Short Papers

Sequential Logic Optimization for Low Power Using Input-Disabling Precomputation Architectures

José Monteiro, Srinivas Devadas, and Abhijit Ghosh

Abstract— **Precomputation is a recently proposed logic optimization technique which selectively disables the inputs of a logic circuit, thereby reducing switching activity and power dissipation, without changing logic functionality. In sequential precomputation, 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. We target a general precomputation architecture for sequential logic circuits, and show that it is significantly more powerful than the architecture previously treated in the literature. The very power of this architecture makes the synthesis of precomputation logic a challenging problem. We present a method to automatically synthesize precomputation logic for this architecture. Up to 66% reduction in power dissipation is possible using the proposed architecture. For many examples, the proposed architecture result in significantly less power dissipation than previously developed methods.**

Index Terms—**Design automation, low power, observability don't-cares, power management, very-large-scale integration.**

I. 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., [6]), and logic optimization methods have been shown to have a significant impact on the power dissipation of combinational logic circuits (e.g., [13]).

In static CMOS circuits, the switching activity of the circuit determines 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., [9], [10]). More recently, efficient and accurate methods of power dissipation estimation for sequential circuits have been developed [15].

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 (e.g., [11]) and retiming [8] algorithms. More recently, techniques that detect self-loops in finite-state machines in order to stop the clock signal have been proposed [3].

A more general 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 are 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 synthesis algorithm 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.

In this paper, we target the first sequential precomputation architecture originally described in [1]. Still, synthesis methods were not developed in [1]. The key difference is that this architecture allows the precomputation logic to be a function of all of the input variables. We term this the *complete input-disabling* precomputation architecture, as opposed to the previous which we call the *subset input-disabling* architecture. We give an example that shows that the *complete input-disabling* architecture can reduce power dissipation for a larger class of sequential circuits than the *subset input-disabling* architecture. We propose an algorithm to synthesize precomputation logic for the *complete input-disabling* architecture.

In Section II, we briefly describe our model for power dissipation. In Section III, we describe various precomputation architectures, and describe their relative merits as well as synthesis issues. New algorithms that synthesize precomputation logic for a general sequential architecture are presented in Section IV. Experimental results are presented in Section V.

II. A Power Dissipation Model

Under a simplified model, the energy dissipation of a CMOS circuit is directly related to the switching activity. In particular, the three simplifying assumptions are as follows.

- The only capacitance in a CMOS logic gate is at the output node of the gate.
- Either current is flowing through some path from V_{DD} to the output capacitor, or current is flowing from the output capacitor to ground.
- Any change in a logic-gate output voltage is a change from V_{DD} to ground or vice versa.

All of these are reasonably accurate assumptions for well-designed CMOS gates [7], and when combined, imply that the energy dissipated by a CMOS logic gate each time its output changes is roughly equal to the change in energy stored in the gate's output capacitance. If the gate is part of a synchronous digital system controlled by a global clock, it follows that the average power dissipated by the gate is given by

$$P_{\text{avg}} = 0.5 \times C_{\text{load}} \times (V_{dd}^2/T_{\text{cyc}}) \times E(\text{transitions}) \qquad (1)$$

where P_{avg} denotes the average power, C_{load} is the load capacitance, V_{dd} is the supply voltage, T_{cyc} is the global clock period, and

Manuscript received October 18, 1995. This work 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 "Praxis XXI," and in part by an NSF Young Investigator Award with matching funds from Mitsubishi Corporation. This paper was recommended by Associate Editor F. Somenzi.

J. Monteiro is with the Department of Electrical Engineering and Computer Science, IST/INESC, 1000 Lisbon, Portugal.

S. Devadas is with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

A. Ghosh is with Synopsys, Inc., Mountain View, CA USA.

Publisher Item Identifier S 0278-0070(98)03083-8.

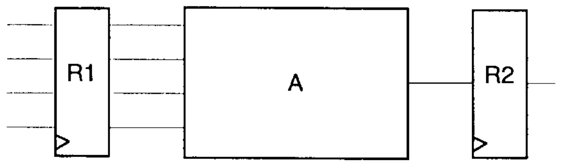

Fig. 1. Original circuit.

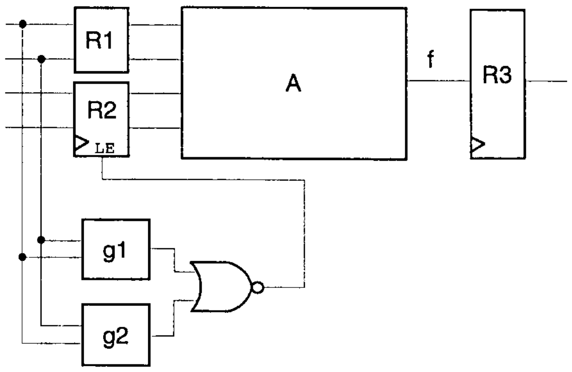

Fig. 2. Subset input-disabling precomputation architecture.

E(transitions) is the *expected value* of the number of gate output transitions per global clock cycle [10], or equivalently, the average number of gate output transitions per clock cycle. All of the parameters in (1) can be determined from technology or circuit layout information except E(transitions), which depends on both the logic function being performed and the statistical properties of the primary inputs.

Equation (1) is used by the power estimation techniques such as [9], [10] to relate switching activity to power dissipation.

III. Precomputation Architectures

We present two precomputation architectures originally described in [1]. Synthesis algorithms developed in [1] only targeted the *subset input-disabling* architecture.

A. Subset Input-Disabling Architecture

Consider the circuit of Fig. 1. We have a combinational logic block $\boldsymbol{A}$ that is bounded by registers R_1 and R_2. While R_1 and R_2 are shown as distinct registers in Fig. 1, they could, in fact, be the same register, i.e., all of the results presented apply equally to pipelined circuits and finite-state machines.

In Fig. 2, the *subset input-disabling* precomputation architecture is shown. The inputs to the block $\boldsymbol{A}$ have been partitioned into two sets, corresponding to the registers R_1 and R_2. The output of the logic block $\boldsymbol{A}$ feeds the register R_3. Two Boolean functions g_1 and g_2 are the *predictor* functions. We require

$$g_1 = 1 \Rightarrow f = 1 \tag{2}$$

$$g_2 = 1 \Rightarrow f = 0. \tag{3}$$

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 (we could alternatively gate the clock signal). 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 $\boldsymbol{A}$ changes, 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 a 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 a small subset of the inputs of f.

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

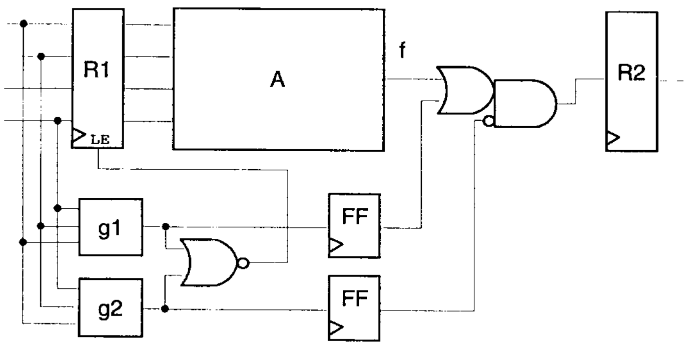

Fig. 3. Complete input-disabling precomputation architecture.

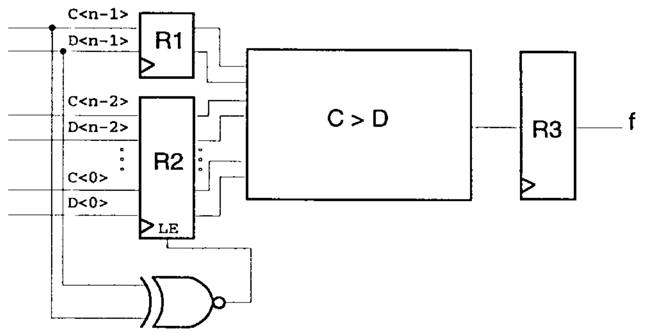

Fig. 4. Comparator example.

B. Complete Input-Disabling 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.

Complete Input-Disabling Precomputation Architecture: In Fig. 3, the new precomputation architecture for sequential circuits is shown. The functions g_1 and g_2 satisfy the conditions of (2) and (3) 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 $\boldsymbol{A}$ 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 (2) and (3).

The important difference between this architecture and the *subset input-disabling* architecture shown in Fig. 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.

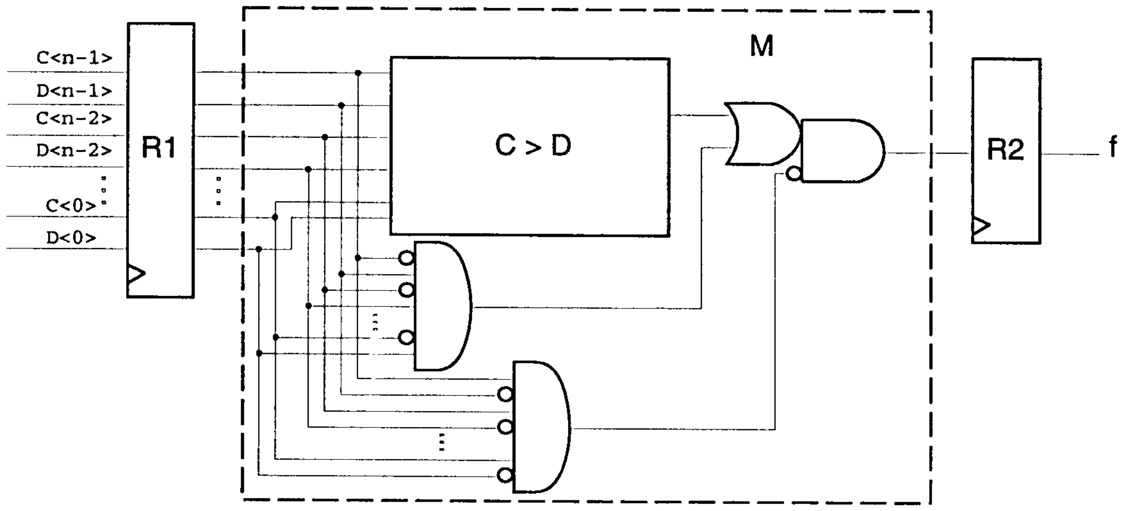

Fig. 5. Modified comparator.

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 $\boldsymbol{A}$ by using these input combinations as an *input don't-care set* for f.

An Example: A simple example that illustrates the effectiveness of the *subset input-disabling* architecture is an n-bit comparator that compares two n-bit numbers C and D and computes the function $C > D$. The optimized circuit with precomputation logic is shown in Fig. 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, i.e., each $C\langle i\rangle$ and $D\langle i\rangle$ has a 0.5 static probability of being a 0 or a 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 close to 50%.

Now, let us consider a modified comparator shown in Fig. 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 vice versa, 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 Fig. 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%.

For each of the sequential 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 combinational or sequential circuit is reduced. Algorithms to "discover" possible g_1 and g_2 functions within the original circuit which can be used to disable unnecessary transitions have been independently developed in [14].

IV. 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.

A. Precomputation Logic for Single-Output Functions

The key tradeoff in selecting the precomputation logic is that we want to include in it as many input combinations as possible, but at the same time, keep this 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 $\boldsymbol{A}$.

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.

Selecting a Subset of Inputs: Given a function f, we are going to select the "best" subset of inputs D 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), it means that for the combination of input variables in c, the value of f will be 1 (or 0) most of the time.

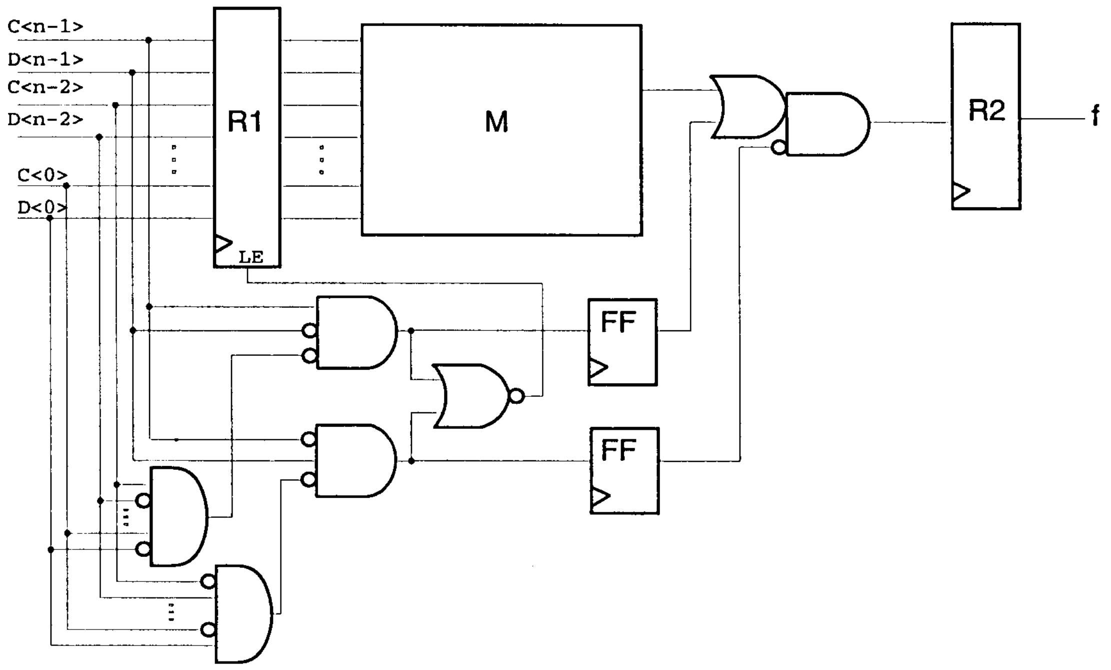

Fig. 6. Modified comparator under complete input-disabling architecture.

Let us consider f with inputs $x_1, x_2, \cdots, x_n$, and assume that we have selected the subset $x_1, x_2, \cdots, x_k$. If the probability of the cofactor of f with respect to $x_1 x_2 \cdots x_k$ being all 1's is high $(\text{prob}(f_{x_1 x_2 \cdots x_k}) \approx 1)$, 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 $x_1 x_2 \cdots x_k \cdot f_{x_1 x_2 \cdots x_k}$ in g_1. Similarly, if the probability of the $f_{x_1 x_2 \cdots x_k}$ is low $(\text{prob}(f_{x_1 x_2 \cdots x_k}) \approx 0)$, 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 $x_1 x_2 \cdots x_k \cdot \overline{f_{x_1 x_2 \cdots x_k}}$ in g_2. Note that in the *subset input-disabling* architecture, we could only do this when $f_{x_1 x_2 \cdots x_k} = 1$ or $f_{x_1 x_2 \cdots x_k} = 0$.

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 that it does not get very large. We estimate how large the precomputation logic is by computing the size of its corresponding ROBDD [5].

In the sequel, we describe a branching algorithm that selects the "best" subset of inputs. The pseudocode is shown in Fig. 7.

The procedure **SELECT_LOGIC** receives as arguments the function f, the desired number of inputs k to select, and the difference α from 0 and 1 that the probability of an approximate cofactor can be in order to be selected. **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 (size(BDD1)/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 (size(BDD1)/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 save the selected set corresponding to the maximum value of the cost function.

The worst case running time of the algorithm described above is exponential in the number of input variables, thus limiting the size of the circuits on which it can be applied. For large circuits, we resort to the approximate algorithm initially proposed in [1]. This algorithm looks at each input individually and chooses the k most promising inputs. For each input, we calculate

$$p_i = \text{prob}(U_{x_i} f) + \text{prob}(U_{x_i} \bar{f})$$

where p_i is the probability that we know the value of f without knowing the value of x_i. If p_i is high, then most of the time we do not need x_i to compute f. Therefore, we select the k inputs corresponding to smaller values of p_i. We now run the cycle in lines 19–33 of the pseudocode of Fig. 7, and add all of the cofactors that make the cost function increase. Note that, although this loop is exponential in k, k is always small, typically less than 10.

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 (ROBDD's) [5]. In the pseudocode of Fig. 7, 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 ROBDD's for g_1 and g_2, these can be converted into a multiplexor-based network (see [2]) or into a sum-of-products cover. The network or cover can be optimized using standard com-

TABLE I
POWER REDUCTIONS USING THE COMPLETE INPUT-DISABLING PRECOMPUTATION ARCHITECTURE

Circuit	Original					Precompute Logic				Optimized		
	I	O	Lits	Delay	Power	I	O	Lits	Delay	Delay	Power	% Red
9sym	9	1	303	19.6	1828	7	1	53	13.8	20.4	1255	31.3
Z5xp1	7	10	163	34.8	1533	2	1	3	2.8	34.8	1325	13.6
alu2	10	6	501	42.2	2988	5	3	24	8.6	44.0	2648	11.4
apex2	39	3	330	15.6	1978	10	3	23	7.2	27.2	984	50.0
cm138	6	8	34	5.8	232	3	8	4	5.4	7.4	136	41.4
cm152	11	1	30	6.4	427	9	1	26	7.8	9.2	301	29.5
cm162	14	5	66	9.8	540	9	5	24	4.8	10.8	370	31.5
cmb	16	4	75	7.0	653	8	4	40	5.4	8.8	224	65.7
dalu	75	16	1271	46.0	7003	6	16	68	11.6	46.3	3720	46.9
mux	21	1	65	9.8	806	1	1	1	1.6	11.2	539	33.1
sao2	10	4	181	24.6	1001	2	4	5	2.4	23.6	406	59.3

TABLE II
COMPARISON OF POWER REDUCTIONS BETWEEN COMPLETE AND SUBSET INPUT DISABLING

Circuit	Original Power	Subset Input Disable				Complete Input Disable			
		Precomp. Logic		Total		Precomp. Logic		Total	
		Literals	Delay	Power	% Red	Literals	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	46.9
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

binational logic optimization methods that reduce area [4] or those that target low-power dissipation [13].

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 $\boldsymbol{A}$ 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 the logic in block $\boldsymbol{A}$, thus leading to a reduction in area, and consequently to a further reduction in power dissipation.

B. Multiple-Output Functions

In general, we have a multiple-output function $f_1, \cdots, f_m$ that corresponds to the logic block $\boldsymbol{A}$ in Fig. 1. The procedures described above can be generalized for 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

$$g = \prod_{i=1}^{m} (g_{1i} + g_{2i}). \tag{4}$$

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

The probability that g, as defined in (4), is 1 may be very low since the number of input combinations that allow precomputation of all outputs may be very small. Thus, we need to select a subset of outputs to maximize a given cost function that is dependent on the probability of the precomputation logic and the number of selected outputs. The fewer outputs selected, the higher the probability that g evaluates to 1 (therefore, the higher the percentage of the time the circuit is being precomputed), but the smaller the fraction of the circuit that is precomputed. We use the same algorithm as described in [1] to select the optimal subset of outputs.

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]).

V. EXPERIMENTAL RESULTS

We present in Table I some results using sequential precomputation under the *complete input-disabling* architecture obtained from logic circuits taken from the MCNC benchmark set. Although these are combinational logic circuits, in our experiments, we assumed that the inputs to the circuits are outputs of flip-flops, and we applied sequential precomputation. All results were obtained using SIS [12], after the original circuits have been optimized using the *rugged* script and mapped to the MSU library.

In the first columns of Table I, 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

```
SELECT_LOGIC( $f$, $k$, $\alpha$ ):
{
    /* $f$: function to precompute, with the set of inputs $X$ */
    /* $k$: # of inputs to select */
    /* $\alpha$: probability interval for selection of approximate cofactors */
    BEST_IN_COST = 0 ;
    SELECTED_SET = $\phi$ ;
    SELECT_RECUR( $f$, $\phi$, $X$, $k$, $\alpha$ ) ;
    return( SELECTED_SET ) ;
}

SELECT_RECUR( $f$, $D$, $Q$, $k$, $\alpha$ ):
{
    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($f_c$) > $1 - \alpha$) {
                $approx = approx + 1$;
                BDD2 = BDD2 + $c \cdot f_c$;
            }
            if(prob($f_c$) < $\alpha$) {
                $approx = approx + 1$;
                BDD2 = BDD2 + $c \cdot \overline{f_c}$;
            }
        }
        $cost = (exact + \frac{\text{size(BDD1)}}{\text{size(BDD2)}} \times approx)/2^{|D|}$ ;
        if( $cost$ > BEST_IN_COST) {
            BEST_IN_COST = $cost$ ;
            SELECTED_SET = $D$ ;
        }
        return ;
    }
    choose $x_i \in Q$ such that $i$ is minimum ;
    SELECT_RECUR( $f$, $D \cup x_i$, $Q - x_i$, $k$, $\alpha$ ) ;
    SELECT_RECUR( $f$, $D$, $Q - x_i$, $k$, $\alpha$ ) ;
}
```

Fig. 7. Procedure to determine the precomputation logic.

columns present results obtained with the new *complete input-disabling* architecture, respectively, the number of inputs in the selected set, number of precomputed outputs, literals and delay of the precomputation logic, the delay and power of the optimized precomputed network, and the percent reduction in power. All power estimates are in microwatts, and were computed using the techniques described in [15]. A clock frequency of 20 MHz, a supply voltage of 5 V, and uniform input probabilities were assumed. We used a general delay model where the gate delays were obtained from the MSU generic library. The *rugged* script of SIS was used to optimize the precompute logic.

We should stress that the delay of the precomputation logic is added to the delay of the *previous* stage in sequential precomputation. The delay numbers in the third to last column correspond to the critical delay of the optimized circuit which includes the output AND–OR gate (Fig. 3). However, the use of don't-care conditions to optimize the circuit once the precomputation logic has been determined can reduce the delay of the optimized circuit.

In Table II, we compare our method with the *subset input-disabling* method. The best results obtained with each method for each of the examples is given. The precomputation logic in the *complete input-disabling* method is typically larger than in the *subset input-disabling* method; however, it results in larger power reductions. The reason for this is twofold. First, the probability of the precomputation logic can be higher in the *complete input-disabling* architecture. Second, the original circuit is simplified due to the don't-care conditions in the *complete input-disabling* architecture.

VI. Conclusions and Ongoing Work

We have proposed new synthesis algorithms that can be used to optimize a given sequential logic circuit for low power dissipation by adding "precomputation logic" which reduces unnecessary transitions in large parts of the given circuit. As opposed to power-down techniques applied at the system level, transition reduction is achieved on a per-clock cycle basis. We are currently exploring techniques to achieve data-dependent power-down at the register-transfer and behavioral levels.

Acknowledgment

Thanks to M. Alidina and M. Papaefthymiou for valuable discussions regarding precomputation architectures.

References

[1] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou, "Precomputation-based sequential logic optimization for low power," *IEEE Trans. VLSI Syst.*, vol. 2, pp. 426–436, Dec. 1994.

[2] P. Ashar, S. Devadas, and K. Keutzer, "Path-delay-fault testability properties of multiplexor-based networks," *INTEGRATION, VLSI J.*, vol. 15, pp. 1–23, July 1993.

[3] L. Benini, P. Siegel, and G. De Micheli, "Automatic synthesis of low-power gated-clock finite-state machines," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 630–643, June 1996.

[4] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. Computer-Aided Design*, vol. 6, pp. 1062–1081, Nov. 1987.

[5] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, pp. 677–691, Aug. 1986.

[6] A. Chandrakasan, T. Sheng, and R. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, pp. 473–484, Apr. 1992.

[7] L. Glasser and D. Dobberpuhl, *The Design and Analysis of VLSI Circuits*. Reading, MA: Addison-Wesley, 1985.

[8] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1993, pp. 398–402.

[9] J. Monteiro, S. Devadas, A. Ghosh, K. Keutzer, and J. White, "Estimation of average switching activity in combinational logic circuits using symbolic simulation," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 121–127, Jan. 1997.

[10] F. Najm, "Transition density: A new measure of activity in digital circuits," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 310–323, Feb. 1993.

[11] K. Roy and S. Prasad, "SYCLOP: Synthesis of CMOS logic for low power applications," in *Proc. Int. Conf. Computer Design*, Oct. 1992, pp. 464–467.

[12] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," in *Proc. Int. Conf. Comput. Design: VLSI in Comput. and Processors*, Oct. 1992, pp. 328–333.

[13] A. Shen, S. Devadas, A. Ghosh, and K. Keutzer, "On average power dissipation and random pattern testability of combinational logic circuits," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 402–407.

[14] V. Tiwari, P. Ashar, and S. Malik, "Guarded evaluation: Pushing power management to logic synthesis/design," in *Proc. Int. Symp. Low Power Design*, Apr. 1995, pp. 221–226.

[15] C.-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. Despain, and B. Lin, "Power estimation methods for sequential logic circuits," *IEEE Trans. VLSI Syst.*, vol. 3, pp. 404–416, Sept. 1995.