Synthesis of Power-Managed Sequential Components Based on Computational Kernel Extraction

Luca Benini, *Member, IEEE*, Giovanni De Micheli, *Fellow, IEEE*, Antonio Lioy, *Member, IEEE*, Enrico Macii, *Senior Member, IEEE*, Giuseppe Odasso, and Massimo Poncino, *Member, IEEE*

***Abstract*—This paper introduces a power optimization paradigm for sequential components based on the concept of computational kernel, a highly simplified logic block whose behavior mimics the steady-state behavior of the original specification. We present a flexible framework that supports a number of algorithmic options for carrying out kernel extraction. We first describe an exact symbolic procedure that is applicable to components for which only a functional specification (i.e., the state transition graph) is available. Due to its computational complexity, this procedure is mainly of theoretical interest and it is not usable for large circuits. We then propose two approximate algorithms that can be adopted in practical situations. The first one is simulation-based and it is suitable to cases where input data streams representing typical operation of the component are available. The second approach performs kernel extraction by iteratively refining a structural representation of the component obtained through synthesis. The impact of the power optimization paradigm based on kernel extraction is demonstrated by the results of extensive experimentation carried out on a number of benchmarks of different characteristics and nature.**

***Index Terms*—Logic synthesis, low-power design, sequential circuitsl.**

I. Introduction

WHEN specifying and designing a complex sequential component, engineers must consider not only the basic, typical behaviors, but also a large number of unusual operating conditions. In many cases, the number and the nature of these conditions (sometimes called *corner cases*) is such that they require considerable attention and design effort. As a result, final specifications are often much larger and more complex than what would be needed to just ensure correct functionality in the average case.

A key consequence of this fact is that sequential components may have an extremely large number of reachable states, but during normal operation, the circuits tend to visit only a relatively small subset of them. This intuitive statement is supported by the evidence provided by the probabilistic analysis of finite-state machines (FSMs) associated to large networks: only a few states have sizable occupation probabilities [1]. A similar situation occurs at the primary outputs; while the circuit walks through the most probable states, only a few distinct output patterns are generated.

The idea of optimizing complex digital systems based on improving their typical behaviors has been extensively exploited by computer architects: cache memories, branch prediction schemes, and variable-latency data paths are just a few notable examples. The main challenge in the implementation of these techniques is to effectively partition a design in such a way that commonly executed computations can follow a highly optimized path without being slowed down by the circuitry needed for dealing with all corner cases.

The main contribution of our work is the introduction of techniques for identifying the most probable behaviors in a sequential component and for automatically building a dedicated logic block that correctly implements such behaviors. The block, which we call *computational kernel*, is usually much smaller, faster, and less power-consuming than the module it is extracted from. Nevertheless, it can replace the original component for a large fraction of the operation time.

After kernel extraction, we still need to guarantee correct operation under any input condition. Hence, the kernel is connected in parallel to the original circuit and a selection logic is added, which, depending on the input patterns, selects either the original circuit or the kernel in a mutually exclusive fashion. By definition of computational kernel, its likelihood of being selected is very high. Therefore, the average computational cost decreases with respect to the original design.

Kernel extraction is key for successful optimization. It should be accurate (i.e., identify the set of most common cases with high precision), efficient (i.e., capable of manipulating large components), and it should produce high-quality implementations through tight integration with a logic synthesis engine.

We first introduce a kernel extraction procedure that is applicable to small sequential components for which the state-transition graph can be fully explored (either explicitly or implicitly). This procedure features binary decision diagram (BDD)-based algorithms that exactly determine the computational kernel through symbolic calculations similar to those employed in reachability analysis of FSMs. Exact symbolic kernel extraction is mainly of theoretical interest and normally not applicable to large components. As the size of the state transition graph (STG) increases, most symbolic operations are no longer affordable for both memory and time reasons.

An approximate variant of the exact symbolic algorithm replaces the most computationally expensive step of the exact algorithm (namely, the probabilistic analysis of the STG) with a cheaper (but less accurate) one. Although this solution extends

Manuscript received October 13, 1999; revised April 12, 2001. This paper was recommended by Associate Editor M. Papaeftthymiou.

L. Benini is with the Università di Bologna, Dipartimento di Elettronica, Informatica, e Sistemistica, Bologna 40136, Italy.

G. De Micheli is with the Computer Systems Laboratory, Stanford University, Stanford, CA 94305 USA.

A. Lioy, E. Macii, G. Odasso, and M. Poncino are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino 10129, Italy.

Publisher Item Identifier S 0278-0070(01)06892-0.

the domain of applicability of the method, it is still of limited usefulness for handling large components. We, thus, propose two extensions to the symbolic procedure. The first one is based on zero-delay functional simulation of typical input streams. In this case, complete, symbolic state occupation probability computation is bypassed and the set of states belonging to the kernel is determined directly from the simulation traces. This approach does not suffer from potential computational blowups as for the symbolic solution and enables kernel extraction for much larger components. In principle, even random pattern simulation can be used for kernel computation; however, the effectiveness of the simulation-based approach clearly depends on the availability of a meaningful stream to be simulated, i.e., one that well represents the typical operational context of the component. As for the two variants of the symbolic method, the simulation-based algorithm also does not require a structural representation (i.e., a gate-level netlist) of the sequential component. It can be applied directly at the register-transfer level (RTL), starting from a cycle-accurate functional specification.

In contrast, the second approximate approach we introduce requires a structural representation of the target module. The algorithm incrementally constructs the computational kernel of the component by iteratively modifying the next-state and output circuitry through logic implication analysis and redundancy removal. The extraction algorithm is driven by a cost function that monitors the quality of the kernel with respect to a given optimization target (e.g., performance, power) and, thus, provides a criterion for stopping the iterations. A side advantage of this solution is that once the kernel is available as a gate-level netlist, it may be exploited for further optimizing the original logic as well, since the input conditions for which the kernel is active actually represent *controllability don't cares* for the original component.

We show the practical significance of the optimization paradigm based on computational kernel extraction by applying it to the problem of reducing the power dissipation in sequential components. This implies a customization of the various extraction procedures to account for power dissipation as primary optimization constraint. Experimental results, obtained on several benchmarks, demonstrate the viability and effectiveness of the proposed optimization paradigm.

The remainder of the manuscript is organized as follows. Section II presents the notation that will be used throughout the paper. In Section III, we provide the basic theory of computational kernels and in Section IV we introduce the kernel-based power optimization paradigm. Section V puts our solution in perspective with related approaches available in the literature. Sections VI and VII are devoted to symbolic and approximate kernel extraction algorithms, respectively, and Section VIII reports the experimental results. Finally, Section IX provides closing remarks.

II. Background

A. Boolean Operators

We assume the reader to be familiar with the basic concepts of Boolean functions and with the data structures commonly used for the symbolic manipulation of such functions, i.e., BDDs. Background material on this subject can be found in [2]. We review here two Boolean operators essential for our purposes: cofactor and existential abstraction.

Given a single-output Boolean function $f(x_1, \ldots, x_n)$, the *positive* and the *negative cofactors* of f with respect to variable x_i are defined as $f_{x_i} = f(x_1, \ldots, x_i = 1, \ldots, x_n)$ and $f_{x_i'} = f(x_1, \ldots, x_i = 0, \ldots, x_n)$, respectively. The *existential abstraction* (or *quantification*) of f with respect to x_i is defined as $\exists_{x_i} f = f_{x_i} + f_{x_i'}$.

B. FSMs

An FSM M is defined as the 5-tuple $M = (X, Z, S, S^0, R)$, where X and Z are the input and output alphabets, S is the finite set of states, S^0 is the unique reset state, and $R \subseteq X \times S \times S \times Z \rightarrow \{0, 1\}$ is the *global relation*. $R(x, s, t, z) = 1$ if and only if, under input $x \in X$, M moves from present state $s \in S$ to next state $t \in S$ outputting $z \in Z$. The *size* of M is the cardinality of set S and it is denoted as $|M|$. M can be represented by an STG, whose states are elements of $s \in S$ and edges are labeled with pairs $(x, z) \in X \times Z$.

Given the global relation R of an FSM M, the *transition relation* T and the *output relation* O of M are defined as $T(x, s, t) = \exists_z R(x, s, t, z)$ and $O(x, s, z) = \exists_t R(x, s, t, z)$, respectively.

The FSM is often the formalism of choice for specifying the behavior of sequential components. We use it in Section III as a vehicle for introducing and illustrating the concept of computational kernel.

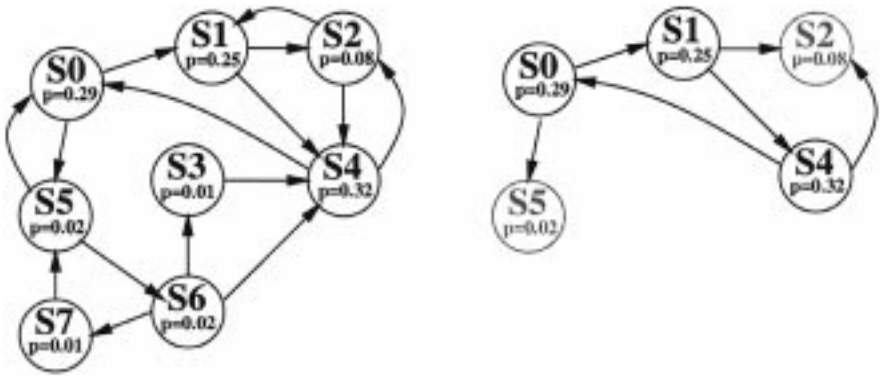
C. FSM Markovian Analysis

The probabilistic behavior of an FSM can be studied by regarding its transition structure as a Markov chain. It is sufficient to label each outgoing edge of each state with the probability for the FSM to make that particular transition to obtain a discrete-parameter Markov chain. On the other hand, studying the behavior of the Markov chain, i.e., computing the state occupation probabilities, is related to performing the reachability analysis of an FSM.

Given the transition relation of an FSM, it is possible to compute the vector **p** whose elements are the steady-state probabilities of the FSM to be in each state s. For small FSMs, the calculation can be carried out in an exact fashion using the algebraic decision diagram (ADD)-based procedures of [1]; for larger machines, the approximate techniques of [3] may be employed. In both cases, complex input probability distributions can be specified in order to have more detailed hardware modeling options.

III. Computational Kernels—Definition and Basic Theory

The functional specification of sequential components is commonly given in a cycle-accurate fashion, using hardware description languages (HDLs) with syntactic restrictions (called *synthesizable subsets*). Consistent with the terminology adopted by commercial synthesis tools, we call such specification style RTL to contrast it with structural or gate-level specifications.



(a) (b)

Fig. 1. (a) Moore-type FSM and (b) its 0.25-order computational kernel.

Each HDL has its own syntax, but most share the same FSM semantic. In other words, all sequential RTL specifications describe a (possibly large) finite-state sequential behavior. To rigorously define the concept of computational kernel, we refer to the FSM semantic of any RTL specification.

Given an FSM $M = (X, Z, S, S^0, R)$, modeling the behavior of a sequential component and given a probability threshold p, we define the p-order *computational kernel* of M, denoted as K, as the following FSM:

$$\mathrm{K} = (X, Z, S_p, S_p^0, R_p)$$

where $S_p = \{s \in S : \text{Prob}\,(s) \geq p\}$ represents the subset of the states of M whose steady-state occupation probabilities are larger than p. S_p^0 equals S^0 in case $S^0 \in S_p$, otherwise S_p^0 is chosen randomly within S_p. Finally, the global relation of K is defined as

$$R_p(x, s, t, z) = S_p(s) \cdot R(x, s, t, z).$$

The global relation of the kernel is incompletely specified: next state and output are uniquely defined only if the present state belongs to S_p. If this is not the case, the relation does not constrain the next-state and output values. In other words, $S_p(s)'$ is the *don't care* set for the next-state and output functions that can be extracted from R_p. An important characteristics of computational kernels is their probability Prob(S_p), which is defined as the cumulative occupation probability of the states in S_p.

As an example, consider the simple FSM shown in Fig. 1(a) in which the input and output values are omitted for the sake of simplicity and the states are annotated with the steady-state occupation probabilities calculated through Markovian analysis of the STG [1]. If we specify a probability threshold $p = 0.25$, the computational kernel of the FSM is depicted in Fig. 1(b). States in black represent set S_p, while states in grey represent valid values for t in $R_p(x, s, t, z)$, but they do not belong to S_p. The kernel probability is Prob $(S_p) = 0.29 + 0.25 + 0.32 = 0.86$.

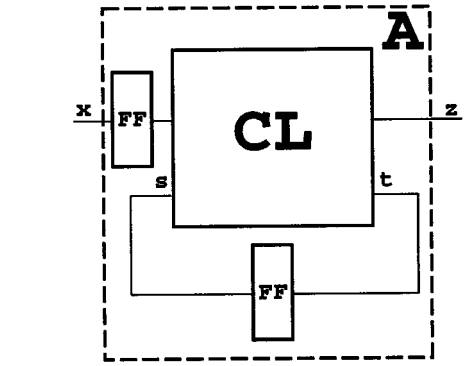
We note that in the implementation of the kernel extraction procedures we propose in Sections VI and VII, threshold probability p is used as a *relative* threshold. More precisely, kernel states have steady-state probability greater than p times the probability of the most frequently occurring state. For example, if $p = 0.8$, all states with steady-state probability $\geq 80\%$ of the most probable state are selected. This is necessary for large circuits, where absolute steady-state probability values are relatively low because the "kernel" behavior is spread over a relatively large number of states. In this case, fixing an absolute value of p will not select any state. In other terms, absolute threshold probability may have conceptual validity, but cannot be used in practice because it is a circuit-dependent quantity. For all the experiments we present in Section VIII, a fixed value of $p = 0.8$ was used. This choice was made after performing sensitivity analysis on a number of benchmark examples. We decided to take a conservative approach in the sense that the threshold is biased toward the selection of kernels with high state probability even at the price of losing some optimization opportunities.

Kernel extraction algorithms require two types of inputs, namely, the specification of M and the information on typical input pattern distribution. The latter is needed for determining the state probability distribution, which is a prerequisite for kernel computation.

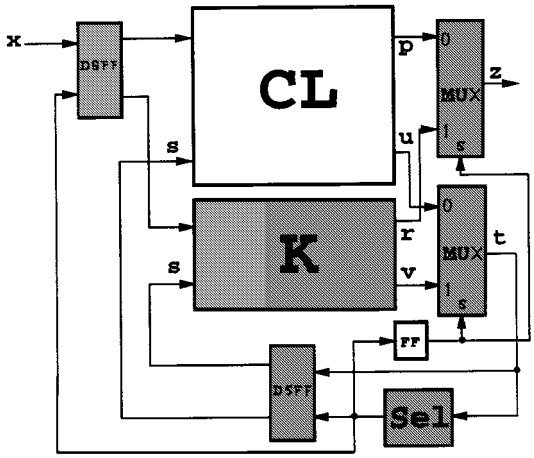
It is intuitively clear that the complexity of kernel extraction depends on the number of states and inputs of the specification. Kernel extraction is a relatively straightforward process for small state machines, where exhaustive state and input pattern enumeration is possible. Most sequential components have large state and input spaces that cannot be enumerated in a reasonable amount of time.

For RTL specifications with associated medium-size FSMs (a few millions of states), computational kernels can be determined by handling simultaneously sets of states represented as characteristics functions [4]. This is possible thanks to the BDD/ADD data structures that enable efficient manipulation of large and complex Boolean and pseudo-Boolean functions. More specifically, the symbolic kernel extraction algorithm we present in Section VI exploits well-established technology developed in the context of FSM reachability analysis [5]–[8] to extract kernels for FSMs that are too large to be handled by explicitly enumerative algorithms.

Unfortunately, large sequential components, which are the most common ones in the design practice, are often beyond the capabilities of the most powerful algorithms based on symbolic manipulations (in particular, those for state occupation probability calculation). For this reason, in Section VII, we formulate



(a)



(b)

Fig. 2. (a) Sequential component and (b) the kernel-based optimized architecture.

approximate kernel extraction algorithms that scale well with the size of the component.

IV. Kernel-Based Power Optimization Paradigm

In this section, we illustrate how the concept of computational kernel can be exploited for power optimization. For the sake of generality, we will refer to the FSM semantic which underlies any RTL specification. A generic FSM can be always represented as a combinational logic block coupled with sequential elements, as shown in Fig. 2(a). The optimized logic block after kernel extraction can be abstractly represented as in Fig. 2(b). The essential elements of the optimized FSM architecture are the combinational logic of the original block (block CL), the computational kernel (block K), the selector function (block Sel), the dual-state flip-flops (DSFF), and the output multiplexors (MUX). The computational kernel can be viewed as a "dense" implementation of the component it has been extracted from. In other terms, K implements the core functions of the original component and because of its reduced complexity, it usually implements such functions in a faster and more efficient way.

The purpose of the selector function Sel is that of deciding what logic block between CL and K will provide the output value and the next-state in the following clock cycle. To take a decision, Sel examines the values of the next-state outputs at clock cycle n. If the output and next-state values in clock cycle $n+1$ can be computed by the kernel K, then Sel takes on the value 1. Otherwise, it takes on the value 0. The value of Sel is fed to a flip-flop, whose output is connected to the multiplexors that select which block produces the correct outputs and next-state. Obviously, the kernel-based power-optimized implementation of the component is functionally equivalent to the original one.

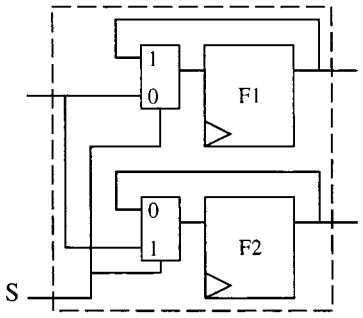


Fig. 3. Functional model of a DSFF.

The scheme of Fig. 2 is just one among several possible architectures. The peculiar feature of the proposed solution concerns the topology of the selection logic. In particular, choosing a selection function that only depends on the next-state outputs is motivated by the need of obtaining a small implementation. Reducing the support of Sel by not including the primary inputs helps in this respect.

The sequential elements indicated as DSFFs replace the ordinary flip-flops present in the original design. A DSFF is functionally equivalent to the schematic of Fig. 3 and it operates as follows. When $\mathrm{Sel}=1$, flip-flop F_2 is loaded with a new value coming from the external data input, while flip-flop F_1 holds its state. The opposite happens when $\mathrm{Sel}=0$. In this way, the state of either the kernel or the original network can be kept unchanged (i.e., frozen in the DSFFs) while the other logic block is being used to produce the output and the next state. Clearly, Fig. 3 only describes the functional behavior of a DSFF. Its actual implementation can be properly optimized to reduce the overhead with respect to the standard flip-flops used in the original sequential network, as discussed in the Appendix.

It can be observed in the first place that besides the automatic generation of the kernel, the logic for the selector function Sel needs to be synthesized as well. Fortunately, such logic can be obtained as a byproduct of the kernel extraction process. In Sections VI and VII, we will show how kernel extraction can be modified to obtain Sel.

The power efficiency of the architecture of Fig. 2(b) stems from the fact that it enforces mutual exclusion: the large block CL is "frozen" when the kernel logic is enabled. In fact, the inputs of CL remain fixed when K is active, thereby nullifying switching activity. In normal operation, K is active most of the time and average switching activity within CL is much reduced. Since the logic in K is simpler than CL, the kernel dissipates much less power than block CL. On the other hand, the modified architecture has some overhead with respect to the standard implementation: DSFFs consume more power than the standard ones, the selector function is always active and dissipates power, and so do the output multiplexors. In general, we can expect significant power savings only if:

1) the kernel is significantly smaller than the original circuit;
2) the selector function is very small and power efficient;
3) the kernel is active most of the time.

The kernel extraction algorithms of Sections VI and VII strive for satisfying all the above requirements.

V. Related Work

The development of computer-aided design techniques for power minimization has been a very active area of research in the last few years (refer to the surveys by Pedram [9] and by Macii *et al.* [10] for further reading). Our paper is related to a number of sequential power optimization techniques that are briefly summarized next.

FSM decomposition for low power [11]–[14] can be seen as a top–down computational kernel extraction procedure that starts from explicit STG specifications (i.e., state tables or equivalent formats). Although decomposition techniques reported sizable power reductions, they can be applied only to small sequential components for which the explicit STG description can be manipulated in reasonable time and memory space. The main advantage of our approach to kernel extraction is that it can be applied to much larger sequential components for which not even the implicit representation of the STG can be constructed.

A few high-level decomposition techniques have also been presented in the literature [15]–[17]. They exploit information available during high-level synthesis to find a good controller and data-path decomposition. Even though we believe that these solutions have very good potential and may scale well for large components, they have some limitations. In fact, the approaches described in [15] and [16] can be applied only to design flows that are based on behavioral synthesis. Furthermore, the method presented in [17] operates directly on the HDL specification using algorithmic transformations such as loop unrolling, constant propagation and partial evaluation, which can be applied only to a restricted subset of the available HDL constructs. Our approach to computational kernel extraction is not subject to all these restrictions: it can be applied to any sequential specification. A consequence of its wider scope of usability, however, is that computational kernel extraction may be less scalable to very large sequential components.

Automated extraction of clock-gating logic [18]–[22] is based on the observation that a sequential component can, under some input conditions, be unobservable or it may not react to input changes. In these cases, the circuit is *idle* and power can be saved by stopping its clock. This is done by creating a *clock-gating function* that stops the clock when idleness is detected. Clearly, the larger the number of idle conditions, the higher the opportunities for power savings. Interestingly enough, we can view idle conditions as a very special computational kernel, i.e., the *null kernel*. In other words, when the circuit is idle, it can be replaced by an empty kernel and kernel extraction simply reduces to the construction of the selection function. From this viewpoint, computational kernel extraction can be seen as a generalization of clock-gating function generation. Therefore, since it exploits other kinds of behaviors than just idleness, it may potentially provide more prominent opportunities for power reduction and it can be applied to sequential blocks for which clock-gating is ineffective.

```
procedure Symbolic_Kernel_Extraction (D(x), R(s,x,z,t), p) {
  P(s) = Markov(D(x), R(x,s,t,z));
  S_p(s) = Threshold(P(s), p);
  T(x,s,t) = ∃_z R(x,s,t,z);
  O(x,s,z) = ∃_t R(x,s,t,z);
  T^p(x,s,t) = S_p(s) · T(x,s,t);
  O^p(x,s,z) = S_p(s) · O(x,s,z);
  Δ^p(x,s) = ((∃_{t≠t_1} T^p(x,s,t))_{t'_1}, ..., (∃_{t≠t_m} T^p(x,s,t))_{t'_m});
  Λ^p(x,s) = ((∃_{z≠z_1} O^p(x,s,z))_{z'_1}, ..., (∃_{z≠z_m} O^p(x,s,z))_{z'_m});
  return (Δ^p(x,s), Λ^p(x,s), S_p(s));
}
```

Fig. 4. Symbolic kernel extraction algorithm.

Precomputation-based power optimization [23], [24] pushes the clock-gating concept even further. The simple idleness detection logic is replaced by more complex precomputation function that can either detect *partial* idle conditions (i.e., conditions in which only part of the original circuit needs to be active to compute correct output and next-state values) or compute the values of the outputs one cycle in advance and deactivate the original circuit. Precomputation can be seen as a kernel extraction procedure, where the selector function is merged with the kernel logic, and both are evaluated one cycle in advance. This choice is suboptimal because it implies that the computational kernel logic cannot be disabled [while the kernel K of Fig. 2(b) is not evaluated when CL is active]. Thus, it always contributes to the overall power consumption. The algorithms for extracting precomputation logic are not effective for kernel extraction because they tend to produce very small kernels with limited probability.

VI. Symbolic Kernel Extraction

A symbolic algorithm for computational kernel extraction is an almost direct implementation of the definition given in Section III. It leverages state-of-the-art implicit BDD-based algorithms for the manipulation of sets of states and Boolean functions and its pseudocode is shown in Fig. 4.

The inputs to the procedure are the following.

1) A multiterminal BDD (or ADD [25]) whose support is the set of input variables x. The leaves of the ADD represent occurrence probabilities of the input patterns. The ADD is a compact representation of function $D(x) : X \rightarrow \Re$ that associates to each input pattern its probability.
2) A BDD representing the global relation $R(x, s, t, z)$ of the component being optimized.
3) A probability threshold p. According to the definition of Section III, all states with steady-state occupation probability greater than p belong to the state set of kernel K.

The first step is to compute steady-state probabilities for all states $s \in S$. This is accomplished by exact symbolic Markov analysis [1]. Procedure `Markov` computes state probabilities and returns an ADD with support s (i.e., the state variables). The leaves of the ADD are state probabilities. The state set S of the original circuit is then pruned using the `Threshold` operator. Such operator takes, as inputs, the ADD of the state probabilities

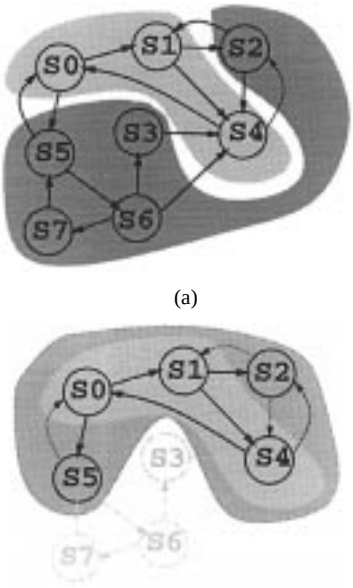


Fig. 5. Example of kernel extraction. (a) Light grey area: occupation probability greater than or equal to p. (b) Light grey (innermost) area: $S_p(s)$. Dark grey (outermost) area: states not belonging to $S_p(s)$.

and the probability threshold p and returns the BDD $S_p(s)$ of the kernel state set.

To obtain the next-state and output functions of the kernel, we first extract the transition relation $T(x,s,t)$ and the output relation $O(x,s,z)$ from the global relation of the original circuit; then, we compute their conjunction with $S_p(s)$. This can be done because the behavior of the computational kernel is specified only when the current state belongs to $S_p(s)$.

The result of the conjunction is illustrated in Fig. 5. In Fig. 5(a), the light grey area identifies the states for which the occupation probability is greater than or equal to p. In Fig. 5(b), on the other hand, the innermost (light grey) shaded area represents $S_p(s)$, while the outer shaded region (dark grey) represents states not belonging to $S_p(s)$ that are valid next states for the next-state function of kernel K because they are associated with present states that do belong to $S_p(s)$. In other words, there is a "corolla" consisting of all the states that are 1-step reachable from $S_p(s)$ for which the behavior of the kernel must be specified and consistent with the original one.

Given T^p and O^p, it is relatively easy to extract the next-state and output functions of the computational kernel by existential quantification followed by cofactoring (in the pseudocode, we assume that there are n state variables and m output variables). Finally, the procedure returns functions Δ^p and Λ^p that completely specify the functionality of block K in Fig. 2(b) and function $S_p(s)$ that will be used for synthesizing the selection logic [block Sel in Fig. 2(b)]. Obviously, Sel is a function of the next-state; therefore, a formal step of variable substitution (from s to t) is done before synthesis. Notice that $S_p(s)'$ represents the *don't care* set that can be used to optimize the implementation of K in case the synthesis of such block of logic still needs to be done.

As mentioned in the introduction, the most computationally intensive phase of the symbolic kernel extraction algorithm is the one that calculates state occupation probabilities. One simple, yet effective way of getting around the complexity problem consists of replacing the `Markov` routine with the implementation of the algorithm for approximate FSM probabilistic analysis of [3].

This solution definitely helps in extending the applicability of the extraction procedure to circuits whose state space is large, at a quite limited cost in terms of achievable power savings. Nevertheless, the bottleneck introduced by the overall symbolic computation engine cannot be eliminated. Consequently, radically different algorithms are needed to make kernel-based power optimization applicable in realistic design environments. We propose two options in Section VII.

VII. Approximate Kernel Extraction

The results of Section VIII-A show that the kernel-based paradigm can provide sizable power reductions. On the other hand, they also prove that the applicability of the symbolic method for kernel extraction described in Section VI is limited to instances of components with associated small-to-medium FSMs. In fact, these are the only cases where state occupation probabilities can be calculated using the symbolic methods summarized in Section II-C. Another limitation of the algorithm is that it assumes the knowledge of the probability of every possible input pattern of the circuit (this is required to obtain ADD $D(x)$). In this section, we describe approximate kernel extraction techniques that scale well with the complexity of the components being handled. In particular, we first focus on a solution that considers RTL components for which only the input–output (I/O) specification is available. Then, we move to an algorithm that requires the availability of gate-level netlists of the components under optimization.

A. Simulation-Based Extraction

We target large RTL components for which symbolic Markovian analysis (either exact or approximate) is infeasible. Furthermore, we assume that knowledge of input pattern distribution is limited to the specification of a set of patterns that represent typical operating conditions under which the circuit is expected to run. Finally, we consider components with black box specifications whose internal structure does not yet exist or it is not available.

The solution we have identified is simulation-based and assumes the existence of a set of patterns representing typical operation conditions. Alternatively, patterns can be automatically generated with a user-specified probability distribution. In the first step, the component is simulated with L input patterns. During simulation, the number of occurrences of each visited state is monitored. At the end of simulation, the probability distribution of the *visited states* is computed by simply dividing the number of occurrences of each state by L.

To compute a p-order kernel, all states with probability larger than p are selected. Notice that during simulation, at most L states are visited. State probability computation and state selection simply involve linear processing of the visited state list. Hence, they are even faster than simulation. The set of selected

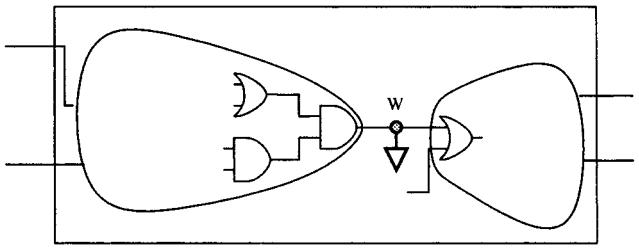


Fig. 6. Elementary transformation for constructing a computational kernel.

states is taken as state set $S_p(s)$ that is used after formal variable substitution (from s to t) as specification of block Sel in Fig. 2(b).

The next-state and output functions of the kernel can be determined by using the procedure of Fig. 4 in which the ADD $P(s)$ constructed from simulation replaces the one built by procedure `Markov`.

B. Structural Extraction

In this section, we present an iterative procedure for determining the computational kernel of a sequential component that works on the structural description of the network. Hence, the method can be employed to optimize components for which a gate-level netlist is available. However, the technique can also be employed to optimize circuits for which an existing implementation is not available. In this case, a preliminary (possibly fast) synthesis step is carried out to generate a gate-level representation on which the procedure described below can be applied.

The algorithm is heuristic by nature and it exploits concepts such as *logic implication* and *redundancy removal* that have been in use for quite a long time in applications like test generation and logic optimization. More specifically, it takes the netlist of the original circuit A and it iteratively computes the kernel K by removing gates and connections from the combinational logic CL of A.

The elementary step of the iterative transformation is shown in Fig. 6. A connection w is selected inside network A. Signal w is replaced with either the constant value 0 (shown in Fig. 6) or 1 and the circuit is simplified by propagating the constant value in the fan-out cone of w and by removing all the fan-out-free logic gates in the transitive fan-in of w. Notice that w can be a primary input or a primary output of the logic network.

The computational kernel K is obtained from the original netlist A by iteratively applying the elementary transformation until a stopping criterion is met. The pseudocode of the procedure is shown in Fig. 7. K^i denotes kernel K after i applications of the transformation (initially, $\mathrm{K}^0 \equiv A$).

The procedure takes, as inputs, the original circuit synthesized to gates A, the cost function F, which controls the stopping criterion, the utility function U_w, which drives the node selection process and the timing constraint T to be used in the synthesis of the selection logic Sel. This is because the speed of block Sel must always be kept under consideration since its delay adds up to the critical delay of the original sequential circuit. The output of the algorithm is the computational kernel K^i obtained after i iterations of the **do while** loop.

```
procedure Structural_Kernel_Extraction(A, F, U_w, T) {
  i = 0; K^i = A;
  do {
    w = SelectNode(A, U_w);
    val = ChooseBestValue(A, w);
    K^{i+1} = PropagateValues(K^i, A, w, val);
    S_i = ∏_{j=1}^{m+n} (ρ_j^{i+1} ≡ ω_j);
    BuildSelectionLogic(S_i, T);
    i++;
  } while (!StopTest(K^i, S_i, F))
  return(K^i, S_i);
}
```

Fig. 7. Structural kernel extraction algorithm.

In the sequel, we discuss in detail the key steps of the algorithm: kernel generation by simplification of the initial netlist, synthesis of the selection logic, and stopping criterion.

1) Node and Value Selection: Performing the selection of the candidate node w on which the elementary transformation of Fig. 6 is applied (procedure `SelectNode`) requires a utility function U_w that estimates the savings that can be achieved if w is replaced by either 0 or 1. The U_w we adopt is approximate since we are interested in fast, yet reasonably accurate estimates of the power savings. It is the sum of the expected power savings on w (χ_w) and the expected power savings on the fan-in and fan-out (ξ_w)

$$U_w = \chi_w + \xi_w. \tag{1}$$

ξ_w can be efficiently computed, given the transition probability $Tp[w]$ and the load capacitance C_w^{load} of node w as $\chi_w = Tp[w] \cdot C_w^{\mathrm{load}}$. On the other hand, ξ_w can be determined by finding how many gates will be eliminated from A by the transformation and summing the total power

$$\mathrm{Pow}_{\mathrm{save}} = \sum_{\forall w_i \text{eliminated}} Tp[i] \cdot C_i^{\mathrm{load}} \tag{2}$$

The number of gates eliminated can be easily determined by evaluating how many gates in the fan-in and fan-out cones have their outputs fixed to some constant value.

Even though the approximate U_w discussed above usually provides estimates that are accurate enough, for critical cases (e.g., portions of logic that are always under stress, controllers), it may be required resorting to exact calculations at the price of an increased runtime of the kernel extraction procedure. The exact solution consists of evaluating the actual power savings resulting by the substitution in the network by running accurate power estimation. This is clearly much more time and memory demanding, since estimation must be invoked for every new substitution; thus, it may be infeasible for large netlists.

Once a candidate node w has been selected, there are two options for the selection of what value is the best to force on w (procedure `ChooseBestValue`). For small- to medium-sized circuits, it may be feasible to force first zero and then one on the network A and then evaluate U_w using (1) for each of the two choices. The selected value *val* will be the one with the highest value of U_w.

For larger circuits in which the propagation of a value can require a certain amount of time, it is better to *estimate* the best value to assign to w. This can be done as follows. The node w will usually have a transition probability $Tp[w]$ that will be close to 0.5; this is because under the assumption of temporally uncorrelated circuit inputs, the transition probability can be obtained as

$$Tp[w] = p_w(1-p_w) + (1-p_w)p_w = 2p_w(1-p_w) \quad (3)$$

where p_w is the signal probability of node w. The first term in (3) indicates the probability of a $1 \rightarrow 0$ transition occurring on node w while the second term represents the probability of the other transition $(0 \rightarrow 1)$. This function has a maximum at $p_w = 0.5$; therefore, values of the signal probability close to 0.5 imply a high transition probability. This means that once w has been selected, the value *val* being assigned should be obtained by the logic expression $p_w > 0.5$. In other terms, w should be assigned to zero if $p_w < 0.5$, to one otherwise, i.e., to the value which is more likely to occur.

2) Synthesis of the Selection Logic: The ON-set of the selection function at the ith iteration of the extraction loop is defined as the set of next-state conditions for which, in the next clock cycle, output and next-state values of K^i are equal to output and next-state values of CL (i.e., K^i can be used in place of CL to compute the output and next-state values). Thus, $\mathrm{Sel_i}$ can be computed in an exact fashion by the following equation:

$$\mathrm{Sel}_i(t) = \prod_{j=1}^{m+n} \left(\rho_j^i \equiv \omega_j\right) \quad (4)$$

where the product stands for logic conjunction, ρ_j^i is the jth output of network K^i [i.e., an element of set (r, v) in Fig. 2], ω_j is the jth output of network CL [i.e., an element of set (p, u) in Fig. 2], x are the primary inputs, m is the number of primary outputs, and n is the number of next-state outputs.

Since BDDs are used to represent both the kernel and the selection function, Sel_i can be computed as long as it is possible to construct its BDD. There may exist circuits for which the BDD of Sel_i can not be constructed. In these cases, since the logic implementing the selection function must be synthesized for power under timing constraints, we use a *subsetting* algorithm [26] that provides an implementation of the selection logic whose ON-set is smaller than that of Sel_i [and whose implementation is fast (or small) enough], but with maximum probability. With subsetting, we sacrifice probability for performance and/or area.

Finally, one additional optimization can be obtained by observing that the original circuit A can be optimized using Sel_i as controllability *don't care* set to reduce the area overhead and to save additional power. This is because when $\mathrm{Sel}_i = 1$, the functionality implemented by A is already computed by K.

3) Stopping Criterion: The iterative construction of K and Sel tends to yield increasingly small selection functions. Intuitively, this corresponds to lowering the probability of using K instead of A to compute the outputs and the next states. After some iterations within the **do while** loop, this probability might become too small, reducing the fraction of time in which K is selected. The process should then be stopped.

An estimate of the power Pow_i dissipated by the ith version of the architecture can be computed as

$$\begin{aligned}\mathrm{Pow}_i =& \mathrm{Pow}(\mathrm{K}^i) \cdot \mathrm{Prob}(\mathrm{Sel}_i) + \mathrm{Pow}(A) \cdot (1 - \mathrm{Prob}(\mathrm{Sel}_i)) \\ &+ \mathrm{Pow}(\mathrm{Sel}_i). \end{aligned} \quad (5)$$

Equation (5) can be interpreted as follows: when $\mathrm{Sel}_i = 1$, the kernel K is operating; thus, it dissipates $\mathrm{Pow}(\mathrm{K}^i)$ for a fraction of time equal to $\mathrm{Prob}(\mathrm{Sel}_i)$. Conversely, when $\mathrm{Sel}_i = 0$, the original circuit A is operating; thus, it dissipates $\mathrm{Pow}(A)$ for a fraction of time equal to $1 - \mathrm{Prob}(\mathrm{Sel}_i)$. Then, the condition to be checked for terminating the iterations occurs when Pow_i stops decreasing from one iteration to the next one.

In practice, however, the first iterations of the loop of Fig. 7 do not always result in a reduction of Pow_i. Therefore, a greedy test on the monotonic decrease of Pow_i would prevent the search of better solutions.

In order to avoid such cases, in the actual implementation of procedure `StopTest`, the stopping condition is not tested for the first few iterations (this is to ensure that the K^i is sufficiently simpler than A). It is then tested again thereafter to make sure that $\mathrm{Prob}(\mathrm{Sel}_i)$ does not become too small.

Concluding the discussion on the approximate simulation-based and structural approaches, we point out that both techniques require the availability of next-state and output function BDDs for the sequential component. Hence, approximate extraction cannot be carried out when BDDs cannot be constructed because of memory blowup.

VIII. Experimental Results

A. Symbolic Extraction

We have implemented the symbolic extraction algorithm as an extension of SIS [27] using CUDD [28] as the underlying BDD package. Experiments were run on a DEC AXP 1000/400 workstation with 256 MB of memory.

We have first applied the algorithm of Fig. 4 to the examples taken from the ISCAS'89 sequential suite [29] (addendum included) for which the exact state occupation probabilities can be computed using the ADD-based method of [1].

The STGs of the circuits have been initially synthesized as networks of multiplexors directly from the BDD representation of the output and transition relation, optimized for area using `script.rugged` (whenever possible) and mapped for area with `map -m0 -AFG` onto a 3.3-V 0.5-μm complementary metal–oxide–semiconductor (CMOS) library containing two– to four-input NAND and NOR gates, inverters and buffers with three different drive strengths, and a flip-flop.

Table I reports the data for the examples for which some power savings have been obtained. In particular, columns *In*, *Out*, *FF*, *Gates*, and *Delay* report the characteristics of the reference circuits. Column $Power$ shows the power in μW of the reference circuit (column *Ref*) and that of the kernel-based architecture (column *Opt*) as well as the obtained savings (column *Sav*). Column $\mathrm{Prob}(\mathrm{Sel})$ tells the probability of the selector function Sel to be one, i.e., the probability of kernel K to be active. Column *Area Overhead* shows the area cost of the modified architecture, expressed as a percentage of the reference

TABLE I
RESULTS—SYMBOLIC EXTRACTION USING EXACT MARKOVIAN ANALYSIS

Circuit	*In*	*Out*	*FF*	*Gates*	*Delay*	*Power*			*Prob(Sel)*	*Area Overhead*	*Delay Overhead*	*CPU Time*
						Ref	*Opt*	*Sav*				
s298	3	6	14	131	20.7	361	178	51%	0.74	39%	9%	53
s349	9	11	15	146	21.4	378	279	27%	0.77	61%	18%	81
s382	3	6	21	178	22.5	453	226	50%	0.75	35%	15%	170
s386	7	7	6	136	22.0	237	126	47%	0.89	51%	14%	29
s400	3	6	21	174	26.5	446	224	50%	0.75	35%	11%	288
s444	3	6	21	175	25.0	460	223	52%	0.75	31%	12%	276
s510	19	7	6	262	32.5	641	586	10%	0.45	60%	16%	193
s526	3	6	21	194	16.0	509	252	51%	0.75	33%	15%	304
s641	35	23	17	188	34.0	448	340	24%	0.63	47%	11%	511
s713	35	23	17	202	27.1	470	334	29%	0.63	62%	15%	506
s820	18	19	5	316	17.3	658	97	85%	0.96	19%	12%	49
s832	18	19	5	276	21.8	605	93	85%	0.96	18%	7%	46
s967	16	23	29	470	17.9	777	602	22%	0.35	25%	15%	853
s991	65	17	19	450	43.6	1055	830	21%	0.25	14%	12%	670
s1488	6	19	6	685	21.9	1414	212	85%	0.91	7%	16%	73
s1494	6	19	6	680	27.3	1425	205	86%	0.91	7%	11%	74
Avg.								51%		34%	13%	

gate count. Similarly, column *Delay Overhead* shows the performance penalty introduced by the use of the kernel-based architecture. Finally, column *CPU Time* indicates the execution time in seconds required by the optimization procedure to complete.

Power estimates within the kernel extraction procedure have been computed using symbolic simulation [30] while those for the initial and final circuits have been determined using the Irsim switch-level simulator [31], assuming a clock frequency of 100 MHz.

Results are very encouraging. An average power savings of approximately 51% has been achieved with peaks of more than 80%. Notice that some circuits of the complete suite are missing in the table, namely, s208, s420, s838, s1196, and s1238. These circuits have the peculiar property of having all states that are equiprobable; this uniform probability distribution clearly prevents the application of the kernel-based optimization paradigm. For the remaining circuits, however, the existence of a dense set of states implementing most of the behavior seems to be the rule. In the cases where the kernel probability Prob(Sel) is very high (e.g., s820, s1196), we have observed that the kernels typically include very few high-probability states.

Concerning the timing of the optimized circuits in spite of the fact that the delay of the selector function Sel adds up to the largest delay between CL and K, the penalty is limited (13% on average). This is because CL is optimized using Sel as *don't care set*. Thus, its delay usually reduces with respect to the original circuit. On the other hand, as it was expected, the area penalty is significant (34% on average). This is because the kernel-based approach suffers, in principle, from the same overhead (logic duplication) that affects any type of parallel implementation.

One final remark concerns the relation between kernel probability and the actual power savings. In principle, a high probability is a good indicator for a significant savings. However, there are other factors that can influence the achievable savings. For example, the power consumed by the selection logic (i.e., selection function and output MUXs) is somehow independent of the kernel probability; rather, it is tightly related to what states the kernel contains.

As mentioned in Section VI, one way of reducing the complexity of the symbolic kernel extraction algorithm consists of replacing the exact Markovian analysis procedure of [1] with the approximate one of [3]. This allows us to handle larger designs at the cost of a decreased accuracy in state probability calculation and, therefore, of smaller power savings. In Table II, we compare the results we have obtained using approximate Markovian analysis to those of Table I.

As expected, introducing an approximation in the calculation of the steady-state occupation probabilities has produced a degradation, quite limited though, of the power savings with respect to those obtained using the exact method. On the other hand, it enhances the computational capabilities of the tool. Circuits s1423 and s1512 are, in fact, not manageable by using exact Markovian analysis.

B. Simulation-Based Extraction

To benchmark the simulation-based extraction method of Section VII-A, we have considered the RTL circuits from [32] for which only a functional description (specified in Esterel) is available. The BDDs for the next-state and output functions have been constructed directly from the Esterel source code using a procedure similar to that described in [33]. The data of the experiments are collected in Table III.

For circuits whose functionality is known, as those in the table, the simulation-based method works well (average power savings are around 34%); this is because kernel extraction has been done by feeding the circuits with meaningful input traces (each of which consisted of approximately 50 000 patterns). On the contrary, for circuits whose functionality is unknown, the only way of providing them with an input trace is through random generation. In this case, the results are expected to be less appealing.

TABLE II
RESULTS—SYMBOLIC EXTRACTION USING EXACT AND APPROXIMATE MARKOVIAN ANALYSIS

Circuit	Power Savings		Prob(Sel)		Area Overhead		Delay Overhead		CPU Time	
	Exact	Approx	Exact	Approx	Exact	Approx	Exact	Approx	Exact	Approx
s298	51%	50%	0.74	0.73	39%	39%	9%	11%	53	40
s349	27%	18%	0.77	0.74	61%	50%	18%	20%	81	52
s382	50%	39%	0.75	0.70	35%	41%	15%	17%	170	59
s386	47%	64%	0.89	0.97	51%	42%	14%	11%	29	25
s400	50%	43%	0.75	0.76	35%	37%	11%	21%	288	64
s444	52%	48%	0.75	0.75	31%	36%	12%	21%	276	69
s510	10%	12%	0.45	0.44	60%	64%	16%	19%	193	59
s526	51%	40%	0.75	0.74	33%	30%	15%	19%	304	73
s641	24%	11%	0.63	0.66	47%	54%	11%	12%	511	86
s713	29%	19%	0.63	0.64	62%	74%	15%	16%	506	84
s820	85%	70%	0.96	0.96	19%	19%	12%	12%	49	29
s832	85%	72%	0.96	0.96	18%	19%	7%	16%	46	26
s967	22%	13%	0.35	0.48	25%	79%	15%	17%	853	93
s991	21%	10%	0.25	0.41	14%	15%	12%	19%	670	84
s1488	85%	79%	0.91	0.91	7%	9%	16%	16%	73	48
s1494	86%	80%	0.91	0.92	7%	9%	11%	11%	74	49
Avg.	51%	42%			34%	39%	13%	16%		
s1423	—	80%	—	0.74	—	70%	—	18%	—	2292
s1512	—	39%	—	0.27	—	26%	—	9%	—	1840

TABLE III
RESULTS—SIMULATION-BASED EXTRACTION

Circuit	In	Out	FF	Gates	Delay	Power			Prob(Sel)	Area Overhead	Delay Overhead	CPU Time
						Ref	Opt	Sav				
Boltzmann	7	21	91	367	15.7	134	50	63%	0.42	45%	13%	674
FifoWriteCntr	2	2	17	141	16.8	57	42	27%	0.62	58%	25%	93
Gcd16	33	17	50	1197	32.6	128	83	35%	0.19	36%	19%	578
Iqc	10	15	36	1169	23.9	445	220	51%	0.90	21%	18%	492
Lan	10	8	19	215	19.4	107	99	7%	0.84	44%	19%	116
Radio	5	16	16	181	14.6	110	77	30%	0.98	35%	23%	76
Watch	4	16	11	108	9.1	67	51	24%	0.98	58%	18%	59
Avg.								34%		42%	19%	

TABLE IV
RESULTS—STRUCTURAL EXTRACTION

Circuit	In	Out	FF	Gates	Delay	Power			Prob(Sel)	Area Overhead	Delay Overhead	CPU Time
						Ref	Opt	Sav				
s1269	18	10	37	468	50	965	902	7%	0.45	61%	7%	21
s1423	17	5	74	602	73	1389	460	67%	0.62	58%	8%	90
s1512	29	21	57	475	42	1003	843	16%	0.80	19%	3%	85
s3271	26	14	116	1045	35	3286	1611	51%	0.78	44%	8%	204
s3384	43	26	183	1393	92	11516	10757	7%	0.38	82%	6%	215
s4863	49	16	104	2022	86	9590	7809	19%	0.97	70%	9%	567
s5378	35	49	164	1132	22	978	345	65%	0.94	52%	3%	418
s6669	83	55	239	2703	163	7083	6254	11%	0.99	61%	3%	355
s13207	31	121	669	2462	44	8806	6831	22%	0.96	52%	6%	633
s15850	14	87	597	3417	72	8814	7227	18%	0.98	48%	5%	1050
Avg.								28%	0.79	55%	6%	

C. Structural Extraction

The benchmarks we have used to check the effectiveness of the structural kernel extraction algorithm of Section VII-B are *all* the large ISCAS'89 sequential circuits [29], including the *addendum* (i.e., a total of 15 examples). The netlists were optimized and mapped as for the experiments of Section VIII-A.

Table IV reports the data for the examples for which some power savings have been obtained (ten cases). Power reductions are still relevant (28% on average), although the area overhead is quite substantial (55% on average). As expected, delay increase is very limited (6% on average) thanks to the fine tuning enabled by the synthesis procedure for the selection logic.

For the remaining five benchmarks in the suite, results are not provided, since the application of our technique to examples s3330 and s9234 did not produce any noticeable power improvement, while circuits s35932, s38417, and s38584 could not be mapped using SIS onto our technology library.

TABLE V
RESULTS—POWER SAVINGS OBTAINED WITH DIFFERENT EXTRACTION TECHNIQUES

Circuit	*Inputs*	*FFs*	*Power Savings*			
			Sym-Exact	*Sym-Approx*	*Sim-Based*	*Struct*
s298	3	14	51%	50%	50%	19%
s713	35	17	29%	19%	20%	15%
s820	18	5	85%	70%	58%	26%
Iqc	10	36	–	86%	51%	23%
Radio	5	16	37%	35%	30%	19%
Watch	4	11	41%	31%	24%	16%
s1423	17	74	–	80%	35%	67%
s1512	29	57	–	39%	11%	16%
s3271	26	116	–	–	6%	51%

D. Comparison

All the kernel extraction techniques we have presented in this paper are directly compared in Table V using a total of nine benchmarks, chosen in equal number from Tables I, III, and IV. The results provided by the two variants of the symbolic method (exact and approximate) are reported in columns *Sym-Exact* and *Sym-Approx*, respectively, while those obtained using the simulation-based method are collected in column *Sim-Based*. Finally, data for the structural technique are shown in column *Struct*.

For circuits s298, s713, and s820, all the methods are applicable. Since their state spaces are small, the *Sim-Based* method is almost as good as the symbolic ones because relatively short simulations are sufficient to bring the circuits into the subsets of states that make the computational kernels. On the other hand, the *Struct* method works quite poorly. This fact can be explained by observing that structural transformations in a small circuit are likely to drastically change the overall behavior, thereby reducing the percentage of time in which the kernel can be activated.

For benchmarks Iqc, Radio, and Watch, the *Sim-Based* method works better than *Struct* because the patterns used for simulation are meaningful and not generated with the assumption of 0.5 input probability. Since these benchmarks have relatively small size, the structural method does not have many opportunities for netlist transformations without drastically altering kernel functionality (and, consequently, its probability of being selected). The same input probabilities as those used for the *Sim-Based* method were used for the two symbolic methods. For example, Iqc, the *Sym-Exact* method did not complete and *Sym-Approx* gives much better results than *Sim-Based* because the state space that is explored with the simulation does not seem to be large enough to identify all the highly probable states of the kernel. Results for Radio and Watch obtained with *Sym-Exact* and *Sym-Approx* are close to those given by *Sim-Based*. This is justified by the fact that since they have a relatively small state space, the simulation allows to properly identify the computational kernel.

For circuits s1423, s1512, and s3271, the *Sym-Exact* is obviously not applicable (exact Markovian analysis does not complete). Also, limited savings are achieved with the *Sim-Based* method because no high-level information is available for the input signals, which are thus assumed to have a static probability of 0.5. Since these circuits are fairly large, they do not reach a computational kernel with the unbiased patterns provided in simulation. In contrast, the structural method gives better results because several structural changes can be applied in the circuits without altering too much their original functionality.

Summarizing, the exact symbolic algorithm produces best results, but it does not scale well to large circuits. The approximate symbolic approach can be applied to slightly larger circuits at the price of marginally inferior power savings. The simulation-based method does not produce good savings for large circuits with unbiased random patterns, but it works well for small circuits where a small number of patterns brings the circuit into a kernel and for large circuits with biased input patterns that speed up the convergence to a computational kernel. The structural technique tends to produce good results for large circuits, but it has insufficiently fine granularity on small circuits.

IX. CONCLUSION

We have described a technique for generating low-power sequential RTL components based on the concept of computational kernel, a small block of logic whose behavior mimics that of the original specification for most of the operation time.

Kernel extraction is at the basis for the success of our method. Therefore, we have proposed a number of algorithmic options for performing this step. Besides an exact symbolic approach that is only applicable to small components, we have introduced approximate solutions that have practical interest. In particular, we have presented a simulation-based technique that is suitable to the optimization of components for which only a description of the I/O behavior is available and a structural method that performs kernel extraction by iteratively modifying the gate-level implementation of a circuit.

The viability and effectiveness of the power optimization technique based on kernel extraction is proved by the results we have obtained on a large set of benchmark examples having different characteristics and nature. Average power savings are around 50% for the symbolic solution, while they are around 30% when the approximate extraction algorithms are used.

APPENDIX
DSFF IMPLEMENTATIONS

DSFFs are critical for a power-efficient hardware implementation of computational kernels. In Section IV, we simply described the functional view of these components. In this

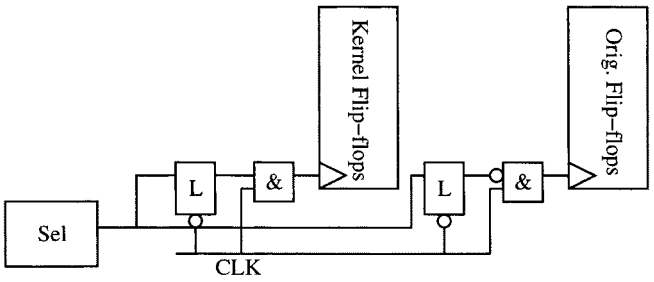


Fig. 8. Computational kernel clustered gated-clock architecture.

appendix, we propose two power-efficient implementations of DSFFs, namely, a *clustered gated-clock* implementation and a low-overhead DSFF library cell.

The main issue with a straightforward implementation of DSFFs, as shown in Fig. 3, is that it doubles the clock load with respect to standard flip-flops. In this case, the computational kernel architecture would be marred by a severe clock loading overhead that scales with the number of state variables in the original sequential circuit. Fortunately, we can overcome this limitation by observing that the computational kernel operates in a mutual exclusive fashion with the original circuit. Hence, we can decompose all DSFFs in standard flip-flop pairs and clock the two flip-flops with mutually exclusive gated-clocks. All flip-flops feeding data to the kernel are clustered together and receive the same clock signal. Similarly, the flip-flops feeding the original circuit are clustered as well under the same gated-clock domain. This clustered gated-clock implementation is shown in Fig. 8. Notice that the two phases of the selection function control clock-gating of the two clusters. Observe also that no multiplexers are needed for recirculating data as in Fig. 3.

The clustered implementation virtually eliminates any clock power overhead because only one cluster at a time receives the clock signal. However, it still imposes some clock routing overhead because it does require routing of two gated-clock signals and skew control on clock-gating logic and wiring. When clock routing and skew are severely constrained or cannot be tightly controlled, a cell-based implementation that does not require clock gating may then be preferable. In this case, we need to design a DSFF library cell with minimal area, speed, and clock load overhead with respect to the corresponding standard flip-flop.

The schematic of a DSFF cell satisfying the requirements listed above is shown in Fig. 9. This implementation is derived from the single-clock flip-flop introduced by Yuan and Svensson [34], which has been shown to be ideally suited for low-power applications [35]. The original D flip-flop described in [34] has a master-slave structure, where the master latch samples the input value and steers the bistable slave. The distinctive characteristics of this flip-flop is its sense-amplifier structure, where the clock signal is used to control virtual ground and power supply that activate the slave and the master, respectively.

We exploit the sense-amplifier architecture to design a DSFF with *exactly the same clock load* as the standard D flip-flop. In our implementation, the clocked PMOS transistor T12 controls

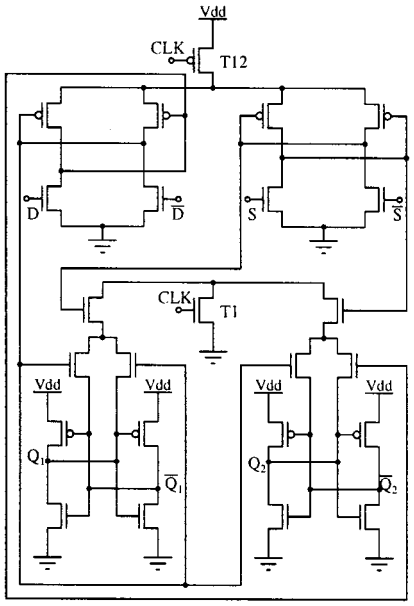


Fig. 9. Implementation of a DSFF library cell.

the activation of two sampling latches: the master data latch and the selection signal latch. Similarly, the clocked NMOS transistor T1 controls the activation of two bistable slaves. Notice that only one of the slaves is written at any given clock cycle because writing of the slaves is controlled by the two phases of the latched selection signal.

Regarding speed overhead, the DSFF is marginally slower than its standard D flip-flop counterpart, mainly because of the increased resistance of the pull down virtual ground path in the slaves. Accurate HSPICE simulation of a DSFF implementation in $\lambda = 0.5$-μm MOSIS CMOS technology showed a clock-to-output delay increase of less than 7% for a DSFF loaded by a single inverter. Area overhead is also well controlled, because the DSFF occupies sightly less than twice the area of the standard D flip-flop.

ACKNOWLEDGMENT

The authors would like to thank F. Somenzi for providing some of the examples used for the experiments and A. Macii and R. Scarsi for running the experiments.

REFERENCES

[1] G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Markovian analysis of large finite state machines," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 1479–1493, Dec. 1996.

[2] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, pp. 79–85, Aug. 1986.

[3] C.-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. M. Despain, and B. Lin, "Power estimation methods for sequential logic circuits," *IEEE Trans. VLSI Syst.*, vol. 3, pp. 404–416, Sept. 1995.

[4] G. D. Hachtel and F. Somenzi, *Algorithms for Logic Synthesis and Verification*. Norwell, MA: Kluwer, 1996.

[5] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill, "Sequential circuit verification using symbolic model checking," in *Proc. ACM/IEEE Design Automation Conf.*, Orlando, FL, June 1990, pp. 46–51.

[6] O. Coudert and J. C. Madre, "A unified framework for the formal verification of sequential circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1990, pp. 126–129.

[7] H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Implicit enumeration of finite state machines using bdds," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1990, pp. 130–133.

[8] H. Cho, G. D. Hachtel, S. W. Jeong, B. Plessier, E. Schwarz, and F. Somenzi, "ATPG aspects of FSM verification," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1990, pp. 134–137.

[9] M. Pedram, "Power minimization in ic design: principles and applications," *ACM Trans. Design Automat. Electron. Syst.*, vol. 1, no. 1, pp. 3–56, 1996.

[10] E. Macii, M. Pedram, and F. Somenzi, "High-level power modeling, estimation, and optimization," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 1061–1079, Nov. 1998.

[11] S. H. Chow, Y. C. Ho, T. Hwang, and C. L. Liu, "Lower power realization of finite state machines—a decomposition approach," *ACM Trans. Design Automat. Electron. Syst.*, vol. 1, no. 3, pp. 315–340, July 1996.

[12] S. Roy, P. Banerjee, and M. Sarrafzadeh, "Partitioning sequential circuits for low power," in *Proc. Int. Conf. VLSI Design*, Chennai, India, Jan. 1998, pp. 212–217.

[13] L. Benini, F. Vermeulen, and G. De Micheli, "Finite state machine partitioning for low power consumption," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 2, Monterey, CA, May 1998, pp. 5–8.

[14] J. Monteiro and A. Oliveira, "Finite state machine decomposition for low power," in *Proc. ACM/IEEE Design Automation Conf.*, San Francisco, CA, June 1998, pp. 763–768.

[15] L. Benini, P. Vuillod, C. Coelho, and G. De Micheli, "Synthesis of low-power selectively clocked systems from high-level specification," in *Proc. IEEE Int. Symp. System Synthesis*, La Jolla, CA, Oct. 1996, pp. 57–62.

[16] E. Hwang, F. Vahid, and Y.-C. Hsu, "FSMD functional partitioning for low power," in *Proc. IEEE Design Automation and Test Eur.*, Munich, Germany, Mar. 1999, pp. 22–28.

[17] G. Lakshminarayana, A. Raghunathan, K. S. Khouri, N. K. Jha, and S. Dey, "Common-case computation: a high-level tecnique for power and performance optimization," in *Proc. ACM/IEEE Design Automation Conf.*, New Orleans, LA, June 1999, pp. 56–61.

[18] L. Benini, P. Siegel, and G. De Micheli, "Automatic synthesis of gated clocks for power reduction in sequential circuits," *IEEE Des. Test Comput.*, vol. 11, pp. 32–40, Dec. 1994.

[19] L. Benini and G. De Micheli, "Transformation and synthesis of FSMs for low power gated clock implementation," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 630–643, June 1996.

[20] M. Onishi, A. Yamada, H. Noda, and T. Kambe, "A method of redundant clocking detection and power reduction at rt level design," in *Proc. IEEE Int. Symp. Low Power Electronics and Design*, Monterey, CA, Aug. 1997, pp. 131–136.

[21] H. Kapadia, L. Benini, and G. De Micheli, "Reducing switching activity on datapath buses with control-signal gating," *IEEE J. Solid-State Circuits*, vol. 34, pp. 405–414, Mar. 1999.

[22] L. Benini, G. De Micheli, E. Macii, M. Poncino, and R. Scarsi, "Symbolic synthesis of clock-gating logic for power optimization of synchronous controllers," *ACM Trans. Design Automat. Electron. Syst.*, vol. 4, no. 4, pp. 351–375, Oct. 1999.

[23] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou, "Precomputation-based sequential logic optimization for low power," *IEEE Trans. VLSI Syst.*, vol. 2, pp. 426–436, Dec. 1994.

[24] J. Monteiro, S. Devadas, and A. Ghosh, "Sequential logic optimization for low power using input-disabling precomputation architectures," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 279–284, Mar. 1998.

[25] R. I. Bahar, C. Gaona, E. Frohm, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic decision diagrams and their applications," *Formal Methods Syst. Des.*, vol. 10, no. 2/3, pp. 171–206, Apr. 1997.

[26] K. Ravi and F. Somenzi, "High-density reachability analysis," in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 1995, pp. 154–158.

[27] E. M. Sentovich, K. J. Singh, C. W. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuits design using synthesis and optimization," in *Proc. IEEE Int. Conf. Computer Design*, Cambridge, MA, Oct. 1992, pp. 328–333.

[28] F. Somenzi, "CUDD: University of Colorado Decision Diagram Package, Release 2.3.0," Dept. ECE, Univ. Colorado, Boulder, CO, 1998.

[29] F. Brglez, D. Bryan, and K. Koźmiński, "Combinational profiles of sequential benchmark circuits," in *Proc. IEEE Int. Symp. Circuits and Systems*, Portland, OR, May 1989, pp. 1929–1934.

[30] J. Monteiro, A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of average switching activity in combinational logic circuits using symbolic simulation," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 121–127, Jan. 1997.

[31] A. Salz and M. Horowitz, "IRSIM: an incremental mos switch-level simulator," in *Proc. ACM/IEEE Design Automation Conf.*, Las Vegas, NV, June 1989, pp. 173–178.

[32] G. Berry and H. Touati, "Optimized controller synthesis using esterel," in *Proc. ACM/IEEE Int. Workshop Logic Synthesis*, Lake Tahoe, CA, May 1993.

[33] S. -I. Minato, "Generation of BDDs from hardware algorithm description," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 1996, pp. 644–649.

[34] J. Yuan and C. Svensson, "New single-clock CMOS latches and flip-flops with improved speed and power savings," *IEEE J. Solid-State Circuits*, vol. 32, pp. 62–69, Jan. 1997.

[35] C. Svensson and J. Yuan, "Latches and flip-flops for low power systems," in *Low Power CMOS Design*, A. Chandrakasan and R. Bridersen, Eds. Piscataway, NJ: IEEE Press, 1998, pp. 233–238.

Luca Benini (S'94–M'97) received the B.S. degree (summa cum laude) in electrical engineering from the University of Bologna, Bologna, Italy, in 1991, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1994 and 1997, respectively.

Since 1998, he has been an Assistant Professor in the Department of Electronics and Computer Science with the University of Bologna. He also holds Visiting Researcher positions at Stanford University and the Hewlett-Packard Laboratories, Palo Alto, CA. He is a member of the organizing commitee of the International Symposium on Low Power Design and a member of the technical program committee for several technical conferences, including Design and Test in Europe, the International Symposium on Low Power Design, and the Symposiom on Hardware-Software Codesign. He has authored or coauthored more than 120 papers in international journals and conferences, a book, and several book chapters. His research interests include all aspects of computer-aided design of digital circuits, with special emphasis on low-power applications, and in the design of portable systems.

Giovanni De Micheli (S'70–M'83–SM'89–F'94) received the Nucl. Eng. degree from the Politecnico di Milano, Milan, Italy, in 1979 and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 1980 and 1983, respectively.

He is a Professor of Electrical Engineering and Computer Science at Stanford University, Stanford, CA. He is member of the technical advisory board of several EDA companies, including Magma Design Automation, Coware and Aplus Design Technologies and was a member of the technical advisory board of Ambit Design Systems.He is author of *Synthesis and Optimization of Digital Circuits* (New York: McGraw-Hill, 1994) and coauthor or coeditor of co-author and/or co-editor of four other books and of over 200 technical papers. His research interests include several aspects of design technologies for integrated circuits and systems, with particular emphasis on synthesis, system-level design, hardware/software codesign, and low-power design.

Dr. De Micheli is a Fellow of ACM. He received the Golden Jubilee Medal for outstanding contributions to the IEEE Circuits and Systems Society in 2000. He received the 1987 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS Best Paper Award and two Best Paper Awards at the Design Automation Conference in 1983 and in 1993.

He is Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. He was Vice President (for publications) of the IEEE Circuits and Systems Society from 1999 to 2000. He was the Program Chair and General Chair of the Design Automation Conference in 1996, 1997, and 2000, respectively, and was also the Program and General Chair of the International Conference on Computer Design in 1988 and 1989, respectively.

Antonio Lioy (S'80–M'89) received the Dr. Eng. degree in electrical engineering and the Ph. D. degree in computer engineering from the Politecnico di Torino, Torino, Italy, in 1982 and 1987, respectively.

He was a Research Assistant from 1987 to 1989 and then an Assistant Professor from 1990 to 1992 with the Politecnico di Torino. In 1993, he became an Associate Professor at the Universitá di Parma, Parma, Italy. He is currently an Associate Professor with the Politecnico di Torino. His research interests include simulation and testing of digital circuits and systems, as well as advanced networking technologies and computer security.

Enrico Macii (M'92–SM'01) received the Dr. Eng. degree in electrical engineering from the Politecnico di Torino, Torino, Italy, in 1990, the Dr. Sc. degree in computer science from the Universitá di Torino, Torino, Italy, in 1991, and the Ph. D. degree in computer engineering from the Politecnico di Torino in 1995.

From 1991 through 1994, he was an Adjunct Faculty Member with the University of Colorado, Boulder. He is currenlty an Associate Professor with the Politecnico di Torino. He has authored or coauthored over 200 journal and conference papers. His research interests include several aspects of computer-aided design of integrated circuits and systems, with particular emphasis on synthesis, optimization, and formal verification.

Dr. Macii received the Best Paper Award at the 1996 IEEE European Design Automation Conference. He was the Technical Program Co-Chair of the IEEE Alessandro Volta Memorial Workshop on Low-Power Design in 1999 and the Technical Program Co-Chair and the General Chair of the ACM/IEEE International Symposium on Low Power Electronics and Design in 2000 and 2001, respectively. He is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS and an Associate Editor of the *ACM Transactions on Design Automation of Electronic Systems*.

Giuseppe Odasso received the Dr. Eng. degree in electrical engineering and the Ph. D. degree in computer engineering from the Politecnico di Torino, Torino, Italy, in 1994 and 2001, respectively.

His research interests include synthesis, verification, simulation, and testing of digital circuits, with special emphasis on low-power and high-performance systems.

Massimo Poncino (M'97) received the Dr. Eng. degree in electrical engineering and the Ph.D. degree in computer engineering from the Politecnico di Torino, Torino, Italy, in 1989 and 1993, respectively.

From 1993 to 1994, he was a Visiting Faculty Member with the University of Colorado, Boulder. He is currently an Assistant Professor with the Politecnico di Torino. His research interests include synthesis, verification, simulation, and testing of digital circuits and systems.