Computational Kernels and their Application to Sequential Power Optimization

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

Stanford University
Computer Systems Laboratory
Stanford, CA 94305

‡ Politecnico di Torino
Dip. di Automatica e Informatica
Torino, ITALY 10129

Abstract

We introduce a new sequential optimization paradigm based on the extraction of computational kernels, i.e., logic blocks whose behavior mimics the steady-state behavior of the original circuit. We present a procedure for the automatic extraction of such kernels directly from the gate-level description of the design. The advantage of this solution with respect to extraction algorithms based on STG analysis is that it can be applied to large circuits, since it does not require to manipulate the STG specification.
We exploit computational kernels for optimization purposes; in particular, we describe an architectural decomposition paradigm whose template is reminiscent of the mux-based scheme adopted in parallel implementations of logic-level descriptions.
We show the usefulness of the new optimization style by applying it to the problem of reducing the power dissipated by a sequential circuit. Experimental results, obtained on standard benchmarks, demonstrate the merit of the proposed approach.

1 Introduction

In spite of the large number of states that are potentially reachable by a sequential design, it is a well established fact that, during normal operation, the circuit tends to run through only a few of such states. An informal, though convincing proof of this statement is given by the results of the probabilistic analysis of the finite state machines associated to large networks: Only a few states have sizable steady-state occupation probability [1]. A similar situation occurs at the network primary outputs. While the circuit walks through the most probable states, a limited number of output patterns is generated.
We call *computational kernel* of a sequential circuit a logic block whose behavior mimics the steady-state 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.
For circuits with a few registers, computational kernels can be calculated in an exact fashion through symbolic procedures for FSM reachability analysis. However, when the network size increases over a few tens of memory elements, the above approaches are no longer applicable for both memory and time reasons. Resorting to approximate, simulation-based techniques that rely on structural circuit analysis, is then mandatory.
In this paper, we discuss heuristics for the automatic extraction of computational kernels, we propose an optimization paradigm based on the exploitation of such kernels, and we illustrate how the technique can be applied to sequential power minimization.

Permission to make digital/hard copy of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DAC 98, San Francisco, California
©1998 ACM 0-89791-964-5/98/06..$5.00

The extraction procedure constructs the computational kernel of a given sequential network incrementally by iteratively reducing the size of the logic block implementing the next-state and the output functions through implication analysis and redundancy removal. The extraction algorithm is driven by a cost function, whose objective is that of monitoring the quality of the kernel with respect to a given optimization target (e.g., delay or power) and thus providing a criterion for stopping the iterations.
Both the original circuit and its kernel are fed by the same primary inputs, and the state and output values are computed by the low cost logic (i.e., the kernel) anytime this is possible. By definition of computational kernel, there are high chances for this situation to happen. Therefore, the average computational cost decreases with respect to the original design. Moreover, once the computational kernel is available as a logic network, it can be exploited for optimizing the original circuit as well, because the input conditions when the kernel is active are *controllability don't cares* for the original circuit.
We show the usefulness of kernel-based optimization by applying it to the problem of reducing the power dissipated by a sequential circuit. In this case, the cost function which drives the kernel extraction procedure must take into account both the switching activity and the capacitive load of the reduced network. Experimental results, obtained on a set of standard benchmarks, demonstrate the validity of the proposed approach.

2 Definition of Computational Kernel

For the sake of simplicity, we provide the definition of computational kernel of a sequential design by referring to the FSM which models the behavior of the circuit. This definition is then extended to the more interesting case of circuits described at the structural-level, that is, to designs whose manipulation is not constrained by the size of the available computer memory.
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 $z \in Z$.
Mealy-type FSMs produce the output value z when the transition from state $s \in S$ to state $t \in S$ labeled x is taken, while Moore-type FSMs produce the output value z when a given state s is reached. Therefore, in Moore-type machines, states, rather than edges, are labeled with output symbols. M can be represented by a *state transition graph* (STG); for Mealy-type FSMs, the vertices of the STG are elements of $s \in S$, and the edges are labeled with pairs $(x, z) \in X \times Z$. On the other hand, the vertices of Moore-type FSMs are labeled with $z \in Z$, while the edges are labeled with $x \in X$.

Given a Moore-type 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:

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

$S_p = S_M \cup idle$, where $S_M = \{s \in S : P(s) \geq p\}$ is the subset of the states of M whose steady-state occupation probabilities are larger than p, and *idle* is an additional state which corresponds to the remaining states of M. S_p^0 equals S^0 in case $S^0 \in S_M$, otherwise $S_p^0 = idle$. X_p is the set of input signals which includes X. Additional input signals are required to specify the destination state in transitions from *idle* to other states in S_p. Finally, $R_p(x,s,t,z) = R(x,s,t,z) - \{(x,s,t,z) : s \notin S_M \wedge t \notin S_M\} + \{(x,s,t,z) : s = idle \wedge t \in S_M\} + \{(x,s,t,z) : s \in S_M \wedge t = idle\} + (x, idle, idle, z)$.
As an example, consider the simple finite state machine shown in 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 computational kernel of the machine is depicted in Figure 1-b.

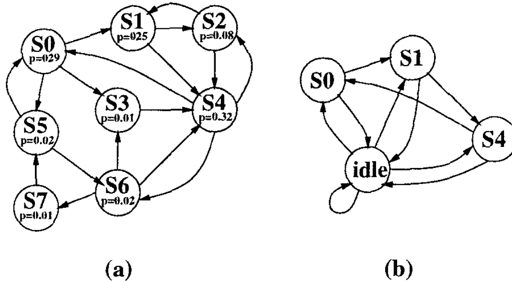

Figure 1: Moore FSM (a); 0.25-Order Computational Kernel (b).

When the FSM modeling the design is of Mealy-type, the definition of computational kernel seen above becomes more complex, since the output and the transition relations of M_p must be determined separately. The reason for this is that the output values generated by the FSM are associated to the edges, rather than to the STG vertices. Then, the way to proceed for determining the kernel consists of extracting the two relations above from the global relation through universal quantification, applying the definition given for Moore-type FSMs to the STGs of such relations, and forcing the two kernels to run in lock-step.
From the discussion above, it is apparent that calculating the computational kernel of a FSM can be done easily through symbolic manipulation, by handling simultaneously sets of states represented as characteristics functions [2]. Well-established technology developed in the context of exact [3, 4, 5, 6] and approximate [7, 8] FSM reachability analysis can be easily applied to solve the kernel calculation problem in the case the STG of the FSM is available or it can be extracted from the circuit description.
If the computational kernel has been determined following a top-down path starting from the STG (implicitly or explicitly represented), state assignment must be performed, and existing logic synthesis and optimization techniques [9] can be applied to generate a sequential network functioning exactly as M_p.

A rather different situation must be faced when the STG construction is prevented by the memory required to (implicitly) store the STG itself. First, exact probabilistic analysis can not be performed in absence of the FSM specification at the STG-level. Approximate algorithms, such as those discussed in [10], must then be employed; consequently, some states may be incorrectly considered as belonging to M_p. Second, state encoding for S_p is not performed, because it may be too time consuming. In the next section, we propose a heuristic approach for simultaneously determining the output and the next-state kernels of a sequential circuit, and to directly obtain a logic-level sequential network for such kernels. The method does not require explicit nor implicit STG manipulation; therefore, it is not memory intensive, and can deal with large designs.

3 Heuristics for Kernel Extraction

In this section, we present an iterative procedure for determining the computational kernels of a sequential circuit that works on the structural description of the network. The algorithm is heuristic by nature, and it exploits concepts such as *logic implication* and *redundancy removal* which have been in use for a long time in applications like test generation and logic optimization. More specifically, it takes the initial sequential 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 Figure 2. A connection w is selected inside network A. Signal w is replaced with either the constant value 0 (shown in the figure) or 1, and the circuit is simplified by propagating the constant value in the fanout cone of w and by removing all the fanout-free logic gates in the transitive fanin of w. Notice that w can be a primary input or a primary output of the logic network.

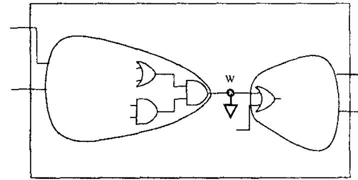

Figure 2: Basic Transformation for Kernel Construction.

The computational kernel K is obtained from the original circuit A by applying a sequence of these elementary transformations, until a stopping criterion is satisfied. We denote with K^i the computational kernel K after i applications of the above transformation. Initially, $K^0 \equiv A$; then, the i-th transformation yields a new circuit K^i from the previous one K^{i-1}. The pseudo-code of the iterative extraction procedure is shown in Figure 3.

```
procedure Compute_Kernel(A,F,U_w) {
    i = 0; K^i =A;
    do {
        w = SelectNode(A,U_w);
        v = ChooseBestValue(A,w);
        K^{i+1} = PropagateValues(K^i,A,w,v);
        i + +;
    } while (!StopTest(K^i,F))
    return(K^i);
}
```

Figure 3: The Compute_Kernel Algorithm.

The procedure receives, as inputs, the original circuit, A, the cost function, F, which controls the stopping criterion, and the utility function, U_w, which drives the node selection process, and it returns the computational kernel, K^i, obtained after i iterations of the **do while** loop.

It is easy to realize that the selection of the nodes in the network to which the transformation is applied (procedure `SelectNode` in the pseudo-code) heavily impacts the result of the kernel extraction. The node selection strategy can thus be customized for different cost functions to be optimized. As an example, in Section 5 we present a heuristics for reducing the power dissipated by the kernel under the constraint that K "covers" most of the behaviors of the original circuit A.

The computational kernel can be seen as a "dense" implementation of the circuit it has been extracted from. In other words, 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. In the next section, we propose an innovative optimization paradigm that takes advantage of this fact.

4 Kernel-Based Optimization

The computational kernels of a sequential circuit, extracted as discussed in Section 3, provide us with an extremely powerful device to be used for various types of logic optimization.

Given a sequential circuit with the well-known topology of Figure 4-a, the paradigm we propose for improving its quality with respect to a given cost function (e.g, power dissipation, timing) is based on the architecture shown in Figure 4-b.

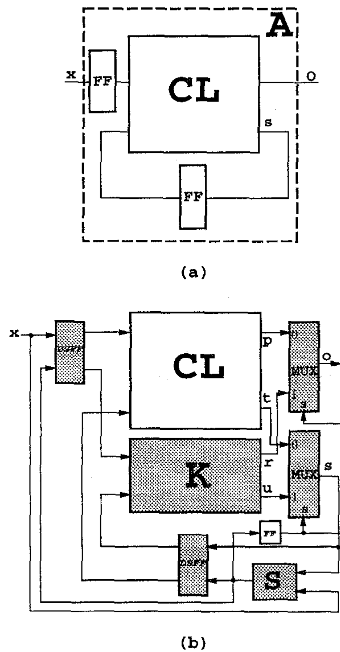

Figure 4: Sequential Circuit (a); Kernel-Based Architecture (b).

The essential elements of the architecture are the following: The combinational portion of the original circuit (block CL), the kernel (block K), the selector function (block S), the double-state flip-flops ($DSFF$), and the output multiplexors (MUX).

The purpose of the 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 primary and present-state inputs that will be fed to blocks CL and K in cycle $n+1$. If the output values in clock 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 multiplexors that select which block produces the output and the next-state. The optimized implementation is functionally equivalent to the original one.

The sequential elements indicated as $DSFF$s are called ***dual-state*** flip-flops, and they replace the ordinary flip-flops present in the original design. A dual-state flip-flop is functionally equivalent to the schematic of Figure 5, and it operates as follows: When $S=1$, 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 $S=0$. In this way, the state of either the kernel or the original network can be kept unchanged (i.e., frozen in the dual-state flip-flops) while the other logic block is being used to produce the output and the next state.

Clearly, Figure 5 only describes the functional behavior of a dual-state flip-flop, whose actual implementation can be properly optimized to reduce the overhead with respect to the standard flip-flops used in the original sequential network.

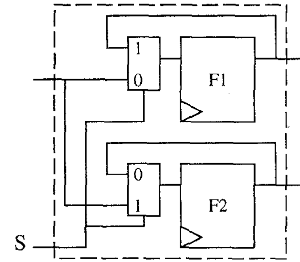

Figure 5: Functional Model of a Dual-State Flip-Flop.

It can be observed in first place that, besides the automatic generation of the kernel (this task is accomplished by procedure `Compute_Kernel` of Section 3), also the logic for the selector function S needs to be synthesized. Fortunately, the logic for S can be obtained as a by-product of the kernel extraction process. In other words, procedure `Compute_Kernel` can be modified to incorporate the automatic synthesis of S.

This is because the ON-set of the selection function, at the i-th iteration of the kernel extraction loop, is defined as the set of primary input and present-state conditions for which each primary and next-state output of K^i is equal to the corresponding primary and next-state output of CL (that is, K^i can be used in place of CL to compute the output and next-state values). Thus, S_i can be simply computed in an exact fashion by the following equation:

$$S_i(x,s) = \prod_{j=1}^{N} (\rho_j^i \equiv \omega_j) \quad (1)$$

where the product stands for logic conjunction, ρ_j^i is the j-th output of network K^i (that is, an element of set (r,u)), ω_j is the j-th output of network CL (that is, an element of set (p,t)), x are the primary inputs, and N is the number of primary and next-state outputs.

Since binary decision diagrams (BDDs) [11] are used to represent both K and S, S_i can be computed as long as it is possible to construct its BDD. There may exist circuits for which the BDD of S can not be constructed. In this case, we have to resort to approximate methods. For performance reasons, such methods must be customarily designed depending on the specific target optimization. In Section 5, we describe a solution which is particularly suited to power minimization.
The revisited pseudo-code is shown in Figure 6. It should be noticed that an extra input parameter is passed to the procedure: The timing constraint T to be used in the synthesis of S. This is because the speed of S must always be kept under consideration, since its delay adds up to the critical delay of the original sequential circuit.

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

Figure 6: The Compute_Kernel_and_Select Algorithm.

5 Application to Power Optimization

Research on logic-level power minimization techniques has always been very active [12]. Most of the optimization approaches at the logic-level target combinational circuits, and the literature on the subject is vast. Concerning sequential networks, on the other hand, only a few solutions have appeared to be successful, namely, state re-encoding, re-timing, clock-gating, and guarded-evaluation. For a detailed discussion of these techniques the interested reader may refer to [13, 14].
In this section, we propose the application of the computational kernel based paradigm discussed in Section 4 to solve the sequential power optimization problem. In other words, we discuss how the basic operations of algorithm Compute_Kernel_and_Select can be implemented when the target of the optimization is a low-power realization of the original circuit.
A few techniques similar to ours have been proposed in the recent literature. *Precomputation* [15, 16] is an approach to power minimization that relies on the idea of computing some of the output values of a circuit through a simplified (and low power consuming) logic running in parallel with the original circuit. The technique works well for circuits with pipelined structure; on the other hand, the extension to the case of sequential circuits with feed-back proposed in [15] is not applicable in practice, as shown by the data presented in [17, 18].
FSM decomposition for low power [19, 20, 21] can be seen as a top-down kernel extraction procedure that starts from explicit STG specifications (e.g., state tables). Although these techniques reported sizable power reductions, they can be applied only to small circuits for which the explicit STG description can be manipulated in reasonable time and memory space. The advantage of our approach to kernel extraction is that it can be applied to circuits for which not even the implicit representation of the STG can be constructed. Kernel extraction for low power is a specialization of the general algorithm of Figure 6.

5.1 Node and Value Selection

In order to perform the selection of the candidate node, w, procedure SelectNode within the algorithm of Figure 6 requires a utility function U_w that estimates the savings that can be achieved if w is replaced by either 0 or 1.
The function U_w we adopt in our method is an approximate one, since we are interested in fast, yet reasonably accurate estimates of the power savings. It is a combination of the expected power savings on w (χ_w) and the expected power savings on the fanin and fanout (ξ_w):

$$U_w = \chi_w + \alpha\xi_w \quad (2)$$

where $0 \leq \alpha \leq 1$ is a scaling coefficient (by default, $\alpha = 1$).
ξ_w can be efficiently computed, given the transition probability $Tp[w]$ and the load capacitance C_w^{load} of node w, as $\chi_w = Tp[w] \cdot C_w^{load}$. 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:

$$Pow_{save} = \sum_{\forall w_i \text{ eliminated}} Tp[i] \cdot C_i^{load} \quad (3)$$

The number of gates eliminated can be easily determined by evaluating how many gates in the fanin and fanout cones have their outputs fixed to some constant value.
Even though the approximate U_w discussed above usually provides accurate enough estimates, for critical cases (e.g., portions of logic which are always under stress, controllers), it may be required to resort to exact calculations, at the price of an increased run-time of the kernel extraction procedure. The exact solution consists of evaluating the actual power savings resulting by the substitution in the network, by running some power estimation tool, such as symbolic simulation [22], or any available commercial tool. This is clearly much more time and memory demanding, and it is applicable for small circuits, since it would be invoked for every new substitution. In the current implementation, it is possible to trigger symbolic simulation to evaluate the exact power savings.
Once a candidate node w has been selected, there are two options for the selection of which value is the best to force on w (procedure ChooseBestValue in Figure 6). For small to medium-sized circuits, it may be feasible to force first 0 and then 1 on the network A, and then evaluate U_w using Equation 2 for each of the two choices. The selected value v 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 is close to 0.5; this is because, under the assumption of temporally uncorrelated circuit inputs, the transition probability can be obtained as:

$$Tp[w] = p_w(1 - p_w) + (1 - p_w)p_w = 2p_w(1 - p_w) \quad (4)$$

where p_w is the signal probability of node w. The first term in Equation 4 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 v to be assigned should be obtained by the logic expression $p_w > 0.5$. In other terms, w should be assigned to 0 if $p_w < 0.5$, to 1 otherwise, that is, to the value which is more likely to occur.

5.2 Stopping Criterion

The iterative construction of K and S will tend to yield increasingly small S's. 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 i-th version of the architecture can be computed as:

$$Pow_i = Pow(K^i) \cdot P(S_i) + Pow(A) \cdot (1 - P(S_i)) + Pow(S_i) \quad (5)$$

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

In order to prevent an early termination, in the implementation of procedure `StopTest` we have added to the right-hand side of Equation 5 a term, O_i, representing an overhead value, that may be adjusted iteration by iteration. In the first few iterations, it should be negative, to force the first transformations. Then, as i increases, it should tend to the value $Pow_{DSFF} + Pow_{MUX}$, where Pow_{DFSS} is the incremental (with respect to regular flip-flops) power dissipation of the dual-state flip-flops, and Pow_{MUX} is the power dissipation of the output multiplexors. Alternatively, the stopping condition may not be tested for the first few iterations, to ensure that the K^i is sufficiently simpler than A, and it is then thereafter to make sure than $P(S_i)$ does not become too small.

5.3 Synthesis of the Selection Logic

The selection logic S should be synthesized for power under timing constraints. In most cases, especially for large circuits, the function S obtained by Equation 1 is too large to be synthesized as is. In these cases, we use a *sub-setting* algorithm [7] that provides an implementation K^r of the selection logic whose ON-set is smaller than that of K (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 circuit A can be optimized using S_i as controllability *don't care* set, to reduce the area overhead and to save additional power. This is because, when $S = 1$, the functionality implemented by A is already computed by K.

6 Experimental Results

We have implemented the procedure for the extraction of the computational kernels of a sequential circuit and to build from it the architecture of Figure 4 as an extension of SIS [23] using CUDD [24] as the underlying BDD package. Experiments have been run on a DEC AXP 1000/400 with 256 MB of memory.

The benchmarks we have used to check the effectiveness of the kernel-based power optimization approach are ALL the large Iscas'89 sequential circuits [25], including the *addendum* (that is, a total of 15 examples). The circuits were initially 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 (10 cases). 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 original circuit A (column *Reference*), and that of the kernel-based architecture (column *Optimized*), as well as the obtained savings (column *Savings*). Column $P(S)$ 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 initial 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, while those for the initial and final circuits have been determined using the Irsim switch-level simulator [26].

The results we have obtained are promising. An average power savings of approximately 13.6% has been achieved, with a peak of 29.3% on circuit s1512.

Five of the benchmarks in the suite did not produce acceptable results. More specifically, the application of our technique to examples s3330 and s9234 did not produce any noticeable power improvement. For circuits s35932, s38417, and s38584, on the other hand, the execution did not complete, due to the size of the original network to be handled.

Concerning the timing of the optimized circuits, in spite of the fact that the delay of the selector function S adds up to the largest delay between CL and K, the penalty is limited (3.71% on average). This is because CL is optimized using S as *don't care set*, thus its delay usually reduces with respect to the original circuit.

As expected, the area penalty is relevant (58.9% on average). This is because the kernel-based approach we have proposed suffers, in principle, from the same overhead (logic duplication) that affects any type of parallel implementation.

7 Conclusions and Future Work

The computational kernels of a sequential circuit are blocks of logic whose behavior mimics the steady-state behavior of the original circuit. The computation of the kernel of a design specified through the state graph of the corresponding finite state machine is feasible only when the state space is of limited size, even though BDD-based, symbolic representation techniques are adopted.

In this paper, we have presented a heuristic method for the automatic extraction of approximate computational kernels directly from the gate-level description of a sequential design. The solution is simulation based, and it is thus applicable to large networks for which the STG can not be extracted and/or manipulated.

We have then proposed the exploitation of computational kernel as a device for general-purpose logic optimization. The method requires the modification of the original circuit according to an architectural decomposition paradigm whose template is reminiscent of the mux-based scheme adopted in most parallel implementations of logic-level descriptions.

We have proved the effectiveness of this optimization paradigm by using it for reducing the power dissipated by a sequential circuit; experimental results, obtained on standard benchmarks, are very promising.

Circuit	*In*	*Out*	*FF*	*Gates*	*Delay*	*Power*			*P(S)*	*Area*	*Delay*	*CPU*
						Reference	*Optimized*	*Savings*		*Overhead*	*Overhead*	*Time*
s1269	18	10	37	468	50.42	1412.96	1339.02	5.23 %	0.45	69%	3.34%	10
s1423	17	5	74	602	73.58	2037.88	1473.64	27.68 %	0.25	38%	0.87%	43
s1512	29	21	57	475	42.71	804.51	568.10	29.38 %	0.93	36%	1.69%	85
s3271	26	14	116	1045	35.21	3436.60	3209.33	6.61 %	0.44	79%	5.43%	132
s3384	43	26	183	1393	92.47	4651.68	4383.40	5.77 %	0.25	85%	6.65%	127
s4863	49	16	104	2022	86.67	5926.34	5618.15	5.20 %	0.99	81%	7.58%	321
s5378	35	49	164	1132	22.30	1891.85	1706.55	9.79 %	0.90	44%	1.61%	452
s6669	83	55	239	2703	163.59	8083.82	7304.13	9.64 %	0.98	63%	4.53%	398
s13207	31	121	669	2462	44.59	3918.45	3230.56	17.55 %	0.97	43%	2.13%	494
s15850	14	87	597	3417	72.73	4971.02	4022.97	19.07 %	0.99	51%	3.28%	795
Avg.								13.59 %	0.72	58.9%	3.71 %	285.7

Table 1: Experimental Results.

We believe that the concept of computational kernel can be exploited in several directions. Performance optimization is an interesting future application of this technology. At this regard, computational kernels can be seen as an extension of the ideas presented in [27]. Moreover, we are considering the possibility of applying computational kernel extraction at higher levels of abstraction.

References

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

[2] G. D. Hachtel, F. Somenzi, *Algorithms for Logic Synthesis and Verification*, Kluwer Academic Publishers, 1996.

[3] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, "Sequential Circuit Verification Using Symbolic Model Checking," *DAC-27*, 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*, 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*, pp. 130-133, Santa Clara, CA, November 1990.

[6] H. Cho, G. D. Hachtel, S. W. Jeong, B. Plessier, E. Schwarz, F. Somenzi, "ATPG Aspects of FSM Verification," *ICCAD-90*, pp. 134-137, Santa Clara, CA, November 1990.

[7] K. Ravi, F. Somenzi, "High-Density Reachability Analysis," *ICCAD-95*, 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. CAD-15, No. 12, pp. 1465-1478, December 1996.

[9] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[10] 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*, Vol. VLSI-3, No. 3, pp. 404-416, September 1995.

[11] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, Vol. C-35, No. 8, pp. 79-85, August 1986.

[12] M. Pedram, "Power Minimization in IC Design: Principles and Applications," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 1, No. 1, pp. 3-56, 1996.

[13] W. Nebel and J. Mermet Editors, *Low-Power Design in Deep Sub-Micron Electronics*, Kluwer Academic Publishers, 1997.

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

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

[16] J. Monteiro, J. Rinderknecht, S. Devadas, A. Ghosh, "Optimization of Combinational and Sequential Circuits for Low Power Using Precomputation," *1995 Chapel Hill Conference on Advanced Research in VLSI*, pp. 430-444, Chapel Hill, NC, March 1995.

[17] L. Benini, G. De Micheli, E. Macii, M. Poncino, R. Scarsi, "Symbolic Synthesis of Clock-Gating Logic for Power Optimization of Control-Oriented Synchronous Networks," *EDTC-97*, pp. 514-520, Paris, France, March 1997.

[18] L. Benini, G. De Micheli, E. Macii, M. Poncino, R. Scarsi, "Integrating Logic-Level Power Management Techniques," *SASIMI-97*, pp. 59-65, Osaka, Japan, December 1997.

[19] 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.

[20] L. Benini, P. Vuillod, C. Coelho, G. De Micheli, "Synthesis of Low-Power Selectively-Clocked Systems from High-Level Specification," *ISSS-96*, pp. 57-62, La Jolla, CA, October 1996.

[21] L. Benini, F. Vermeulen, G. De Micheli, "Finite-State Machine Partitioning for Low Power," *ISCAS-98*, Monterey, CA, May 1998, To Appear.

[22] J. Monteiro, A. Ghosh, S. Devadas, K. Keutzer, J. White, "Estimation of Average Switching Activity in Combinational Logic Circuits Using Symbolic Simulation," *IEEE Transactions on CAD*, Vol. CAD-16, No. 1, pp. 121-127, January 1997.

[23] E. M. Sentovich, K. J. Singh, C. W. Moon, H. Savoj, R. K. Brayton, A. Sangiovanni-Vincentelli, "Sequential Circuits Design Using Synthesis and Optimization," *ICCD-92*, pp. 328-333, Cambridge, MA, October 1992.

[24] F. Somenzi, *CUDD: University of Colorado Decision Diagram Package*, Release 2.1.2, Technical Report, Dept. of ECE, University of Colorado, Boulder, CO, April 1997.

[25] F. Brglez, D. Bryan, K. Koźmiński, "Combinational Profiles of Sequential Benchmark Circuits," *ISCAS-89*, pp. 1929-1934, Portland, OR, May 1989.

[26] A. Salz, M. Horowitz, "IRSIM: An Incremental MOS Switch-Level Simulator," *DAC-26*, pp. 173-178, Las Vegas, NV, June 1989.

[27] L. Benini, E. Macii, M. Poncino, "Telescopic Units: Increasing the Average Throughput of Pipelined Designs by Adaptive Latency Control", *DAC-34*, pp. 22-27, Anaheim, CA, June 1997.