation is 42 $\mu W$. Notice that the efficacy of the activation function in stopping the clock allows substantial power savings (24%) even if the total capacitance is larger (35%). This is due to the fact that the locally-Moore machine has 5 states, and its combinational logic is more complex. In contrast, with a complete Moore transformation the minimum power implementation has 196 transistors and total nodal capacitance of 3.39 pF. Its power dissipation is 48 $\mu W$.*

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 two columns show the power improvement (computed as $100(P_{mealy}/P_{gated}-1)$) and the $\alpha$ factor used in the solution of CPML leading to the best result. Notice

| Cir. | Original | | Locally-M. | | Gated | | % | $\alpha$ |
|---|---|---|---|---|---|---|---|---|
| | Size | P | Size | P | Size | P | | |
| bbara | 330 | 67 | 422 | 72 | 408 | 34 | 97 | 1 |
| bbsse | 640 | 121 | 742 | 137 | 736 | 119 | 2 | 1 |
| bbtas | 142 | 56 | 138 | 57 | 164 | 44 | 27 | .93 |
| keyb | 721 | 128 | 754 | 132 | 820 | 114 | 12 | .91 |
| lion9 | 188 | 60 | 226 | 60 | 248 | 52 | 15 | .25 |
| s298 | 7492 | 899 | 7496 | 900 | 7502 | 810 | 11 | 1 |
| s420 | 544 | 132 | 544 | 132 | 602 | 108 | 22 | .75 |
| scf | 3222 | 437 | 3222 | 437 | 3169 | 400 | 9 | 1 |
| styr | 1474 | 159 | 2468 | 230 | 2534 | 208 | 0 | .75 |
| test | 348 | 73 | 442 | 76 | 374 | 32 | 128 | .88 |

Table 1: Results of our procedure applied to MCNC benchmarks. Size is number of transistors. P (power) is in $\mu$W.

that, if there is no power improvement the improvement is set to 0.

The tool is able to process all 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. For this class of FSMs our tool gives its best results.

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 low probability. Moreover, the transformation to locally-Moore machine has a too large area overhead in this case, therefore, even if there are power savings with respect to the locally-Moore implementation without clock, the smaller Mealy implementation has the lowest power consumption.

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

From the analysis 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. To address this problem, a procedure that splits only on the states with high probability self-loops is under development in the final version of our tool. 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. Notice however that for some examples the efficiency of the activation function in stopping the clock was such that the power was sensibly reduced even with large area overhead.

## 6 Conclusions and future work

We have described a technique for the automatic synthesis of gated clocks for Mealy and Moore FSMs. 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 important power reductions for a large class of finite state machines, where the probability of being in a self-loop (idle) is high.

We have discussed a new transformation for Mealy FSMs that makes them suitable for gated-clock implementation, we have proposed a new logic optimization problem, called "constrained probability minimum literals" problem, and we have described its exact and heuristic solutions.

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.

## 7 Acknowledgements

This research is supported by NSF under contract MIP-9421129.

## References

[1] M. Alidina, J. Monteiro, et al., "Precomputation-based sequential logic optimization for low power," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 4, pp. 426–436, Jan. 1995.

[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] L. Benini and G. De Micheli, "Transformation and synthesis of FSMs for low power gated clock implementation " *International Symposium on Low Power Design*, pp. 21–26, April 1995.

[4] P. E. Landman and J. M. Rabaey, "Activity-sensitive architectural power analysis for the control path," *International Symposium on Low Power Design*, pp. 93–98, April 1995.

[5] 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.

[6] 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, 1993.

[7] 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.

[8] 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.

[9] P.McGeer, J. Sanghavi, et al., "ESPRESSO-SIGNATURE: a new exact minimizer for logic functions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 4, pp. 432–440, Dec. 1993.

[10] N. Yeung, et al., "The design of a 55SPECin92 RISC processor under 2W," in *IEEE International Solid-State Circuits Conference*, pp. 206–207, Feb. 1994.

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

[12] D. Pham, et al., "A 3.0W 75SPECint92 85SPECfp92 superscalar RISC microprocessor," in *IEEE International Solid-State Circuits Conference*, pp. 212–213, Feb. 1994.

[13] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design (Second Edition).* Addison-Wesley, 1992.

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

[15] 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.*, pages 214 – 218, February 1994.

[16] G. Hachtel, E. Macii, A. Pardo and F. Somenzi, "Probabilistic analysis of large finite state machines," in *DAC, Proceedings of the Design Automation Conference*, pp. 270–275, June 1994.

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

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

[19] M. R. Garey and D. S. Johnson, *Computers and intractability. A guide to the Theory of NP-completeness.* Freeman, 1983.

[20] 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.

[21] 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.

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

[23] 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.

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

[25] J. Monteiro, S. Devadas and B. Lin, "A methodology for efficient estimation of switching activitiy in sequential logic circuits," in *DAC, Proceedings of the Design Automation Conference*, pp. 315–321, June 1994

[26] 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*, pages 187 – 196, August 1989.

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