Kernel-Based Power Optimization of RTL Components: Exact and Approximate Extraction Algorithms

L. Benini * G. De Micheli # E. Macii ‡ G. Odasso ‡ M. Poncino ‡

* Università di Bologna
Bologna, ITALY 40136

Stanford University
Stanford, CA 94305

‡ Politecnico di Torino
Torino, ITALY 10129

Abstract

Sequential logic optimization based on the extraction of computational kernels has proved to be very promising when the target is power minimization. Efficient extraction of the kernels is at the basis of the optimization paradigm; the extraction procedures proposed so far exploit common logic synthesis transformations, and thus assume the availability of a gate-level description of the circuit being optimized. In this paper we present exact and approximate algorithms for the automatic extraction of computational kernels directly from the functional specification of a RTL component. We show the effectiveness of such algorithms by reporting the results of an extensive experimentation we have carried out on a large set of standard benchmarks, as well as on some designs with known functionality.

1 Introduction

When designing a complex sequential circuit, engineers must consider not only the basic, typical behavior, but also a large number of unusual operating conditions. In many cases the number and the nature of these conditions (also known as *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. Then, 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 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 patterns are generated.

Very often designers go through a number of time-consuming refinement steps trying to obtain a circuit that is fast and power-efficient under typical input stimuli, but still operates correctly in unusual conditions. In [2] we have proved that this type of optimization can be efficiently automated by exploiting the concept of *computational kernel*. According to [2], we call computational kernel of a sequential circuit a logic block that mimics the typical behavior of the original network. Usually, such block is much smaller, faster, and less power consuming than the circuit it is extracted from. Nevertheless, it can replace the original network for a large fraction of the operation time.

DAC 99, New Orleans, Louisiana

Fundamental for a successful application of the kernel-based optimization paradigm is the procedure adopted for kernel extraction. In [2] we have proposed an algorithm that relies on iterative simplification of the original network by redundancy removal; because of its intrinsic topological nature, this approach is suitable for optimizations being performed at the logic level, that is, in cases where a gate-level netlist of the design being optimized does exist.

In this work, we raise the level of abstraction at which the kernel-based optimization strategy can be exploited. More specifically, we show how RTL components for which only a functional specification (i.e., the state transition graph (STG) represented in either an explicit or an implicit form) is available can be fruitfully optimized using computational kernels. We do so by first providing a precise definition of computational kernel that was only informally introduced in [2]. We then present an innovative technique for computational kernel extraction directly from the functional specification of a RTL module. Given the STG specification, the proposed algorithm exactly calculates the kernel through symbolic procedures similar to those employed for FSM reachability analysis.

Unfortunately, as the size of the STG increases, exact solutions are no longer applicable for both memory and time reasons. Approximate variants of the exact method are thus required in order to extend its domain of applicability. We propose two modifications to the basic procedure. The first one simply replaces the most computationally expensive step of the exact algorithm (namely, the probabilistic analysis of the STG) with an approximate version. The second solution is based on RTL simulation of a given (random or user-provided) stream. In this case, full, symbolic state probability computation is by-passed, and the set of states belonging to the kernel is determined directly from the simulation traces. This approach does not suffer from potential computational blow-ups as for the exact solution, and enables kernel extraction for much larger components. Although in principle even random pattern simulation can be used for kernel computation, the effectiveness of the simulation-based approach clearly depends on the availability of a meaningful stream, that is, one that well represents the typical operational context of the circuit.

Both the exact and the approximate algorithms primarily target the extraction of a set of states with high steady-state occupation probability. This is in contrast with the optimization procedure presented in [2], whose main limitation is the lack of control on the occupation probability of the kernel.

We show the usefulness of computational kernel extraction by applying it to the problem of reducing the power dissipated by a sequential circuit. Experimental results, obtained on a large set of benchmarks, demonstrate the validity of the proposed RTL kernel extraction procedures.

2 Computational Kernels: Basic Theory

We provide the definition of computational kernel of a sequential design by referring to the finite state machine which models the behavior of the circuit. A *finite state machine* (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 $t \in S$ with output $z \in Z$.

Given a finite state machine, $M = (X, Z, S, S^0, R)$, modeling the behavior of a sequential circuit, and given a probability threshold, p, we define the p-order *computational kernel* of M, denoted as M_p, as the following finite state machine:

$$M_p = (X, Z, S_p, S_p^0, R_p)$$

where $S_p = \{s \in S : Prob(s) \geq p\}$ is 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 M_p is defined as:

$$R_p(x,s,t,z) = \mathcal{S}_p(s) \cdot R(x,s,t,z)$$

where $\mathcal{S}_p(s)$ is the characteristic function of set S_p. 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, $\mathcal{S}_p(s)'$ is the *don't care* set for the next state and output functions that can be extracted from R_p. A key characteristics of computational kernels is their probability $Prob(R_p)$. This quantity is defined as the cumulative occupation probability of the states in S_p.

As an example, consider the simple FSM of Figure 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 kernel of the FSM is depicted in Figure 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 of R_p is $Prob(R_p) = 0.29 + 0.25 + 0.32 = 0.86$.

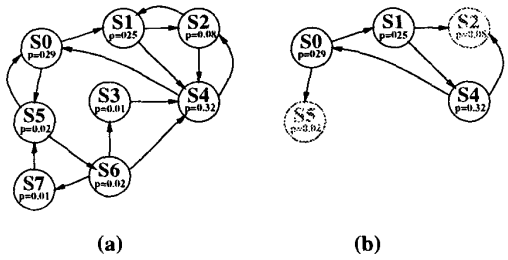

Figure 1: Moore-Type FSM and Corresponding Kernel.

In this work, we focus on efficient algorithms for extracting the kernel from a functional specification of a RTL component. The extraction requires 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, the main prerequisite for kernel computation.

Kernel extraction is a relatively straight-forward process for small FSMs where exhaustive state and input pattern enumeration is possible. However, most sequential circuits have large state and input spaces that cannot be enumerated in a reasonable amount of time. For medium-size FSMs, the computational kernel can be determined through BDD-based operations by exploiting well-established technology developed in the context of FSM reachability analysis [3, 4, 5, 6, 7, 8, 9]. The exact algorithm we present in Section 3 serves this purpose.

Unfortunately, large sequential components are often beyond the capabilities of the most powerful techniques based on symbolic manipulations (especially those for FSM probabilistic analysis). For this reason, in Section 4 we propose an approximate kernel extraction algorithm that scales well with circuit size.

2.1 Kernel-Based Architecture

The computational kernel of a sequential design, once extracted from a given circuit specification, provides us with an extremely powerful device to be used for various types of logic optimization. Given a sequential circuit with the conventional topology of Figure 2-a, the paradigm proposed in [2] for improving its quality with respect to a given cost function (e.g., power dissipation, timing) is based on the architecture shown in Figure 2-b.

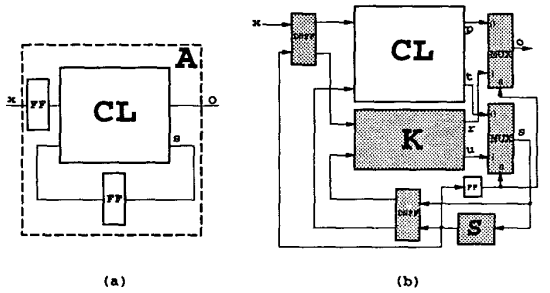

Figure 2: Kernel-Based Optimized Architecture.

The basic elements of the architecture are: The combinational portion of the original circuit (block CL), the computational kernel (block K), the selector function (block S), the double-state flip-flops ($DSFF$), and the output multiplexors (MUX). It is worth noting that the block diagram of Figure 2 bears some resemblance with several low-power architectures proposed in the past, e.g., precomputation [10, 11], gated-clocks [12, 13, 14], decomposed FSMs [15, 16]. The interested reader may refer to [2] for an analysis of the relationship between such architectures and computational kernels, and to [17, 18] for a more comprehensive discussion on gate/RTL power management techniques.

The computational kernel can be seen as a "dense" implementation of the circuit it has been extracted from. In other terms, K implements the core functions of the original circuit, and because of its reduced complexity, it usually implements such functions in a faster and more efficient way.

The purpose of selector function S 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, S examines the values of the next-state outputs at clock cycle n. If the output and next-state values in cycle $n+1$ can be computed by the kernel K, then S takes on the value 1. Otherwise, it takes on the value 0. The value of S is fed to a flip-flop, whose output is connected to the MUXes that select which block produces the output and the next-state. The optimized implementation is functionally equivalent to the original one.

The scheme of Figure 2 is just one among several possible architectures. The peculiar feature of the proposed solution concerns the topology of the selection logic. In particular, the choice of having a selection function that only depends on the next-state outputs is dictated by the need of obtaining a small implementation. Reducing the size of the support of S, i.e., not including the primary inputs, is one way of pursuing this objective.
The sequential elements in Figure 2 indicated as *DSFFs* are called *dual-state* flip-flops [2], and they replace the flip-flops in the original design. A dual-state flip-flop has two outputs, one additional control signal and contains two state variables. Depending on the value of the control signal, only one state variable (and the corresponding output) is updated, while the other is kept frozen at its last value. Dual-state flip-flops are functionally equivalent to two flip-flops with two multiplexors, but they can be more effectively designed at the transistor-level and instantiated as an atomic cell.

2.2 Exact Gate-Level Kernel Extraction

The approximate kernel extraction algorithm presented in [2] targets components specified at the gate-level. It is based on redundancy removal, and it can be summarized as follows: First, the signal probability of all nodes in the the next-state and output logic of the target circuit is computed. Then, nodes are selected and stuck either to zero or to one. The next-state and output logic is then simplified by propagating the constant value (i.e., by removing the redundancies introduced in the circuit). Node selection and simplification are iterated until a stopping criterion is met. Node selection is driven by a heuristic criterion that privileges nodes with high load and switching activity. The stopping test is based on the estimated power savings.
One limitation of this algorithm is that it does not allow to control kernel probability. When selecting nodes, the extraction procedure does not test the impact of node removal on the behavior of the circuit. Even eliminating a single node may cause drastic changes in such behavior. As a result, the probability for the modified circuit to work as the original one (i.e., the probability of the kernel) may decrease very rapidly with the number of removed nodes. Another issue is that redundancy removal operates on gate-level netlists, hence the approach is not suitable for macros for which only a functional specification is available. Both limitations are addressed by the exact and approximate algorithms presented in the following sections.

3 Exact RTL Kernel Extraction

The exact kernel extraction algorithm is an almost direct implementation of the definition given in Section 2 that leverages state-of-the-art implicit BDD-based algorithms for the manipulation of sets of states and Boolean functions. The pseudo-code of the exact algorithm is reported in Figure 3.

```
procedure ExtractKernel (D(x), R(s, x, z, t), p) {
1   P(s) = Markov(D(x), R(s, x, z, t));
2   S_p(s) = Threshold(P(s), p);
3   T(s, x, t) = ∃_z R(x, s, t, z);
4   O(s, x, z) = ∃_t R(x, s, t, z);
5   T^p(s, x, t) = S_p(s) · T(s, x, t);
6   O^p(s, x, z) = S_p(s) · O(s, x, z);
7   Δ^p(x, s) = ((∃_{t≠t_1} T^p(s, x, t))_{t'_1}, ..., (∃_{t≠t_n} T^p(s, x, t))_{t'_n});
8   Λ^p(x, s) = ((∃_{z≠z_1} O^p(s, x, z))_{z'_1}, ..., (∃_{z≠z_m} O^p(s, x, z))_{z'_m});
9   Kernel(s, x) = (Δ^p(x, s), Λ^p(x, s));
10  return (Kernel, S_p(s));
}
```

Figure 3: Symbolic Kernel Extraction Algorithm.

The inputs to the procedure are:

- A multi-terminal BDD (also known as ADD [19]) 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.
- A BDD representing the global relation $R(s, x, z, t)$ of the target circuit.
- A probability threshold p. According to the definition of Section 2, all states with steady-state occupation probability greater than p belong to the state set of kernel M_p.

The first step is to compute steady-state probabilities for all states $s \in S$. This is accomplished in Line 1 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 states probabilities. The state set S is then pruned in Line 2 using the `Threshold` operator. Such operator takes, as inputs, the ADD of the state probabilities and the probability threshold p, and returns a BDD that represents the characteristic function $S_p(s)$ of the state set S_p of the kernel.
To obtain the next-state and output functions of the kernel, we first extract the transition relation $T(s, x, t)$ and the output relation $O(s, x, t)$ from the global relation of the original circuit (Lines 3 and 4) and we compute their conjunction with S_p (Lines 5 and 6). This operation is the key step of the algorithm. Its intuitive meaning is that the behavior of the computational kernel is specified only when the current state belongs to S_p.
The result of the conjunction is illustrated in Figure 4. In the picture on the left, the light grey area identifies the states for which the occupation probability is greater than or equal to p. In the picture on the right, the outer shaded region (dark grey) represents states not belonging to S_p that are valid next states for the next-state function of M_p, because they are associated with present states that do belong to S_p. In other words, there is a "corolla" consisting of all the states that are 1-step reachable from states $s \in S_p$ for which the behavior of the kernel must be specified and consistent with the original one.

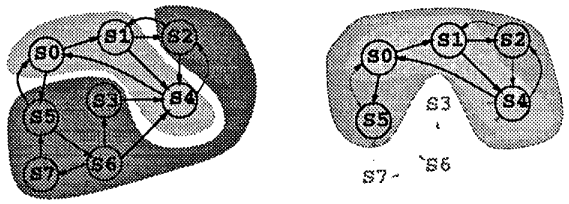

Figure 4: Example of Kernel Extraction.

Given T^p and O^p, it is relatively easy to extract the next state and output functions of the computational kernel (Lines 7 and 8) by existential quantification followed by cofactoring (in the pseudo-code, we assume that there are n state variables and m output variables). The procedure returns Δ^p (the next-state function) and Λ^p (the output function) that completely specify the functionality of block K in Figure 2-b, and function S_p that specifies the ON-set of function S in Figure 2-b. Notice that S'_p represents the *don't care* set that can be used to optimize the implementation of K.

Circuit	*In*	*Out*	*FF*	*Gates*	*Delay*	*Power*			*Prob(S)*	*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%	

Table 1: Experimental Results for the Exact Extraction Procedure.

3.1 Experimental Results

We have implemented the exact extraction algorithm as an extension of SIS [20], using CUDD [21] as the underlying BDD package. Experiments have been run on a DEC AXP 1000/400 with 256MB of memory.

We have applied the algorithm of Figure 3 to the examples taken from the Iscas'89 sequential suite [22] 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 library containing two to four-input NAND and NOR gates, inverters and buffers with three different drive strengths, and a flip-flop.

Table 1 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 *Prob(S)* tells the probability of the selector function S to be 1, that is, 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 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, expressed in seconds, required by the complete optimization procedure to terminate.

Power estimates within the kernel extraction procedure have been computed using symbolic simulation, while those for the initial and final circuits have been determined using the Irsim switch-level simulator [23].

Results are very promising. An average power savings of approximately 51% has been achieved, with peaks of more than 80%. Notice that some circuits included in the original Iscas'89 set are missing from the table, namely s208, s420, s838, s1196, and s1238. These circuits have the peculiar property that all states are equiprobable; this uniform probability distribution clearly prevents the application of the kernel-based optimization paradigm, which is most effective in cases where a small kernel has a high cumulative occupation probability.

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(s)* is very high (e.g., benchmarks s820, s1196), we have observed that the computational kernels typically include very few, highly-probable states.

Concerning the speed of the optimized circuits, although the delay of the selection function S always adds up to the largest delay between combinational blocks CL and K, the overall timing penalty is quite limited (13% on average). This is because the logic of module CL is optimized using function S as external *don't care set*, thus its delay usually reduces with respect to the original circuit.

On the other hand, as it could be easily predicted, the impact of the transformation on circuit area 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.

4 Approximate RTL Kernel Extraction

The results of Section 3.1 show that the kernel-based paradigm can provide sizable power reductions. On the other hand, the applicability of the exact method described in Section 3 is limited to instances of small-to-medium circuits. Only in these cases, in fact, the state occupation probabilities can be calculated using the symbolic method of [1]. For larger circuits, the computation simply becomes too memory-intensive. Another limitation of the exact RTL kernel extraction 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 a more realistic setting, the size of the circuit can be such that Markovian analysis is infeasible, and knowledge of input pattern distribution is limited to the specification of a set of patterns that represent typical operating conditions under which the system is expected to run. Under these assumptions, we need a technique that can extract computational kernels for large circuits based on the available, limited knowledge of input probability distribution.

We have identified two simple, yet effective ways of getting around the probability calculation problem. The first one consists of simply replacing the `Markov` routine in Line 1 of Figure 3 with the implementation of the algorithm for approximate FSM probabilistic analysis of [24].

Circuit	Power Savings		Prob(S)		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%	45%	0.89	0.92	51%	44%	14%	14%	29	38
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%	10%	0.42	0.45	60%	63%	16%	19%	193	71
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%	41%			34%	38%	13%	16%		

Table 2: Experimental Results: Comparison of Exact and Approximate Extraction Procedures.

Circuit	In	Out	FF	Gates	Delay	Power			Prob(S)	Area Overhead	Delay Overhead	CPU Time	Markov Type
						Ref	Opt	Sav					
s1423	17	5	74	602	73.6	1809	368	80%	0.74	70%	18%	2292	Markov
s1512	29	21	57	475	42.7	155	95	39%	0.27	26%	9%	1840	Markov
Boltzmann	7	21	91	367	15.7	134	50	63%	0.42	45%	13%	674	Simul
FifoWriteCntr	2	2	17	141	16.8	57	42	27%	0.62	58%	25%	93	Simul
Gcd16	33	17	50	1197	32.6	128	83	35%	0.19	36%	19%	578	Simul
Iqc	10	15	36	1169	23.9	445	220	51%	0.90	21%	18%	492	Simul
Lan	10	8	19	215	19.4	107	99	7%	0.84	44%	19%	116	Simul
Radio	5	16	16	181	14.6	110	77	30%	0.98	35%	23%	76	Simul
Watch	4	16	11	108	9.1	67	51	24%	0.98	58%	18%	59	Simul
Avg.								40%		44%	18%		

Table 3: Experimental Results for the Approximate Extraction Procedures.

The second procedure 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 circuit 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 states is taken as the ON-set of characteristic function $\mathcal{S}_p$, which will be used as a specification for block S of Figure 2-b.

For both solutions, the next-state and output functions of K can be constructed by exploiting the procedure of Figure 3, in which the ADD $P(s)$ constructed through approximate Markovian analysis [24] or from simulation replaces the one built implicitly by procedure `Markov`.

4.1 Experimental Results

We have implemented the two variants, discussed above, of the approximate kernel extraction procedure. In the sequel we present the results of two sets of experiments. In the first one we compare the data of the approximate algorithm which is based on the Markovian analysis of [24] to those of the exact procedure; this is to measure the quality of the approximation. The results are shown in Table 2.

As expected, introducing an approximation in the calculation of the state occupation probabilities has produced a degradation in the power savings with respect to those obtained using the exact method. The amount of degradation is however acceptable, if we consider the higher speed and the lower complexity of the approximate method. Then, we can claim with reasonable confidence that the approximate approach provides us with a viable solution for the optimization of designs for which the use of the exact method is either not affordable or infeasible.

In a second set of experiments, we have run the two approximate extraction procedures on some examples for which the exact method is inapplicable. In particular, we have chosen some of the large Iscas'89 benchmarks (addendum included), as well as some RTL circuits taken from [25] for which only a functional description (specified in Esterel) was available. The BDDs for the output and next-state functions for the latter circuits have been constructed directly from the Esterel source code using a procedure similar to that described in [26].

Table 3 reports the data of the experiments. Notice that the indication of which algorithm has been used for collecting the state occupation probability information is reported in column *Markov Type* of such table.

For circuits whose functionality is known, the method based on simulation works well; this is because kernel identification and extraction has been done by feeding the circuits with meaningful and realistic input traces (each of which consisted of approximately 50.000 binary patterns). This solution is then preferable to the approximate Markovian analysis approach, since execution times are much lower with optimization results of comparable quality.

For circuits whose functionality is unknown (as in the case of the Iscas'89 benchmarks), the only way of providing them with an input trace is through random generation. In this case, the results that we have obtained are much less appealing (basically power savings were negligible, since the size of both the kernel and the selection function was considerable). On the other hand, resorting to the approximate Markovian analysis has produced much higher savings (these are the data reported in Table 3). It should also be observed that both circuits s1423 and s1512 were used in [2] to benchmark the synthesis-based kernel extraction procedure proposed in that paper. Although the reference circuits were different, the achieved power savings were much lower (27% and 29%, respectively) than those reported in Table 3. This was somehow expected, since the extraction algorithms introduced in this paper allow a finer tuning of the generated kernel.

5 Conclusions

In this paper we have presented methods for the automatic extraction of computational kernels starting from a RT-level description of a sequential component. Solutions relying on exact and approximate probabilistic analysis of the STG representations have been proposed

We have shown the effectiveness of the new extraction algorithms in the context of sequential synthesis for low power by reporting the results of an extensive experimentation we have carried out on a large set of standard benchmarks, as well as on some designs with known functionality.

Acknowledgments

We wish to thank Fabio Somenzi for providing us with some of the examples we have used for the experiments. The help of Alberto Macii and Riccardo Scarsi in running the experiments is also acknowledged.

References

[1] G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi, "Markovian Analysis of Large Finite State Machines," IEEE Transactions on CAD, Vol. 15, No. 12, pp. 1479-1493, December 1996.

[2] L. Benini, G. De Micheli, A. Lioy, E. Macii, G. Odasso, M. Poncino, "Computational Kernels and their Application to Sequential Power Optimization", DAC-35: ACM/IEEE 1998 Design Automation Conference, pp. 764-769, San Francisco, CA, June 1998.

[3] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, "Sequential Circuit Verification Using Symbolic Model Checking," DAC-27: ACM/IEEE Design Automation Conference, pp. 46-51, Orlando, FL, June 1990.

[4] O. Coudert, J. C. Madre, "A Unified Framework for the Formal Verification of Sequential Circuits," ICCAD-90: IEEE International Conference on Computer-Aided Design, pp. 126-129, Santa Clara, CA, November 1990.

[5] H. Touati, H. Savoj, B. Lin, R. K. Brayton, A. Sangiovanni-Vincentelli, "Implicit Enumeration of Finite State Machines Using BDDs," ICCAD-90: IEEE International Conference on Computer-Aided Design, pp. 130-133, Santa Clara, CA, November 1990.

[6] H. Cho, G. D. Hachtel, S. W. Jeong, B. Plessier, E. Schwars, F. Somenzi, "ATPG Aspects of FSM Verification," ICCAD-90: IEEE International Conference on Computer-Aided Design, pp. 134-137, Santa Clara, CA, November 1990.

[7] K. Ravi, F. Somenzi, "High-Density Reachability Analysis," ICCAD-95: IEEE/ACM International Conference on Computer-Aided Design, pp. 154-158, San Jose, CA, November 1995.

[8] H. Cho, G. D. Hachtel, E. Macii, B. Plessier, F. Somenzi, "Algorithms for Approximate FSM Traversal Based on State Space Decomposition,", IEEE Transactions on CAD, Vol. 15, No. 12, pp. 1465-1478, December 1996.

[9] H. Cho, G. D. Hachtel, E. Macii, M. Poncino, F. Somenzi, "Automatic State Space Decomposition for Approximate FSM Traversal Based on Circuit Structural Analysis," IEEE Transactions on CAD, Vol. 15, No. 12, pp. 1451-1464, December 1996.

[10] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, M. Papaefthymiou, "Precomputation-Based Sequential Logic Optimization for Low Power," IEEE Transactions on VLSI Systems, Vol. 2, No. 4, pp. 426-436, December 1994.

[11] J. Monteiro, S. Devadas, A. Ghosh, "Sequential Logic Optimization for Low Power Using Input-Disabling Precomputation Architectures," IEEE Transactions on CAD, Vol. 17, No. 3, pp. 279-284, March 1998.

[12] L. Benini, P. Siegel, G. De Micheli, "Automatic Synthesis of Gated Clocks for Power Reduction in Sequential Circuits," IEEE Design and Test of Computers, Vol. 11, No. 4, pp. 32-40, December 1994.

[13] L. Benini, G. De Micheli, "Transformation and Synthesis of FSMs for Low Power Gated Clock Implementation," IEEE Transactions on CAD, Vol. 15, No. 6, pp. 630-643, June 1996.

[14] L. Benini, G. De Micheli, E. Macii, M. Poncino, R. Scarsi, "Symbolic Synthesis of Clock-Gating Logic for Power Optimisation of Synchronous Controllers," ACM Transactions on Design Automation of Electronic Systems, To Appear.

[15] S. H. Chow, Y. C. Ho, T. Hwang, C. L. Liu, "Lower Power Realization of Finite State Machines - A Decomposition Approach," ACM Transactions on Design Automation of Electronic Systems, Vol. 1, No. 3, pp. 315-340, July 1996.

[16] J. Monteiro, A. Oliveira, "Finite State Machine Decomposition for Low Power," DAC-35: ACM/IEEE 1998 Design Automation Conference, pp. 763-768, San Francisco, CA, June 1998.

[17] L. Benini, G. De Micheli, Dynamic Power Management: Design Techniques and CAD Tools. Kluwer Academic Publishers, 1998.

[18] E. Macii, M. Pedram, F. Somenzi, "High-Level Power Modeling, Estimation, and Optimisation", IEEE Transactions on CAD, Vol. 17, No. 11, November 1998.

[19] R. I. Bahar, C. Gaona, E. Frohm, G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi, "Algebraic Decision Diagrams and Their Applications", Formal Methods in System Design, Vol. 10, pp. 171-206, 1997.

[20] E. M. Sentovich, K. J. Singh, C. W. Moon, H. Savoj, R. K. Brayton, A. Sangiovanni-Vincentelli, "Sequential Circuits Design Using Synthesis and Optimisation," ICCD-92: IEEE International Conference Computer Design, pp. 328-333, Cambridge, MA, October 1992.

[21] F. Somenzi, CUDD: University of Colorado Decision Diagram Package, Release 2.3.0, Technical Report, Dept. of ECE, University of Colorado, Boulder, CO, September 1998.

[22] F. Brgles, D. Bryan, K. Kośmiński, "Combinational Profiles of Sequential Benchmark Circuits," ISCAS-89: IEEE International Symposium on Circuits and Systems, pp. 1929-1934, Portland, OR, May 1989.

[23] A. Sals, M. Horowits, "IRSIM: An Incremental MOS Switch-Level Simulator," DAC-26: ACM/IEEE Design Automation Conference, pp. 173-178, Las Vegas, NV, June 1989.

[24] C. Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. M. Despain, B. Lin, "Power Estimation Methods for Sequential Logic Circuits," IEEE Transactions on VLSI Systems, Vol. 3, No. 3, pp. 404-416, September 1995.

[25] G. Berry, H. Touati, "Optimised Controller Synthesis using Esterel," IWLS-93: ACM/IEEE International Workshop on Logic Synthesis, Paper 5b, Lake Tahoe, CA, May 1993.

[26] S.-I. Minato, "Generation of BDDs from Hardware Algorithm Description," ICCAD-96: IEEE/ACM International Conference on Computer-Aided Design, pp. 644-649, San Jose, CA, November 1996.