FSM Decomposition by Direct Circuit Manipulation Applied to Low Power Design

José C. Monteiro
IST-INESC
Lisbon, Portugal
Tel: +351-1-3100283
Fax: +351-1-3145843
jcm@inesc.pt

Arlindo L. Oliveira
Cadence European Labs / IST-INESC
Lisbon, Portugal
Tel: +351-1-3100228
Fax: +351-1-3145843
aml@inesc.pt

Abstract— Clock-gating techniques are very effective in the reduction of the switching activity in sequential logic circuits. In particular, recent work has shown that significant power reductions are possible with techniques based on finite state machine (FSM) decomposition. A serious limitation of previously proposed techniques is that they require the state transition graph (STG) of the FSM to be given or extracted from the circuit. Since the size of the STG can be exponential on the number of registers in the circuit, explicit techniques can only be applied to relatively small sequential circuits. In this paper, we present a new approach to perform FSM decomposition by direct manipulation of the circuit. This way, we do not require the STG, either explicit or implicit, thus further avoiding the limitations imposed by the use of BDDs. Therefore, this technique can be applied to circuits with very large STGs. We provide a set of experimental results that show that power consumption can be substantially reduced, in some cases by more than 70%.

I. Introduction

Power consumption has become a major design parameter in the project of integrated circuits. Two independent factors have contributed for this. On one hand, low power consumption is essential to achieve longer autonomy for portable devices. On the other hand, increasingly higher circuit density and higher clock frequencies are creating heat dissipation problems, which in turn raise reliability concerns and lead to more expensive packaging.

In static CMOS circuits, the probabilistic average switching activity of the circuit is a good measure of the average power dissipation of the circuit. Methods that can efficiently compute the average switching activity, and thus power dissipation, in CMOS combinational [11] and sequential [16] circuits have been developed.

Techniques based on disabling the input/state registers when some input conditions are met have been proposed and shown to be among the most effective in reducing the overall switching activity in sequential circuits [1, 2, 14]. This class of techniques is sometimes referred to as *logic level* or *dynamic power management*.

One method that falls into this class of techniques is clock-gated finite state machines (FSM) decomposition. This method is based on the computation of two sub-FSMs that together have the same functionality as the original FSM. For all the transitions within one sub-FSM, the clock for the other sub-FSM is disabled. To minimize the average switching activity, a small cluster of states with high stationary state probability is selected to be in the small sub-FSM. So far, this methodology has been limited to the cases where the state transition graph (STG) can be extracted and explicit decomposition performed. Since the size of the STG can be exponential on the number of registers in the circuit, explicit techniques can only be applied to relatively small sequential circuits.

In this paper, we propose a new approach for the decomposition of FSMs based on direct circuit manipulation. We perform the decomposition solely by manipulating the original circuit, creating the desired sub-machines and their interconnections without ever needing an explicit or implicit representation of the STG. Although there exist BDD-based techniques that can represent very large STGs implicitly [15], they are still limited by the size of the required BDD. Our approach does not have this limitation.

Three main issues had to be addressed to enable the approach described in this paper:

- how to select the states for each sub-machine.
- given these states, how to create each sub-machine.
- finally, given these sub-machines, how to set a sub-machine to the correct state when a transition coming from a state in the other sub-machine is exercised.

We targeted this decomposition approach to the reduction of power dissipation in sequential logic circuits. We applied the technique to the ISCAS benchmark set and were able to run it on all the examples in a reasonable amount of time even for the largest example. We present results that show that power savings close to 80% are possible in some cases.

In Section II, we introduce some basic definitions used throughout the paper. An overview of the problem of FSM decomposition if given in Section III. Section IV presents related work on logic level power management, focusing on techniques that use FSM decomposition. We describe our implicit method for decomposition in Section V. Finally, Sections VI and VII describe the experimental results and the conclusions.

II. Basic Definitions

We use the standard definition of finite state machines:

Definition II.1 *A finite state machine is a tuple* $M = (\Sigma, \Delta, Q, q_0, \delta, \lambda)$ *where* $\Sigma \neq \emptyset$ *is a finite set of input symbols,* $\Delta \neq \emptyset$ *is a finite set of output symbols,* $Q \neq \emptyset$ *is a finite set of states,* $q_0 \in Q$ *is the initial "reset" state,* $\delta(q,a) : Q \times \Sigma \rightarrow Q$ *is the transition function, and* $\lambda(q,a) : Q \times \Sigma \rightarrow \Delta$ *is the output function.*

The specification of the finite state machine is equivalent to the specification of a state transition graph (STG), where each state corresponds to one node in the graph, and there exists an edge e_{ij} between two states q_i and q_j with label a/b iff $\delta(q_i, a) = q_j$ and $\lambda(q,a) = b$.

Under specific input line probabilities, it is possible to compute the stationary state and transition probabilities. The stationary state probability, $P(q_i)$, is the probability of the finite state machine being in state q_i in a given clock cycle. In this work, we will always use the absolute transition probabilities, indicated as $P(e_{ij})$. $P(e_{ij})$ represents the probability that, at any specific time, transition e_{ij} is active. $P(e_{ij})$ can be computed using:

$$P(e_{ij}) \;=\; P(q_i) \times P(a) \tag{1}$$

where $P(a)$ represents the probability of the primary inputs combinations that trigger the transition e_{ij}.

For each state q_i we can write an equation:

$$P(q_i) \;=\; \sum_{k=1}^{|Q|} P(e_{ki}) \tag{2}$$

We obtain $|Q|$ equations out of which any one equation can be derived from the remaining $|Q|-1$ equations. Since at any specific time the machine is in one and only one state, we have a final equation:

$$\sum_{k=1}^{|Q|} P(q_i) \;=\; 1 \tag{3}$$

This linear set of $|Q|$ equations can be solved to obtain the different $P(q_i)$'s. This system of equations is known as the Chapman-Kolmogorov equations for a discrete-time discrete-transition Markov process. Indeed, if the Markov process satisfies the conditions that it has a finite number of states, its essential states form a single-chain and it contains no periodic-states, then the above system of equations will have a unique solution [12].

III. Decomposition Of Finite State Machines

The decomposition of finite state machines has been addressed by a number of authors, the first work dating back to 1960 by Hartmanis [8]. Several decomposition strategies have been proposed and are described in detail elsewhere [6]. Cascade and parallel decomposition, shown in Figure 1 (a) and (b), respectively, are simple, but of limited use with finite state machines that do not have a very regular structure. On the other hand, the more general form of finite state machine decomposition, shown in Figure 1 (c), is applicable to any finite state machine. The structure of the resulting finite state machine is shown in Figure 2. Each of the sub-machines receives, as inputs, not only the primary inputs and its own state variables, but also the state variables of the other sub-machine. Conceptually, the STGs of each of the two sub-machines are created in the following way:

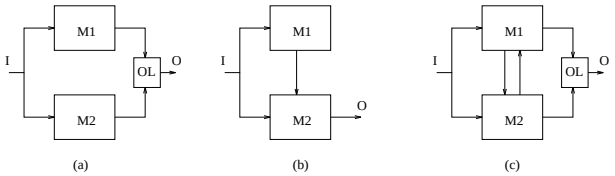

Fig. 1. Cascade, parallel and general approaches to finite state machine decomposition.

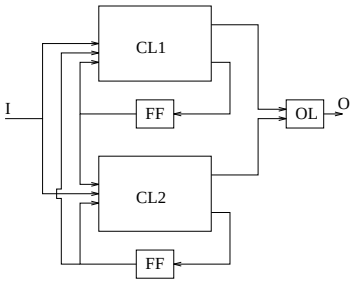

Fig. 2. Structure of decomposed finite state machine.

- Select a subset of the states in the original STG to belong to the first sub-machine, and let the remaining states belong to the second machine.
- Generate two STGs, one for each sub-machine. Create a new state, the RESET state, in each of the two new STGs.
- All transitions entirely inside each of these STGs are copied unmodified from the original STG.
- Transitions between a state in the first sub-STG and a state in the second sub-STG are replaced by two transitions: one to the RESET state in the first sub-machine, and one from the RESET state in the second sub-machine. The reverse is true for the symmetric case.

To illustrate this procedure, consider the state transition graph for a simple sequence detector depicted in Figure 3. Assume that a decomposition of the corresponding FSM is desired, with states A and B to belong to machine M2, and all the other states to belong to machine M1. The original STG is transformed into two smaller STGs. Transitions between the states in M1, as well as transitions between the states in M2, are copied without transformation. As shown in Figure 2, the number of inputs to each sub-FSM is enlarged with the value of the state lines of the other sub-FSM, each FSM also providing as outputs the value of the state lines.

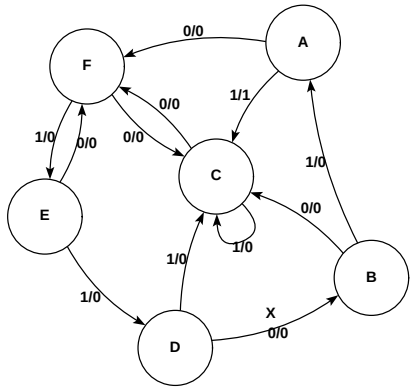

Fig. 3. State transition graph of the sequence detector.

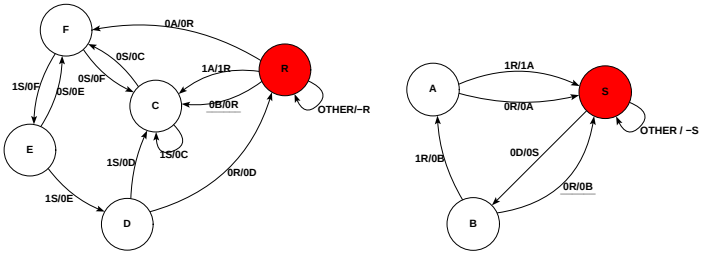

Fig. 4. State transition graphs for the sequence detector after decomposition.

Transitions between a state in M1 and a state in M2 require a slightly more complex treatment. Consider, for example, the transition from state B to state C in the original FSM. This transition corresponds to two new transitions, one in each of the STG of the sub-FSMs. These transitions are underlined in Figure 4. When input 0 is received in state B, machine M2 goes to its reset state S and machine M1 goes out of its reset state, R. Incidentally, this corresponds to one situation where both machine are active. For transitions between states in the same sub-machine, only that sub-machine needs to be active.

All input combinations not explicitly shown in the edges leaving the reset state of each machines are self-loops, and they signal the fact that the given machine is to stay inactive until it receives the right input combination. This is represented by the transition with input label *other*.

The procedure shown here can be implemented directly if the STG for the original machine can be extracted in a reasonable amount of time. In practice, care must be taken with the state encoding procedure, since there must exist a correspondence between the outputs of each machine and the state encoding used in the other one [5]. Apart from that, the procedure to obtain the decomposed machine is straightforward, and can be accomplished using standard logic synthesis tools.

We are, however, concerned, with a methodology to perform this process without actually generating the STG for either the original or the decomposed machines. This methodology is described in the next section.

IV. FSM Decomposition Targeting Low Power

So called power management techniques that shutdown blocks of hardware for periods of time in which they are not producing useful data are effective methods to reduce the power consumption of a circuit. Shutdown can be accomplished by either turning off the power supply or by disabling the clock signal. A system-level approach is to identify idle periods for entire modules and turn off the clock lines for these modules for the duration of the idle periods ([4], Chapter 10).

Power management techniques have also been proposed at the logic level [1, 2, 14]. One key difference is that the disabling of the input/state registers is decided on a clock-cycle basis and can be done either by using a register load-enable signal or by gating the clock. A common feature in these methods is the addition of extra circuitry that is able to identify input conditions for which some or all of the input/state registers can be disabled. In this situation there will be zero switching activity in the logic driven by input signals coming from the disabled registers. The objective of these techniques is to minimize the size of the extra circuitry, which translates to area and power overhead, at the same time maximizing the fraction of time that this circuitry is disabling registers.

Decomposition of finite state machines targeted for low power has been recently proposed [5, 10] as an alternative and/or addition to the techniques described above. The basic idea is to decompose the original finite state machine into two machines. Except for transitions that involve going from one state in one sub-machine to a state in the other, only one of the sub-machines needs to be clocked.

There is a small difference between the architectures used in each of these approaches, as shown in Figure 5.

The basic difference is that the registers in [5] are shared between both sub-machines whereas [10] follows closely the standard general decomposition topology (Figure1(c)), but uses additional signals that permit one sub-machine to disable the registers of the other when the next transition will be between states in that sub-machine.

The assignment of states to each sub-machine is such that transitions within sub-machines are maximized and transitions between sub-machines minimized, corresponding to the cost function:

$$F = \sum_{q_i,q_j \in Q_1} P(e_{ij}) - \beta\Big(\sum_{q_i \in Q_1, q_j \in Q_2} P(e_{ij}) + \sum_{q_j \in Q_1, q_i \in Q_2} P(e_{ij}) \Big) \quad (4)$$

where $P(e_{ij})$ are the static edge probabilities computed as explained in Section II and β a parameter that the user can use to experiment with different weights for the two terms. The best value for β depends from circuit to circuit, yet the authors have found empirically that a value between 0.5 and 1.0 works well for most circuits.

The technique of [10] attempts to maximize the power savings by searching first for a small number of states for one of the sub-machine, the *small sub-machine*. If a small a cluster of states with high stationary probability is found, then there will be a small amount of logic active (the *small sub-machine*) that

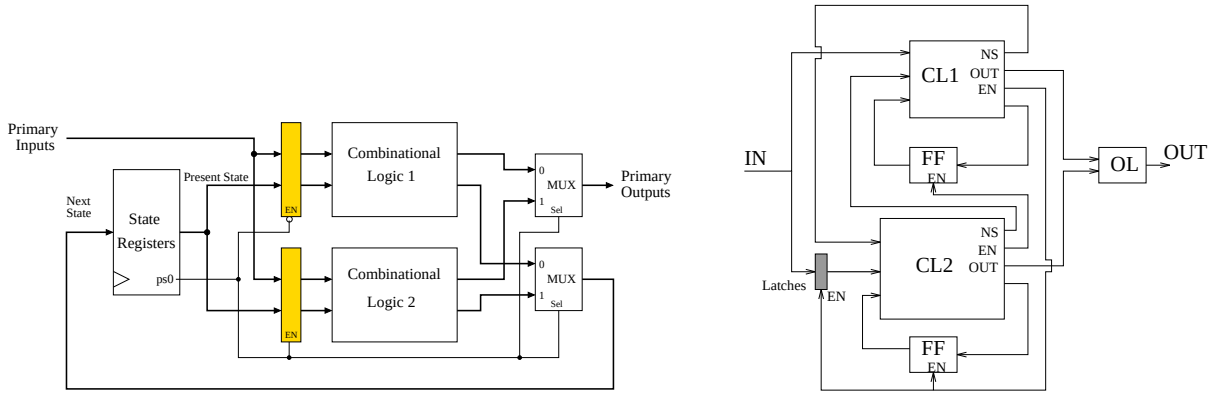

Fig. 5. Architectures used in explicit FSM decomposition: (a) [5]; (b) [10].

most of the time will be disabling all the state registers in the larger sub-machine.

The overhead associated with the FSM decomposition makes this technique not very effective for typical FSMs with a small number of states. However, for larger machines, very large power savings of over 80% have been reported. Given this fact and the severe limitation that these explicit techniques have from being based on information extracted from the STG of the FSM, have motivated this work on implicit decomposition.

A different technique that can be regarded as performing a FSM decomposition and that does not require an explicit representation of the STG was proposed in [3], thus making it close to the work we present in this paper. However, the partitioning is not based on state selection, but on the selection of nodes in the combinational logic of the original circuit.

There are two major limitations of the technique of [3]:

1. requiring BDDs for the operators during node selection limits the size of the circuits that can be handled.
2. since the analysis is done at node and not state level, the circuit may be constantly switching between both *sub-machines*, thus increasing power dissipation.

These observations are confirmed by the results presented in the paper. On the other hand, being a different approach than the one we are presenting, there will be circuits for which it is effective and ours is not and vice-versa.

V. FSM Decomposition Using Direct Circuit Manipulation

Given the limitations exhibited by the explicit algorithms for FSM factorization described in the previous section, and their potential when applied to the reduction of power dissipation, it is of fundamental importance to find ways to perform a similar decomposition without ever actually enumerating the state transition graph.

Given the specific way in which the original FSM is decomposed, an implicit method can not be obtained by a trivial transformation from the explicit approach described previously.

One possible approach is that of computing the transition relation in implicit form using BDDs as the underlying data structure. This approach, however, requires the ability to build the state transition relation, a procedure that is, in practice, limited to a small fraction of the total number of interesting circuits. Even if the state transition relation can be computed, it is not effective in this case, because there is no straightforward way to perform the following required operations:

1. Computation of the steady state transition probabilities, by solving Equations 2 and 3.
2. Selection of a state partition.
3. Actual decomposition of the STG of the original FSM.
4. Re-encoding and re-synthesis of the sequential circuit.

The main contribution of this work is exactly a methodology to perform the three major computations described above, by direct manipulation of the original network, avoiding both an explicit extraction of the state transition graph or the computation of the transition relation.

The following sub-sections describe how each of the four computations described above can be either approximated or performed directly on the network itself.

A. Approximate Computation Of The State Transition Probabilities

For machines with small state transition graphs, it is possible to obtain the stationary transition probabilities using a variety of methods. However, if the state transition graph is too large to be extracted, it is not possible to compute these probabilities. We are saved by the fact that, for our approach, we are only interested in the transition probabilities that add up to a significant amount. In fact, one would like to select a set of

states such that the sum of the transition probabilities between them is high enough to justify the overhead incurred by our factorization procedure.

It is therefore possible to use Monte Carlo simulation to compute approximate state transition probabilities, and to use this approximation in the partition algorithm. If there are clusters of states with very high stationary transition probabilities, this type of simulation is very likely to identify them. The procedure may fail if the diameter of the state transition graph is very high, but will, in general, give very approximate results for the cases where a cluster of interest exists.

In the case where no compact group of states exhibits very high stationary transition probabilities, the results may be a very poor approximation to the real distribution. However, in this case, the basic idea underlying the decomposition approach is unlikely to yield good results, and therefore the failure of this computation does not pose any additional restriction.

B. Selection of a State Partition

The arguments used in [10] for state selection still apply. The potential for the gains in power dissipation obtainable from the decomposition techniques described in Section IV are larger if one selects a partition that exhibits the following characteristics:

1. One of the machines (the *small* machine) has a state transition diagram with a small number of states, and is therefore simple and dissipates a small amount of power.

2. The sum of the transition probabilities between two states in the *small* machine other than the RESET state is as large as possible.

3. The sum of the transition probabilities involving the RESET state in the *small* machine is as small as possible.

We also use Kernighan-Lin algorithm [9] to perform the state selection with the same cost function (Equation 4). However, instead of exact transition probabilities, the approximate probabilities obtained by simulation are used.

If N_1 is the number of states to select for the small submachine, and N the total number of states that where visited during the Monte Carlo simulation, then the Kernighan-Lin procedure has complexity $O(N_1^3 N_2)$, where $N_2 = N - N_1$. However, N_1 is always a small number, because it is the number of states in the *small* machine. Typically, N_1 is never superior to 15, and N_1^3 is therefore bounded from above by a constant, leading to a total complexity that is linear in N, the total number of states that were visited. This number is always bounded by the choice made for number of vectors used in this simulation. States with very low stationary probabilities may never get visited, and are not considered by the Kerninghan-Lin procedure. This makes the method applicable to very large machines, the total number of states having no influence on the complexity of the partition procedure.

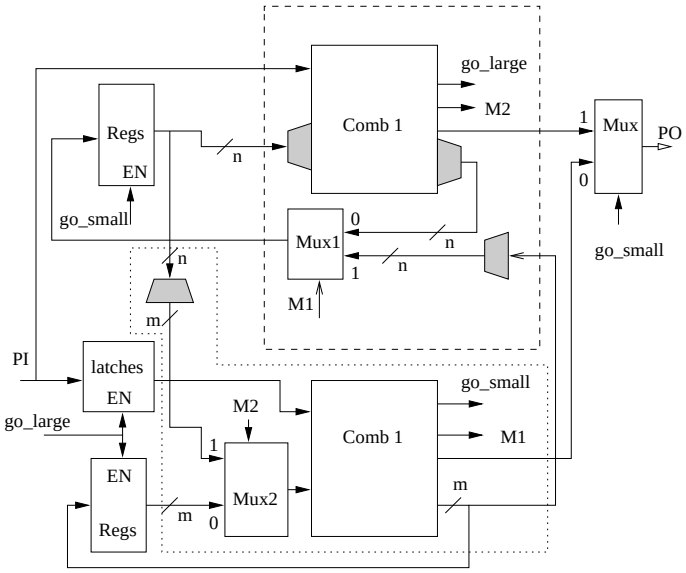

Fig. 6. Implicit FSM decomposition. The block *COMB1* represents the combinational logic the original FSM, while the shaded blocks represent transcoders that translate between the encodings used in each of the factor machines.

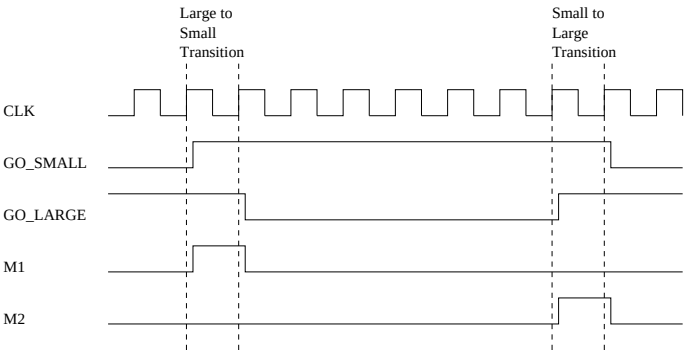

Fig. 7. Waveforms for the four control signals used in the decomposition procedure.

C. Decomposition By Direct Circuit Manipulation

We decompose an FSM into two communicating FSMs, as shown in Figure 6. We will refer to each of the sub-machines as the *large* and the *small* machine. The large contains all the complex functionality of the original machine, while the small will only implement the functionality that corresponds to a small set of states. Assume that a set of states $S = \{s_1, s_2 \ldots s_k\}$, and that the large machine has *m* state lines while the small machine has *n* state lines.

The circuit is controlled by 4 control signals. The waveforms of these control signals are shown in Figure 7.

Signals *go_large* and *go_small* are the enables of the small and large machines, respectively. Signal *M2* is active only during a transition that is transferring control from the small to the large machine. Signal M1 performs the same role on the symmetric condition. To avoid combinational loops, these signals

are generated in a slightly different way from each other.

The control signal *go_large* is used to disable both the registers in large machine and a set of latches that keep transitions from the primary inputs from propagating into the combinational logic.

Signal *go_small* disables only the registers in the small machine. Disabling the primary inputs is possible, but uninteresting since this machine will be working most of the time, at least when a good partition of the states can be found.

The shaded blocks in Figure 6 represent transcoders that convert between the encodings used in each of the machine. Note that these blocks are relatively simple, since S contains only a small number of states. By composing a *small-to-large* and a *large-to-small* transcoder with the original combinational block, it is possible to use the original logic block *Comb1* to generate both the outputs and the next state logic of the small machine, as shown on the top of Figure 6.

To illustrate the behavior of the circuit, consider the STGs in Figure 4 and the transitions underlined there. In this situation, *M2* is 1, indicating that the small machine is entering its reset state and that the large machine should leave it. Since *M2* is 1, the lower-most combinational logic block *Comb 1* sees at its inputs the encoding for state *B* and outputs the right next state value, *C*. The dotted line shows the new combinational block for the large machine. The inputs for this block are the original primary inputs enlarged with n lines, the state codes from the other machine.

D. Re-encoding and Re-synthesis

The decomposition procedure described above actually solves directly the last problem raised in the beginning of this section, that of re-encoding and re-synthesis of the decomposed machine. However, the decomposition performed as shown above would be uninteresting if the combinational logic of the small machine could not be reduced. Under those conditions, no power saving could take place since the small machine will dissipate at least as much power as the large machine.

Yet, note that the small machine has a much smaller number of states than the original one. Consider the logic block in the top of Figure 6, constituted by a *small-to-large* transcoder, block *COMB1* and a *large-to-small* transcoder. If k states are selected for the small machine, only $k+1$ state combinations are possible at its input. Using the degrees of freedom permitted by the use of controllability don't cares, it is usually possible to reduce this circuit enormously, and therefore obtain an implementation to the small machine that is compact and power efficient.

Although in principle the same could be done for the large machine, the expected gains are much smaller, since only a reduced number of possible input combinations are removed from the large machine. We therefore choose not to apply this optimization procedure to the large FSM.

Once the states to be included in the small machine are chosen, a state encoding has to be chosen for this machine. The estimated transition probabilities between the states in the small machine are known, but using this information is non-trivial, since the two objectives of low switching activity and compact circuit size are not necessarily correlated.

We are currently using a simple heuristic that tries to minimize the number of bits that commute for transitions with very high probability, and are actively working on improving the state assignment procedure.

TABLE I
STATISTICS OF THE CIRCUITS.

Circuit	PI	PO	Regs	Lits	Power
`s386`	7	7	6	191	606
`s499`	1	22	22	329	994
`s635`	2	1	32	335	1134
`s820`	18	19	5	433	958
`s832`	18	19	5	453	980
`s953`	16	23	29	620	1969
`s967`	16	23	29	622	1906
`s1488`	8	19	6	943	1567
`s1494`	8	19	6	951	1569
`s35932`	5	320	1728	15383	84712

VI. EXPERIMENTAL RESULTS

We have applied the implicit FSM decomposition technique proposed in the previous sections on circuits from the ISCAS89 benchmark set. We present a set of results on the power savings we have obtained. All the results were obtained with SIS [13] on a Sun Ultra I, running at 170MHz, with 384Mb of main memory.

Table I shows some statistics on the circuits we present results for. All circuits where initially optimized with `script.algebraic` and mapped to the `msu` library. The name, number of inputs, number of outputs and number of registers for each of the circuits used is given in the first four columns. We also give the number of literals in column five, which is a good measure of circuit area. The power dissipation (in μW) of the original circuit is shown in the last column, assuming a supply voltage of 5V, a clock frequency of 20MHz and a zero-delay model.

Table II presents the results obtained. To collect the state transition statistics, we run in parallel with 31 (fitting in a long integer) simulations with a sequence of 1,000 input vectors. We used randomly generated input vectors simply because we did not have information about these circuit. User-specified inputs could have been used as easily. The length of the simulation run is actually conservative. As we expected, if a good cluster of states exists, a shorter simulation (e.g., 100 vectors) will be able to identify it. Yet, the simulation process is very fast finishing the 1,000-vectors run in less than 2 minutes even for the largest circuit in the benchmark set. On the other hand, we have experimented with longer runs and found no improve-

TABLE II
RESULTS OBTAINED AFTER DECOMPOSITION.

Circuit Name	Orig. Power	States Visited	States Small	Cost Function	Lits		Power		CPU time
					Opt.	%	Opt.	%	
s386	606	10	4	21606	314	39	511	16	4s
s499	994	22	6	5403	634	48	688	31	8s
s635	1134	9	4	28641	545	39	236	79	7s
s820	958	9	4	28276	678	36	435	55	7s
s832	980	9	4	28276	699	35	441	55	7s
s953	1969	441	16	14750	1272	51	1672	15	131s
s967	1906	482	8	12721	1094	43	1540	19	22s
s1488	1567	25	8	30392	1332	29	561	64	12s
s1494	1569	25	8	30392	1335	29	562	64	12s
s35932	84712	6359	6	11986	25391	39	56453	33	3.4h

ment on the best cluster found for any of the circuits.

Under column *States Visited* in Table II we give the states that were visited during the simulation of the circuit. We have observed that this number is a good indicator of the existence of a good decomposition for low power. If this number is low, then there is a high probability that the FSM stays in a small number of states most of the time. We just need to identify these states and extract them to the small sub-machine. If this number is close to the total number of simulated vectors, then probably no such cluster exists. The extreme case is when for each input vector a different state is visited, meaning that the decomposition technique we propose in this paper can never be effective. This was the case for circuit s3271, s5378, s6669, s13207, s15850 and s38417 of the ISCAS benchmark set, where 31,000 states where visited in the simulation process (some other circuits in this set were also close to 31,000). However, note that for the case of circuit s35932 the number of states visited is high and significant power savings are still possible.

Since the algorithm for state selection described in Section B is very efficient, we run the algorithm for different values of the number of states in the small sub-FSM. In Table II we present the best result found where the number of states used for the small sub-machine is given under column *States Small*. The cost function given in Equation (4) was used with $\beta = 0.7$, which we have found empirically to give good results. Column *Cost Function* lists the value of this function. Intuitively, this value represents the number of transitions inside the small machine, less the number of transitions between the two sub-machines multiplied by β. Clearly, the maximum value for this function is the total number of input vectors simulated (in this case, 31,000), and, in many cases, values close to that limit are obtainable, even with a small number of states.

In the remaining columns of Table II we give the number of literals (i.e., area) and percentage increase and power dissipation and percentage savings of the circuit after decomposition. As we can observe from the table, significant power savings can be obtained for many circuits, and can be close to 80%. Still, the technique is not effective for other circuits in the benchmark set. Our claim is not that the implicit decomposition approach we are proposing is applicable to all FSMs, but that large power savings can be achieved for many circuits. We do incur in some area penalty which on average is around 40%, to which we need to add the extra registers used in the small sub-machine.

Though not shown in the results table, there is a small delay increase. The maximum delay of the decomposed circuit is determined by the large sub-machine. Recall that we do not attempt any optimization for this sub-machine, so it is still just like the original circuit except that we have introduced a block of multiplexors to set the correct state when there a transition from a state in the small to a state in the large sub-machine (block *Mux2* in Figure 6). If the designer is not willing to allow even this small delay increase, we need to re-synthesize the large sub-machine.

Finally, note that we were able to run the technique on all the circuits in the ISCAS benchmark set (even if a good decomposition for low power was not found) in a reasonable amount of CPU time. The run time for the smaller circuits is very small, most run in just a few seconds. For the largest circuit in the set, it took about 3 hours of CPU time.

VII. CONCLUSIONS AND FUTURE WORK

We presented a new methodology for the decomposition of finite state machines using direct circuit manipulation. Given a state partitioning, we have shown how to construct each sub-machine and how to interconnect them such that the behavioral of the original machine is maintained. This procedure is applicable to very large FSMs even those for which implicit STG representation is not possible.

We have used this technique in the optimization of sequential circuits for low power dissipation. Approximate transition probabilities are computed from simulation and from these a

state partitioning is obtained. Register-disabling signals are added to the decomposed circuit so that overall switching activity is minimized. The results show that power savings of over 70% are possible in some of the examples tested. The use of the proposed implicit decomposition makes this technique applicable to very large circuits in acceptable CPU time.

An interesting direction for future research is on the automatic selection of the size of the *small* machine. Although, in some cases, significant gains can be obtained with a variety of sizes for this machine, in other cases the result depends strongly on the adequate selection of the value of this parameter. In general, a larger value of this parameter leads to a higher value of the merit function, but this gain is offset by the increased complexity of the small machine and the encoders used. It may be possible to modify the objective function to automatically include a term that depends on the number of states, thereby removing the need for the user to specify the value of this parameter or to perform a search for its right value.

We would also like to improve the state assignment methodology. Currently, the small machine encodings are selected using a simple heuristic, given the estimated inter-state transition probabilities. Improvements on the encoding used should significantly decrease switching levels in the small machine and the total power dissipation incurred [7, 17].

Finally, we are investigating other applications for the implicit FSM decomposition methodology developed.

ACKNOWLEDGMENTS

This research was supported by the Portuguese "Fundação para a Ciência e Tecnologia" under project "Praxis XXI".

REFERENCES

[1] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou. Precomputation-Based Sequential Logic Optimization for Low Power. *IEEE Transactions on VLSI Systems*, 2(4):426–436, December 1994.

[2] L. Benini and G. De Micheli. Transformation and Synthesis of FSMs for Low Power Gated Clock Implementation. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 21–26, April 1995.

[3] L. Benini, G. De Micheli, A. Lioy, E. Macii, G. Odasso, and M. Poncino. Computational Kernels and their Application to Sequential Power Optimization. In *Proceedings of the* 35^{th} *Design Automation Conference*, June 1998.

[4] A. Chandrakasan, T. Sheng, and R. Brodersen. Low-Power CMOS Digital Design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, April 1992.

[5] S-H. Chow, Y-C. Ho, and T. Hwang. Low Power Realization of Finite State Machines – A Decomposition Approach. *ACM Transactions on Design Automation of Electronic Systems*, 1(3):315–340, July 1996.

[6] S. Devadas and A. Newton. Decomposition and Factorization of Sequential Finite State Machines. *IEEE Transactions on Computer-Aided Design*, 8(11):1206–1217, November 1989.

[7] G. Hachtel, M. Hermida, A. Pardo, M. Poncino, and F. Somenzi. Re-Encoding Sequential Circuits to Reduce Power Dissipation. In *Proceedings of the International Conference on Computer-Aided Design*, pages 70–73, November 1994.

[8] J. Hartmanis. Symbolic Analysis of a Decomposition of Information Processing. *Information Control*, 3:154–178, June 1960.

[9] B. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, pages 291–307, February 1970.

[10] J. Monteiro and A. Oliveira. Finite State Machine Decomposition for Low Power. In *Proceedings of the* 35^{th} *Design Automation Conference*, pages 758–763, June 1998.

[11] F. Najm. A Survey of Power Estimation Techniques in VLSI Circuits (*Invited Paper*). *IEEE Transactions on VLSI Systems*, 2(4):446–455, December 1994.

[12] A. Papoulis. *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, 2^{nd} edition, 1984.

[13] E. Sentovich et al. *SIS: A System for Sequential Circuit Synthesis*. University of California, Berkeley, April 1992.

[14] V. Tiwari, P. Ashar, and S. Malik. Guarded Evaluation: Pushing Power Management to Logic Synthesis/Design. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 221–226, April 1995.

[15] H. Touati, H. Savoj, B. Lin, R. Brayton, and A. Sangiovanni-Vincentelli. Implicit State Enumeration of Finite State Machines using BDD's. In *Proceedings of the International Conference on Computer-Aided Design*, pages 130–133, November 1990.

[16] C-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. Despain, and B. Lin. Power Estimation Methods for Sequential Logic Circuits. *IEEE Transactions on VLSI Systems*, 3(3):404–416, September 1995.

[17] C-Y. Tsui, M. Pedram, C-A. Chen, and A. Despain. Low Power State Assignment Targeting Two- and Multi-level Logic Implementations. In *Proceedings of the International Conference on Computer-Aided Design*, pages 82–87, November 1994.