# 18-760 Fall'01 VLSI CAD Paper Review 2

**Out: Wed Oct. 31, 2001.** **Due: TBD in class**

## 1.0 Intent

One of the goals of 18-760 is that you acquire enough about the "fundamental" ideas of CAD algorithms to be able to read new papers and see where they borrow from known techniques, and where they innovate. Even just this far through this class, you (should) now know a **lot** about basic Boolean representation, manipulation, verification, etc.

The intent here is for you to write a short review (not to exceed **4** typed 8.5 X 11 pages, in a reasonable font, *including* any figures and tables) analyzing the attached paper:

- V. Tiwari, S. Malik, P. Ashar, "Guarded Evaluation: Pushing Power Management to Logic Synthesis/Design," *IEEE Transactions on CAD*, 1996.

You already know about logic synthesis and mapping. Turns out there are many tricks to "restructure" or "augment" logic-level designs so as to minimize the overall power consumption. This is one particularly successfull approach.

## 2.0 Objectives

We want you to summarize the paper, analyze what new ideas it is offering, connect it to other ideas that are already well-understood (in this case, logic synthesis), and critique how well the results presented actually measure up to the goals set forth by the author.

You can regard this as preparation for one of two scenarios:

- Your boss in some company has seen this paper and thinks it may offer a solution to a pressing CAD problem. But your boss is a busy person; she's got other stuff to do than just read these things and figure out if they really work. So she asks **you** to write a summary **evaluating** whether this looks like a good idea.

- You actually have to write a program to solve a problem like the one being described here. To clarify your own thinking, you want to write up a summary for you and your fellow CAD hackers that tries to ferret out whether the assumptions, proposed solution techniques, and experimental results, really make **sense** and offer a viable solution strategy for this problem.

# 3.0 Report Style

The scoring sheet at the end lists the various components of the write-up that we are expecting to see. Here we enumerate a couple of "failure modes" to avoid:

- **Repeating without Summarizing:** We have the paper in front of us. We don't need to see whole paragraphs recopied and passed off as analysis. **Do** summarize very briefly the background technical material of the paper in your report. **Do** provide a detailed summary of what the paper is about technically, with the critical ideas explained. But, **don't** just copy it all down wholesale. We want you to summarize the interesting stuff, to extract the essential ideas and write them in your own style. Repeat when you need to put a chunk of results or assumptions from the paper into the summary, but not when analyzing these results or decisions.

- **Repeating without Clarifying:** Some of the details are going to be messy in the paper. It's your job to read the text, figure out the "big ideas" of what's going on, and summarize these in your report. Remember, you are trying to **explain** this to someone (perhaps someone who decides if you get a raise or a corner office with a window). This will require some thinking (preferably before writing).

- **Diagrams vs. Verbiage:** Sometimes a small, clear picture is much easier to understand than 1 full page full of contorted prose. On the other hand, sometimes a lucid little paragraph beats a tortured figure (especially if the figure was inexplicable in the original paper). The same goes for putting in equations: sometimes they help if done carefully. It's your judgement call about how to mix these up so that the result is maximally understandable. Given that there are *not* a lot of pictures in the paper, probably drawing a small example and carefully showing how the ideas in the paper would apply to it, to illustrate exactly how the algorithm works, would be a good idea.

- **Long versus short:** It is in fact possible to do this in a fairly short (< 4 pages) report, but it's rather hard, especially if you don't have a lot of experience writing these sorts of summaries. If in doubt, go for clarity: use the 4 page limit. But, don't go over 4 pages, or we knock off points. 4 pages means **4 pages**, not 4.5, not 6, and certainly not 10+.

- **Read the scoring criteria!!** Every year a few people write a report that entirely misses points we expected to see, because we listed them on the grading sheet on the following page. Don't let this happen to you. **Read the directions.**

# 18-760 Fall'01 VLSI CAD
## Paper 1: Guarded Evaluation [100 pts]

**NAME**:

**Problem Goals and Motives [20 pts]:** What is the paper trying to solve? What assumptions are they making that determine the style of their solution?

**Contrast with "Ordinary" Logic Synthesis [20 pts]:** You already know about multilevel synthesis. Obviously, this paper has some new ideas. What are the central differences?

**Solution Strategies [20 pts]:** How do they actually solve their formulation of the problem? What are the extensions to known prior approaches? What are the new ideas they offer? How well do they actually describe these strategies? Are there any holes or vague parts?

**Experimental Plan and Results [20 pts]:** What sort of experiments do they do to suggest that their ideas really work? Are the benchmarks, measured results, *etc*., convincing? Are there any holes or suspicious decisions here? Does anything really not work the way they advertised it should?

**Writing Style (Yours, not Theirs) [20 pts]:** Professional, neat, word-processed, coherent, grammatically clean, *etc*. Good diagrams or equations where/if they make sense. Does your prose convince us of the correctness of your opinions about the ideas in the paper? Would we read this and give you *raise*, or an office in the *basement*?

A copy of the paper is attached here.

# Guarded Evaluation: Pushing Power Management to Logic Synthesis/Design

Vivek Tiwari, Sharad Malik                     Pranav Ashar
Dept. of Electrical Engineering            C&C Research Labs.
Princeton Univ.                                    NEC USA

**Abstract**

The need to reduce the power consumption of the next generation of digital systems is clearly recognized at all levels of system design. At the system level, power management is a very powerful technique and delivers large and unambiguous savings. The ideas behind power management can be extended to the logic level. This would involve determining, which parts of a circuit are computing results that will be used, and which are not. The sections that are not needed are then "shut off". This paper describes an approach termed *guarded evaluation*, which is an implementation of this idea. A theoretical framework and the algorithms that form the basis of the approach are presented. The underlying idea is to automatically determine the parts of the circuit that can be disabled on a per clock cycle basis. This saves the power used in all the useless transitions in those parts of the circuit. Initial experiments indicate substantial power savings and the strong potential of this approach for a large number of benchmark circuits. While this paper presents the development of these ideas at the logic level of design – the same ideas have direct application at the register transfer level of design also.

## 1 Guarded Evaluation

We believe in the strength of power management and its unambiguous power savings. We also believe that this idea can be pushed to lower levels of the digital system design. In particular, in this paper, we demonstrate the use of power management at logic level synthesis/design using a technique we call *guarded evaluation*. The essential idea here is to dynamically detect, on a per clock cycle basis, which parts of a logic circuit are being used and which are not. The ones that are not, can then be shut off. This is done by ensuring that no logic transitions propagate through this logic. Gating the clock inputs of existing latches/flip-flops/registers in a given RTL description is one way to do this. This is effective when it is known that the logic fed by the latch is not being utilized during the current clock cycle. This idea has been used in the functional aspects of logic design for a long time. Its utility in terms of power reduction is also known by now, but not completely exploited [1, 2].

This idea can be pushed further to achieve power savings that may not be possible through just the gating of existing latches/registers. As an example, consider a two operation ALU which is used for either addition or shifting. This is typically implemented using an adder and a shifter, and then selecting the result of one of them using a multiplexor as shown in Figure 1. In any clock cycle only one of the two
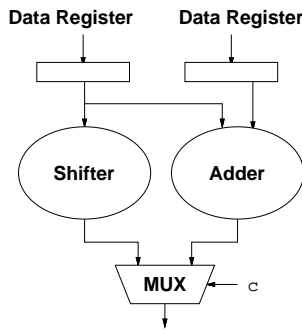
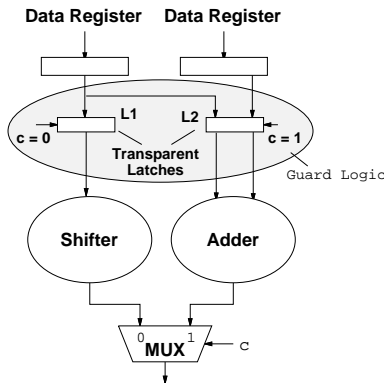Figure 1: Example RTL Circuit: Dual Operation ALU



Figure 2: Example RTL Circuit: Dual Operation ALU with Guard Logic

functions, addition or shifting, needs to be computed. However, the multiplexor does the selection only *after* both units have completed their evaluation. Clearly the evaluation of one of the two units could have been avoided. Direct gating of the clock input of the data registers will not work in this case. This is because the same data register feeds both the adder and the shifter. Duplicating this register is certainly a possibility, but may not be an acceptable solution if this register could be one of many possible ones from a register file. The duplication would involve duplicating the entire register file – certainly an expensive proposition. Further, if the inputs to the adder and shifter were from some other logic or a bus, even this would not be a possibility.

We propose a technique termed *guarded evaluation* that overcomes both of these limitations and accomplishes the task of preventing logic computation in modules when the results will not be used. We place *guard logic*, which consists of a transparent latch with an enable, at the input to each of the parts of the circuit that need to be selectively turned off. If the module is to be active in a clock cycle, the enable signal makes the latch transparent, permitting normal operation. If not, the latch retains its previous state and no transitions propagate through the inactive module. This is illustrated in Figure 2. The idea of avoiding transitions in operators that are not selected by multiplexors has also been independently reported as a low power logic design technique in [7]. Thus, such design techniques are beginning to see industrial use. The contribution of this paper is to provide a generalization of this technique, as well as a fully automated

implementation of this.

On a more abstract note, consider the operation of an arbitrary combinational logic circuit in any one clock cycle. Events propagate from the primary inputs through the circuit, and finally result in events that possibly cause the primary outputs to change. While there is switching activity at a large number of gates in the circuit, not all of this switching is useful. A large number of events in the circuit will never propagate to the primary outputs, instead being blocked somewhere in the circuit. An event is said to be blocked at a gate, if it does not influence the output of the gate. For example, consider a 2-input AND gate, with one input already set to 0. Any switching at the second input is blocked, since it cannot change the output of the gate from its 0 value. Thus, this switching is useless. It is precisely this switching that this work attempts to eliminate. The idea is to determine on a per clock cycle basis, which events in the circuit will be useless and prevent them from occurring.

The idea of using a transparent latch as a signal barrier is not new, it has been used in the past to prevent glitches from propagating through logic in the design of multipliers [10, 12]. However, the enabling condition on the latches in that case is activated after certain time has elapsed, and not by a logical condition that is true. Also, in the work on pre-computation based logic synthesis [1], this use of latches as barriers has been suggested. We would like to emphasize that the contribution of this paper is not the use of transparent latches as barriers, but rather the development of an automated technique that exploits the fact that different parts of a logic circuit are not performing useful functions in different clock cycles. A theoretical framework and algorithms are presented to automatically determine these unneeded portions, which are then "disabled" or "powered down". The use of latches as guarding barriers is just an obvious mechanism for this disabling or powering down.

The following section provides an overview of the formal concepts behind guarded evaluation. Section 3 describes how these ideas have been implemented into an automated method. The details of the experimental procedure used to evaluate these ideas and the results obtained are discussed in Section 4.

## 2   Formal Overview

Consider an arbitrary combinational logic circuit $C$. Let $x$ be some signal in the circuit. Let $F$ be the set of gates in $C$ that are being used to compute $x$ and no other signal. Let $I$ be the set of inputs to $F$. This is illustrated in Figure 3(a). Let $ODC_x$ refer to the set of primary input assignments to $C$ for which the value at $x$ has no influence on the value of the primary outputs [4]. These are the *observability don't care* primary input assignments for $x$. Thus, for these primary input assignments the value on $x$ is not required to compute the primary outputs. Let $s$ be any arbitrary signal in $C$ which satisfies the condition $s \Rightarrow ODC_x$, i.e $\bar{s} + ODC_x \equiv 1$. Thus, when $s = 1$, the value on $x$ is not needed to compute the primary outputs. Let $t_e(I)$ be the earliest time (with respect to the clock edge origin) that any signal in $I$ can switch when $s = 1$. Let $t_l(s)$ be the latest time that $s$ stabilizes at value 1. If $t_l(s) < t_e(I)$
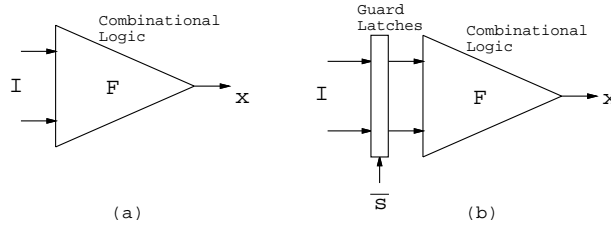
3

Figure 3: Pure Guarded Evaluation

then $s$ can be used to control the guard logic for $F$ as shown in Figure 3(b). In this figure the latches are enabled when $s = 0$ and disabled when $s = 1$. Since $x$ is not needed to compute the primary outputs when $s = 1$, it is logically correct to "shut off" $F$, by disabling the latches at the inputs of $F$. Disabling the latches ensures that the inputs to $F$ do not switch, and thus none of the gates in $F$ switch. The condition $t_l(s) < t_e(I)$ ensures that this shut off is "in time", i.e. the latches are disabled before any of its inputs can make a transition. This ensures that for this primary input vector, none of the gate outputs in $F$ make any transitions and thus guarantees maximal power reductions. If this condition is not met exactly, then this may result in some transitions passing through the latches and propagating through the logic in $F$. However, it does not compromise the correctness of the logical operation of the circuit. Thus, inaccuracies in timing modeling and estimation, which are inevitable at the logic synthesis level, can at worst result in some loss of power savings. While this is undesirable, it is certainly acceptable. This application of the idea of guarded evaluation is referred to as *pure guarded evaluation*; it directly shuts off parts of the logic that will not be used in a clock cycle by means of the guard logic, without modifying the logic in any other way. Thus, carefully hand-crafted logic by expert designers is left largely untouched.

The applicability of this idea can be extended easily if some additional change in the logic is permitted. Let us relax the logical condition on signal $s$. Let us assume that $s$ satisfies the condition $s \Rightarrow (x + ODC_x)$, i.e. $\bar{s} + x + ODC_x \equiv 1$. Clearly this is a weaker condition since it contains the condition $s \Rightarrow ODC_x$. Let us assume that the temporal condition $t_l(s) < t_e(I)$ still holds. Consider the use of $\bar{s}$ as the enabling condition on the guard latches in Figure 3(b). Consider the following possible cases:

- For primary input assignments for which $s = 0$: In this case there is no problem, since the logic in $F$ is being used to compute $x$.

- For primary input assignments which are contained in $ODC_x$ and for which $s = 1$: Again the circuit in Figure 3(b) is logically correct, since the value of $x$ is not needed at the primary outputs for these assignments.

- For primary input assignments which are not contained in $ODC_x$ and $s = 1$: In this case, there is a problem since $F$ is being shut off, while the the value at $x$ is needed to compute the primary outputs. Thus, this circuit will function incorrectly for these assignments. Note, however, that in this case
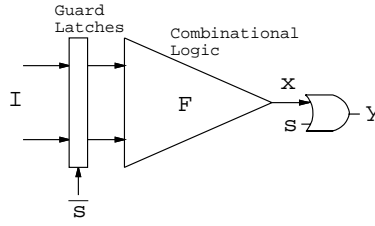
4

Figure 4: Extended Guarded Evaluation

$x$ must be 1, since $s \Rightarrow (x + ODC_x)$. Thus, if in this case $x$ could be set to a 1 while $F$ was shut down, then correct functionality will be restored. This is accomplished by using a simple OR gate as shown in Figure 4 and using signal $y$ wherever $x$ was needed. In this figure, when $s = 0$, $F$ is used to compute $x$, and $y$ is the same as $x$. When $s = 1$, then either the value of $x$ is not needed, or it should be 1. In either case, $y$ is set to 1. This logic transformation is similar to what is done in logic synthesis using global flow [3, 11]. The motivation there is to use this additional gate to help simplify other parts of the logic. Our motivation is to find a larger set of conditions under which we can shut off parts of the logic.

The condition $s \Rightarrow (x + ODC_x)$ is actually the contrapositive of the following condition used in automatic test pattern generation (ATPG): $\bar{x} \overset{D}{\Rightarrow} \bar{s}$. This is read as: $x = 0$ $D$-implies $s = 0$ [11]. In the context of test pattern generation for stuck at faults, this condition indicates that in order to test the stuck-at fault, $x$ stuck-at-1, $s$ must be set to 0, i.e. there are no test vectors for this fault with $s = 1$. Thus, existing ATPG tools can be directly used to determine the pairs $(s, x)$ for which $\bar{x} \overset{D}{\Rightarrow} \bar{s}$, or equivalently, $s \Rightarrow (x + ODC_x)$ holds.

The exposition in this section has been in terms of only one polarity for $s$ and $x$. All possible combinations of their polarities are actually used.

This application of guarded evaluation is referred to as *extended guarded evaluation*, since it involves the addition of some additional logic besides the guard logic. The advantage of using extended guarded evaluation over the pure form is that it permits the shut off of $F$ under a larger set of conditions. However, this comes at a price of adding some additional logic which contributes to additional delay and area.

## 2.1 Relationship with Pre-computation

Recently a powerful class of techniques collectively called *logic pre-computation* has been proposed as a way to reduce the power consumption of logic circuits [1]. Pre-computation also uses the idea of eliminating transitions in logic blocks by using the enable inputs of storage elements (equivalent to gating clocks), or using additional transmission gates and latches. Thus, both pre-computation and guarded evaluation share the common mechanism of power reduction by means of transition blocking. While this mechanism is the same, the two approaches differ in how and where the transitions are blocked. The goal of pre-
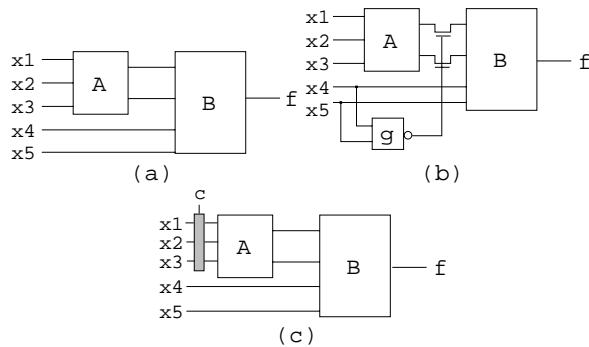
Figure 5: Pre-computation and Guarded Evaluation

computation, as the name suggests, is to derive a pre-computation circuit, that, under some conditions does the computation for all or part of the circuit. Thus, under these conditions, the corresponding circuit/sub-circuit does not have to be active. In order to accomplish this, the original circuit may need to be resynthesized. For example, in the MUX based pre-computation architecture, a given circuit computing $F$ needs to be resynthesized to derive circuits for $F(x = 0)$ and $F(x = 1)$ where $x$ is the control variable for the multiplexor. In this case, carefully hand designed logic cannot be directly used. The goal of guarded evaluation, again as the name suggests, is to determine when parts of the original circuit can be shut down using existing signals from the circuit, i.e., the sub-circuit evaluation is guarded by these signals. The original circuit does not have to be resynthesized to discover these possibilities. It does not need derive any new circuit to dynamically substitute for the main circuit or some sub-circuit in it. While some additional circuit elements are added to enable guarding, most of the original circuit is left untouched.

Since pre-computation is a collection of techniques and not a single algorithm, it is hard to do a more direct comparison of the two approaches. The pre-computation work presented in [1] mostly focusses on sequential pre-computation, where the pre-computation is done one cycle before the computation results are needed. Combinational pre-computation has been introduced in that paper but only a brief description is given there. Figures 5 (a) and (b) are taken from that paper and illustrate the combinational pre-computation described there. Function $f$ is being computed using two sub-functions $A$ and $B$ as shown in Figure 5(a). Function $g$ is used to control the transmission gates in the pre-computation based circuit shown in Figure 5(b). $g = 0$ is the set of conditions under which $f$ does not depend on the inputs $x1, x2, x3$ and has been derived accordingly. Thus, when $g = 0$, the transmission gates can be shut off and transitions occurring at the output of block $A$ will not propagate through block $B$. Figure 5(c) shows what guarded evaluation would do in this case. It would search for a signal $c$ in the circuit (as opposed to synthesizing $g$) such that the output of block $A$ is not being used when $c = 0$, and use that to control the guarding latches *at the input of block $A$*, as opposed to the outputs. If no such $c$ can be found, then the circuit will not be modified at all. The reason for placing the latches at the inputs (and not the outputs) of $A$ is that in this case the transitions that occur in block $A$ can also be saved and are not needed. In the pre-computation

6

circuit shown in Figure 5(b), there will be transitions in block $A$, even when $g = 0$. Another practical illustration of the difference between pre-computation and guarded evaluation is provided by the circuit of Figure 1. None of the prescribed precomputation techniques can save the switching activity of the inactive functional block, without duplication of the data register. As discussed in Section 1, this duplication may not always be acceptable or feasible.

# 3    Implementation

As described in the previous section, extended guarded evaluation involves guard latches (referred to as *guards* from here on), logic that generates the controlling (or guarding) signals for the latches (referred to as *guarding signal logic* from here on), and some extra gates that are needed to preserve circuit functionality (referred to as *extension gates* from here on). The most general statement for the problem of guarded evaluation is - determine the guarding conditions and the associated overhead, such that the resulting circuit has the least power consumption. The overall space for choosing the guarding conditions is very large. Any arbitrary function of the inputs of the circuit can be used as a guarding signal if it satisfies the conditions described in Section 2. The space gets even larger under extended guarded evaluation, which is the model that is used in the subsequent discussion. Searching this space can be very computationally expensive. In addition, large guarding signal logic is undesirable due to the potentially larger power, area, and delay overhead. Therefore we restrict the choice of guarding signals to signals that pre-exist in the circuit. If a pre-existing signal in the circuit is used for guarding, then that signal can directly control the guards with no other logic overhead. Greater power savings may be attained, however, if more than one pre-existing signal is used for guarding. This is because (1) A single signal may be effective in guarding only a particular portion of the entire circuit. Other signals may be more effective for other parts. Using more than one signal may help to guard a greater part of the circuit. (2) The number of input vectors for which a guard is effective can be increased if a Boolean OR of more than one signal is used to control the guard. While there is the additional overhead of the guarding signal logic (an OR gate), the guard itself is shared.

## 3.1    Implementation Overview

The chosen implementation method works as follows: In the first phase, single pre-existing signals are evaluated in terms of the potential power savings attained if each signal was to act *alone*. In the second phase, a limited number of candidate single signals are selected. Different combinations of the candidate signals are then evaluated to determine the power savings attained, when all the signals in the given combination are used *together* for guarding. The overall flow of the implementation methodology is shown below. Steps 0 and 1 constitute phase 1, and steps 2, 3, and 4 constitute phase 2.

**Step 0:** Initial circuit

**Step 1:** Evaluate single controlling signals

    **Step 1.1:** Select signals to evaluate

    *for each selected signal*

        **Step 1.2:** Determine portion of the circuit guarded, guards, extension gates, and potential benefit

**Step 2:** Select subset of controlling signals

**Step 3:** Evaluate combinations of controlling signals

    **Step 3.1:** Generate a combination

    **Step 3.2:** Evaluate combination

**Step 4:** Select final combination and generate circuit

## 3.2  Implementation Details

### 3.2.1  Step 0:

The initial circuit can be either mapped or un-mapped. However, mapped circuits are preferable, since more accurate delay analysis using library parameters can then be used.

### 3.2.2  Step 1:

In this step, single signals are evaluated for the potential benefit obtained, if each signal was to act alone.

### 3.2.3  Step 1.1:

For guarding to be most effective, the guarding signal should arrive at the controlling input of a guard earlier than the transition that travels through the shortest path from the primary inputs to the guard. Therefore, signals which arrive early can potentially guard more gates than signals that arrive later. Signals are thus ranked in increasing order of arrival times. A user-defined fraction of the earliest signals is then processed in Steps 1.2 and 1.3. This pruning step is not necessary but is simply an efficiency tradeoff, as the later arriving signals are less likely to be better candidates than the earlier signals.

### 3.2.4  Step 1.2:

Given a candidate signal $s$ and a phase $a$, $a \in \{0, 1\}$, this step determines what parts of the circuit a given signal $s$ can guard, when $s = a$, $a \in \{0, 1\}$. For this the following needs to be determined:

a) The set of gates that are guarded by $s = a$

b) The set of locations where guards are required

c) The set of locations where extension gates are required

This information is obtained by a procedure whose basic flow is as follows: First all gates in the transitive fanout of the transitive fanin of $s$ are listed in a *depth first* order, i.e., a gate precedes all gates that are in its transitive fanin. All gates are initially unmarked. The top unmarked gate is then considered. A check is now performed to see if if $s = a$ can guard the logic that computes $x$. As described in Section 2, under the extended guarded evaluation model, a signal $s$ can guard the logic that computes $x$ when $s = a$, if $(s = a) \Rightarrow ((x = b) + ODC_x)$, for $a, b \in \{0, 1\}$. The current implementation, however, does not use this form of implications. Due to the presence of the $ODC$ term, obtaining these implications after each iteration is very expensive. Exact logical implications are used instead, i.e., $(s = a) \Rightarrow (x = b)$, for $a, b \in \{0, 1\}$. Though these implications represent a restricted condition, their use is much more computationally efficient. The current implementation determines the logical implications using OBDDs [5]. These can also be determined using ATPG-like search techniques [11, 6].

If $(s = a)$ does imply a value on $x$, an extension gate is recorded for this gate. The exact extension gate required depends on the value implied on x. Then, starting from $x$, the gates in its transitive fanin are visited in a depth-first recursive fashion. Each gate that is visited is marked and is recorded as a gate that is guarded.

The terminating condition for recursion is when a gate $y$ is reached, for which the earliest arrival time $t_e(y)$ (corresponding to shortest path from primary inputs) is less than $t_l(s) + t$, where $t_l(s)$ is the (latest) arrival time of the guarding signal $s$, and $t$ is a user-defined threshold value. In this case guards will be placed at the outputs of the gates fed by $y$. The appropriate value of $t$ depends on the circuit parameters of the guard and a positive value indicates a conservative approach to ensure that no transitions leak through the guard. $t_e(y)$ is adjusted to reflect the fact that the load due to a guard may be different from the original load seen by $y$.

Whenever recursion reaches a gate $y$ with multiple fanouts, an implication check is made to see if $s = a$ implies a value on $y$. If no value is implied, recursion is terminated, and a guard has to be placed on $y$. For example, in Figure 6(a), if recursion has flowed through the fanout branch 1, a guard should be placed at the output of $y$, *but* only on branch 1. If guards are placed anywhere on the transitive fanin of $y$, or if branches 2 or 3 are fed through the guard output rather than the original output, the functionality of the logic fed by branches 2 and 3 will change. The reason is that the functionality restoring extension gates are only present on the transitive fanouts of branch 1. Of course, if a guard was already present at gate $y$, branch 1 can now be fed through the guard and no additional guard is needed. This is illustrated in Figure 6(b), where a guard was already present on branch 2. Another scenario is when all the fanouts of $y$ are fed by a guard as shown in Figure 6(c). In this case, when $s = a$, $y$ is not needed by any of its fanouts, and thus, the transitions that occur in the logic that computes $y$ are useless. The guard at $y$ is removed and $y$ is treated as a new starting point for the recursive procedure. In effect, guards will now be placed

9
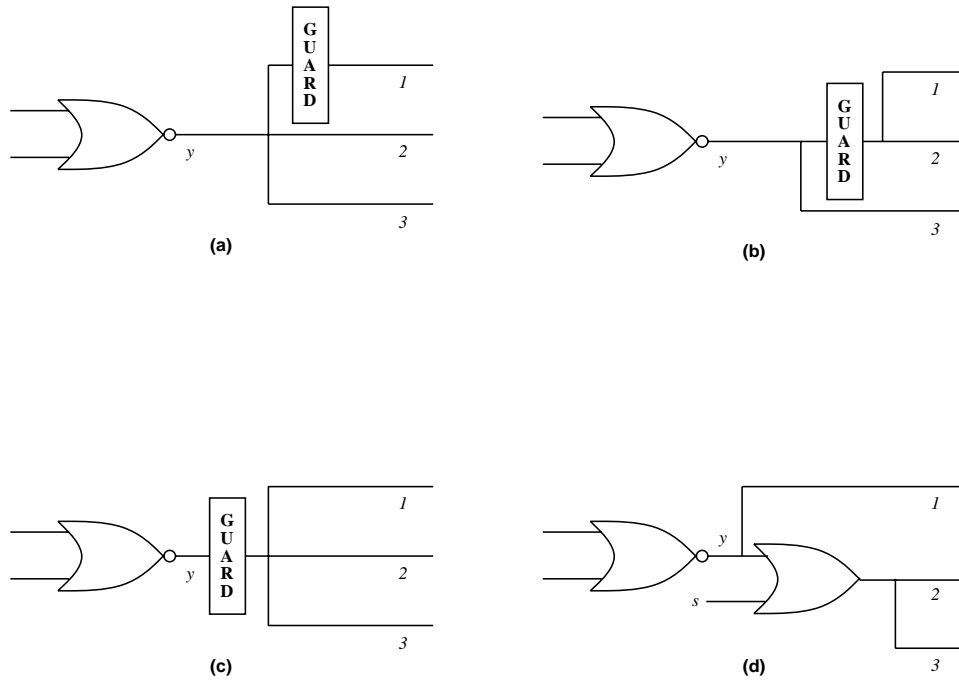
Figure 6: Handling Multiple-fanout Points

somewhere on the transitive fanins of $y$.

The final case to consider during recursion is when a multiple fanout gate $y$ is reached, where either $y$ or $\bar{y}$ *is* implied by $s = a$. In this case recursion can continue beyond $y$, i.e. $y$ and its fanins can be considered as guarded. What is required is that the other fanouts of $y$ be fed by an appropriate extension gate. For example, in Figure 6(d), if $(s = 1) \Rightarrow (y = 1)$, and recursion has reached $y$ through fanout branch 1, fanout 2 and 3 should be fed by $s$ OR $y$. If in subsequent steps, recursion reaches $y$ through branch 2, it can then be fed directly by $y$. Recursion will also stop at $y$ in this case, since the presence of the extension gate means that $y$ and its fanins have already been visited and counted. If all the fanouts of $y$ ultimately get directly fed by $y$, the extension gate can be removed.

When recursion returns back to the initial root gate, the next unmarked gate that can be guarded is selected from the list of unmarked gates. The above recursive procedure is then repeated on its transitive fanin. The procedure ends when all the gates have been marked.

The procedure described above and the current implementation does not support multiple-output gates, but it is possible to extend the procedure to handle these. The extension is straightforward and involves handling these gates in a manner analogous to what is done for multiple fanout points. The difference is that unlike the branches of a multiple fanout node, the different outputs of a multiple output gate may have different global functions and this has to be accounted for.

The computationally dominant part of this entire method is implication checking. From the above description it would appear that $O(n^2)$ implication checks are required, where $n$ is the number of signals

10

in the circuit. In practice though, the number of implication checks is only a fraction of $n^2$. The pruning in Step 1.1 significantly reduces the number of signals that are evaluated as controlling signals. Further, in Step 1.2, implications checks are only made where necessary. For circuits for which OBDDs can be built, running times have been found to very short for most circuits, and acceptable for others.

For circuits for which OBDDs cannot be constructed, ATPG-like search techniques are needed for implication checks [11, 6]. Additionally, logic simulations with random inputs are performed as a pre-processing step. This eliminates the need to perform a large number of implication checks. For example, if signal $A = 0$ and signal $B = 1$ for a certain input vector, we know that $A = 0$ cannot imply $B = 0$, and $B = 1$ cannot imply $A = 1$. Storing the results for some random input vectors can significantly reduce the number of checks that need to be fully evaluated. This leads to acceptable running times even for very large industrial circuits. A version of guarded evaluation based on this idea has been implemented as part of an industrial CAD framework. Details of the industrial implementation are beyond the scope of this paper and will not be further discussed. The results presented in the next section are for the OBDD based implementation.

The number of gates guarded is a measure of the guarding effectiveness of $s = a$. However, the actual impact of guarding by $s = a$ on the average power consumption of the whole circuit also depends on how frequently the condition $s = a$ is expected to occur over the typical input-space. Therefore, the figure of merit used to evaluate the guarding effectiveness of $s = a$ is:

$$P(s = a) \times num\_gates\_guarded_{s=a}$$

$P(s = a)$ is equal to $P_s$, if $a = 1$, and $1 - P_s$, if $a = 0$, where $P_s$ is the signal probability of $s$ and has been used by researchers in the past to estimate power consumption [13, 8]. Given the signal probabilities of the primary inputs, $P_s$ is obtained by a traversal of the OBDD of $s$. The accuracy of the above figure of merit depends on the accuracy of the probability values supplied for the primary inputs. In a real application environment, the input signal probabilities should be determined from an analysis of typical input traces. Since this information is not available for the standard benchmark circuits, signal probabilities of primary inputs are assumed to 0.5 for the purpose of the results presented in this paper. The method, however, is completely general and can use any set of input signal probabilities. As an alternative to probabilistic methods, $P_s$ can also be obtained directly by simulation over typical input traces.

### 3.2.5   Step 2:

In this step, a specified number, $n$, of controlling signals are selected as candidates for evaluating combinations of multiple signals. The signals are first ranked in decreasing order of their figures of merit, which were determined in the previous step. Selection proceeds by picking up the current top candidate $s_1$ and testing if for equivalence against the candidates already selected. If $s_1$ is functionally equivalent to an

already selected candidate $s_2$, then $s_1$ is not selected. In addition, if $s_1$ is contained in the set of signals guarded by an already selected signal $s_2$, it is again not selected. This is because the use of $s_2$ as a controlling signal can lead to a change in the logic functionality of $s_1$, invalidating any guarding opportunities that require $s_1$ as control.

### 3.2.6 Step 3:

This step estimates the power savings possible by the simultaneous use of multiple guarding signals. Different combinations of the $n$ signals, selected in the previous step, are generated, and the power savings attained by each combination are evaluated. The following sub-steps are needed.

### 3.2.7 Step 3.1:

If $n$ is small, all combinations of the selected signals can be tried out. Currently this is the method used. For larger $n$, for the sake of efficiency, it may be beneficial to adopt a faster, though possibly less effective, search strategy.

### 3.2.8 Step 3.2:

This step evaluates the power saving possible when a given subset(combination) of selected signals, $(s_1 \ldots s_n)$, is used for guarding. Without going into the actual implementation details, the basic idea is as follows. First, determine the complete set of guards and extension gates needed. Let these be $L$ and $E$, respectively. Also determine the complete set of gates guarded, $G$. Then estimate the power savings attained due to guarding of the gates in $G$. Let this be $\mathcal{P}_G$. $\mathcal{P}_G$ is calculated as follows. Consider a gate $g \in G$. Without loss of generality, let $g$ be included in the set of gates guarded by $(s_1 \ldots s_k), k \leq n$. Let $s = s_1 + s_2 \ldots + s_k$, where $+$ indicates Boolean OR. Now using the traditional, zero delay, temporal independence model for power calculation [13, 9], the power consumption of gate $g$ is $\mathcal{P}_g = (2 \times P_g \times (1 - P_g) \times C_g \times A)$, where $P_g$ is the signal probability of $g$, and $C_g$ is the total capacitance at the output of $g$, and $A$ is a constant[1]. Since $g$ is guarded whenever $s = 1$, the power savings may appear to be $P_s \times \mathcal{P}_g$, where $P_s$ is the signal probability of $s$, i.e., probability that $s$ equals 1.

However, this is not completely accurate, since it is not necessary that $g$ would have had a transition in the original circuit, for every input vector for which $s = 1$. The probability of $g$ having a $1 \rightarrow 0$ transition in the original circuit, for an input vector for which $s = 1$ is given by $P_g \cdot P_{\bar{g} \cdot s}$, where $P_{\bar{g} \cdot s}$ is the signal probability of $\bar{g} \cdot s$ and "$\cdot$" stands for Boolean AND. Similarly the probability of a $0 \rightarrow 1$ transition in the original circuit, when $s = 1$ is given by $P_{\bar{g}} \cdot P_{g \cdot s}$. From this it follows that the total power saving for all the

---

[1] $A = 0.5 \times V_{DD}^2$, where $V_{DD}$ is the supply voltage

guarded gates under the given combination of controlling signals is:

$$\mathcal{P}_G = \sum_{g \in G} (P_g \cdot P_{\bar{g} \cdot s} + P_{\bar{g}} \cdot P_{g \cdot s}) \times C_g \times A$$

where for each $g$, $s$ is the Boolean $OR$ of the subset of the controlling signals that guard $g$. $C_g$ is obtained from the library parameters of the given gates [15], and the signal probabilities are obtained from OBDDs.

The power consumed in the guards, extension gates, and the guarding signal logic constitutes the power overhead associated with guarded evaluation. To estimate the power consumed in the guards, it should be noted that a guard's output switches only when the guarding condition is not true, i.e., when the controlling signal on the guards allows transitions to pass through. Using the reasoning followed above, the power consumed in the guards is given by:

$$\mathcal{P}_L = \sum_{l \in L} (P_l \cdot P_{\bar{l} \cdot \bar{s}} + P_l \cdot P_{l \cdot \bar{s}}) \times C_l \times A$$

where $P_l$ is the signal probability of the node at which the guard is present, and for each guard $l$, $s$ is the Boolean $OR$ of the subset of the controlling signals that share $l$.

The extension gates also consume power and this power is estimated. Let $\mathcal{P}_E$ be the sum of the power consumption of all the extension gates. Various combinations of the controlling signals may be required to control the different guards and feed the different extension gates. The logic associated required to generate these combinations also consumes power. The power consumed in this logic is also estimated. Let this be $\mathcal{P}_K$.

Thus, given a subset of controlling signals $S = (s_1 \ldots s_n)$, the figure of merit for evaluating the combination is the net power saving achieved, and this is given by:

$$\mathcal{P}_S = \mathcal{P}_G - \mathcal{P}_L - \mathcal{P}_E - \mathcal{P}_K$$

The above estimates have some sources of inaccuracy. First, the use of the zero-delay model for calculating transition probabilities ignores the effect of *glitches* (spurious transitions). Power consumption due to glitches may increase the power cost of the original circuit. On the other hand, power saved by the use of guarding, $\mathcal{P}_G$, may also be higher in this case, since guards block glitches too. However, consideration of glitches requires extra computational effort that may be prohibitively high for larger circuits. The zero-delay model is used since it is generally accepted as representing a reasonable accuracy versus efficiency tradeoff. Another source of inaccuracy is that the insertion of guards, extension gates, and guarding signal logic can change the timing relationships between signals. Therefore, it is recommended that the estimated power savings be validated with accurate simulation. This has been done for the results presented in the next section.

### 3.2.9   Step 4:

The combination of controlling signals that yields the maximum power savings is selected and the final circuit, incorporating the guards, guarding signal logic, and extension gates is generated.

## 4   Experimental Results

The implementation described in the previous section has been carried out in the framework of the SIS synthesis system [14]. Experiments were conducted on a large number (84) of benchmark circuits from the MCNC and ISCAS suites.

All circuits were initially mapped using standard cell libraries. Four different libraries were tried - lib2, mcnc, msu, and NEC. The first three libraries are distributed with the SIS package. NEC is a proprietary industrial library. There is some variation in the results obtained with different libraries. The reason is that the set of implying and implied signal pairs that exist in a circuit depends on the result of mapping, since implications can be checked only for signals that are exposed at the outputs of complex gates. In addition, different mappings lead to different circuit structures, and different relative delays between signals. Of the above four libraries, lib2 and NEC are the only ones that are fully characterized for low level area, delay, and load information. Thus, for the sake of brevity, results are presented for only these two libraries. The results from the other two libraries, however, were also along the same lines.

Table 1 shows the power saving obtained by the application of the method described in the previous section. The circuit name is shown in Column 1 and the power savings obtained for the lib2 and NEC libraries are shown in Columns 2 and 3, respectively. The pruning fraction defined in Section 3.2.3 was 1, i.e., all the signals in the circuits were evaluated in Step 1 of the algorithm. This was done for the sake of completeness in the experiments, and also because the required computational effort was not prohibitively high for any of the circuits tried. The number of controlling signals that were evaluated in Step 2 of the process described in Section 3 is 3, for all circuits. Power for the final guarded circuit, incorporating all the guards, extension gates and guarding signal logic was measured using the method described in Section 3.2.6. The experiments were executed on a 60MHz SUN SPARCstation-20 workstation. The CPU times ranged from 2 seconds for C17, to 70 minutes for des.

The blank entries represent cases where no power savings were obtained, i.e., every combination of the selected controlling signals only led to an estimated increase in power. Net power saving were obtained in several cases and power savings of greater than 10% are shown in italics. Maximum savings were obtained for sao2-hdl - 65% for lib2, and 63.6% for NEC.

Table 2 provides data about the guarding mechanism used in each circuit that demonstrated net power savings. Column 2 shows the number of primary inputs and Column 3 shows the number of primary outputs. Columns 4 to 7 and Columns 8 to 11 show data for the lib2 and NEC libraries, respectively.

14

| Circuit | % Power reduction | | Circuit | % Power reduction | |
|---------|--------|--------|---------|--------|--------|
|         | lib2   | NEC    |         | lib2   | NEC    |
| 5xp1     |        |        | frg1     |        | 4.656  |
| 5xp1-hdl |        |        | frg2     | 27.087 | 18.564 |
| 9sym     | 1.279  |        | i1       |        |        |
| 9sym-hdl |        |        | i3       |        |        |
| 9symml   |        |        | i6       | 2.296  |        |
| C1355    |        |        | i8       |        |        |
| C17      |        |        | i9       |        |        |
| C432     |        |        | k2       |        |        |
| C499     |        |        | lal      |        |        |
| C880     |        |        | majority |        |        |
| alu2     |        |        | misex1   |        |        |
| alu4     | 0.396  |        | misex2   |        |        |
| alupla   |        |        | misex3   |        | 4.906  |
| apex7    |        |        | misex3c  |        |        |
| b1       |        |        | mux      | 21.434 | 9.146  |
| b9       |        |        | my_adder |        |        |
| bw       |        |        | parity   |        |        |
| c8       |        |        | pcle     |        | 20.265 |
| cc       |        |        | pcler8   |        | 16.054 |
| cht      | 7.164  |        | pm1      |        |        |
| cm150a   | 12.988 |        | rd53     |        |        |
| cm151a   | 0.012  |        | rd53-hdl |        |        |
| cm152a   |        |        | rd73     |        |        |
| cm162a   |        |        | rd73-hdl |        |        |
| cm163a   |        |        | rd84     |        |        |
| cm42a    |        |        | rd84-hdl |        |        |
| cm82a    |        |        | rot      |        |        |
| cm85a    |        |        | sao2     | 35.809 | 54.583 |
| cmb      |        |        | sao2-hdl | 65.061 | 63.593 |
| comp     |        |        | t        |        |        |
| con1     |        |        | tcon     |        |        |
| cordic   |        |        | term1    | 27.463 | 22.005 |
| count    |        |        | too_large| 39.789 | 35.226 |
| cu       | 2.474  |        | ttt2     | 6.252  |        |
| dalu     |        | 18.698 | unreg    | 7.278  |        |
| decod    | 1.037  |        | vda      |        |        |
| des      | 6.130  | 4.028  | vg2      |        |        |
| duke2    | 0.523  | 16.022 | x1       |        |        |
| example2 |        |        | x2       |        |        |
| f2       |        |        | x3       | 12.097 | 1.972  |
| f51m     |        |        | x4       |        |        |
| f51m-hdl |        |        | z4ml     |        |        |

Table 1: Power savings for lib2 and NEC libraries

Column 4 shows the number of controlling signals used in the final circuit. These signals are determined by Step 4 of the procedure described in Section 3. Column 5 shows the number of guards and Column 6 shows the number of extension gates used. Column 7 shows the percent of the gates of the original circuit that are fully guarded in the final circuit. A gate is fully guarded if there is a guard on every path from the gate to the primary inputs. The numbers in this column provide insight about the potential effectiveness of guarding for a given circuit. The actual power saving estimates, however, depend on the probability of the guarding conditions and the overhead associated with guarding. These estimates are shown in the previous table.

The variation in the effectiveness of guarded evaluation has to do with the logical and topological structure of the subject circuit. The logical structure determines the existence of guarding opportunities. The topological structure of the circuit plays a role in determining the overhead involved in exploiting any observed guarding opportunities. The relative delays of signals, and points of multiple fanout and reconvergence determine the number of guards and extension gates that are needed. In certain cases, the power overhead associated with these elements can outweigh any power savings due to guarding. As can be seen from the table, this method is not universally applicable. However, there are several cases where large net power savings are obtained. The key point to be noted is that guarded evaluation provides an automated method for identifying such cases. It is only for cases where the method reports substantial power savings that the guarded version of the circuit should be generated and considered for further analysis. Further analysis involves evaluating the area and delay overheads, and possibly detailed simulation for determining the exact power savings.

Table 3 shows the area and delay overhead associated with guarding, for the circuits that exhibited net power savings in Table 1. The area overhead comes from the area of guards, extension gates, and the guarding signal logic. The insertion of guards and extension gates can also increase the delay of critical paths. The outputs of the guarding signal logic may have to feed a large number of guards. Thus, fanout buffer optimization may be needed for these signals. Columns 2 and 7 show the number of gates in the original mapped circuit for the lib2, and NEC libraries, respectively. Columns 3, 5, 8, and 10, show the original area and delay for the lib2, and NEC mapped circuits, respectively. These values are in terms of the basic units used in the respective libraries. The percent increase in area and delay after guarding is shown in Columns 4, 5, 9, and 11. Fanout buffer optimization of the guard controlling signals actually reduces the final circuit delay in some cases. This can happen when the controlling signal comes directly from the original circuit, as opposed to the output of an added guarding signal logic gate. The area and delay overhead show a great amount of variation across the circuits. It should be noted that most of the larger overheads are for the smaller circuits - decod, cu, cm151a, etc. The small size of the circuits makes the percent overhead associated with guarding seem disproportionately high.

The estimation methodology described in Section 3.2.6 provides an estimate of the power savings

| Circuit | PI | PO | lib2 | | | | NEC | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | GSignals | Guards | Extras | %Guarded | GSignals | Guards | Extras | %Guarded |
| 9sym | 9 | 1 | 2 | 11 | 5 | 7 | | | | |
| alu4 | 14 | 8 | 1 | 74 | 10 | 56.3 | | | | |
| cht | 47 | 36 | 1 | 47 | 36 | 77.7 | | | | |
| cm150a | 21 | 1 | 1 | 21 | 1 | 75.3 | | | | |
| cm151a | 12 | 2 | 1 | 12 | 2 | 72.7 | | | | |
| cu | 14 | 11 | 1 | 14 | 10 | 71.6 | | | | |
| dalu | 75 | 16 | | | | | 2 | 118 | 24 | 48.1 |
| decod | 5 | 16 | 1 | 5 | 16 | 84.4 | | | | |
| des | 256 | 245 | 3 | 137 | 128 | 19.6 | 1 | 122 | 60 | 31.4 |
| duke2 | 22 | 29 | 1 | 33 | 20 | 50.0 | 2 | 54 | 36 | 77.4 |
| frg1 | 28 | 3 | | | | | 1 | 27 | 4 | 79.5 |
| frg2 | 143 | 139 | 2 | 163 | 86 | 68.3 | 2 | 156 | 86 | 68.7 |
| i6 | 138 | 67 | 2 | 82 | 113 | 39.8 | | | | |
| misex3 | 14 | 14 | | | | | 1 | 26 | 24 | 60.8 |
| mux | 21 | 1 | 1 | 21 | 1 | 78.3 | 2 | 25 | 6 | 82.5 |
| pcle | 19 | 9 | | | | | 3 | 12 | 10 | 54.7 |
| pcler8 | 27 | 17 | | | | | 3 | 20 | 18 | 53.9 |
| sao2 | 10 | 4 | 1 | 10 | 4 | 89.5 | 2 | 11 | 5 | 92.7 |
| sao2-hdl | 10 | 4 | 2 | 11 | 6 | 95.0 | 2 | 11 | 5 | 96.6 |
| term1 | 34 | 10 | 3 | 73 | 8 | 70.9 | 3 | 71 | 9 | 80.0 |
| too_large | 38 | 3 | 1 | 38 | 3 | 92.8 | 1 | 38 | 3 | 92.9 |
| ttt2 | 24 | 21 | 1 | 25 | 15 | 56.9 | | | | |
| unreg | 36 | 16 | 2 | 1 | 4 | 2.0 | | | | |
| x3 | 135 | 99 | 1 | 126 | 79 | 63.4 | 1 | 123 | 79 | 68.4 |

Table 2: Guarding Statistics for lib2 and NEC libraries

| Circuit | lib2 | | | | | NEC | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Gates | Area | | Delay | | Gates | Area | | Delay | |
| | | Orig | %Ovh | Orig | %Ovh | | Orig | %Ovh | Orig | %Ovh |
| 9sym | 100 | 216224 | 18.88 | 25.68 | -7.64 | | | | | |
| alu4 | 446 | 748896 | 30.11 | 59.94 | -10.93 | | | | | |
| cht | 211 | 290000 | 63.68 | 20.58 | 24.28 | | | | | |
| cm150a | 85 | 107184 | 58.44 | 18.54 | 18.90 | | | | | |
| cm151a | 44 | 54288 | 76.92 | 13.87 | 20.73 | | | | | |
| cu | 67 | 93264 | 62.69 | 9.84 | 33.19 | | | | | |
| dalu | | | | | | 1690 | 3614 | 11.48 | 14.37 | 2.62 |
| decod | 32 | 52896 | 73.68 | 8.21 | 65.98 | | | | | |
| des | 3069 | 5349456 | 10.81 | 131.60 | 0.11 | 3669 | 8400 | 6.00 | 43.48 | -0.26 |
| duke2 | 322 | 641712 | 19.38 | 39.09 | 3.29 | 474 | 1139 | 21.77 | 6.44 | 25.86 |
| frg1 | | | | | | 161 | 271 | 35.79 | 4.97 | 25.62 |
| frg2 | 1013 | 1626784 | 35.62 | 58.10 | 14.11 | 1120 | 2446 | 26.86 | 25.63 | 7.96 |
| i6 | 560 | 841232 | 47.60 | 77.76 | -21.70 | | | | | |
| misex3 | | | | | | 569 | 1460 | 9.18 | 8.71 | 0.34 |
| mux | 97 | 129920 | 46.79 | 19.04 | 20.75 | 120 | 205 | 47.80 | 4.77 | 50.40 |
| pcle | | | | | | 95 | 152 | 43.42 | 4.54 | 46.14 |
| pcler8 | | | | | | 128 | 201 | 52.74 | 4.79 | 50.40 |
| sao2 | 95 | 171680 | 21.08 | 19.54 | 14.28 | 138 | 303 | 16.83 | 5.91 | 27.22 |
| sao2-hdl | 201 | 321552 | 14.29 | 51.31 | 9.59 | 298 | 582 | 8.76 | 15.18 | 13.56 |
| term1 | 275 | 479312 | 48.60 | 27.46 | 3.04 | 380 | 832 | 30.05 | 5.85 | 27.48 |
| too_large | 532 | 956768 | 12.08 | 42.98 | 8.41 | 542 | 1403 | 9.19 | 9.27 | 14.43 |
| ttt2 | 181 | 294176 | 33.28 | 16.19 | 8.81 | | | | | |
| unreg | 121 | 164256 | 6.50 | 15.22 | 1.18 | | | | | |
| x3 | 756 | 1172992 | 39.99 | 42.86 | 13.09 | 892 | 1859 | 24.93 | 6.09 | 29.37 |

Table 3: Area and delay overhead for lib2 and NEC libraries

| Circuit | Orig Power (mW) | %Power Red | %Est Red |
|---|---|---|---|
| dalu | 19.678 | 21.10 | 18.69 |
| des | 50.346 | 3.79 | 4.02 |
| duke2 | 4.601 | 15.69 | 16.02 |
| frg1 | 1.223 | -7.20 | 4.65 |
| frg2 | 9.989 | 1.62 | 18.56 |
| misex3 | 5.835 | 11.91 | 4.90 |
| mux | 1.082 | -22.55 | 9.14 |
| pcle | 0.603 | 7.79 | 20.26 |
| pcler8 | 0.782 | -0.64 | 16.05 |
| sao2 | 1.209 | 37.55 | 54.58 |
| sao2-hdl | 6.098 | 51.48 | 63.59 |
| term1 | 3.676 | 7.34 | 22.00 |
| too_large | 4.78 | 26.03 | 35.22 |
| x3 | 8.037 | -7.70 | 1.97 |

Table 4: Results using simulation for the internal NEC library

obtained through the use of guarding. This is not an exact estimate due to the limitations of the zero-delay probability based power model, as well as the fact that the timing information may have changed in the final circuit. Thus, detailed simulations were performed in order to validate the estimated power savings. The simulator used is an accurate, gate-level, event-driven, internal NEC simulator. The simulator uses detailed timing information, and accounts for attenuation of glitches due to inertial delays. The simulator works for only the NEC library. Thus, simulations were performed only for circuits mapped with this library. The results are shown in Table 4. Since real input traces are not specified for the benchmark circuits, random input vectors were used as stimulus for simulation. 10,000 input vectors were used and these were more than sufficient to ensure convergence in all cases. Column 1 shows the power consumption of the original circuit in $mW$. Column 2 shows the percent power reduction for the guarded circuit as observed through simulation. Column 3 shows the estimated power savings, as obtained from Table 1. The observed power savings are very close to the estimates for some examples, such as the first three. For others, the actual power reduction is less than the estimate for all but one circuit. In some of the smaller examples - frg1, mux, and pcler8, the power consumption actually increases in the final circuit. However, large power savings are still exhibited for other examples - sao2-hdl, too_large, dalu, etc. This reinforces the observation that guarded evaluation is extremely effective for certain circuits. Large savings estimated by the automated methods described in the previous sections help to identify these circuits. Accurate pre-layout, and possibly, post-layout simulation can then be performed for these circuits in order to validate the estimates.

It should be noted that guards were implemented as latches for the simulation results reported above. The only latches available in the NEC library are large general purpose latches. These also have an additional, complemented output pin, which is never used, but which does contribute to the power cost. These latches, therefore, tend to be very expensive in terms of power. It is possible to design lower power latches that are more suitable as guards. Use of these latches will improve the power savings.

In addition, if dynamic logic is used, a guard can be implemented at a much lower cost, through the use of a single transmission gate, resulting in potentially much higher power savings.

We also believe that the power savings will be even greater when these techniques are applied on complete logic descriptions, rather than on individual blocks of combinational logic that are represented in the benchmark suite. The larger combinational logic that is obtained by combining several smaller combinational blocks will offer larger guarding opportunities. We intend to explore this by applying these ideas to complete digital IC logic descriptions.

The use of extension gates can create redundancies in the internal logic. This has implications for the testability of the guarded circuit. A discussion of the testing techniques that are applicable for these circuits is beyond the scope of this paper. The use of extension gates also provides an opportunity to optimize the logic in the guarded portion of the circuit, through some traditional logic synthesis techniques [3, 11].

These optimizations can lead to a reduction in area, delay, and an additional reduction in power.

## 5  Conclusions

The battle for low power VLSI is being fought on several fronts. The efforts in the diverse areas of device physics, circuit design, system design and register-transfer/logic design are all complimentary and provide gains that are independent of each other. At the device level, lowering the supply voltage and other innovations provide clear and unambiguous power savings. The same can be said of innovative circuit design techniques, as and when they become part of common design practice. At the system level, power management provides this ability by shutting off modules that are not needed. Unfortunately, at the logic design level, the emphasis thus far, has been largely on techniques that alter logic designs so as to reduce the number of transitions that occur in the circuit. While in principle this sounds good; in practice these gains tend to be small and even difficult to evaluate. This is because of the difficulty in measuring the power reduction due to a change in the number of transitions resulting from small structural modifications in the logic design.

The focus of this paper is to use the essential idea of power management, i.e., shutting of parts of the circuit that are not needed, at the logic level. At a philosophical level, this is very interesting, since it attempts to isolate the "useful" part of a logic circuit for a given input. We provide a clean characterization of how this can be done. This is followed by a specific algorithm that shows how this is implemented in practice. This is a first implementation of this idea, and we point out several areas of obvious improvement here. Even with this preliminary implementation, we have been able to demonstrate significant power savings on a set of benchmark circuits. We believe the savings will be even greater when this idea is applied to complete systems, rather than restricted to individual blocks of combinational logic. In addition, these ideas translate directly to the register-transfer level. This is part of our future research work in this area.

## References

[1] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou. Precomputation-based sequential logic optimization for low power. *IEEE Transactions on VLSI Systems*, pages 426–436, December 1994.

[2] L. Benini, P. Siegel, and G. De Micheli. Automatic synthesis of gated-clocks for power reduction in sequential circuits. *IEEE Design and Test*, December 1994.

[3] L. Berman and L. Trevillyan. Global flow optimization in automatic logic design. *IEEE Transactions on Computer-aided design*, 10(5), May 1991.

[4] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS: A Multiple-Level Logic Optimization System. In *IEEE Transactions on Computer-Aided Design*, pages 1062–1081, November 1987.

[5] R. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35:677–691, Aug. 1986.

[6] S. T. Chakradhar and V. D. Agrawal. A transitive closure based algorithm for test generation. In *Proceedings of the Design Automation Conference*, pages 353–358, June 1991.

[7] A. Correale. Overview of the power minimization techniques employed in the IBM PowerPC 4xx embedded controllers. In *Proceedings of the International Symposium on Low Power Design*, pages 75–80, Dana Point, CA, April 1995.

[8] A. Ghosh, S. Devadas, K. Keutzer, and J. White. Estimation of average switching activity in combinational and sequential circuits. In *Proceedings of the Design Automation Conference*, pages 253–259, June 1992.

[9] A. Ghosh, A. Shen, S. Devadas, and K. Keutzer. On Average Power Dissipation and Random Pattern Testability. In *Proceedings of the International Conference on Computer-Aided Design*, November 1992. To appear.

[10] U. Ko, P. Balsara, and W. Lee. A self-timed method to minimize spurious transitions in low power CMOS circuits. In *Proceedings of the 1994 IEEE Workshop on Low Power Electronics*, October 1994.

[11] W. Kunz and P. R. Menon. Multi-level logic optimization by implication analysis. In *Proceedings of the International Conference on Computer-Aided Design*, pages 6–13, November 1994.

[12] C. Lemonds and S. S. Shetti. A low power 16 by 16 multiplier using transition reduction circuitry. In *Proceedings of the 1994 Intl. Workshop on Low Power Design*, pages 139–142, April 1994.

[13] F. Najm. Transition Density, A Stochastic Measure of Activity in Digital Circuits. In *Proceedings of the Design Automation Conference*, pages 644–649, June 1991.

[14] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli. Sequential circuit design using synthesis and optimization. In *Proceedings of the International Conference on Computer Design*, pages 328–333, October 1992.

[15] V. Tiwari, P. Ashar, and S. Malik. Technology mapping for low power. In *Proceedings of the Design Automation Conference*, pages 74–79, June 1993.