Synthesis of Low-Power Asynchronous Circuits in a Specified Environment*

Steven M. Nowick *Michael Theobald*
Department of Computer Science
Columbia University, New York

Abstract — We introduce a new method for the synthesis of power-optimal asynchronous control circuits. The method includes two steps: (i) an exact algorithm for 2-level synthesis, and (ii) heuristic algorithms for multi-level synthesis. Unlike most existing synchronous algorithms, we incorporate both *temporal dependence* and *glitch activity* in our model to guide synthesis. Results, using only our 2-level minimization, show power reduction up to 33%.

1 INTRODUCTION

Interest in low-power design has grown considerably in recent years. The increasing market for battery-powered portable devices has made low power a critical concern. Low-power operation can reduce the need for expensive packaging, and can extend the lifetime of components by providing a less stressful operating environment.

In this paper, we focus on *asynchronous circuits* for low power. Asynchronous systems have the potential for low power operation for two reasons [3]. First, these systems have no global clock; in contrast, clock distribution is a major source of power consumption in synchronous systems. Second, asynchronous circuits have an inherent *automatic power-down operation*: modules are activated only when their operations are needed. Low-power design is a major focus of recent asynchronous design, including a low-power infrared communications chip, an asynchronous implementation of the ARM microprocessor, and an asynchronous error corrector for a DCC player [3].

Many techniques have been developed to reduce power consumption in *synchronous circuits*. These methods approach the problem at different levels of synthesis, including algorithmic, architectural or structural, logic synthesis, and IC device technology (see [4, 6]). Other techniques include "precomputation", "guarded evaluation" and "gated clocking" at the logic level (see [6]) and "voltage scaling" at the system level [4]. At the logic synthesis level, in particular, a number of power-optimization methods have been proposed. These include 2-level logic minimization [9], multi-level (extraction and factorization) [8], Boolean re-synthesis [1], clustering [16], and technology mapping and post-mapping optimization [6, 5].

These methods have been effective; however, they have two significant drawbacks in their power model:

> **temporal independence assumption:** Each of these methods assumes that primary inputs are temporally independent [11]. In reality, inputs are often correlated. For example, in controllers, input sequences (and resulting state sequences) are often known [2]. Monteiro and Devadas [10] show that the assumption of temporal independence may result in errors in power estimation up to 44%.
>
> **0-delay assumption:** Most of the above methods assume a 0-delay model, where glitch power dissipation is ignored. In fact, glitch power dissipation may contribute 15-20% of total power consumption in control circuits (in some datapath circuits, a much higher contribution is possible) [7].

* This work was supported by NSF under Grant no. MIP-9501880 and by an Alfred P. Sloan Research Fellowship.

Of the above logic synthesis methods, only Bahar and Somenzi [1] consider glitch power dissipation, but restricted to the context of a mapped circuit during resynthesis.

Contributions. This paper makes two contributions.

First, we introduce a power model which incorporates both *temporal dependence* and *glitch activity*. That is, we use a more accurate model than nearly all existing synchronous approaches. Temporal dependence and glitch activity are incorporated analytically, and are used in both technology-independent and dependent synthesis steps. Temporal dependence is naturally given in asynchronous controller specifications, such as burst-mode specifications [13, 12, 17], where the *input- and state-sequencing* are specified. Glitch activity is determined analytically, at the logic level. Our goal is to synthesis power-optimal circuits *with respect to* a given environment, *while considering* glitching activity. We call our synthesis approach *environment-driven*.

Second, we apply this model to a new synthesis method, for *power-optimal* asynchronous control circuits. We present a synthesis path consisting of (i) a technology-independent and (ii) a technology-dependent step. First, we present an exact 2-level minimization algorithm targeting low power for asynchronous circuits, i.e. where temporal dependence is provided by sequencing information. Second, we present power-optimal (a) *clustering* and (b) *decomposition* algorithms, targeted to CMOS complex gates. The complex gates are generally AOI-type, and are used to cluster products to further reduce power.

While our focus is on asynchronous circuits, we believe our techniques can be applied to synchronous circuits in the future.

2 PRELIMINARIES

2.1 Specified Environment

Our synthesis approach for a Boolean function f is based on a given *specified environment* (T, P), where T is the set of possible multi-input changes (also known as input transitions, i.e. "a change from one input vector to another") and P is the probability distribution of T. The assumption of a specified environment is very appropriate for asynchronous circuits [13, 12, 17].

2.2 Synthesis Techniques and their Power Models

There are three major sources of power consumption in CMOS circuits [3, 11]. *Switching energy* is associated with transitions on gate outputs. *Short circuit energy* consumption is caused by simultaneous conduction of *pull-up* and *pull-down* stacks, allowing current flow directly from the power supply to ground. Finally, *leakage energy* occurs in standby mode, and is determined by technology factors. In most CMOS circuits, switching power dominates the other two. Therefore, total energy consumption can usually be approximated by switching energy.

In synchronous circuits, power consumption of an AND-gate q_i, considering both input and output switching, is given by the following formula [9]:

$$W(q_i) = \frac{{V_{DD}}^2}{2 \cdot T_{cycle}} \cdot (C_{AND} \cdot \phi(q_i) + \sum_{l_b \in lit(q_i)} (C_{IN} \cdot \phi(l_b))),$$

where $\phi(q_i)$ and $\phi(l_b)$ are the number of transitions on q_i and l_b, respectively, during the clock cycle time T_{cycle}; lit(q_i) is the set of literals of implicant q_i; C_{AND} is the load seen by the AND-gate output; and C_{IN} is the load on each of its inputs.

This notion of power consumption is a poor match for asynchronous design, since an asynchronous circuit has no clock period T_{cycle}. A more appropriate metric is *energy per input transition* [2]. Thus, energy can be quantified over the sequence of input changes driven by the environment.

Energy consumption of a static CMOS gate, for a given computation, is often reasonably approximated by $E = \frac{1}{2} C {V_{DD}}^2 N$, where C is the load capacitance of the gate, and N is the number of output transitions for a given computation. Assuming uniform gate capacitances, the energy consumption can be normalized to N, the number of output transitions [3].

The amount of energy dissipated by a 2-level circuit $f = p_1 + p_2 + \ldots + p_n$, *with respect to a specified input transition* t, is expressed as follows: $E(p_i, t) = N(p_i, t) + \sum_{x \in lit(p_i)} N(x, t)$, and $E(f, t) = N(f, t) + \sum_{1 \le i \le n} E(p_i, t)$.

Here, $N(x, t)$ is the number of transitions on input x during transition t, $N(p_i, t)$ is the number of gate output transitions on AND-gate p_i, and $N(f, t)$ is the number of gate output transitions on OR-gate f.

Given a set T of input transitions, the *total energy*, consumed by a 2-level circuit, is therefore: $E(p_i) = \sum_{t \in T} E(p_i, t)$, and $E(f) = \sum_{t \in T} N(f, t) + \sum_{1 \le i \le n} E(p_i)$.

$E(p_i)$ is the power cost of an implicant p_i, over all input transitions. The average power cost per transition is therefore $\frac{E(p_i)}{|T|}$.

In general, in this paper, our goal is to produce a circuit with *minimum energy consumption per input transition*. Although there is no clock period, it should be clear that low-energy optimization will result directly in a low-power design. The "time unit" of computation is an input transition. Therefore, we will sometimes loosely interchange the terms "energy of a transition" and "power consumption of a transition".

3 Background On Hazard-Free Synthesis

3.1 Combinational Hazards

For the following discussion, a combinational circuit model is assumed where gates and wires may have arbitrary finite delays. Since we are concerned with the dynamic behavior of a combinational circuit as its inputs change value, we need to formalize the notion of a "multiple-input change", or "input transition".

A *transition cube* [14] is a cube with a *start point* and an *end point*. Given input states A and B, the transition cube $[A, B]$ has start (end) point A (B) and contains all minterms that can be reached during a transition from A to B. The cube describes a *multiple-input change* or *input transition* from A to B. Inputs are assumed to change monotonically (*i.e.*, at most once) in any order and at any time. Once a multiple-input change occurs, no further inputs may change until the circuit has stabilized.

A function f which does not change monotonically during an input transition is said to have a **function hazard** in the transition.

If a transition has a function hazard, *no* implementation of the function is guaranteed to avoid glitches during the transition (assuming our circuit model of arbitrary gate and wire delays) [14].

Therefore, we consider only input transitions which are *function-hazard-free* [1]. For such transitions, a circuit for f may still glitch due to delays in the actual gates and wires. In this case, the circuit is said to have a **logic hazard** for the input transition. The *logic hazard* is *static* for an input transition from A to B if $f(A) = f(B)$ and *dynamic* if $f(A) \neq f(B)$.

3.2 Conditions for a Hazard-Free Transition

We now describe conditions to avoid logic hazards in a sum-of-products implementation (for details, see [14]).

For the $1 \rightarrow 1$ case, the transition cube is called a *required cube*, which must be completely contained in some product to insure no hazards. For the $1 \rightarrow 0$ case, first, each $1 \rightarrow 1$ sub-transition must be hazard-free, so the corresponding required cubes must each be contained in some product. Second, no product in the cover may *illegally intersect* the $1 \rightarrow 0$ transition, i.e. intersect, but not contain the transition's start point, otherwise a dynamic hazard will result. Satisfying these conditions may require the use of redundant and non-prime implicants. There are two remaining transitions to consider. A $0 \rightarrow 1$ transition may be regarded as a $1 \rightarrow 0$ transition in reverse, and the same conditions apply. Finally, for a $0 \rightarrow 0$ transition, there are no additional constraints.

3.3 Hazard-Free Covers

A *hazard-free cover* is a cover of a function which is hazard-free for a *set* of specified input transitions. The following theorem formulates the hazard-free covering problem [14].

Theorem 3.1. A set of implicants *C* is a hazard-free cover for function *f* with respect to a specified set of input transitions if and only if: (a) each *required cube* of *f* is contained in some implicant in *C*; and (b) no implicant of *C illegally intersects* any specified dynamic transition.

An implicant which does not illegally intersect any dynamic transition is called a **dynamic-hazard-free implicant** (or **dhf-implicant**). Only dhf-implicants may appear in a hazard-free cover. A *dhf-prime implicant* is a dhf-implicant contained in no other dhf-implicant. Using the above theorem, the *two-level hazard-free logic minimization problem* is to find a minimum-cost cover of a function using only dhf-prime implicants where every required cube is covered. This unate covering problem is a variant of the classic two-level minimization problem, where each ON-set minterm of a function must be covered by a prime implicant. An exact hazard-free two-level minimizer has been developed [14], based on the above theorem.

4 A Complete Synthesis Path: Overview

Synthesis of combinational circuits is usually performed in two steps: *technology-independent logic minimization* followed by a *technology-mapping* step. This approach is aimed at reducing the complexity of the problem. We propose an approach to synthesis for low power in a specified environment (i.e. assuming complete knowledge about possible input transitions) that follows the same paradigm. An outline of our synthesis path for a function f and a specified environment (P, T) is as follows:

1. Exact two-level hazard-free logic minimization targeting low-power consumption;
2. Technology mapping of the obtained sum of products to complex CMOS gates.

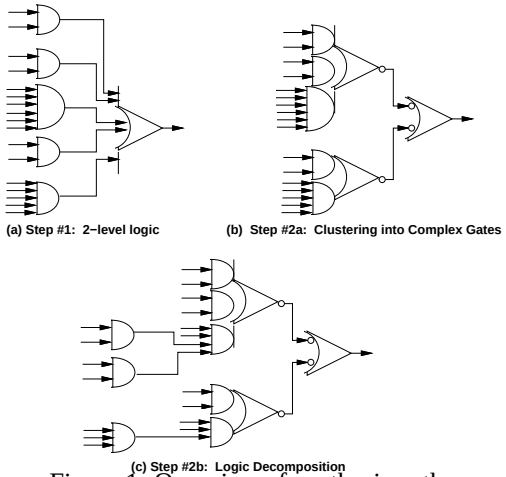

Figure 1: Overview of synthesis path

Our approach is illustrated in Figure 1. The first step produces a two-level logic representation of f that has minimum power (for the specified set of transitions T) among all hazard-free implementations.

The second step is a mapping step that combines products into complex gates to exploit technology-dependent properties to reduce the amount of dissipated power (*clustering* may "hide" transitions). Finally, since complex CMOS gates usually have a limited stack size, products containing more than k literals (k is usually 3 or 4) need to be split into several levels of CMOS gates.

The next two sections explain the steps in detail.

5 Step #1: Two-Level Hazard-Free Minimization for Low Power

The first step of the synthesis path is a technology-independent two-level hazard-free minimization for low power. As is standard in many asynchronous sequential synthesis methods [12, 17], the inputs to this step are: (i) an incompletely-specified Boolean function, and (ii) a set of specified input transitions. The input transitions typically are chained in an *input sequence*, specifying primary-input changes and present-state changes, in order. The goal of this synthesis step is to produce a hazard-free two-level circuit consuming minimum energy per input transition.

To simplify the presentation, we make two restrictions. First, we assume that each specified input transition $t \in T$ occurs with equal

[1] Sequential synthesis methods, which use hazard-free minimization as a substep, include constraints in their algorithms such that no transitions with function hazards are generated [13, 17].

probability, i.e. we assume a uniform distribution of input transitions [2]. Second, we use a cost function for products that is an upper bound for its switching activity for the given set of specified transitions. Both restrictions can easily be relaxed: the first one by simply rewriting the expressions for energy consumption ($E(p_i)$ and $E(f)$) so that they take non-uniform distributions into account, and the second by considering Boolean walks for one of the subcases (for details, see [15]). Thus, it is possible to allow for arbitrary distributions and also to capture the exact cost of an implicant. That is, if the relative frequencies of input changes are given (see [2]), the result is exactly power-optimal for the given environment.

5.1 Problem Statement

The hazard-free minimization problem for low power can be stated as follows. Given a function f, and a set, T, of *specified* function-hazard-free input transitions of f, each of which occurs with uniform probability $\frac{1}{|T|}$, find a power-minimum hazard-free cover of f. (The size of a cover is incorporated as a secondary cost function, i.e. among all solutions of minimum power a solution of minimum size is selected.) We show that the problem can be reduced to a weighted unate covering problem.

5.2 Intuition

Before we move on to the formal description of the problem and its solution, an example gives some intuition of the subtle and somewhat counter-intuitive issues that are introduced by targeting low power. Note that, in order to simplify the intuitive ideas in the examples in this section, we do not consider the amount of power dissipated by gate inputs – the actual algorithm does.

Hazard-free minimization computes a cover of dhf-primes of minimum cardinality. The following example shows that a cover of dhf-primes of *minimum* cardinality may consume more power than a cover of dhf-primes of *higher* cardinality.

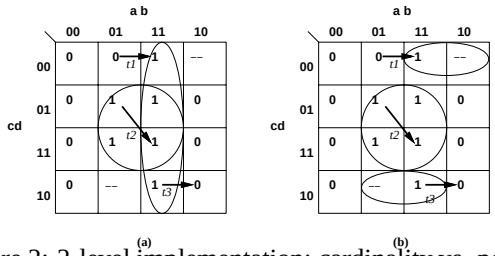

Figure 2: 2-level implementation: cardinality vs. power

Example. Consider the Boolean function in Figure 2. The left cover contains 2 dhf-prime-implicants while the right cover contains 3 dhf-prime-implicants. Consider the specified 2 dynamic transitions ($1 \rightarrow 0$ and $0 \rightarrow 1$) and 1 static $1 \rightarrow 1$ transition.[3] In both covers, each dynamic transition causes exactly one product to make a transition. However, for the static $1 \rightarrow 1$ transition, product AB in the left cover makes a transition, while no product in the right cover makes a transition. Therefore, the cover of greater cardinality has less power consumption. This example shows that a different cost function for dhf-prime-implicants is needed. □

Similarly, one can show that it is not sufficient to consider dhf-prime-implicants only. A function may have a power-minimum cover that also includes non-dhf-prime implicants [15].

The main idea can now be summarized as follows: We generate the set of "interesting" implicants, which we call *dhf-power-prime-implicants* to follow the terminology of Iman and Pedram. We assign them a specific cost function and then solve the corresponding weighted unate covering problem. Although this is related to Iman and Pedram's underlying idea, our approach is different, since we model both glitching energy and temporal dependence, while they do not. Furthermore, we additionally have to impose constraints to ensure that the generated dhf-power-prime-implicants are also dhf-implicants (i.e. do not cause glitches at the function output).

[2] It is important to note that we are *not* assuming a uniform distribution of input *changes*: we *still* observe temporal dependence, since an input transition is a *correlated* change from one input vector to another.

[3] Note that the input transitions are not "chained" in a connected sequence in this example. In reality, they would be, but we ignore this to simplify the exposition.

5.3 Cost of an Implicant

Unlike the 2-level logic minimization problem targeting area where each product is assigned the cost 1 (or a cost according to its literal count), in logic minimization for low power, the cost of a product must be computed from its behavior in the specified environment. Our analysis considers both glitching activity[4] and input dependence (*i.e.*, specified multiple input changes).

In Section 2 we expressed the energy consumption of products $E(p_i)$, and sum of products $E(f)$, in terms of transitions (Nterms). We now define the computation of the N terms.

$N(f,t) = 1(0)$ if t is a dynamic (static) transition. In a hazard-free implementation, the function output f (*i.e.* OR-gate) changes at most once per transition. It changes if and only if t is a dynamic transition.

$N(x,t) = 1(0)$ if literal x changes (does not change) in transition t. Each input x changes at most once during an input transition (see Section 3).

$N(p,t)$ represents the number of switches on product p during transition t. The switching activity on gate p depends on *how*, or *whether*, p intersects the transition t. There are 5 cases – interestingly, since the cover must be hazard-free, the analysis is simplified since only certain intersections may occur.

1. p does not intersect t, then $N(p,t) = 0$. In this case, p never turns on during transition t.
2. t is a dynamic transition and p intersects t, then $N(p,t) = 1$. This simple result is due to the hazard-free covering requirement. From Section 3, no product p can intersect t unless it intersects the start point of t. Other intersections are illegal. As a result, p will switch only once.
3. t is a static $1 \rightarrow 1$ transition and p covers t, then $N(p,t) = 0$. In this case, p remains at 1 throughout the transition; it does not switch.
4. t is a static $1 \rightarrow 1$ transition and p covers either its start or endpoint but not both, then $N(p,t) = 1$. In this case, p is 1 initially (0), and goes to 0 eventually (1). It therefore switches once.
5. t is a static $1 \rightarrow 1$ transition and p intersects t but intersects neither its start point nor end point, then $N(p,t) = 2$ (p *could* switch twice.)

Note that above cost functions can be extended to multi-output implicants, by weighting the gate transitions by the gate's fanout.

5.4 Implicant Generation

The set of dhf-power-prime implicants (dhf-PPI's) can now be defined as:

$$dhf(p) \wedge (\forall q : dhf(q) \wedge p \subset q \Rightarrow E(p) < E(q))$$
$$\wedge (\forall r : dhf(r) \wedge p \supset r \Rightarrow E(p) \leq E(r) \vee req(p) \supset req(r))$$

Here, $req(p)$ represents the set of required cubes covered by p, and $dhf(p)$ is a predicate which is true if and only if p is a dhf-implicant.

Our approach is related to that of Iman and Pedram [9], but with several important differences. First, not every product contained in a dhf-implicant is a dhf-implicant: further reduction may be needed to reach the next dhf-implicant. Therefore, the early cutoff criterion of Iman and Pedram cannot be used. Second, the cost function used is different from [9], since ours is environment-driven (*i.e.*, driven by specified input transitions and considering glitching behavior). Finally, we "skip over" a dhf-implicant (*e.g.*, p), during dhf-PPI generation, if it has a sub-dhf-implicant (*e.g.*, r) that covers the same required cubes but has less power cost (E(r)). As a result, the initial dhf-primes are not all dhf-PPI's. (In contrast, in [9], all primes are PPI's.) A recursive generation algorithm can be found in [15].

5.5 2-Level Power Minimization

The exact hazard-free minimization problem for low power can now be summarized as follows. Given a function-hazard free Boolean function f, and a specified environment (T, P):

1. Compute the set of dhf-power-prime-implicants $dhf\text{-}PPIs$.
2. Find a cover $\{p_1, \ldots, p_n\}$ of f such that $\sum_{1 \leq i \leq n} E(p_i)$ is minimal by solving the weighted covering problem (set of required cubes, dhf-PPIs, E(.)).

[4] Note that in a hazard-free 2-level circuit, individual products may glitch. However, the function output may not glitch.

6 STEP #2: CLUSTERING AND TECHNOLOGY MAPPING TO COMPLEX CMOS GATES

The technology-mapping step starts from a given sum of products generated by Step #1. The main idea is that a good packing of products into complex CMOS gates may hide transitions at AND-gate outputs. (AOI-type complex gates are used to combine the products.) For example, assume two products switch from 0 to 1 for a certain transition. Placing them into the same complex gate means that only one output transition occurs for the input transition, i.e. the amount of energy is reduced.

This step is composed of two substeps. The first substep is aimed at clustering the products (with possible overlap) into complex gates. Given a hazard-free 2-level implementation of $f = p_1 + \cdots + p_n$, and the maximal number of products placeable into a cluster max, we compute a power-minimal clustering $C_1, \ldots C_l$ of f, i.e. $\cup_i C_i = \{p_1, \ldots, p_n\}$ and $\forall i : |C_i| \leq max$. An algorithm that generates all possible clusters is given in [15]. In general, the cost of a cluster may be significantly lower than the sum of costs of the products which it contains (as pointed out above). In particular, the *cost* $E(C,t)$ of C with respect to input transition t is defined as follows: $E(C,t) = N(C,t) + \sum_{p_i \in C} \sum_{x \in lit(p_i)} N(x,t)$. Here, $N(C,t)$ represents the number of switches of C for transition t. $N(C,t)$ is given as follows (4 different cases): If no product $p \in C$ intersects t, then $N(C,t) = 0$. If t is a dynamic transition and some product $p \in C$ intersects t, then $N(C,t) = 1$. If t is a static $1 \rightarrow 1$ transition and there exists a product $p_i \in C$ that completely contains t, then $N(C,t) = 0$. If t is a static $1 \rightarrow 1$ transition and no product $p_i \in C$ completely covers t, then $N(C,t) = k_1 + k_2 + k_3$. Here, $k_1(k_2)$ is 1 if at least one product covers the start (end) point, and otherwise 0. $k_3 = \sum_{p_j \in C} N(p_j,t)$ such that p_j intersects t, but contains neither the start point nor end point. The clustering problem can therefore also be reduced to a weighted unate covering problem, where each product must be covered.

The second substep is a simple (heuristic) decomposition to actual gates, which takes into account that the stack size of a complex gate is bounded, by splitting products with more than a technology-dependent constant of inputs. The goal of this step is to perform the decomposition in a power-optimal way (for details, see [15]). The result is a low-power hazard-free circuit mapped to complex gates.

7 EXPERIMENTAL RESULTS

Table 3 compares *Base* 2-level circuits with circuits obtained from our synthesis method, using *Step #1* (2-level) and *Step #2a* (clustering). The Base circuits were synthesized using an existing exact (non-power-optimal) hazard-free 2-level algorithm [14]. The circuits are optimized for both product count and literal count. That is, the Base circuits are already area-optimal, which is known to often have a positive impact on power reduction.

Column *Step #1* shows power consumption of the circuits synthesized using our new power-optimal 2-level algorithm. Column *Step #1 + #2a* shows power consumption using both our 2-level algorithm and clustering algorithm. In each case, only single-output optimization is performed. Energy is estimated by counting the transitions on primary inputs, as well as transitions on AND-gate (Step #1) and AOI-Gate (Step #2a) outputs, *over the specified set of multiple-input changes* (both state and input changes, which are provided by the FSM specification). These transition counts are added up over all the individual outputs of the circuit. a% reports the percentage of the average power consumption with respect to Base. b% reports the percentage of power consumption (wrt. Base) for an output where power was reduced most. The reduction in power dissipation of circuits after Step #1 is up to 33%, and up to 38% after Steps #1 and #2a.

Although we cannot compare directly with Pedram's non-hazard-free benchmarks [9], our results are still promising. For a single-output function, our best power reduction is 33% (*stetson-p1*), while their best reduction for single-output functions was 11% (averaged over 63 functions with same on-set ratio).

Our results after the combined Steps #1 and #2a (2-level minimization and clustering for low power) cannot easily be compared with the Base case of column 1 (2-level logic minimization for area). However, the reduction in power from column 4 to 5 still shows the benefits of our method. Power consumption is very often dominated by *input switching*, rather than *gate output switching* (this was observed while running the benchmarks shown in the table). Our savings are especially promising, since *Step #2a only affects gate output switching.* In addition, Step #2a (and #2b) are limited to *single-output* optimization.

		Base	**Our Method**	
name	*in out*		*#1/ a%/ b%*	*#1+#2a /a%/b%*
*#cache-ctrl	20 23	15315	14837/ 97/ 93	13914/ 91/ 80
dram-ctrl	9 8	362	350/ 97/ 88	290/ 80/ 63
pe-send-ifc	12 10	927	888/ 96/ 91	793/ 86/ 78
*#stetson-p1	32 33	3204	2852/ 89/ 67	2546/ 79/ 62
stetson-p2	18 22	1143	1066/ 93/ 85	952/ 83/ 73
stetson-p3	6 4	83	83/100/100	71/ 86/ 76
pscsi-ircv	8 7	156	156/100/100	132/ 85/ 75
pscsi-isend	11 10	501	489/ 98/ 89	423/ 84/ 76
*pscsi-pscsi	16 11	8004	7596/ 95/ 84	7429/ 93/ 81
pscsi-tsend	11 10	519	501/ 97/ 83	433/ 83/ 82
pscsi-tsend-bm	11 11	521	507/ 97/ 83	450/ 86/ 82
sd-control	18 22	1312	1270/ 97/ 86	1131/ 86/ 78
sscsi-isend-bm	10 9	559	555/ 99/ 95	479/ 86/ 77
sscsi-trcv-bm	10 9	531	516/ 97/ 93	444/ 84/ 78
sscsi-tsend-bm	11 10	413	383/ 93/ 82	328/ 79/ 71

Figure 3: Comparison of power consumption of area-minimized circuits with power consumption of circuits synthesized using our synthesis path. *Due to size, only dhf-pi's (not dhf-ppi's) were generated for some outputs (power reduction using dhf-ppi's should be better). #Due to size, mincov's heuristic-mode was used.

8 CONCLUSIONS AND FUTURE WORK

This paper has made two contributions. First, we have defined a power model which incorporates both (i) *temporal dependence*, and (ii) *glitching activity*. This model can be used in both technology-independent and technology-dependent synthesis. Second, we have incorporated this model into a new synthesis method, for power-optimal asynchronous control circuits. The method includes an exact 2-level minimization step (technology-independent) and a heuristic clustering and decomposition step (technology-dependent). Initial results appear quite promising.

In the future, we believe that our *environment-driven* approach can easily be extended to *multi-level*/*multi-output* optimization algorithms such as cube- and kernel-extraction [8]. These algorithms should have further impact on both (i) area and (ii) input switching, since these algorithms significantly reduce primary input fanout and load.

REFERENCES

[1] I. Bahar and F. Somenzi. Boolean techniques for low power driven re-synthesis. In *ICCAD*, pages 428–432, November 1995.

[2] P.A. Beerel, K.Y.Yun, S.M. Nowick, and P.-C. Yeh. Estimation and bounding of energy consumption in burst-mode control circuits. In *ICCAD*, 1995.

[3] G. Birtwistle and A. Davis, editors. *Asynchronous Digital Circuit Design*. Springer-Verlag, 1995.

[4] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, April 1992.

[5] O. Coudert and R. Haddad. Integrated resynthesis for low power. In *Int. Symp. on Low-Power Design*, pages 169–174, August 1996.

[6] S. Devadas and S. Malik. A survey of optimization techniques targeting low power VLSI circuits. In *32nd DAC*, pages 242–247, June 1995.

[7] M. Favalli and L. Benini. Analysis of glitch power dissipation in CMOS IC's. In *Int. Symp. on Low-Power Design*, pages 123–128, April 1995.

[8] S. Iman and M. Pedram. Logic extraction and factorization for low power. In *32nd DAC*, pages 248–253, June 1995.

[9] S. Iman and M. Pedram. Two-level logic minimization for low power. In *ICCAD*, pages 433–438, November 1995.

[10] J. Monteiro and S. Devadas. Techniques for the power estimation of sequential logic circuits under user-specified input sequences and programs. In *Int. Symp. on Low-Power Design*, pages 33–38, April 1995.

[11] F. Najm. Power estimation techniques in integrated circuits. In *ICCAD*, pages 492–499, November 1995.

[12] S.M. Nowick and B. Coates. Uclock: Automated design of high-performance unclocked state machines. In *ICCD*, October 1994.

[13] S.M. Nowick and D.L. Dill. Synthesis of asynchronous state machines using a local clock. In *ICCD*, pages 192–197, October 1991.

[14] S.M. Nowick and D.L. Dill. Exact two-level minimization of hazard-free logic with multiple-input changes. *IEEE Transactions on CAD*, 14(8):986–997, 1995.

[15] S.M. Nowick and M. Theobald. Synthesis of low-power asynchronous circuits in a specified environment. Technical Report CUCS-020-97, Columbia University, 1997.

[16] H. Vaishnav and M. Pedram. Delay optimzal partitioning targeting low power VLSI circuits. In *ICCAD*, pages 638–643, November 1995.

[17] K.Y. Yun and D.L. Dill. Automatic synthesis of 3D asynchronous finite-state machines. In *ICCAD*, November 1992.