ILP-Based Optimization of Sequential Circuits for Low Power

Feng Gao and John P. Hayes
Advanced Computer Architecture Lab.
University of Michigan, Ann Arbor, MI 48109, USA
{fgao, jhayes}@umich.edu

Abstract

The power consumption of a sequential circuit can be reduced by decomposing it into subcircuits which can be turned off when inactive. Power can also be reduced by careful state encoding. Modeling a given circuit as a finite-state machine, we formulate its decomposition into submachines as an integer linear programming (ILP) problem, and automatically generate the ILP model with power minimization as the objective. A simple, but powerful state encoding method is used for the submachines to further reduce power consumption. We present experimental results which show that circuits designed by our approach consume 30% to 90% less power than conventional circuits.

Categories and Subject Descriptors:

B.7 INTEGRATED CIRCUITS. Additional classification: F.1 COMPUTATION BY ABSTRACT DEVICES.

General Terms:

Algorithms, design, experimentation.

Keywords

Finite-state machine, decomposition, low power, integer linear programming.

1. INTRODUCTION

With increasing integration scale and clock frequency, power dissipation forms an important design constraint for integrated circuits. Various technologies, from the transistor and gate levels to the operating system and application levels, have been studied to reduce system power consumption [7], [12].

For a sequential circuit, an effective way to reduce power dissipation is to turn off part of the circuit when inactive. Finite state machine (FSM) decomposition is usually applied to facilitate this approach [1], [2], [8], [9]. The states of the target FSM M are partitioned to form a set of submachines, and each submachine implements the functions of M in one state partition. When one submachine is active, we can disable all the other submachines, leading to a reduction of the switching activities involved in state transitions.

Two different FSM decomposition structures have been studied. The first approach lets the submachines maintain their own states and compute their own state transitions [1], [8], [9]. Figure 1*a* shows the circuit structure of this type of decomposition. We refer to this approach as *sequential logic decomposition* (*SLD*). The second approach separates the state update and state transition computation. Dedicated logic is used to maintain circuit states and schedule submachine activities, while the submachines are used just for computing the next states and outputs [2]. This can be regarded as the decomposition of the combinational logic of the target system, and is hence called *combinational logic decomposition* (*CLD*). The circuit structure produced by this method is illustrated in Figure 1*b*. The approaches in [1], [2], [8], [9] decompose the target FSMs incrementally using heuristic or generic algorithms, and are only able to explore a very limited design space.

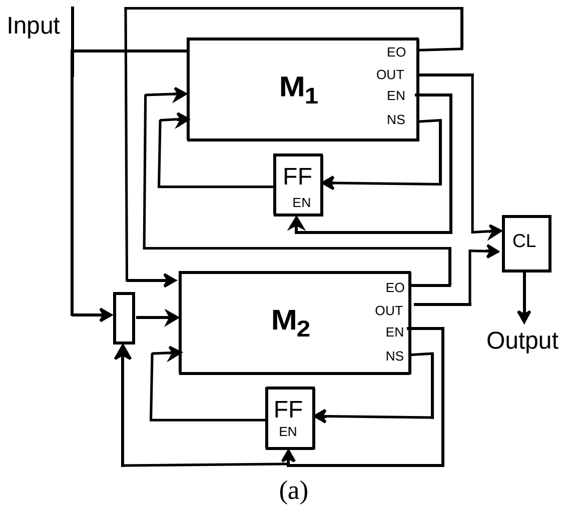
Circuit power consumption can be further reduced by taking advantage of the flexibility possible in submachine state encoding. FSM encoding has been studied for decades. Traditionally, encoding algorithms aim to minimize area [4], [17]. Several encoding algorithms explicitly targeting low power have also been proposed recently [10], [13], [16]. However, the submachines obtained by decomposition differ from traditional FSMs in that they have transitions to and from other submachines. Therefore, the state assignment of one submachine affects the state assignments of other submachines.

In this paper, we formulate FSM decomposition as an integer linear programming (ILP) problem with power minimization as the objective, and generate the ILP model automatically. A simple, but powerful way to encode interactive submachine states is also proposed. We present experimental results which show that the decomposed FSMs consume 30% to 90% less power than conventional designs, and incur 20% to 120% area overhead. Experiments on technology-mapped circuits show similar results (30% to 70% power reduction at the cost of 10% to 100% area overhead).

The remainder of this paper is as follows. Some notation and our system model are introduced in Section 2. The optimization procedure, including the ILP model generation and the state encoding for interacting machines, are detailed in Section 3. In Section 4, we present our experimental results. Section 5 concludes the paper.

2. NOTATION AND SYSTEM MODEL

A *finite-state machine (FSM) M* is defined as a 6-tuple $\{I, S, \delta, O, \lambda, S_0\}$, where I is the set of inputs, S is the set of states, $\delta: I \times S \rightarrow 2^S$ is the state transition function, O is the set of outputs, $\lambda: I \times S \rightarrow 2^O$ is the output function, and $S_0 \subseteq S$ is the set of initial states [6]. We use $|S|$ to represent the size of set S. An FSM can also be described using a state transition graph (STG). The STG of M is denoted by $G_M(V_M, E_M)$, where V_M is the set of nodes, representing the state set of M and $E_M = \{<u, v>, u, v \in V_M\}$ is the set of edges, representing the state transition set of M. Each edge $<u, v>$ is labeled with a set of input-output pairs, each of which denotes an input that triggers the state transition and the corresponding output. A



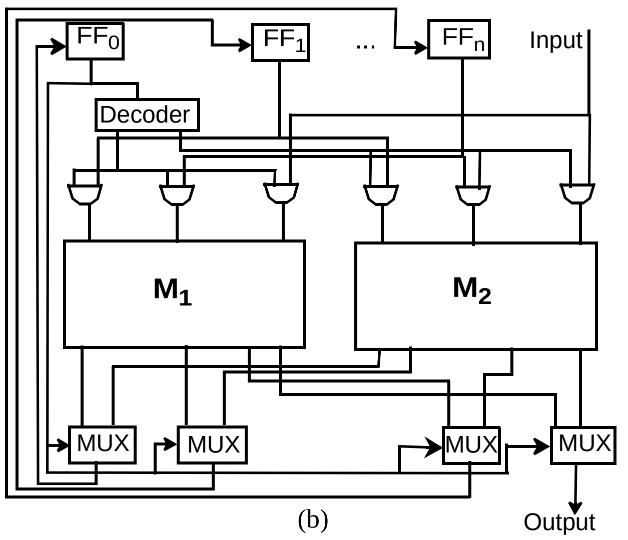


Figure 1. Two FSM decomposition methods: (a) sequential logic decomposition (SLD), and (b) combinational logic decomposition (CLD)

submachine of M based on state set S'$\subseteq$ S contains all the transitions starting from or ending at states in S'. Consider FSM *dk27* in Figure 2. We divide its states into two subsets $S_1 = \{s_1, s_4, s_6\}$ and $S_2 = \{s_2, s_3, s_5, s_7\}$ and obtain submachines M_1 and M_2. Note that a submachine differs from a classical FSM in that it contains transitions that start from or end at states in other submachines. The existence of such transitions introduces inter-dependence in the submachines' state encoding, and hence complicates the design problem.

We adopt the FSM decomposition structure shown in Figure 1*b* and limit the number of submachines to two. Suppose the FSM M has u input signals, w output signals and N states. The scheduling logic, which controls the activities of submachines, has a 1:2 decoder, $2(u + n)$ AND gates, $2(n + 1 + w)$ 2:1 multiplexers, and $n + 1$ state flip-flops. The flip-flops are allocated as follows. We use n flip-flops for submachine states, and the remaining one (FF$_0$) as a scheduling flip-flop to identify the submachine that is active in the current cycle. The flip-flop FF$_0$ determines which submachine inputs should be enabled, and which submachine outputs should be chosen as the outputs of the circuit. Since the state flip-flops are shared by both submachines, they should be able to hold the states in the larger submachine, whose number is between $\lceil N/2 \rceil$ and N. The number of flip-flops to store these n states is hence between $\lceil \log\lceil N/2 \rceil \rceil$ and $\lceil \log N \rceil$, which differ by at most one. Therefore, the number of AND gates and multiplexers varies only slightly (at most by two), so does the area of the scheduling logic.

The *state probability* $Pr(s)$ for state s in M is the probability that M is in state s, while the *state transition probability* $Pr(s_1, s_2)$ from s_1 to s_2 is defined as the probability that M transits from s_1 to s_2. The state probability $Pr(M_i)$ of a submachine M_i is the sum of the state probabilities of all states in M_i, that is,

$$Pr(M_i) = \sum_{k=1, s_k \in M_i}^{|S|} Pr(s_k) \quad (1)$$

On the other hand, the *state transition probability* $Pr(M_i, M_j)$ from submachine M_i to M_j is the sum of the state transition probabilities from any state in M_i to any state in M_j.

$$Pr(M_i, M_j) = \sum_{s_k \in M_i}^{s_l \in M_j} Pr(s_k, s_l) \quad (2)$$

In the bipartition case where there are just two submachines, we have

$$Pr(M_1) = Pr(M_1, M_1) + Pr(M_2, M_1) \quad (3)$$

If a transition is within the same submachine M_i, the submachine label does not change. Therefore, the inputs to the other submachine are kept disabled, causing no activity in that submachine. On the other hand, if a transition occurs from submachine M_i to submachine M_j, the inputs to M_i change to all-0s, while the primary inputs of M_j change to new values. Consequently, both machines become active, and the probability $P(M_1)$ that M_1 is active is $Pr(M_1) + Pr(M_1, M_2)$. Consider *dk27* in Figure 2, whose state probabilities are given in the figure beside the states. $Pr(M_1) = Pr(s_1) + Pr(s_4) + Pr(s_6) = 1/2$. $Pr(M_1, M_2) = Pr(s_7, s_6) + Pr(s_5, s_1) = 1/2 * 7/42 + 1/2 * 1/21 = 9/84$. $P(M_1) = Pr(M_1) + Pr(M_1, M_2) = 51/84$.

The area of the combinational logic (CL) associated with a state s is denoted $A(s)$, and the area of a submachine M_i is denoted $A(M_i)$. The power consumption of the decomposed FSM $Pw(M)$ is $P(M_1) \times A(M_1) + P(M_2) \times A(M_2) + P(SL) \times A(SL)$, where $A(SL)$ is the area of the scheduling logic and $P(SL)$ is the probability that the scheduling logic is active. Since the scheduling logic is never turned off, $P(SL)$ is always one. Furthermore, $A(SL)$ is also approximately a constant, as shown before. The power consumption of the scheduling logic is hence approximately a constant.

3. ILP-BASED FSM DECOMPOSITION

The optimization algorithm follows the procedure depicted in Figure 3. Given an FSM, we first calculate the state probability and the CL area of each state. The ILP generator automatically constructs the ILP model using the calculated data. The ILP model is then exported to a commercial ILP solver cplex [5], which computes a state partition as its solution. The state partition defines a decomposition of the original FSM. We use jedi [14] to encode the submachines. Finally, the circuit is

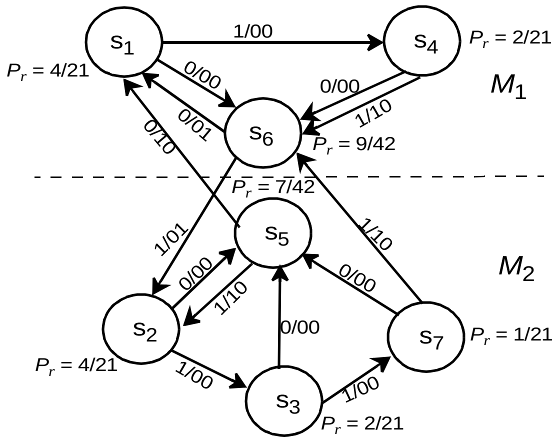


Figure 2. An example FSM dk27

optimized using SIS [14] and the Synopsys Design Analyzer [15] CAD tools. We describe these steps in detail in this section.

3.1 State probability and area estimation

The state probabilities for a completely specified FSM can be computed with the Chapman-Kolmogorov equations for a discrete-time, discrete-transition process [11]. For each state s_k, we have the following equation

$$\sum_{i=1}^{|S|} Pr(s_i) \times Pr(s_k|s_i) = Pr(s_k) \quad \forall\, k \in [1, |S|] \tag{4}$$

where $Pr(s_k \mid s_i)$ is the conditional probability that the machine makes a transition to state s_k if it is in state s_i. An additional constraint is that the overall state probability of all states is 1, namely,

$$\sum_{i=1}^{|S|} Pr(s_i) = 1 \tag{5}$$

We next estimate the area of the combinational logic (CL). For the purpose of area measurement, we view the CL of a state as a PLA and estimate its number of literals [2]. Each row of a PLA, represented as $\{<PI, CS>, <NS, PO>\}$, has $u + n$ inputs, where $u = |PI|$ and $n = |CS| = |NS|$ are the numbers of primary inputs and state flip-flops, respectively, of the FSM. The PLA has $w + n$ outputs, where $w = |PO|$ is the number of primary outputs. We estimate the number of literals in the PLA as follows. Assume that each row corresponds to $w + n$ cubes, and the size of a cube is proportional to its number of inputs $u + n$. Suppose the CL for state s has m rows; its area is hence approximated by $m \times (u + n) \times (w + n)/2$. Note that we divide the last term by two because the probability for a bit in $<NS, PO>$ to be zero is assumed to be 0.5, in which case the cube should not be counted.

To estimate the area of a submachine, we must also estimate the area of the CL shared by each state pair. We also view the CL in terms of PLAs and count the number of shared literals. We compare any two rows of the two PLAs $\{<PI_i, CS_i>, <NS_i, PO_i>\}$ and $\{<PI_j, CS_j>, <NS_j, PO_j>\}$. If $PI_i = PI_j$ and $NS_i = NS_j$, they share $n = |CS|$ cubes. If $PI_i = PI_j$ and one bit of PO_i equals the corresponding bit in PO_j, they share one cube. Therefore, the total number of shared cubes is $(sw + sn \times n)/2$, where sw is the number shared outputs, and sn is the number of shared next states. For the same reason as before, this number is divided by two. The number of literals shared by two PLAs is hence $(u + n) \times (sw + sn \times n)/2$.

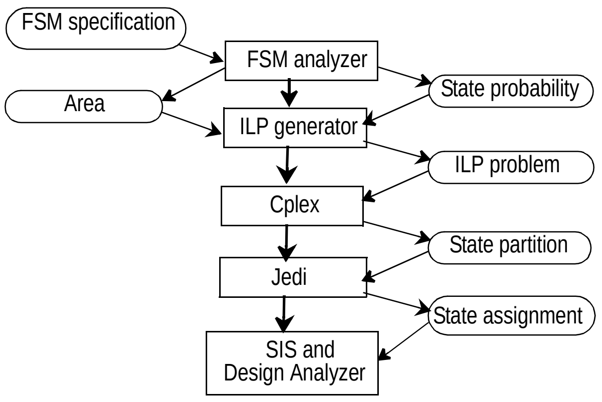


Figure 3. Flowchart of the optimization process

3.2 Problem formulation

The objective function to be minimized is the power consumption of the whole circuit. Since the scheduling logic consumes a constant amount of power, we exclude it from our objective function, which is therefore as follows.

$$\begin{aligned}&(Pr(M_1) + Pr(M_1, M_2)) \times A(M_1) + (Pr(M_2) + Pr(M_2, M_1))\\ &\qquad \times A(M_2)\\ &\quad = Pr(M_1) \times A(M_1) + Pr(M_2) \times A(M_2) + ((Pr(M_1, M_2)\\ &\qquad \times A(M_1) + Pr(M_2, M_1) \times A(M_2))\end{aligned}$$

Since the inter-submachine transitions activate both submachines and should be avoided as much as possible, we only consider decompositions where the probability of inter-machine transitions is less than 0.1. Under this constraint, the last term $Pr(M_1, M_2) \times A(M_1) + Pr(M_2, M_1) \times A(M_2)$ in the objective function can be neglected, leading to the simpler objective function

$$Pr(M_1) \times A(M_1) + Pr(M_2) \times A(M_2)$$

If M_1 has only two states s_i and s_j, its state probability is $Pr(s_i) + Pr(s_j)$. Its area is $A(s_i) + A(s_j) - SA(s_i, s_j)$, where $SA(s_i, s_j)$ denotes the area of the CL shared by s_i and s_j. Although the power estimation is more accurate if we consider logic sharing among three or more states, the estimation complexity increases significantly. Consequently, we only consider logic sharing between any two states of a submachine.

$$\sum_{k=1}^{2} \sum_{i \neq j} (Pr(s_i) + Pr(s_j)) \times (A(s_i) + A(s_j) - SA(s_i, s_j)) \tag{6}$$

Let $s_i s_j$ be a binary variable, which is 1 if and only if s_i and s_j are in the same submachine. The objective function can be restated as:

$$\sum_{i,j=1, i \neq j}^{|S|} s_i s_j \times (Pr(s_i) + Pr(s_j)) \times (A(s_i) + A(s_j) - SA(s_i, s_j)) \tag{7}$$

Note that $Pr(s_i)$, $A(s_i)$, and $SA(s_i, s_j)$ can be calculated beforehand. Therefore, the above function is actually a linear function of s_is_j, and so is used as the objective function of the ILP problem. We will try to minimize the objective function under the constraints derived below.

The ILP constraints fall into two main categories: quality constraints and correctness constraints. The quality constraints guide the solution to the desired decomposition. First, the interaction between submachines should be limited. In addition, we would like to generate one submachine with fewer states and larger state probability than the other. To this end, we define variables p_1s_i, p_2s_i for all $i \in [1, |S|]$, which are 1 if and only if s_i is in M_1 or M_2, respectively. The above quality constraints are formulated as follows.

$$\sum_{i,j=1}^{|S|} (1 - s_is_j) \times Pr(s_i, s_j) < 0.1 \tag{8}$$

$$\sum_{i=1}^{|S|} p_1s_i \le \sum_{i=1}^{|S|} p_2s_i \tag{9}$$

$$\sum_{i=1}^{|S|} p_1s_i \times Pr(s_i) \ge \sum_{i=1}^{|S|} p_2s_i \times Pr(s_i) \tag{10}$$

The correctness constraints ensure the variables have reasonable values. First, a state must belong to one and only one submachine.

$$p_1s_i + p_2s_i = 1, \text{ for } i \in [1, |S|]$$

Second, if s_i and s_j are both in M_1 or M_2, they belong to the same submachine. This constraint is formulated as

$$s_is_j = p_1s_i \text{ AND } p_1s_j \text{ OR } p_2s_i \text{ AND } p_2s_j$$

Because of the complementary relation between p_1s_i and p_2s_i, the above formulation is equivalent to

$$s_is_j = p_1s_i \text{ XNOR } p_1s_j \tag{11}$$

We linearize (11) as follows.

$$p_1s_i + p_1s_j + 1 = 2c_{i,j} + s_is_j \tag{12}$$

where $c_{i,j}$ is also a binary variable.

The ILP model is summarized in Figure 4. Solving the ILP model via cplex, we obtain an FSM bipartition that minimizes the objective function. We can further reduce circuit power consumption by taking advantage of the flexibility of the submachine state assignment, as described in the next subsection.

3.3 Submachine state assignment

We know of no FSM state assignment tool specifically designed for interactive submachines. We therefore modify the submachines so that we can solve their state assignment problem with the available FSM state encoding tools.

We first add a dummy state for the outgoing and incoming transitions of each submachine M_i. A new output signal is also added to M_i, which is for the scheduling flip-flop of the next cycle. Then we perform state assignment for M_i using the traditional state assignment tool jedi from the SIS suite [14], with the command *jedi -e c*. The dummy state is added to represent the inactive state, which is the all-0s state in our implementation. Therefore, we shift the submachine state assignments such that the dummy state is assigned the all-0s state. This shift significantly impacts the submachine power consumption. Consider the example FSM *dk27* and the indicated decomposition in Figure 2 again. We encode the states in M_1 as $\{s_1 = 000, s_4 = 010, s_6 = 011, \text{dummy} = 001\}$, and the states in M_2 as $\{s_2 = 001, s_3 = 111, s_5 = 100, s_7 = 101, \text{dummy} = 000\}$. The power consumption estimated by SIS is 250μW assuming a 20MHz operating clock frequency and a 5V supply voltage. However, if we shift the state assignment for M_1, the overall power consumption reduces to 140μW.

After state assignment, we generate the FSM implementation in SIS's blif format. We optimize the circuits in SIS using the standard optimization script *script.rugged*, followed by the command *full_simplify*. We also performed technology mapping using the Synopsys Design Analyzer to estimate the area and power consumption of the technology-mapped circuits.

4. EXPERIMENTAL RESULTS

To examine the effectiveness of our approach, we applied it to a subset of the MCNC FSM benchmarks [3]. We compared the area and power consumption of circuits designed using our ILP approach with those using the standard SIS (no decomposition) and the CLD approaches. We carried out area and power evaluation both before and after technology mapping. The run times of the ILP solver cplex varied from seconds to 20 minutes.

First, we use SIS to estimate the area and power consumption of the pre-mapping netlists. The power is estimated using the command *power_estimate -t S*, where 5V supply voltage and 20MHz working frequency are assumed. The results are presented in Figure 5. Under the FSM column, we list the FSMs and the number of their states, input signals and output signals, respectively. The area, in terms of the number of literals, and the corresponding power consumption for each FSM are then listed. SIS refers to a design directly optimized by SIS, while CLD and ILP refer to designs obtained with the

Minimize

$$\sum_{i,j=1, i \ne j}^{|S|} s_is_j \times (Pr(s_i) + Pr(s_j)) \times (A(s_i) + A(s_j) - SA(s_i, s_j))$$

subject to the following constraints.

$$\sum_{i,j=1}^{|S|} (1 - s_is_j) \times Pr(s_i, s_j) < 0.1$$

$$\sum_{i=1}^{|S|} p_1s_i \le \sum_{i=1}^{|S|} p_2s_i$$

$$\sum_{i=1}^{|S|} (p_1s_i \times Pr(s_i)) \ge \sum_{i=1}^{|S|} p_2s_i \times (Pr(s_i))$$

$$p_1s_i + p_2s_i = 1, \text{ for } i \in [1, |S|]$$

$$p_1s_i + p_1s_j + 1 = 2c_{i,j} + s_is_j, \text{ for } i,j \in [1, |S|]$$

All variables are binary.

Figure 4. The ILP model

FSM				SIS [2]		CLD[2]				ILP			
Name	**States**	**Inputs**	**Outputs**	**Area A_s**	**Power P_s**	**Area A_c**	**Power P_c**	**A_c/A_s**	**P_c/P_s**	**Area A_i**	**Power P_i**	**A_i/A_s**	**P_i/P_s**
s1494	48	8	19	514	1276	625	938.3	1.22	0.74	829	87.3	1.61	0.07
tbk	32	6	3	330	978.2	293	744.0	0.89	0.76	725	369.1	2.20	0.38
styr	30	9	10	407	1073	501	695.8	1.23	0.65	613	252.5	1.51	0.24
bbsse	16	7	7	103	373.0	152	318.0	1.48	0.85	212	138.7	2.06	0.37
cse	16	7	7	186	422.0	245	257.0	1.32	0.61	299	112.5	1.61	0.27
sand	32	11	9	527	1525	488	635.0	0.93	0.41	778	880.8	1.48	0.58
pma	24	8	8	202	965.1	237	622.0	1.17	0.64	423	188.5	2.09	0.20
s1488	48	8	19	503	1448	593	592.9	1.18	0.41	785	94.6	1.56	0.06
dk16	27	2	3	224	1238	251	868.0	1.12	0.70	314	874.0	1.40	0.71
ex1	20	9	19	214	789.8	257	409.0	1.20	0.52	255	200.8	1.19	0.25
keyb	19	7	2	168	599.0	246	492.0	1.46	0.82	290	135.0	1.72	0.24

Figure 5. Comparison of area and power consumption of FSMs obtained using the SIS, ILP and CLD techniques.

approaches proposed in [2] and in this paper, respectively. For the FSMs obtained using CLD and ILP methods, we also compare their area and power ratios to those in SIS. The data for the SIS and CLD approaches are taken from [2].

Compared with the FSMs obtained by SIS without considering power consumption, our approach generated FSMs with 30% to 90% less power at the cost of 20% to 120% area overhead, as shown in the ratio columns. In particular, we obtained two designs with extremely lower power consumption. Circuits *s1488* and *s1494* consume less than 10% power of the SIS designs. The reason is that both circuits have one or two states whose state probabilities are very high. We can hence disable most of the CL in most cycles, leading to large power reduction. Compared with the data of the CLD approach, circuits generated by our ILP method consume half or less power in nine of the eleven benchmarks. The ILP design of *s1494* consumes only around one tenth the power of its CLD counterpart. The area overhead varies from 20% to 100% more than that of the CLD approach. We believe that the power reduction is mainly due to the larger design space our technique can explore and its efficient state assignment algorithm.

We also evaluated these benchmarks after technology mapping. We export the same netlists as in the first set of experiments to the Synopsys Design Compiler and set the frequency to 20MHz. We used the TSMC18 0.18μm static CMOS standard cell library. The area and power consumption are estimated by Design Compiler. Figure 6 shows the experimental results in a similar way to Figure 5. We do not list the results for the CLD approach because we do not know the implementation of the decomposed circuits in [2]. The FSMs obtained using the ILP approach consume less power than the original implementation, with a reduction between 30% and 70%. The area overhead ranges from 10% to 100%.

Finally, we estimated the post-layout power consumption of these circuits. we performed place and routing using Cadence's Silicon Ensemble tool. We set the aspect ratio to 1.0 and row utilization to 0.85. The areas of the designs were manually measured, and the parasitic parameters were extracted. We then exported the parasitic capacitance to Design Compiler, which calculated the power consumption. The results are presented in Figure 7, and can be seen to be quite consistent with those in Figure 6, with just small variations.

5. DISCUSSION

We have presented a new way to optimize sequential circuits for low power. The problem of decomposing the circuits is modeled using integer linear programming, which explores a relatively large design space. An efficient method to encode interacting submachines has also been proposed. Experimental

FSM	SIS		ILP			
	Area A_s	**Power P_s**	**Area A_i**	**Power P_i**	**A_i/A_s**	**P_i/P_s**
s1494	4643.7	653.4	6266.9	278.6	1.35	0.43
tbk	3216.6	678.9	5235.8	478.7	1.63	0.71
styr	3752.2	755.2	4996.3	324.3	1.33	0.43
bbsse	1131.0	233.9	1393.8	168.2	1.23	0.72
cse	1789.6	303.7	2511.4	170.9	1.40	0.56
sand	4733.5	904.0	5814.6	422.8	1.23	0.47
pma	1739.7	279.7	3699.0	118.0	2.13	0.42
s1488	4820.0	682.0	6270.3	222.3	1.30	0.33
dk16	2162.2	498.1	2627.9	280.2	1.22	0.56
ex1	2085.7	407.0	2222.0	117.3	1.07	0.29
keyb	1909.4	423.5	2434.9	199.2	1.28	0.47

Figure 6. Comparison of area and power after technology mapping.

FSM	SIS		ILP			
	Area A_s	**Power P_s**	**Area A_i**	**Power P_i**	**A_i/A_s**	**P_i/P_s**
s1494	5402.3	1149.0	7430.4	481.3	1.38	0.42
tbk	3757.7	1163.0	6178.0	834.8	1.64	0.72
styr	4422.3	1343.4	5969.8	568.6	1.34	0.42
bbsse	979.7	366.5	1608.0	269.1	1.64	0.73
cse	2450.3	512.3	2926.8	280.7	1.19	0.55
sand	5520.5	1621.8	6806.3	825.2	1.23	0.51
pma	2304.0	444.8	4984.4	200.8	2.16	0.45
s1488	5745.6	1179.2	7430.4	378.4	1.29	0.32
dk16	2550.3	844.5	3433.96	471.2	1.16	0.56
ex1	2430.5	677.3	2560.4	206.0	1.05	0.30
keyb	2180.9	686.0	2840.9	314.9	1.30	0.46

Figure 7. Comparison of area and power after placement and routing.

results show that the ILP problem can be solved quickly and the resulting circuits consume much less power (30% to 90% less) than those obtained without considering power consumption. The results for technology-mapped and post-layout circuits show similar trends.

The ILP model is greatly simplified by the constraint that limits inter-submachine state transitions. However, this constraint also limits the design space being explored. In some cases where the number of states is very large, or many states are tightly coupled and hard to partition, an ILP solution may not be found, even though one exists. A possible way of tackling this problem is to add a pre-processing step that forms low-power state clusters and allows the inter-submachine transition constraint to be relaxed.

Acknowledgment

This research was supported by the National Science Foundation under Grant No. CCR-0073406.

6. REFERENCES

[1] L. Benini, G. De Micheli, and F. Vermulen, “Finite State Machine Partitioning for Low Power”, *Proc. International Symposium on Circuits and Systems*, 1998, pp. 5-8.

[2] S. H. Chow, Y. C. Ho, T. Hwang and C. L. Liu, “Low Power Realization of Finite State Machines−A Decomposition Approach”, *ACM TODAES*, vol. 1, 1996, pp. 315-340.

[3] Collaborative Benchmarking Lab., North Carolina State Univ., http://www.cbl.ncsu.edu/benchmarks.

[4] S. Devadas, *et al*., “MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations”, *IEEE Trans. on CAD*, vol. 7, 1988, pp. 1290-1300.

[5] ILOG cplex webpage. http://www.ilog.com/products/cplex/.

[6] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, Englewood Cliffs, NJ: Prentice-Hall, 1981.

[7] E. Macii, M. Pedram, and F. Somenzi, “High-Level Power Modeling, Estimation, and Optimization”, *IEEE Trans. on CAD*, vol. 17, 1998, pp. 1061-1079.

[8] J. C. Monterio and A. L. Oliveria, “Finite State Machine Decomposition for Low Power”, *Proc. Design Automaton Conference*, 1998, pp. 758-763.

[9] J. C. Monterio and A. L. Oliveria, “FSM Decomposition by Direct Circuit Manipulation Applied to Low Power Design”, *Proc. Asia South Pacific Design Automation Conference*, 2000, pp. 351-358.

[10] E. Olson and S. Kang, “Low-Power State Assignment for Finite State Machines”, *Proc. International Symposium on Low Power Design*, 1994, pp. 63-68.

[11] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, 2nd Edition, New York: McGraw-Hill, 1984.

[12] M. Pedram, “Power Minimization in IC Design: Principles and Application”, *ACM TODAES*, vol. 1, 1996, pp. 3-56.

[13] K. Roy and S. Prasad, “Circuit Activity Based Logic Synthesis for Low Power Reliable Operations”, *IEEE Trans. on VLSI*, vol. 1, 1993, pp. 503-513.

[14] E. Sentovich, *et al*., “Sequential Circuit Design Using Synthesis and Optimization”, *Proc. International Conference on CAD*, 1992, pp. 328-333.

[15] Synopsys, Design Analyzer data sheet, http://www. synopsys.com/products/logic/design_compiler.html.

[16] C. Y. Tsui, M. Pedram, and A. Despain, “Low-Power State Assignment Targeting Two and Multilevel Implementations”, *IEEE Trans. on CAD*, vol. 17, 1998, pp. 1281-1291.

[17] T. Villa and A. Sangiovanni-Vincentelli, “NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementation”, *IEEE Trans. on CAD*, vol. 9, 1990, pp. 905-924.