Guarded Evaluation: Pushing Power Management to Logic Synthesis/Design

Vivek Tiwari, Sharad Malik
Dept. of Electrical Engineering
Princeton Univ.

Pranav Ashar
C&C Research Labs.
NEC USA

Abstract

The need to reduce the power consumption of the next generation of digital systems is clearly recognized. At the system level, power management is a very powerful technique and delivers large and unambiguous savings. This paper describes the development and application of algorithms that use ideas similar to power management, but that are applicable to logic level synthesis/design. The proposed approach is termed *guarded evaluation*. The main idea here is to determine, on a per clock cycle basis, 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", thus saving the power used in all the useless transitions in that part of the circuit. Initial experiments indicate substantial power savings and the strong potential of this approach. 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 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.

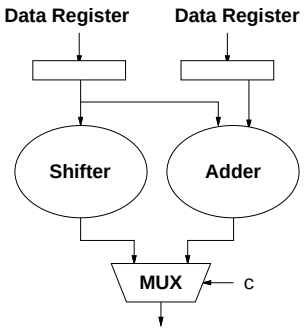

Figure 1: Example RTL Circuit: Dual Operation ALU

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.

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.

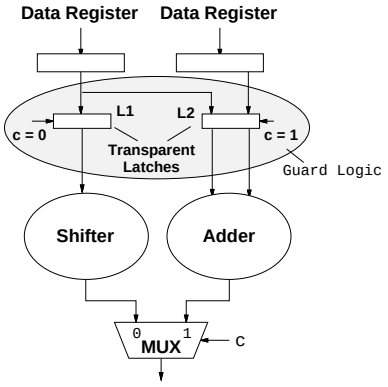

Figure 2: Example RTL Circuit: Dual Operation ALU with Guard Logic

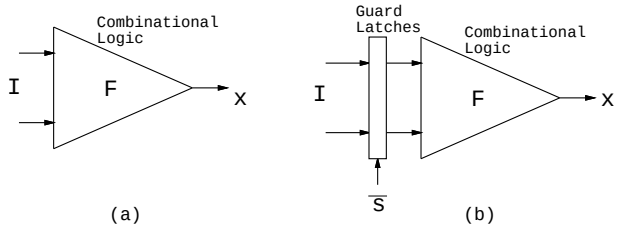

Figure 3: Pure Guarded Evaluation

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 [8, 10] in the design of a multiplier. 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 recognition and exploitation of the fact that different parts of a logic circuit are not performing useful functions in different clock cycles, and thus can be effectively "powered down". The use of latches as guarding barriers is just an obvious implementation of this idea.

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 $\overline{s}+ODC_x \equiv 1$. 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 $t_l(s) < t_e(I)$ then s can be used to control the guard logic

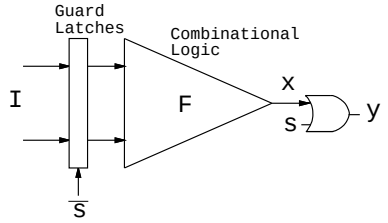

Figure 4: Extended Guarded Evaluation

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. This application of the idea of guarded evaluation is referred to as *pure guarded evaluation*; it directly shuts of 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. $\overline{s}+x+ODC_x \equiv 1$. 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 $\overline{s}$ 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 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, 9]. 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.

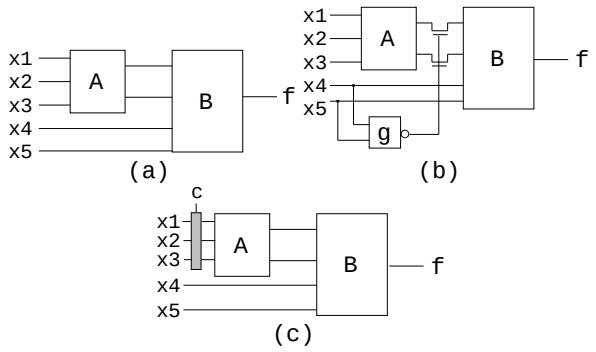

Figure 5: Pre-computation and Guarded Evaluation

The condition $s \Rightarrow (x + ODC_x)$ is actually the contrapositive of the following condition used in automatic test pattern generation (ATPG): $\bar{x} \stackrel{D}{\Rightarrow} \bar{s}$. This is read as: $x = 0$ D-implies $s = 0$ [9]. 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 $\bar{x} \stackrel{D}{\Rightarrow} \bar{s}$, 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-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. 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.

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 circuit shown in Figure 5(b), there will be transitions in block A, even when $g = 0$.

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).

3.1 Implementation Overview

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. As mentioned in Section 2, only existing signals in the circuit are used for guarding. Greater power savings may be attained using multiple pre-existing signals for guarding 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 multiple signals helps 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.

The chosen method works as follows: In the first phase, single pre-existing signals are evaluated in terms of the potential power savings they can achieve *alone*. Next, a limited number of candidate single signals are selected. Different combinations of the candidates are then evaluated to determine the combined power savings attained. The overall flow of the implementation methodology is shown below.

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 gates implied by signal
    Step 1.3: Determine 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

Step 0: A mapped initial circuit is preferable to ensure "in-time" guarding.
Step 1: 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.
Step 1.2: The signals that can be guarded by each signal chosen in Step 1.1 are determined here. This determination is done based on the relative arrival times and logical implication (*cf* Section 2). Implication incorporating ODCs is not used since obtaining such implication after each iteration is very expensive. Logical implication is determined using OBDDs [5] in the current implementation. It can also be determined using ATPG-like search algorithms.
Step 1.3: Given a candidate signal s and a phase a, $a \in \{0, 1\}$, this step determines the following for the whole circuit:

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: The set of gates x, such that, $s = a$ implies x or $\bar{x}$ are first 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. An extension gate is recorded for this gate. The exact extension gate required depends on the phases of s and x. Starting from this gate, the gates in the 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 recursion terminates when a gate y for which the earliest arrival time $t_e(y)$ is less than $t_l(s) + t$ is reached. $t_l(s)$ is the latest arrival time of the guarding signal s, and t is a user-defined threshold value. Guards are 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.

Another terminating condition is when a gate y with multiple fanouts is reached, and $s = a$ does *not* imply y or $\bar{y}$. 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

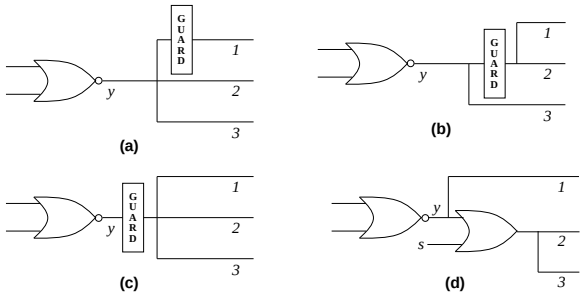

Figure 6: Handling Multiple-fanout Points

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 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 is selected from the list of implied gates, and the above recursive procedure is repeated. When all the implied gates have been visited, the procedure finally ends.

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 [11, 6]. P_s is obtained by a traversal of the OBDD of s.

Note that the exact positioning of guards has an impact on the number of gates guarded and the power savings obtained. In the above discussion, guards were moved as close to the inputs as is allowed by the timing constraint imposed by the arrival time of the guarding signal. This is because more gates can be guarded if the guards are closer to the inputs. However, it may sometimes be more beneficial to place a guard at a gate that is the sink of a re-convergent section of

the circuit. Pushing the guards beyond this sink, towards the primary inputs, can lead to guarding of more gates, but can also require more guards, whose power consumption can affect the power saving obtained. A related effect can sometimes occur at the source of a re-convergent fanout. If the source gate has a large number of fanouts, it-may be beneficial to push the guards placed on its immediate transitive fanout, to its inputs, even if that violates the timing constraints. Fewer guards are now required, though the data inputs of the guards may have to be delayed a bit to satisfy the timing constraints. The issue of the ideal positioning of latches will be explored further as part of future work.

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. Note again, that the two phases of a controlling signal are considered as separate cases.

Step 3: 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.

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 be beneficial to adopt a faster, though possibly less effective, search strategy.

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 for the evaluation 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 [11, 7], 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 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 [12], and the signal probabilities are obtained from OBDDs.

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

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, note 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 above reasoning, the power consumed in the guards is given by:

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

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:

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

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

An implementation of the algorithm has been carried out in the SIS framework. All circuits were mapped using inverters and 2-input NOR gates from the `lib2.genlib` library [2]. Only these basic gates were used because 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. Different mappings can expose different signals leading to different guarding opportunities. Using the two basic gates eliminates this degree of freedom, thus simplifying the presentation of results, while simultaneously exposing a greater number of signals, which provides for greater guarding opportunities. Our method, however, is completely general, and works with any library. In practice, a circuit should first be decomposed into basic gates to determine the guarding opportunities. It can subsequently be re-mapped using more complex gates, with the stipulation that nodes that have guards or extension gates at their outputs should be retained as gate outputs.

A large number of circuits from the IWLS 91 benchmark set were evaluated. Varying amounts of power reduction were obtained, only a subset of the circuits that yielded at least 15% power savings are shown here. We feel that this is the minimum amount of power savings that will make this method acceptable for a circuit. Table 1 shows the circuit statistics for the initial mapped circuits.

Table 2 shows the results after the application of guarded evaluation. The number of controlling signals that were

[2] The `lib2.genlib` library is distributed with the SIS package

Circuit	#Gates	#PIs	#POs	Delay
dalu4	2894	75	16	118.98
duke2	1124	22	29	47.63
frg2	2687	143	139	62.62
k2	5297	45	45	92.22
misex3	1524	14	14	54.92
mux	154	21	1	24.33
sao2-hdl	389	10	4	68.67
term1	783	34	10	32.46
too_large	1752	38	3	56.60
x3	1845	135	99	37.62

Table 1: Circuit Statistics for the Benchmark Circuits

Circuit	%Area Ovh	%Delay Ovh	%Power Red
dalu	13.5	-20.4	23.9
duke2	30.1	-12.4	39.2
frg2	19.0	11.8	38.7
k2	18.6	21.1	21.5
misex3	14.5	-37.7	29.1
mux	48.5	28.0	24.1
sao2-hdl	8.4	8.9	67.4
term1	27.3	4.7	38.6
too_large	21.5	3.2	51.6
x3	22.9	26.4	17.7

Table 2: Statistics for the Guarded Circuits

evaluated in phase 2 of the process described in Section 3 is 3, for all circuits. This number was observed to be a good choice for a majority of the circuits. Column 2 shows the percent increase in area and Column 3 shows the percent increase in delay. The area and delay overhead comes from the guards, extension gates, and the guarding signal logic. The outputs of the guarding signal logic may have to feed a large number of guards. Thus, fanout buffer optimization is used to reduce the delay of these signals. This actually reduces the final circuit delay in some cases, as shown in Column 3. This can happen when the controlling signal from the guard comes directly from the original circuit, as opposed to the output of an added guarding signal logic gate. Depending on the exact arrival times in the final circuit, the inputs of some guards may have to be delayed to prevent power loss due to early glitches, or the guards maybe moved forward in the circuit. The actual power reduction obtained is shown as a percentage in Column 6. Power for the original circuit was estimated using a zero-delay model, with capacitance values obtained from library parameters [12, 6]. The capacitance switching per transition of a guard was considered to be 3 times that of an inverter. 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. Large power savings are indicated, with the maximum being 67.4% for `sao2-hdl`. The experiments were executed on SUN SPARC 2 workstations. The CPU times ranged from 32 minutes for `dalu`, to 10.8 seconds for `cc`.

We 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. 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. 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, 9].

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] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

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