BDD-Based Circuit Restructuring for Reducing Dynamic Power

Quang Dinh, Deming Chen, Martin D. F. Wong
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
{qdinh2, dchen, mdfwong}@illinois.edu

Abstract—**As advances in process technology continue to scale down transistors, low power design is becoming more critical. Clock gating is a dynamic power saving technique that can freeze some flip-flops and prevent portion of the circuit from unneeded switching. In this paper, we consider fine-grained clock gating through pipelining, in which control signals from one pipeline stage are used to freeze some logic in the next pipeline stage. We present a novel BDD-based decomposition algorithm to restructure the circuit and expose possible control signals that would maximize power saving. We then use ILP formulation to select the optimal set of control signals for the circuit. We show that the constraint matrix is totally unimodular, and solve this selection problem optimally using linear programming. Comparing to a previous work [7], we get similar and 9% better dynamic power saving for small and medium circuits, respectively. For the largest MCNC circuits, which the previous technique cannot handle, we get an average of 19% dynamic power saving with 9.3% area overhead comparing to the original, non-restructured circuits.**

I. INTRODUCTION

As advances in CMOS process technology enable more and faster transistors to be packed into the same silicon area, heat is becoming a pressing issue. Heat can be reduced through design for low power, which helps reduce packaging and cooling costs as well as improve reliability. Low power design is also critical in the rapidly expanding mobile applications, as it enables longer usage on limited battery power.

There are two types of power sources in CMOS devices: *static power* and *dynamic power*. Static power is due to leakage current, when there are no signal transitions. Dynamic power is due to switching and short-circuits currents associated with signal switching activities. Although static power is increasing with submicron technology, dynamic power is still the major contributor to total power consumption. Techniques to reduce dynamic power are actively studied.

Clock gating is a technique to reduce dynamic power. In this scheme, if the intermediate output of a sub-circuit is not required for the correct evaluation of the final circuit output during a clock cycle, this sub-circuit can be disabled for this cycle by clock gating its input flip-flops. This prevents the flip-flops from changing their values, saving the sub-circuit from unneeded switching activities. Thus dynamic power consumption is reduced without affecting circuit functionality.

In this paper, we investigate a fine-grained clock gating technique for gate-level circuits. We restructure the original circuit into two pipeline stages by inserting flip-flops. Control signals from one pipeline stage are used to control the clock gating of the flip-flop inputs of the next pipeline stage.

Unlike previous works that work on fixed circuits, we actively restructure circuits in such ways that could generate new, derived control signals with high probabilities of clock gating the subsequent sub-circuits. We also search for circuit modifications in which large sub-circuits with high switching activities can be clock gated. The novelty of our restructuring algorithm is based on the application of binary decision diagram, or *BDD*. We show that BDD decomposition through linear expansion offers an effective way to restructure logic for the purpose of reducing dynamic power through clock gating.

Besides generating high impact control signals through BDD decomposition, we also solve the problem of selecting the optimal set of control signals in a circuit for maximum dynamic power saving through clock gating. Because control signals are not independent (one affects others control signals downstream), we cannot simply select all available control signals or select any arbitrary sets of control signals. Thus, in order for this clock gating technique to be effective, we present an ILP formulation to the optimal selection problem. We show that this ILP has a totally unimodular constraint matrix, therefore it can be solved optimally by linear programming techniques.

Experimental results based on industrial FPGA design tools show that our technique can reduce dynamic power by 19% on average for the 20 largest MCNC circuits, with an average area overhead of 9.3% over the unmodified circuits. Comparing to a previous work [7], our technique scales much better for large circuits, saves 9% more dynamic power for medium-size circuits, and has similar saving for small circuits.

Overall, our contribution can be summarized as follows:

1. We apply BDD-based decomposition technique to restructure a Boolean function, searching for a representation with control signals that allow maximum dynamic power reduction when applying clock gating.
2. We use ILP formulation to optimally select the set of candidate control signals in a circuit to be used for clock gating, so that the total dynamic power saving is maximized. Due to the totally unimodular property of the formulated ILP, it is solved optimally through linear programming relaxation.
3. We test our technique with the commercial Quartus FPGA design tools from Altera, using the industrial quality power estimator PowerPlay Analyzer.

The rest of the paper is organized as follows. Section II provides some related works and background information on clock gating and BDD linear expansion. Section III details our restructuring technique, including BDD decomposition and control signal selection. Section IV presents our experimental results. We conclude the paper in section V.

978-1-4244-8935-0/10/$26.00 ©2010 IEEE

II. PRELIMINARIES AND RELATED WORKS

A. Dynamic Power Reduction and Clock Gating

The dynamic power dissipation of a CMOS circuit is:

$$P_{Dynamic} = 0.5 \times f \times V_{dd}^2 \times \sum_{i=1}^{n} C_i S_i \quad (1)$$

where n is the total number of gates, f is the clock frequency, V_{dd} is the supply voltage, C_i is the load capacitance for gate i, and S_i is the switching activity for gate i. Switching activity is the average number of transitions ($0 \rightarrow 1$ and $1 \rightarrow 0$) a signal switches per unit time. Reducing switching activity is an effective way to lower dynamic power.

Switching activity estimation is based on the concepts of *transition density* and *signal probability*, first developed in [1]. The *transition density* (or *switching activity*) of a logic signal is defined as the average number of transitions per unit time. The *signal probability* is defined as the fraction of the time that the signal is in the logic 1 state. [1] presented an efficient method of computing signal probabilities and switching activities of all signals in a circuit. We use this method for our work.

Switching activity can be reduced through retiming. Some of the works are discussed in [2][3][4], where flip-flops are moved around so that high switching activity signals can be latched and fewer transitions are propagated to subsequent logic. Another approach is glitch minimization, such as the works presented in [5] and [6].

A clock gating study was presented in [7]. In this work, some control signals are selected to clock gate other logic blocks. When a control signal takes on its controlling value, it dominates the clock gated logic and thus can safely prevent that block of logic from switching, without affecting the overall functionality of the circuit. However, in [7], the circuit structure is fixed, and the control signals can only be among those already present in the given circuit. In this work, we actively try decomposing the circuit in different ways, exposing new signals that can act as control signals. We can search for a better restructuring of the circuit that allows more clock gating opportunities. The algorithm presented in [7] relies on brute-force search of all possible combinations of control points, limiting its application to only very small circuits. In this paper, we present a novel ILP-based approach to find the optimal set of control points that are proven to be polynomial.

B. BDDs and Linear Expansion

In this section, we provide a brief review of BDDs and linear decomposition based on BDD. BDDs were introduced by Lee [11] and popularized by Akers [12]. Then Bryant developed *reduced ordered BDDs* (ROBDDs) in [13], together with a set of efficient ROBDD operators. A BDD is a directed acyclic graph with two terminal nodes 0 and 1, representing a Boolean function. Each internal node is associated with an input variable of the Boolean function (an input variable can correspond to many nodes). This node has two outgoing edges: 0-edge pointing to the next node when the corresponding input variable is of logic 0, and similarly the 1-edge for logic 1. A ROBDD is a BDD where input variables are in the same fixed order for all paths of the graph, and every node represents a distinct function. In this paper, we refer to ROBDDs as BDDs. An example BDD with five input variables is illustrated in Fig. 2.

A BDD can be decomposed through cutting the BDD. Its direct effect is to re-express the BDD by two or more sub-BDDs. In this paper, we use the idea of linear expansion [14] to decompose a BDD. Fig. 1 shows the idea of linear expansion. On the left hand side, $\{v_1, v_2, ..., v_k\}$ is a cut set of the BDD. Based on this cut, the BDD can be decomposed into a summation of products of sub-BDDs shown on the right hand side. Such a decomposition exposes possible control signals that could maximize power saving through clock gating.

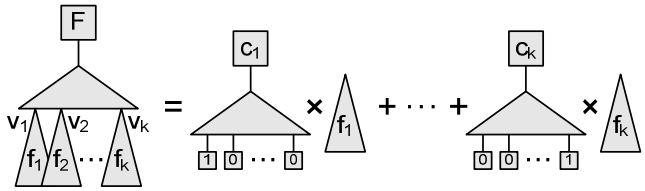


Fig. 1. Linear expansion of a BDD

The structure of BDDs provided a natural way of enumerating cuts for linear expansion. The left hand side of Fig. 2 shows five possible cuts of a BDD at different depth levels. Each cut partitions the input variables into two disjoint sets, the upper set and the lower set. Corresponding to each cut is a linear expansion. The right hand side of Fig. 2 illustrates the linear expansion on *cut2*. The different linear expansions produce immediate candidates for control signals, which can be evaluated for clock gating potential. Linear expansion also enables a dynamic programming approach, which can efficiently search through the decomposition space and generate the best cut for power saving through clock gating. This dynamic programming approach is similar to what was proposed in [10] although we use it for a totally different purpose. More details will be introduced later.

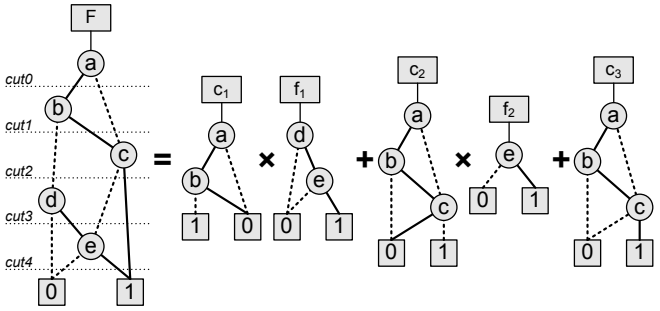


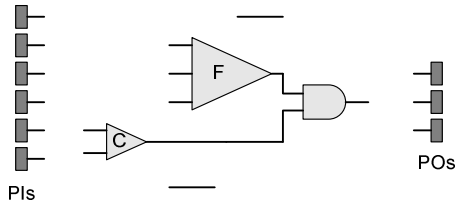
Fig. 2. An example BDD with 5 variables and its linear expansion on *cut2*: $F = c_1f_1 + c_2f_2 + c_3$

III. CIRCUIT RESTRUCTURING FOR LOW-POWER

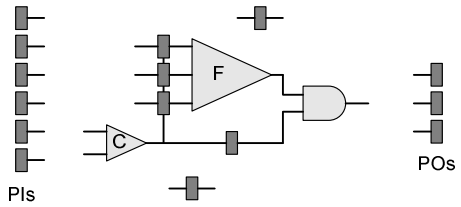
A. Overview

1. Clock Gating and Pipeline

We implement clock gating by breaking the circuit into pipeline stages. As illustrated in Fig. 3a, in the original circuit, block F is controlled by block C. If the control signal C has the controlling value of logic 0, then the correct value of F is not required for the correct functionality of the circuit. Fig. 3b shows how F can be clock gated by C. The restructured circuit now has two pipeline stages: block C is in the first stage and block F is in the second stage. The flip-flops at the inputs of F only load new values when control signal C is of logic 1. Other flip-flops are inserted to keep signals synchronized.



(a) Original Circuit



(b) Pipelined and clock gated circuit

Fig. 3. Pipelining and Clock Gating.

The inserted flip-flops become overhead of this restructuring technique. They consume power and take up silicon area. We would like to keep this overhead under control, by limiting the number of flip-flops. Therefore, the overhead of a clock gating decomposition is estimated based on the number of inserted flip-flops. For the example in Fig. 3, this overhead is computed from the number of inputs of block F plus 1 (the flip-flop for the control signal) and the dynamic power of one flip-flop Pwr_{FF} :

$$FF_{overhead}(F) = \left(\left|inputs(F)\right| + 1\right) \times Pwr_{FF} \quad (2)$$

When the clock gated form is implemented, block F does not consume dynamic power when signal C takes the controlling value, which is logic 0 for the example in Fig. 3. Therefore, the expected dynamic power saving is:

$$Pwr_{Saved} = \Pr\{C = 0\} * Pwr(F) - FF_{overhead}(F) \quad (3)$$

Here $\Pr\{x = 0\}$ and $\Pr\{x = 1\}$ denote the probability of x being 0 and 1, resp.; and $Pwr(f)$ denotes the dynamic power of function f. The estimation of these values is presented later in section III.C.3.

2. Two-stage Pipeline and Exclusivity Constraint

As illustrated in Fig. 4, when a combinational path in the circuit is clock gated twice or more at different locations, the overall delay on this path would be two clock cycles or more. This would make the restructuring impractical due to too much pipeline delay. Furthermore, deep pipeline requires more flip-flops for signal synchronization, and these extra flip-flops would introduce too much overhead. In this work, we limit the restructuring to a two-stage pipeline.

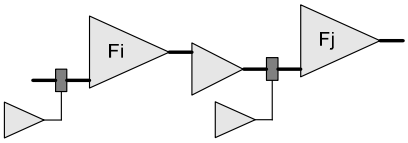


Fig. 4. Multiple clock gating on one combinational path

This means that there would be exactly one flip-flop inserted on every combinational path from PIs to POs (flip-flops in the original circuits are also considered as pseudo PIs and POs). This flip-flop is used either for clock gating or just for signal synchronization. We call this the exclusivity constraint: any flip-flop inserted would exclude all its fan-in and fan-out edges from having another flip-flop inserted. This is illustrated in Fig. 5.

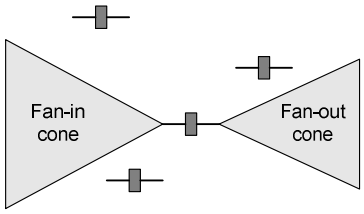


Fig. 5. Exclusivity Constraint: no flip-flops can be in the fan-in or fan-out cones of the flip-flops

3. Three Steps for Restructuring

Our proposed circuit restructuring technique is divided into 3 steps: clustering and partial collapsing into supernodes, decomposition of each supernode into optimal clock gating form, and selection of supernodes to be clock gated.

In the first step, we use clustering and partial collapsing to turn the initial netlist into a network of supernodes. It is based on an iterative elimination framework that iteratively merges pairs of nodes to form larger nodes. The Boolean function of each supernode is represented by a BDD.

In the second step, we decompose each supernode into a form that is beneficial for clock gating. We take advantage of BDD linear expansion to quickly identify such power saving decompositions. We use a dynamic programming approach to efficiently search through the optimization space.

In the final step, we identify the optimal set of supernodes to be clock gated. Due to signal and timing dependency, not all supernodes can be clock gated using the decomposed form found in step 2. Formulating this problem as a max-cut problem, we then show how to solve the optimization problem optimally through a linear programming technique.

The overall algorithm is as follows:

Algorithm 1: Overall algorithm

Input: A logic network
Output: A restructured circuit

1. Collapse the network into a set of connected supernodes;
2. **foreach** supernode **do**
   Perform BDD-based decomposition, find best clock gating form and the associated power saving;
   end
3. Compute optimal set of supernodes to be clock gated;
   Replace selected supernodes with their clock gating forms;

B. Clustering and Partial Collapsing

In this step, the initial circuit netlist is turned into a network of supernodes. This is similar to other logic synthesis systems [15][16]. We use the clustering and partial collapsing algorithm in [10], which is based on an iterative elimination framework. In each iteration, a list of mergable node pairs is generated, and then node

pairs are merged in decreasing order of merging gains. The Boolean function of each node is represented by a BDD. The merging gains are assigned based on the number of BDD nodes after the merge. The algorithm terminates when there remain no mergable node pairs. An example of the resulting network of supernodes is shown in Fig. 6. Note that while circuits generally have multiple outputs, each supernode BDD is single output.

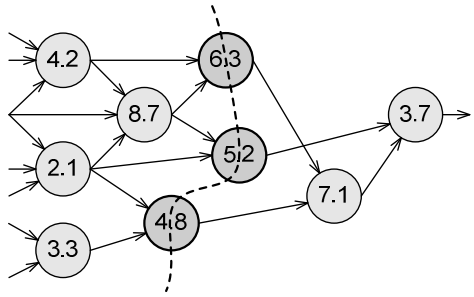


Fig. 6. A network of supernodes and its partitioning into two pipeline stages (section III.D)

C. BDD Decomposition of Each Supernode

1. Clock gating based on BDD Linear Expansion

The linear expansion of a BDD enables efficient exploration of different clock gating options for a given Boolean function. For example, assume that a function F is decomposed into $F = (c_1 * f_1) + (c_2 * f_2) + (c_3 * f_3)$, then some possible clock gating options and their potential savings are (illustrated in Fig. 7):

(a) c_1 (when 0) is used to clock gate f_1. The potential power saving is: $\Pr\{c_1 = 0\} * Pwr(f_1) - FF_{overhead}(f_1)$.

(b) f_2 (when 0) is used to clock gate c_2. The potential power saving is: $\Pr\{f_2 = 0\} * Pwr(c_2) - FF_{overhead}(c_2)$.

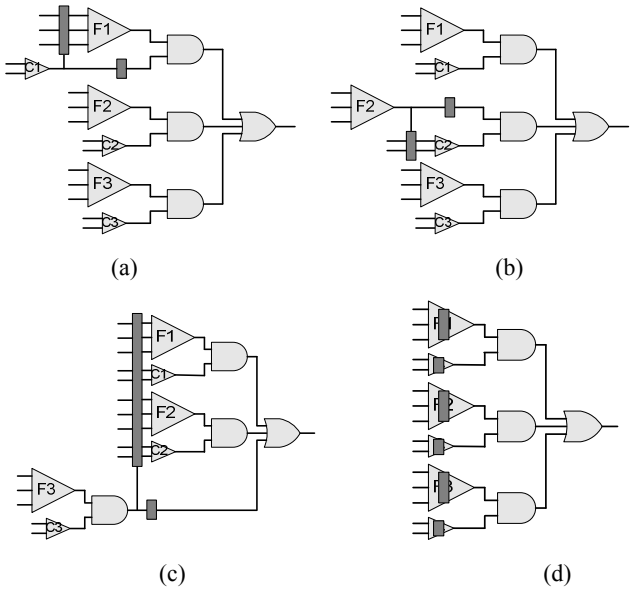


Fig. 7. Four different clock gating options for one linear expansion

(c) $c_3 * f_3$ (when 1) is used to clock gate $c_1 * f_1 + c_2 * f_2$. The potential power saving is:

$$\Pr\{c_3 * f_3 = 1\} * Pwr(c_1 * f_1 + c_2 * f_2) - FF_{overhead}(c_1 * f_1 + c_2 * f_2)$$

(d) Each of the sub-functions c_1, f_1, c_2, f_2, c_3, f_3 has a clock gated form itself. These clock gated form can be used directly in the decomposed structure of F. The potential power saving is simply the sum of power saving from each sub-function.

Some options are independent and can be combined together. An example is options (a) and (b) — their power savings can be added together. Other options, however, are inter-dependent and require some checking. For example, selecting option (c) would affect the power saving of option (a) and (b) (part of it is already counted for in (c)). Furthermore, because of the exclusivity constraint discussed in section III.A.2, clock gating option (c) would exclude options (a) and (b) from being clock gated.

2. Dynamic Programming

As seen in option (d) above, to find the clock gating option for F that offers the most power saving, we would need to know the best power saving of its sub-functions c_1, f_1, c_2, f_2, c_3, f_3 if these sub-functions are clock gated. This means we would need to recursively decompose a BDD into smaller and smaller sub-BDDs. The efficient way to do this is by dynamic programming, similar to the details presented in [10]. While [10] uses dynamic programming to optimize delay and identify a good mapping solution for FPGA, our goal is to find the best cut and clock gating option that would offer the most dynamic power saving when clock gated, which is completely different. Here we provide an overview of this dynamic programming algorithm, shown in Algorithm 2 below.

Algorithm 2: Decomposition of one BDD for best clock gating

Input: A BDD *B*
Output: The clock gating form with the most power saving

```
for l = 0 to (number of input variables of B) do
  foreach sub-BDD S of depth l do
    BestSaving = 0;
    foreach cut C of S do
      Generate the linear expansion of S on cut C;
      foreach clock gating option O of the expansion do
        Compute power saving Saving offered by option O;
        if Saving > BestSaving then
          BestSaving = Saving;
          Record this clock gating form O for sub-BDD S;
        end
      end
    end
    PowerSaving(S) = BestSaving;
  end
end
```

We decompose all possible sub-BDDs of a BDD in the order of increasing depth: all depth-0 sub-BDDs are decomposed first, then all depth-1 sub-BDDs, and so on until the final BDD. In this way, when it comes to evaluate different way of clock gating for a sub-BDD, the best clock gated forms and the corresponding power savings of all its sub-functions are already computed and available from previous iterations.

For each sub-BDD, all possible cut levels are enumerated to

generate different linear expansions (refer to Fig. 2). For each of these linear expansions, all possible clock gating options—as discussed in the previous section—are considered (refer to Fig. 7). Again, these include the option (d) of directly using the best clock gated forms of the sub-functions, which is enabled by the dynamic programming approach. Finally, we pick out the clock gating option that offers the most power saving for this sub-BDD. The best clock gating option and its corresponding power saving is recorded for later uses when decomposing larger sub-BDDs.

The complexity of this dynamic programming algorithm is $O(n^2N^2)$, where n is the number of input variables, and N is the size of the BDD. As analyzed in [10], the number of sub-BDDs is $O(nN^2)$, the number of cuts for each sub-BDD is $O(n)$, and the number of clock gating options for each cut is $O(1)$. We limit the BDD size in our algorithm (up to 200), so the algorithm is fast.

3. Signal Probability and Power Estimation

To compare between different clock gating forms and select the best form, we need to estimate their power saving potentials. This requires estimations of the signal probabilities of the gating signals, as well as the dynamic power of the clock gated blocks. We estimate both of these values from the signal probabilities.

We first estimate signal probability $\Pr\{n=1\}$ for each node n in a BDD, propagating from the two terminal nodes up to the root node; this is a standard operation on BDDs [1]. Then, the switching activity of a node n is estimated as the probability of signal transition on n in any cycle [1]:

$$SW(n) = 2\times \Pr\{n=0\}\times \Pr\{n=1\} \quad (4)$$

Finally, the dynamic power of a BDD (or a sub-BDD) is estimated as the sum of the switching activities of all its internal nodes.

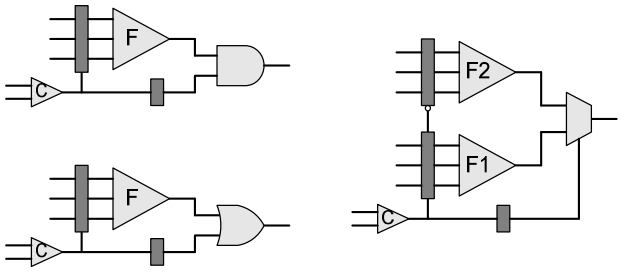


Fig. 8. Special decompositions for clock gating: AND, OR, MUX

4. Special Decompositions

There are four special decompositions that could offer better decompositions than linear expansion [10]. Three of these decompositions—AND, OR, and MUX—are suitable for clock gating, because of an available control signal that allows clock gating when it takes the controlling value (logic 0 for AND and logic 1 for OR). For MUX decomposition, the control signal would gate either of the two options F1 or F2 depending on its logic value. These are illustrated in Fig. 8. The remaining XNOR decomposition is not suitable for clock gating, as it has no possible control signal. We check these special decompositions and use them instead of linear expansion when encountered.

D. Selection of Supernodes

1. Problem

After we have found the optimal decomposed form and the power saving potential for each supernode, the final problem is to select a set of supernodes in the network to be clock gated. Due to the exclusivity constraint, we cannot simply clock gate all supernodes. Instead, we need to find the best set of supernodes that have the maximum total power saving while satisfying the exclusivity constraint.

To address this problem, we view the circuit as being partitioned into two pipeline stages. The pipeline boundary cut through some of the supernodes. Those supernodes can then be clock gated in the optimal decomposed forms found in section III.C, contributing their share of power saving. This is illustrated in Fig. 6. In this example, the three darker supernodes on the pipeline boundary provide a total power saving of 16.3.

2. Max-Cut Formulation

The partitioning of the circuit into two pipeline stages can be viewed as a cut through the circuit graph. Similar to the well-known min-cut problem, we would like to assign costs, or rewards, to the edges. This can be obtained by splitting each supernode into two nodes, with an edge connecting these two nodes assigned the reward value of the power saving of the supernode. All original edges have zero rewards. Fig. 9 demonstrates this transformation.

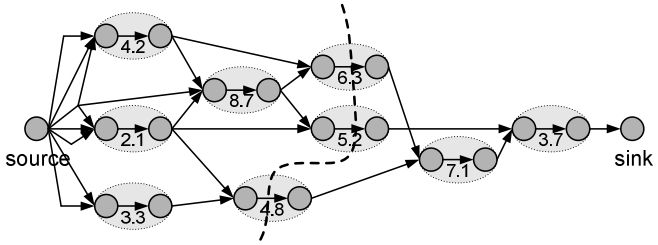


Fig. 9. Max-Cut Formulation for the network in Fig. 6

Max-Cut Problem:

Input: A directed, acyclic graph $G(V,E)$, with a source node and a sink node. Each edge is assigned a reward. Our formulation presented below allows negative rewards.

Output: A cut of the graph that maximizes the sum of rewards of cut edges. The cut has to satisfy the constraints below.

Constraints: The exclusivity constraint requires that any path from source to sink is cut only once. Or, equivalently, cut edges are of the same forward direction: all predecessor nodes of cut edges belong to the half containing the source node, and all successor nodes of cut edges belong to the half containing the sink node.

Although we use the terms *source* and *sink*, this is not a flow problem.

3. Integer Linear Programming Formulation

We formulate the max-cut problem as an integer linear programming problem.

Decision variables: for each node i, we define a binary variable $x_i \in \{0,1\}$. $x_i = 1$ when node i belongs to the source half; $x_i = 0$ when node i belongs to the sink half. The number of variables is the number of nodes in the graph $|V|$, or twice the number of supernodes in the circuit.

Constraints: For any edge (i,j) from node i to node j, the

only possible values for the pair (x_i, x_j) are $(1,1)$ (both nodes are in the source half), $(1,0)$ (a cut edge), and $(0,0)$ (both nodes are in the sink half). It cannot take value $(0,1)$ because that would mean the edge is cut in the reverse direction, violating the constraints. Thus, for each edge (i, j), we have a constraint:

$$x_i - x_j \geq 0 \tag{5}$$

We observe that the number of constraints is the same as the number of edges in the graph, $|E|$.

Cost function: Let R_{ij} be the reward of edge (i, j). If the edge is cut, then $x_i = 1$, $x_j = 0$, and its contribution to the total reward is R_{ij}, or $R_{ij} \times (x_i - x_j)$. If the edge is not cut, then $x_i = x_j$, and its contribution to the total reward is 0, or $R_{ij} \times (x_i - x_j)$ also. Thus, the cost function, or the negative of the total reward, is:

$$Cost = -\sum_{\forall edge(i,j)} R_{ij} \times (x_i - x_j) \tag{6}$$

With R_{ij} given, the cost function is a linear function of all the decision variables x_i.

4. Linear Programming Equivalence

In the integer linear programming problem above, all the constraints are of the integer-difference form. Therefore, the constraint matrix is totally unimodular [8]. Thus, we can remove the integer requirements and still have an equivalent optimization problem [9]. The resulting linear programming problem can be readily solved by established techniques, such as the simplex algorithm.

The final linear programming problem is:

Minimize:

$$Cost = \sum_{\forall edge(i,j)} R_{ij} x_j - R_{ij} x_i \tag{7}$$

Subject to:

$$x_i - x_j \geq 0 \quad \text{for all edges } (i, j) \tag{8}$$

$$0 \leq x_i \leq 1 \quad \text{for all } x_i \tag{9}$$

IV. EXPERIMENTAL RESULTS

We evaluate the restructuring technique on the 20 benchmarks from the MCNC benchmark suites: the 10 largest combinational and the 10 largest sequential circuits. Before restructuring, we add flip-flops to the PIs and POs of the circuits to make them synchronized and more suitable for pipelining. Because one clock cycle in the original circuits is broken into two pipeline stages in the restructured circuits, the clock frequency of a restructured circuit should be twice the clock frequency of its original circuit. After we obtain the maximum clock frequency f_{max} for a restructured circuit from the synthesis tools, the synthesis flow for its original circuit is configured to target a clock frequency of at least $0.5f_{max}$. Note that although the frequency of the restructured circuit is doubled, the actual data switching rate is not doubled, as each pipeline is kept idle every other clock cycle to ensure signal synchronization to be exactly the same as in the original circuit.

To obtain high confident power estimations, we use the industrial tool Altera's PowerPlay Analyzer on placed and routed circuits on FPGAs. Note that our algorithm can target both ASICs and FPGAs. We feed both the original and the restructured circuits to the same Altera's Quartus II flow. The devices are Stratix II family, chosen automatically by Quartus II. After synthesis, placement, and routing (with commands quartus_map, quartus_fit, quartus_asm, quartus_tan), we use the gate-level power estimator PowerPlay Analyzer to collect power estimations. For each benchmark, PowerPlay Analyzer uses switching activity files generated by the Quartus II simulator from 1000 input vectors. The input vectors are randomly generated with a signal probability of 0.5 and a switching activity of 0.5. Note that we use two different clock frequencies for original and restructured circuits ($0.5f_{max}$ and f_{max}, resp.), and this is accurately taken into account during power simulation. Thus the dynamic powers of the two configurations are compared at the same level of performance.

Table 1 presents the dynamic power for each circuit, as reported by PowerPlay Analyzer. We see that on average, the restructured circuits are able to reduce dynamic power by 19% over the original, unmodified circuits. Some restructured circuits can save more than 30% dynamic power. Modern circuits adopted advanced process technologies (e.g. triple-oxide, dual V_{th} transistors, metal gate, and high-k dielectric etc.) to reduce leakage power. Thus, dynamic power is still the dominating portion in the total power. However, in our experiments, the mapped benchmarks are much smaller than the available LUTs in the Stratix II devices, and relatively more power is consumed statically by the unused LUTs. This makes total power estimations not that meaningful for comparisons. Therefore we only report the dynamic power consumptions.

Table 1 also includes the area overhead (number of extra LUTs used) of our restructured circuits. This overhead includes the flip-flops for the pipelining. On average, our restructuring increased circuit size by 9.3%. For deep submicron technology, this trade-off of some increase of circuit size for considerable reduction of dynamic power is very practical. Power consumption and heat are the more concerned issues, rather than the number of transistors as in the past.

We also carry out a comparison study with [7]. We implement the algorithm presented in that paper, then run through the same synthesis flow described above. Due to the limitation of their approach mentioned in [7] (their proposed method is difficult to apply to multi-level circuits), only small and some medium benchmarks (after conversion to two-level circuits) can be used for comparison, large benchmarks cannot be handled. [7] only reported results with small benchmarks. For larger circuits, converting them to two-level representation leads to the explosion of circuit sizes, which result in very large circuits that consume much more dynamic power than the original circuits. Therefore, we didn't use the largest MCNC benchmarks for this comparison, and only used small and medium benchmarks. The results are presented in Table 2. For small benchmarks, our restructuring algorithm has similar performance (0.7% better) and runs 3X slower, but in absolute term only 2 seconds slower. For medium benchmarks (still small comparing to real world circuits), our algorithm is 9.3% better in dynamic power reduction and runs 7X faster. More importantly, this shows that our algorithm scales much better for larger circuits.

Although the dynamic programming and supernodes selection steps are optimal in our algorithm, the final results are not, as can be seen in Table 2, where some circuits lose out to [7]. This is because the supernode network generated in III.B is based on heuristics, and more importantly, our dynamic power savings are only estimations at high level, without knowledge of physical steps

such as mapping, placement, and routing.

Table 1. Dynamic power comparison between original and restructured circuits, area overhead, and restructuring runtime

Circuit	Dynamic Power			Area overhead (%)	Run-time (s)
	Original (mW)	Restruct (mW)	Saving (%)		
alu4	35.63	31.42	11.8	5.5	97
apex2	29.77	21.51	27.8	8.0	152
apex4	21.12	13.79	34.7	11.0	274
des	81.59	69.48	14.8	6.6	87
ex1010	56.07	47.67	15.0	13.5	1325
exp5p	18.71	13.02	30.4	11.1	259
misex3	29.62	22.98	22.4	14.5	182
pdc	49.94	44.95	10.0	6.4	1157
seq	27.33	17.69	35.3	4.5	240
spla	43.54	38.24	12.2	10.0	2933
bigkey	66.96	52.01	22.3	3.7	850
clma	66.48	57.22	13.9	10.8	3763
diffeq	17.13	14.61	14.8	4.2	193
dsip	52.83	49.50	6.3	14.6	729
elliptic	45.38	31.17	31.3	13.2	1721
frisc	30.90	28.00	9.4	15.9	2816
s298	21.80	17.33	20.5	12.8	1376
s38417	110.12	83.64	24.0	7.8	3987
s38584.1	96.05	86.67	9.8	4.9	3391
tseng	17.97	14.02	22.0	6.0	271
Avg.			**19.4**	**9.3**	

Table 2. Scalability and dynamic power comparison with [7]

Circuit	Dynamic power saving (%)			Runtime (s)	
	[7]	our algorithm	Comp.	[7]	our algorithm
Small Circuits					
cht	17.4	15.0	–2.4	0.9	2.1
cm150	23.7	20.7	–3.0	0.7	4.3
cordic	17.8	21.7	3.9	1.3	3.6
mux	25.3	23.6	–1.7	1.4	1.8
pcle	34.1	37.2	3.1	0.7	2.2
pcler8	17.0	21.4	4.4	1.1	3.2
Avg.			**0.7**	**1.0**	**2.8**
Medium Circuits					
apex7	9.2	19.3	10.1	85.7	12.3
b9	14.3	27.4	13.1	62.1	9.7
c8	10.7	20.0	9.3	44.9	6.0
f51m	17.8	22.9	5.1	26.0	6.3
Avg.			**9.3**	**54.5**	**8.6**

V. CONCLUSIONS

In this paper, we present a novel BDD-based restructuring approach to clock gating that reduces dynamic power. We use linear decomposition of BDDs to rearrange the logic functions, exploring alternate forms that are more suitable for clock gating. We then discuss the problem of optimal selection of clock gating points. By viewing this as a max-cut problem, we can have an ILP formulation that has the totally unimodular property, which allows a linear programming relaxation and optimally solves the problem in polynomial time. We then evaluate the algorithm with the industrial tools from Altera. PowerPlay Analyzer reports that our restructured circuits save 19% dynamic power over the original circuits after placement and routing. The area overhead is kept at a reasonable level of 9%. Comparing to the previous work [7], our algorithm saves more power and is much more scalable.

ACKNOWLEDGEMENTS

This work is partially supported by NFS grants CCF-0746608, CCF-0701821, and SHF-1017516.

REFERENCES

[1] F. Najm. Transition density: a new measure of activity in digital circuits. In *IEEE Trans. Comp. Aided Des. Of Integrated Circuits and Systems*. Feb. 1993.

[2] R. Fischer, K. Buchenrieder, and U. Nageldinger. Reducing the power consumption of FPGAs through retiming. In *Proc. of the 12th Conference and Workshops on Engineering of Computer-Based Systems,* Apr. 2005.

[3] Y. Hu, Y. Lin, L. He, and T. Tuan. Physical synthesis for FPGA interconnect power reduction by dual-Vdd budgeting and retiming. In *ACM Trans. Des. Autom. Electron. Syst.* 13, 2, Apr. 2008.

[4] U. Narayanan, P. Pan, and C. Liu. Low power logic synthesis under a general delay model. In *Proc. of the International Symposium on Low Power Electronics and Design,* Aug. 1998.

[5] A. Raghunathan, S. Dey, and N. Jha. Register transfer level power optimization with emphasis on glitch analysis and reduction. In *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 18(8):1114-1131, Aug 1999.

[6] L. Cheng, D. Chen, and M. Wong. GlitchMap: An FPGA Technology Mapper for Low Power Considering Glitches. In *Proc. Design Automation Conference*, 2007.

[7] Y. Hsu and S. Wang. Retiming-based logic synthesis for low-power. In *Proc. Intl. Symp. on Low Power Electronics and Design*, Aug. 2002.

[8] J. Cong , Z. Zhang. An efficient and versatile scheduling algorithm based on SDC formulation. In *Proc. Design Automation Conference*, Jul. 2006.

[9] A. J. Hoffman and J. B. Kruskal. Integral boundary points of convex polyhedra. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 22-46. Princeton University Press, 1956.

[10] L. Cheng, D. Chen, and M. Wong. DDBDD: delay-driven BDD synthesis for FPGAs. In *Proc. Design Automation Conference*, Jun. 2007.

[11] C. Y. Lee. Representation of switching circuits by binary-decision programs. In *Bell Syst. Tech. J.*, vol. 38, no. 4, Jul. 1959.

[12] S. B. Arkers. Functional testing with binary decision diagrams. In *Proc. Conf. Fault Tolerant Comput.*, 1978.

[13] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. In *IEEE Trans. Comput.*, vol. C-35, no. 8, Aug. 1986.

[14] C. Yang. BDD-based logic synthesis system. Ph.D. dissertation, EECS Dept., Univ. Massachusetts, Amherst, MA, Tech. Rep., 2000.

[15] C. Yang and M. Ciesielski. BDS: A BDD-Based Logic Optimization System. In *IEEE Trans. on CAD*, 21(7):866–876, 2002.

[16] R. Rudell. Logic Synthesis for VLSI Design. Ph.D thesis, EECS Department, Univ. of California, Berkeley, 1989.