Optimization of Combinational and Sequential Logic Circuits for Low Power Using Precomputation

José Monteiro, John Rinderknecht, Srinivas Devadas
Department of EECS
MIT, Cambridge, MA

Abhijit Ghosh
MERL
Sunnyvale, CA

Abstract

Precomputation is a recently proposed logic optimization technique which selectively disables the inputs of a sequential logic circuit, thereby reducing switching activity and power dissipation, without changing logic functionality. In this paper, we present new precomputation architectures for both combinational and sequential logic and describe new precomputation-based logic synthesis methods that optimize logic circuits for low power.

We present a general precomputation architecture for sequential logic circuits and show that it is significantly more powerful than the architectures previously treated in the literature. In this architecture, output values required in a particular clock cycle are selectively precomputed one clock cycle earlier, and the original logic circuit is "turned off" in the succeeding clock cycle. The very power of this architecture makes the synthesis of precomputation logic a challenging problem and we present a method to automatically synthesize precomputation logic for this architecture.

We introduce a powerful precomputation architecture for combinational logic circuits that uses transmission gates or transparent latches to disable parts of the logic. Unlike in the sequential circuit architecture, precomputation occurs in an early portion of a clock cycle, and parts of the combinational logic circuit are "turned off" in a later portion of the same clock cycle. Further, we are not restricted to perform precomputation on the primary inputs.

Preliminary results obtained using the described methods are presented. Upto 66 percent reductions in switching activity and power dissipation are possible using the proposed architectures. For many examples, the proposed architectures result in significantly less power dissipation than previously developed methods.

1: Introduction

Average power dissipation has recently emerged as an important parameter in the design of general-purpose and application-specific integrated circuits. Optimization for low power can be applied at many different levels of the design hierarchy. For instance, algorithmic and architectural transformations can trade off throughput, circuit area, and power dissipation (e.g., [5]), and logic optimization methods have been shown to have a significant impact on the power dissipation of combinational logic circuits (e.g., [11]).

In CMOS circuits, the probabilistic average switching activity of the circuit is a good measure of the average power dissipation of the circuit. Average power dissipation can thus be computed by estimating the average switching activity. Several methods to estimate power dissipation for CMOS

combinational circuits have been developed (e.g., [6, 8]). More recently, efficient and accurate methods of power dissipation estimation for sequential circuits have been developed [12].

In this work, we are concerned with the problem of optimizing logic-level circuits for low power. Previous work in the area of sequential logic synthesis for low power has focused on state encoding [9] and retiming [7] algorithms. Recently a new sequential logic optimization method has been presented that is based on selectively *precomputing* the output logic values of the circuit one clock cycle before they are required, and using the precomputed values to reduce internal switching activity in the succeeding clock cycle [1]. The primary optimization step is the synthesis of the precomputation logic, which computes the output values for a *subset* of input conditions. If the output values can be precomputed, the original logic circuit can be "turned off" in the next clock cycle and will not have any switching activity. Since the savings in the power dissipation of the original circuit is offset by the power dissipated in the precomputation phase, the selection of the subset of input conditions for which the output is precomputed is critical. The precomputation logic adds to the circuit area and can also result in an increased clock period.

The work presented in [1] suffers from the limitation that if a logic function is dependent on the values of several inputs for a large fraction of the applied input combinations, then no reduction in switching activity can be obtained.

We introduce two new precomputation architectures in this paper. The first architecture targets sequential logic circuits and allows the precomputation logic to be a function of a subset or all of the input variables. We give an example that shows that the new architecture can reduce power dissipation for a larger class of sequential circuits than the previously developed architecture.

The second precomputation architecture targets combinational circuits. The reduction in switching activity is achieved by introducing transmission-gates or transparent latches in the circuit which can be disabled when the signal going through them is not necessary to determine the output values. This architecture is more flexible than any of the sequential architectures since we are not limited to precomputation over primary inputs. However, these degrees of freedom make the optimization step much harder.

The model we use to relate switching activity to power dissipation is the standard model described in recent literature such as [5, 6]. In Section 2 we describe the precomputation architecture presented in [1]. The two new precomputation architectures are presented in Section 3. Algorithms that synthesize precomputation logic for the new sequential architecture and for the combinational architecture are presented, respectively, in Sections 4 and 5. Preliminary experimental results are presented in Section 6.

2: Previous Work

In this section we describe the *Subset Input Disabling* precomputation architecture introduced in [1].

Consider the circuit of Figure 1. We have a combinational logic block **A** that is separated by registers R_1 and R_2. While R_1 and R_2 are shown as distinct registers in Figure 1 they could, in fact, be the same register.

In Figure 2 the *Subset Input Disabling* precomputation architecture is shown. The inputs to the block **A** have been partitioned into two sets, corresponding to the registers R_1 and R_2. The output of the logic block **A** feeds the register R_3. Two Boolean functions g_1 and g_2 are the *predictor* functions. We require:

$$g_1 = 1 \Rightarrow f = 1 \tag{1}$$

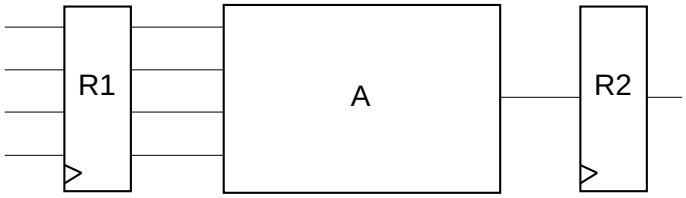

Figure 1. Original circuit

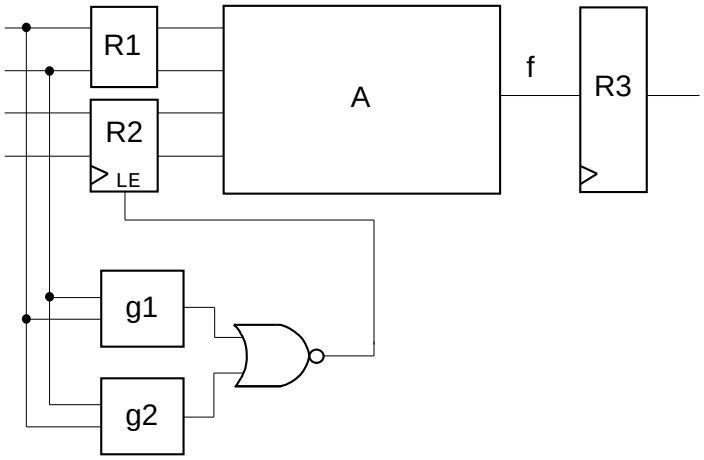

Figure 2. Subset Input Disabling precomputation architecture

$$g_2 = 1 \;\Rightarrow\; f = 0 \tag{2}$$

Therefore, during clock cycle t if either g_1 or g_2 evaluates to a 1, we set the load enable signal of the register R_2 to be 0. This implies that the outputs of R_2 during clock cycle $t+1$ do not change. However, since the outputs of register R_1 are updated, the function f will evaluate to the correct logical value. A power reduction is achieved because only a subset of the inputs to block **A** change implying reduced switching activity.

The choice of g_1 and g_2 is critical. We wish to include as many input conditions as we can in g_1 and g_2. In other words, we wish to maximize the probability of g_1 or g_2 evaluating to a 1. To obtain reduction in power with marginal increases in circuit area and delay, g_1 and g_2 have to be significantly less complex than f. This architecture achieves this by making g_1 and g_2 depend on significantly fewer inputs than f.

In [1] exact and approximate algorithms for the selection of the subset of inputs such that power savings are maximized are given.

3: New Precomputation Architectures

We describe two new precomputation architectures, the first targeting sequential circuits which is more general than the architecture presented in the previous section, and the second targeting combinational circuits.

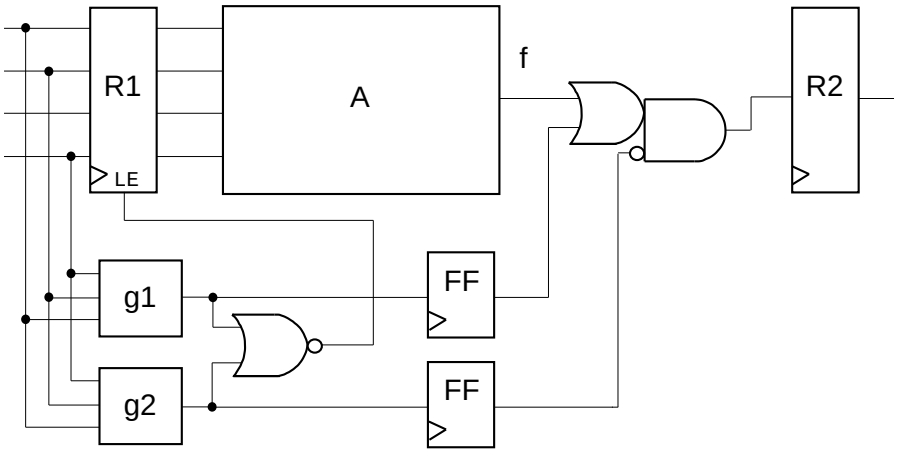

Figure 3. Complete Input Disabling precomputation architecture

3.1: New Sequential Precomputation Architecture

The basic limitation of the *Subset Input Disabling* architecture is that having chosen a subset of inputs for the precomputation logic we can only disable the input registers when the output is the same for *all* combinations over all inputs not in the selected subset. Thus even if there is only one combination for which this is not true, we cannot precompute output values since we need to know the value of input variables that are not in the precomputation logic. The *Complete Input Disabling* precomputation architecture proposed in the following section is able to handle these cases.

3.1.1 Complete Input Disabling Precomputation Architecture

In Figure 3 the new precomputation architecture for sequential circuits is shown. The functions g_1 and g_2 satisfy the conditions of Equations 1 and 2 as before. During clock cycle t if either g_1 or g_2 evaluates to a 1, we set the load enable signal of the register R_1 to be 0. This means that in clock cycle $t+1$ the inputs to the combinational logic block **A** do not change. If g_1 evaluates to a 1 in clock cycle t, the input to register R_2 is a 1 in clock cycle $t+1$, and if g_2 evaluates to a 1, then the input to register R_2 is a 0. Note that g_1 and g_2 cannot both be 1 during the same clock cycle due to the conditions imposed by Equations 1 and 2.

The important difference between this architecture and the *Subset Input Disabling* architecture shown in Figure 2 is that the precomputation logic can be a function of all input variables, allowing us to precompute any input combination. We have additional logic corresponding to the two flip-flops marked **FF** and the AND-OR gate shown in the figure. Also the delay between R_1 and R_2 has increased due to the addition of this gate.

Note that for all input combinations that are included in the precomputation logic (corresponding to $g_1 + g_2$) we are not going to use the output of f. Therefore we can simplify the combinational logic block **A** by using these input combinations as an *input don't-care set* for f.

3.1.2 An Example

A simple example that illustrates the effectiveness of the *Subset Input Disabling* architecture is a n-bit comparator that compares two n-bit numbers C and D and computes the function $C > D$.

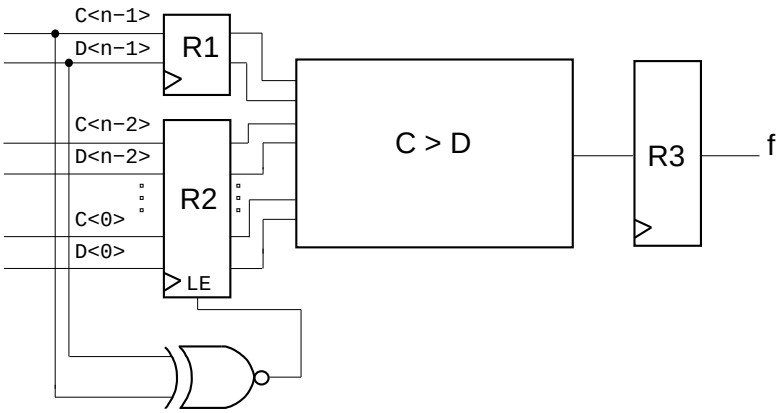

Figure 4. A comparator example

The optimized circuit with precomputation logic is shown in Figure 4. The precomputation logic is as follows:

$$g_1 = C\langle n-1\rangle \cdot \overline{D\langle n-1\rangle}$$
$$g_2 = \overline{C\langle n-1\rangle} \cdot D\langle n-1\rangle$$

Clearly, when $g_1 = 1$, C is greater than D, and when $g_2 = 1$, C is less than D. We have to implement

$$\overline{g_1 + g_2} = C\langle n-1\rangle \otimes D\langle n-1\rangle$$

where $\otimes$ stands for the exclusive-nor operator.

Assuming a uniform probability for the inputs[1], the probability that the XNOR gate evaluates to a 1 is 0.5, regardless of n. For large n, we can neglect the power dissipation in the XNOR gate, and therefore, we can achieve a power reduction of close to 50%.

Now let us consider a modified comparator shown in Figure 5 in which if C is equal to the all 0's bit-vector and D is equal to the all 1's bit-vector the result should still be 1 and vice-versa, if C is equal to the all 1's bit-vector and D is equal to the all 0's bit-vector the result should still be 0. This circuit is not precomputable using the *Subset Input Disabling* architecture because knowing that $C\langle n-1\rangle = 0$ and $D\langle n-1\rangle = 1$ or $C\langle n-1\rangle = 1$ and $D\langle n-1\rangle = 0$ is not enough information to infer the value of f. Thus, although the input combination C equal to the all 0's bit-vector and D equal to the all 1's, and the input combination C equal to the all 1's bit-vector and D equal to the all 0's bit-vector have a very low probability of occurrence, they invalidate this precomputation architecture.

Using the *Complete Input Disabling* architecture, since we have access to all input variables for the precomputation logic, we can simply remove these input combinations from g_2 and g_1, respectively. This is illustrated in Figure 6. This way we will still be precomputing all other input combinations in $C\langle n-1\rangle \otimes D\langle n-1\rangle$, meaning that the fraction of the time that we will precompute the output value is still close to 50%.

3.2: Combinational Precomputation Architecture

Given a combinational circuit, any sub-circuit within the original circuit can be selected. Assume that this sub-circuit has n inputs and m outputs as shown in Figure 7. In an effort to reduce switching

[1]The assumption here is that each $C\langle i\rangle$ and $D\langle i\rangle$ has a 0.5 static probability of being a 0 or a 1.

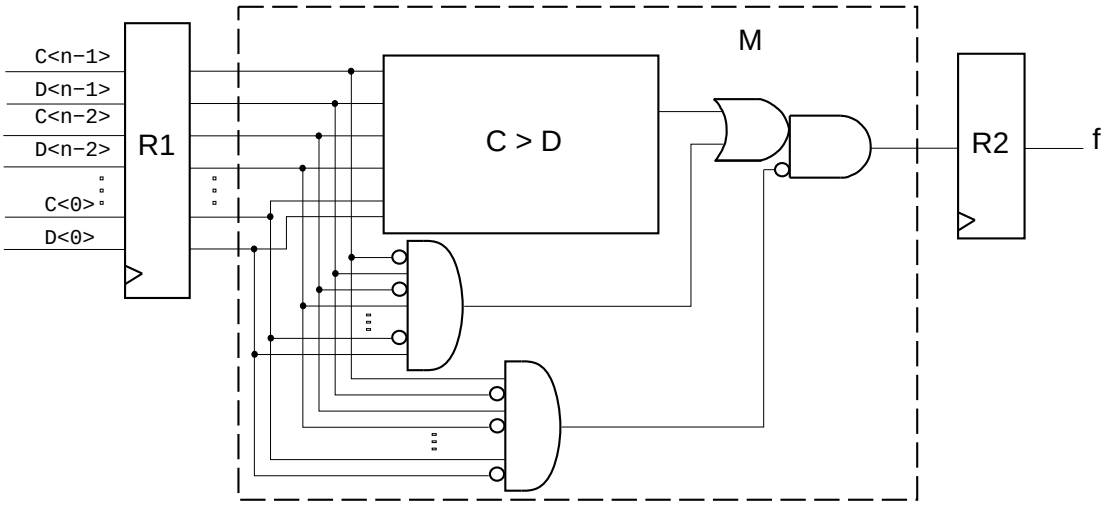

Figure 5. A modified comparator

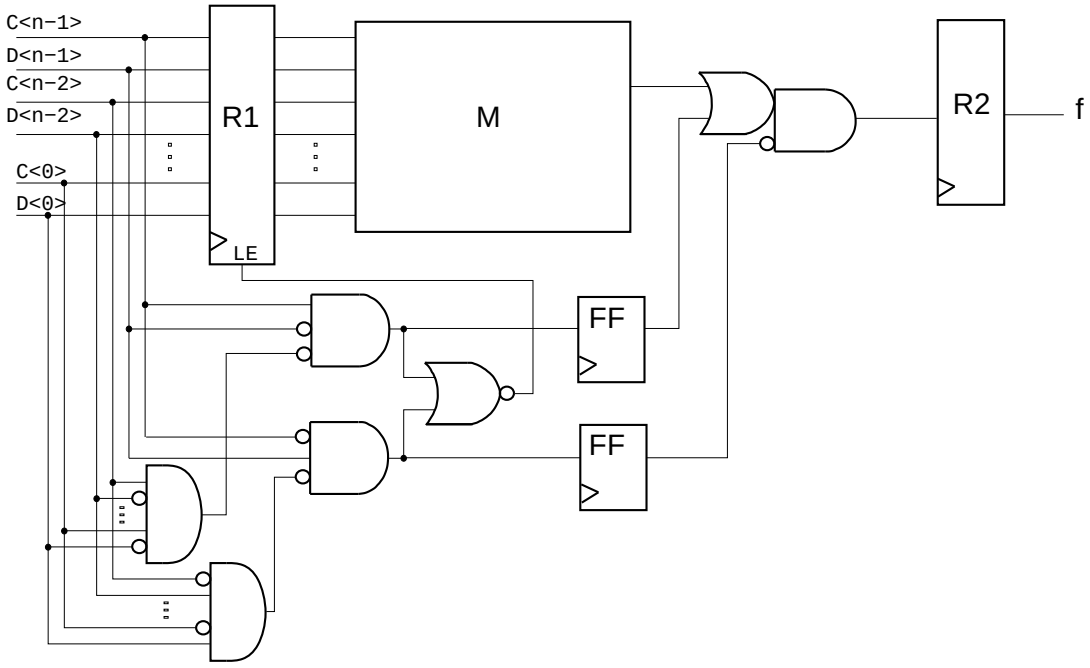

Figure 6. Modified comparator under Complete Input Disabling architecture

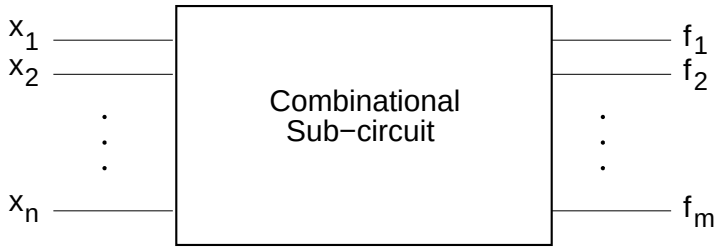

Figure 7. Original sub-circuit

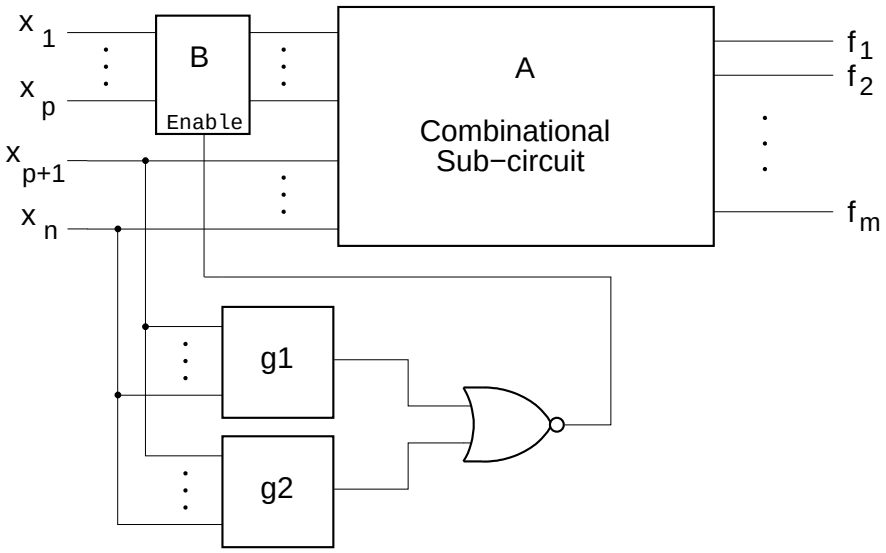

Figure 8. Sub-circuit with input disabling circuit

activity, the algorithm will "turn off" a subset of the n inputs using the circuit shown in Figure 8. The figure shows p inputs being "turned off", where $1 \leq p < n$.

The term "turn off" means different things according to the type of circuit style that is being used. If the circuit is built using static logic gates, then "turn off" means prevent changes at the inputs from propagating through block **B** to the sub-circuit (block **A**) thus reducing the switching activity of the sub-circuit. In this case block **B** may be implemented using one of the transparent latches shown in Figure 9. If the circuit is built using Domino logic, then "turn off" means prevent the outputs of block **B** from evaluating high no matter the value of the inputs. This can be implemented using 2-input AND gates as shown in Figure 10.

Blocks g_1 and g_2 determine when it is appropriate to turn off the selected inputs. The selected inputs may be "turned off" if the static value of all the outputs, f_1 through f_m, are independent of the selected inputs. To fulfill this requirement, outputs g_1 and g_2 are required to satisfy Equations 1 and 2.

If either g_1 or g_2 is high, the inputs may be "turned off". If they are both low, then the selected inputs are needed to determine the outputs, and the circuit is allowed to work normally.

Given the sequential and combinational architectures, algorithms are needed to determine which inputs to turn off and to determine the functions g_1 and g_2 so that the power consumption of the

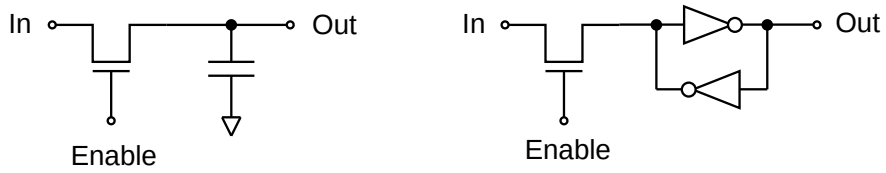

Figure 9. Disabling inputs in combinational circuits

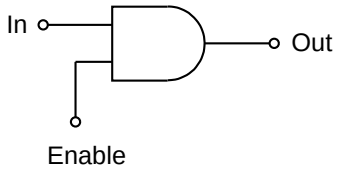

Figure 10. Disabling inputs in Domino circuits

combinational or sequential circuit is reduced. The details of these algorithms are given in the next two sections.

4: Complete Input Disabling Precomputation

In this section, we describe methods to determine the functionality of the precomputation logic for the *Complete Input Disabling* architecture targeting sequential circuits.

4.1: Precomputation Logic for Single Output Functions

The key tradeoff in selecting the precomputation logic is that we want to include in the logic as many input combinations as possible but at the same time keep the logic simple. The *Subset Input Disabling* precomputation architecture ensures that the precomputation logic is significantly less complex than the combinational logic in the original circuit by restricting the search space to identifying g_1 and g_2 such that they depend on a relatively small subset of the inputs to the logic block **A**.

By making the precomputation logic depend on all inputs, the *Complete Input Disabling* architecture allows for a greater flexibility but also makes the problem much more complex. The algorithm to determine the precomputation logic that we present in this section extends the algorithm of [1] to exploit this greater flexibility.

We will be searching for the subset of inputs that are necessary, a large fraction of the time, to determine what the value of f is. We follow a strategy of keeping the precomputation logic simple by making the logic depend *mostly* on a small subset of inputs. The difference is that now we are not going to restrict ourselves to those input combinations for which this subset of inputs defines f, we will allow for some input combinations that need inputs not in the selected set.

4.1.1 Selecting a Subset of Inputs

Given a function f we are going to select the "best" subset of inputs S of cardinality k such that we minimize the number of times we need to know the value of the other inputs to evaluate f. For each subset of size k, we compute the cofactors of f with respect to all combinations of inputs in

the subset. If the probability of a cofactor of f with respect to a cube c is close to 1 (or close to 0), that means that for the combination of input variables in c the value of f will be 1 (or 0) most of the time.

Let us consider f with inputs $x_1, x_2, \cdots, x_n$ and that we have selected the subset $x_1, \cdots, x_k$. If the probability of the cofactor of f with respect to $x_1 \cdots x_k$ being all 1's is high (prob($f_{x_1 \cdots x_k}$) ≈ 1), then over all combinations of $x_{k+1}, \cdots, x_n$ there are only a few for which f is not 1. So we can include $x_1 \cdots x_k \cdot f_{x_1 \cdots x_k}$ in g_1. Similarly if the probability of the $f_{x_1 \cdots x_k}$ is low (prob($f_{x_1 \cdots x_k}$) ≈ 0), then over all combinations of $x_{k+1}, \cdots, x_n$ there are only a few for which f is not 0, so we include $x_1 \cdots x_k \cdot \overline{f_{x_1 \cdots x_k}}$ in g_2. Note that in the *Subset Input Disabling* architecture we would only do this if $f_{x_1 \cdots x_k} = 1$ or $f_{x_1 \cdots x_k} = 0$.

Since there is no limit on the number of inputs that the precomputation logic is a function of, we need to monitor its size in order to ensure it does not get very large. We estimate how large the precomputation logic is by computing the size of its corresponding ROBDD [4].

In the sequel we describe a branching algorithm that selects the "best" subset of inputs. The pseudo-code is shown in Figure 11.

The procedure **SELECT_LOGIC** receives as arguments the function f and the desired number of inputs k to select. **SELECT_LOGIC** calls the recursive procedure **SELECT_RECUR** with four arguments. The first is the function to precompute. The second argument D corresponds to the set of input variables currently selected. The third argument Q corresponds to the set of "active" variables, which may be selected or discarded. Finally, the argument k corresponds to the number of variables we want to select.

If $|D| + |Q| < k$ it means that we have dropped too many variables in the earlier levels of recursion and we will not be able to select a subset of k input variables.

At each recursion we compute the cofactors of f with respect to all combinations over the input variables currently in D. We want to keep those cofactors that have a high probability of being 0 or 1. Our cost function is the fraction of exact cofactors found (exact meaning that the selected inputs determine the value of f) plus a factor $\frac{\text{size(BDD1)}}{\text{size(BDD2)}}$ times the fraction of approximate cofactors found (with these cofactors we still need variables *not* in D to be able to precompute f). The factor $\frac{\text{size(BDD1)}}{\text{size(BDD2)}}$ tries to measure how much more complex the precomputation logic will be by selecting these approximate factors. We can tune the value of α thus controlling how many approximate cofactors we select. The more we select, the more input combinations will be in the precomputation logic therefore increasing the fraction of the time that we will be disabling the input registers. On the other hand, the logic will be more complex since we will need more input variables. (Note that in the extreme case of $\alpha = 0$, the input selection will be the same as in [1] as all the selected input combinations depend only on the inputs that are in subset D.)

We store the selected set corresponding to the maximum value of the cost function.

4.1.2 Implementing the Logic

The Boolean operations of OR and cofactoring required in the input selection procedure can be carried out efficiently using reduced, ordered Binary Decision Diagrams (ROBDDs) [4]. In the pseudo-code of Figure 11 we show how to obtain the $g_1 + g_2$ function. We also need to compute g_1 and g_2 independently. We do this in exactly the same way, by including in g_1 the cofactors corresponding to probabilities close to 1 and in g_2 the cofactors corresponding to probabilities close to 0.

Once we have ROBDDs for g_1 and g_2, these can be converted into a multiplexor-based network

```
SELECT_LOGIC( f, k ):
{
    /* f = function to precompute */
    /* k = # of inputs to select */
    BEST_IN_COST = 0 ;
    SELECTED_SET = φ ;
    SELECT_RECUR( f, φ, X, k ) ;
    return( SELECTED_SET ) ;
}

SELECT_RECUR( f, D, Q, k ):
{
    if( |D| + |Q| < k )
        return ;
    if( |D| == k) {
        exact = approx = 0;
        BDD1 = BDD2 = 0;
        foreach combination c over all variables in D {
            if(prob(f_c) == 1 or prob(f_c) == 0) {
                exact = exact + 1;
                BDD1 = BDD1 + c;
                BDD2 = BDD2 + c;
            }
            if(prob(f_c) > 1 − α) {
                approx = approx + 1;
                BDD2 = BDD2 + c·f_c;
            }
            if(prob(f_c) < α) {
                approx = approx + 1;
                BDD2 = BDD2 + c·f̄_c;
            }
        }
        cost = (exact + size(BDD1)/size(BDD2) × approx)/2^|D| ;
        if( cost > BEST_IN_COST) {
            BEST_IN_COST = cost ;
            SELECTED_SET = D ;
        }
        return ;
    }
    choose x_i ∈ Q such that i is minimum ;
    SELECT_RECUR( f, D ∪ x_i, Q − x_i, k ) ;
    SELECT_RECUR( f, D, Q − x_i, k ) ;
}
```

Figure 11. Procedure to determine the precomputation logic

(see [2]) or into a sum-of-products cover. The network or cover can be optimized using standard combinational logic optimization methods that reduce area [3] or those that target low power dissipation [11].

4.1.3 Simplifying the Original Combinational Logic Block

Whenever g_1 or g_2 evaluate to a 1, we will not be using the result produced by the original combinational logic block **A**, since the value of f will be set by either g_1 or g_2. Therefore all input combinations in the precomputation logic are new don't-care conditions for this circuit and we can use this information to simplify this logic block, thus leading to a reduction in area and consequently to a further reduction in power dissipation.

4.2: Multiple-Output Functions

In general, we have a multiple-output function $f_1, \cdots, f_m$ that corresponds to the logic block **A** in Figure 1. All the procedures described thus far can be generalized to the multiple-output case.

The functions g_{1i} and g_{2i} are obtained by computing the cofactors of f_i separately. The function g whose complement drives the load enable signal is obtained as:

$$g = \prod_{i=1}^{m} (g_{1i} + g_{2i})$$

The function g corresponds to the set of input conditions that control the values of *all* the f_i's.

4.2.1 Selecting a Subset of Outputs

We describe an algorithm, which given a multiple-output function, selects a subset of outputs *and* the corresponding precomputation logic so as to maximize a given cost function that is dependent on the probability of the precomputation logic and the number of selected outputs. This algorithm is described in pseudo-code in Figure 12.

The inputs to procedure **SELECT_OUTPUTS** are the multiple-output function F, and a number k corresponding to the size of the set in the input selection.

The procedure **SELECT_ORECUR** receives as inputs two sets G and H, which correspond to the current set of outputs that have been selected and the set of outputs which can be added to the selected set, respectively. Initially, $G = \phi$ and $H = F$. The cost of a particular selection of outputs, namely G, is given by $prG \times$ gates($F - H$)/total_gates, where prG corresponds to the signal probability of the precomputation logic, gates($F - H$) corresponds to the number of gates in the logic corresponding to the outputs in G and not shared by any output in H, and total_gates corresponds to the total number of gates in the network (across all outputs of F).

There are two pruning conditions that are checked for in the procedure **SELECT_ORECUR**. The first corresponds to assuming that all the outputs in H can be added to G without decreasing the probability of the precomputation logic. This is a valid condition because the quantity $proldG$ in each recursive call can only decrease with the addition of outputs to G. We then set a lower bound on the probability of the precomputation logic prior to calling the input selection procedure. Optimistically assuming that all the outputs in H can be added to G without lowering the precomputation logic probability, we are not interested in a precomputation logic probability for G that would result in a cost that is equal to or lower than BEST_OUT_COST.

```
SELECT_OUTPUTS( F = {f_1, ..., f_m}, k ):
{
    /* F = multi-output function to precompute */
    /* k = size of set in the input selection */
    BEST_OUT_COST = 0 ;
    SEL_OP_SET = φ ;
    SELECT_ORECUR( φ, F, 1, k ) ;
    return( SEL_OP_SET ) ;
}

SELECT_ORECUR( G, H, proldG, k ):
{
    lf = gates(F − H)/total_gates × proldG ;
    if( lf ≤ BEST_OUT_COST )
        return ;
    if( G ≠ φ )
        if( SELECT_LOGIC( G, k ) == φ )
            return ;
    prG = BEST_IN_COST ;   /* BEST_IN_COST is set in SELECT_LOGIC */
    cost = prG × gates(G)/total_gates ;
    if( cost > BEST_OUT_COST) {
        BEST_OUT_COST = cost ;
        SEL_OP_SET = G ;
    }
    choose f_i ∈ H such that i is minimum ;
    SELECT_ORECUR( G ∪ f_i, H − f_i, prG, k ) ;
    SELECT_ORECUR( G, H − f_i, prG, k ) ;
}
```

Figure 12. Procedure to determine the set of outputs to precompute

4.2.2 Logic Duplication

Since we are only precomputing a subset of outputs, we may incorrectly evaluate the outputs that we are *not* precomputing as we disable certain inputs during particular clock cycles. If an output that is not being precomputed depends on an input that is being disabled, then the output will be incorrect. However, an appropriate duplication of registers and logic will ensure that the outputs which are not selected are still implemented correctly (as described in [1]). The algorithm of Figure 12 attempts to minimize this duplication.

5: Combinational Precomputation

In order to synthesize precomputation logic for the architecture of Figure 8, algorithms are needed to accomplish several tasks. First, an algorithm must divide the circuit into sub-circuits. Then for each sub-circuit, algorithms must: a) select the subset of inputs to "turn off," and b) given these inputs, produce the logic for g_1 and g_2 in Figure 8. For each of these steps, the goal is to maximize the savings function

$$net\ savings \ = \sum_{all\ subcircuits} (savings(A) - cost(B) - cost(g)) \tag{3}$$

where g is defined as $g = g_1 + g_2$.

We must divide the original combinational circuit into sub-circuits so that Equation 3 is maximized. Note that the original circuit can be divided into a set of maximum-sized, single-output sub-circuits. A maximum-sized, single-output sub-circuit is a single-output sub-circuit such that no set of nodes from the original circuit can be added to this sub-circuit without creating a *multi-output* sub-circuit. An equivalent way of saying this is, the circuit can be divided into a minimum number of single-output sub-circuits. Such a set exists and is unique for any legal circuit. A linear time algorithm for determining this set is given in Figure 13.

Next, note that there is no need to analyze any sub-circuit that is composed of only a part of one of these maximum-sized, single-output sub-circuits. If a part of a single-output sub-circuit including the output node is in some sub-circuit to be analyzed, then the rest of the nodes of the single-output sub-circuit can be added to the sub-circuit *at no cost* since the outputs remain the same. Adding these nodes can only result in more savings. Further, if a part of a single-output sub-circuit not including the output node is in some sub-circuit to be analyzed, then the rest of the nodes of the single-output sub-circuit can be added to the sub-circuit because the precomputability of the outputs can only become less restrictive. Therefore, even in the worst case, the disable logic can be left the same so that there is no additional cost yet additional savings are achieved because of the additional nodes.

Based upon this theory, an algorithm to synthesize precomputation logic would 1) create the set of maximum-sized, single-output sub-circuits, 2) try different combinations of these sub-circuits, and 3) determine the combinations that yield the best net savings. Given the maximum-sized single-output sub-circuits, we use the algorithms of the previous section to determine a subset of the sub-circuits and a selection of inputs to each sub-circuit that results in relatively simple precomputation logic and maximal power savings.

```
GET_SINGLE_OUTPUT_SUBCIRCUITS( circuit ):
{
    arrange nodes of circuit in depth-first order outputs to inputs;
    foreach node in depth order ( node ) {
        if ( node is a primary output ) {
            subcircuit = create_new_subcircuit();
            mark node as part of subcircuit;
        }
        else {
            check every fanout of node;
            if ( all fanouts are part of the same sub-circuit )
                subcircuit = sub-circuit of the fanouts;
            else
                subcircuit = create_new_subcircuit();
            mark node as part of subcircuit;
        }
    }
}
```

Figure 13. Procedure to find the minimum set of single-output sub-circuits

6: Experimental Results

We present in Table 1 some preliminary results on random logic circuits taken from the MCNC benchmark set. In the first columns we present for each circuit the circuit name, number of inputs, outputs, literals, the maximum delay in nanoseconds, and power of the original circuit. The remaining columns present results obtained by the new *Complete Input Disabling* architecture, respectively, the number of inputs in the selected set, number of precomputed outputs, literals and levels of the precomputation logic, the final power and the percent reduction in power. All power estimates are in micro-Watts and are computed using the techniques described in [12]. A clock frequency of 20MHz was assumed. We used a general delay model where the gate delays were obtained from the `msu` generic library. The rugged script of **sis** [10] was used to optimize the precompute logic.

In Table 2 we compare our method with the *Subset Input Disabling* method. The best results obtained by both methods for each of the examples is given.

As it can be observed, using this new precomputation architecture we achieve significant power reductions over and beyond both the original circuit as well as the circuit synthesized under the *Subset Input Disabling* architecture. The reason for this is twofold. First the probability of the precomputation logic can be higher in the *Complete Input Disabling* architecture. Secondly, the original circuit is simplified due to the don't-care conditions in the *Complete Input Disabling* architecture.

Circuit	Original					Precompute Logic				Optimized	
	I	O	Lits	Delay	Power	I	O	Lits	Delay	Power	% Red
9sym	9	1	303	19.6	1828	7	1	53	13.8	1255	31.3
Z5xp1	7	10	163	34.8	1533	2	1	3	2.8	1325	13.6
alu2	10	6	501	42.2	2988	5	3	24	8.6	2648	11.4
apex2	39	3	330	15.6	1978	10	3	23	7.2	984	50.0
cm138	6	8	34	5.8	232	3	8	4	5.4	136	41.4
cm152	11	1	30	6.4	427	9	1	26	7.8	301	29.5
cm162	14	5	66	9.8	540	9	5	24	4.8	370	31.5
cmb	16	4	75	7.0	653	8	4	40	5.4	224	65.7
cordic	23	2	93	9.4	928	15	2	114	12.2	553	40.0
dalu	75	16	1271	46.0	7003	6	16	68	11.6	3720	46.9
mux	21	1	65	9.8	806	1	1	1	1.6	539	33.1
sao2	10	4	181	24.6	1001	2	4	5	2.4	406	59.3

Table 1. Power reductions for random logic circuits

7: Acknowledgements

Thanks to Mazhar Alidina and Marios Papaefthymiou for valuable discussions regarding precomputation architectures. This research was supported in part by the Advanced Research Projects Agency under contract DABT63-94-C-0053, in part by the Portuguese "Junta Nacional de Investigação Científica e Tecnológica" under project "Ciência" and in part by a NSF Young Investigator Award with matching funds from Mitsubishi Corporation.

References

[1] Mazhar Alidina, José Monteiro, Srinivas Devadas, Abhijit Ghosh, and Marios Papaefthymiou. Precomputation-based sequential logic optimization for low power. In *International Workshop on Low Power Design*, pages 57–62, April 1994.

[2] P. Ashar, S. Devadas, and K. Keutzer. Path-Delay-Fault Testability Properties of Multiplexor-Based Networks. *INTEGRATION, the VLSI Journal*, 15(1):1–23, July 1993.

[3] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS: A Multiple-Level Logic Optimization System. In *IEEE Transactions on Computer-Aided Design*, volume CAD-6, pages 1062–1081, November 1987.

[4] R. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[5] A. Chandrakasan, T. Sheng, and R. W. Brodersen. Low Power CMOS Digital Design. In *Journal of Solid State Circuits*, pages 473–484, April 1992.

[6] A. Ghosh, S. Devadas, K. Keutzer, and J. White. Estimation of Average Switching Activity in Combinational and Sequential Circuits. In *Proceedings of the* 29^{th} *Design Automation Conference*, pages 253–259, June 1992.

[7] J. Monteiro, S. Devadas, and A. Ghosh. Retiming Sequential Circuits for Low Power. In *Proceedings of the Int'l Conference on Computer-Aided Design*, pages 398–402, November

Circuit	Original Power	Subset Input Disable				Complete Input Disable			
		Lits	Delay	Power	% Red	Lits	Delay	Power	% Red
9sym	1828	40	11.0	1610	11.9	53	13.8	1255	31.3
Z5xp1	1533	3	2.8	1390	9.3	3	2.8	1325	13.6
alu2	2988	8	4.0	2683	10.2	24	8.6	2648	11.4
apex2	1978	15	5.3	1196	39.5	23	7.2	984	50.0
cm138	232	3	2.6	146	37.0	4	5.4	136	41.4
cm152	427	5	2.6	395	7.5	26	7.8	301	29.5
cm162	540	2	1.4	466	13.7	24	4.8	370	31.5
cmb	653	13	3.8	436	33.2	40	5.4	224	65.7
cordic	928	13	5.2	798	14.0	114	12.2	553	40.0
dalu	7003	16	5.6	4292	38.7	68	11.6	3720	56.9
mux	806	0	0	591	26.7	1	1.6	539	33.1
sao2	1001	2	1.4	446	55.4	5	2.0	406	59.3

Table 2. Comparison of power reductions between the two architectures

1993.

[8] F. Najm. Transition Density, A Stochastic Measure of Activity in Digital Circuits. In *Proceedings of the* 28^{th} *Design Automation Conference*, pages 644–649, June 1991.

[9] K. Roy and S. Prasad. SYCLOP: Synthesis of CMOS Logic for Low Power Applications. In *Proceedings of the Int'l Conference on Computer Design: VLSI in Computers and Processors*, pages 464–467, October 1992.

[10] 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 Int'l Conference on Computer Design: VLSI in Computers and Processors*, pages 328–333, October 1992.

[11] A. Shen, S. Devadas, A. Ghosh, and K. Keutzer. On Average Power Dissipation and Random Pattern Testability of Combinational Logic Circuits. In *Proceedings of the Int'l Conference on Computer-Aided Design*, pages 402–407, November 1992.

[12] C-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. Despain, and B. Lin. Exact and Approximate Methods for Switching Activity Estimation in Sequential Logic Circuits. *IEEE Transactions on VLSI Systems*, 3(1), March 1995. to appear.