BDD-based Logic Partitioning for Sequential Circuits[*]

Ming-Ter Kuo, Yifeng Wang, Chung-Kuan Cheng, and Masahiro Fujita[†]

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093, USA
e-mail: {mtkuo, yifeng, kuan}@cs.ucsd.edu

[†]Fujitsu Laboratories of America
Santa Clara, CA 95054, USA
e-mail: fujita@flab.fujitsu.com

Abstract— This paper presents a BDD-based approach to perform logic partitioning for sequential circuits. We use a sequential machine to model a circuit and represent the machine by its transition relation. A heuristic algorithm based on the BDD representation of the transition relation is proposed to partition the sequential machine with minimum number of input/output pins. Using BDDs and their operations, we have developed an efficient method to iteratively improve a partition. Experimental results show that our sequential logic partitioning algorithm significantly outperforms partitioning algorithms at the netlist level.

1. Introduction

Circuit partitioning is a well known problem and has been extensively studied. It aims to separate a large circuit design into smaller parts which satisfy certain constraints, and usually has an objective of minimizing the interconnections between the parts. Many approaches have been proposed to solve various partitioning problems. However, most of the partitioning algorithms operate on the physical implementations (the netlists) of the circuits and formulate the problems as graph or hypergraph partitioning problems. One disadvantage of this approach is that the quality of the partitioning results is strongly dependent on the circuit implementation. In contrast, by partitioning a circuit at the functional level, i.e. decomposing the logic function of the circuit, we can explore a larger solution space that is independent of the physical implementation.

To perform partitioning on the functions, an efficient data structure is required to represent a sequential circuit. Ordered Reduced Binary Decision Diagrams (ORBDDs or simply BDDs) are graph based data structures that provide efficient and canonical representations for Boolean functions. They have been widely used in various aspects of logic synthesis and verification. Particularly, BDDs have been used in function decomposition [4][5] and communication based logic partitioning [1][8]. However, these approaches mainly apply to combinational logics, not sequential circuits.

[*]This work was supported in part by grants from the NSF project MIP-9529077 and the California MICRO program.

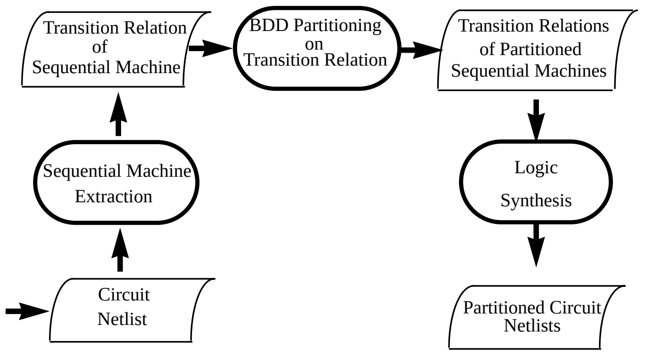
Transition Relation of Sequential Machine
BDD Partitioning on Transition Relation
Transition Relations of Partitioned Sequential Machines
Sequential Machine Extraction
Logic Synthesis
Circuit Netlist
Partitioned Circuit Netlists

Fig. 1. Circuit partitioning flow.

In this paper, we propose a BDD-based circuit partitioning approach for sequential logics. It can also be used to partition combinational logics. Fig. 1 depicts the partitioning flow of our approach for circuit partitioning. First, we extract the behavior of the circuit from its netlist by modeling the circuit as a sequential machine. The behavior of the sequential machine can be captured by a transition relation that describes the relationship among the inputs, outputs, present state and next state. We construct a BDD to represent the transition relation, where each computation path of the BDD from the source to sink '1' defines valid transitions of the sequential machine. The BDD of the transition relation is used in a heuristic algorithm to find a partition of the input/output and state variables of the relation. From the variable partition, we can derive two communicative sequential machines. The communication complexity between the two machines is evaluated by the dependency between blocks of variables in the partition. Using BDD representation of the transition relation, it is easy to observe the relationship between the variables and efficient to evaluate the number of interconnections in the partition.

The remainder of this paper is organized as follows. In Section 2, we will first give some preliminaries on sequential machines and BDD-based function decomposition. We will then state the sequential machine partitioning problem and its relation to function decomposition in Section 3. In Section 4, the BDD-based partitioning algorithm for sequential machines will be presented. We will given the experimental results in Section 5 and conclude the paper in Section 6.

ASP-DAC '97
0-89791-851-7$5.00 © 1997 IEEE

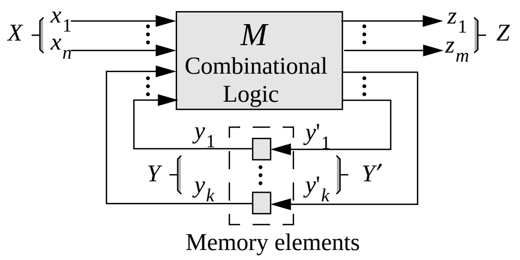

Fig. 2. Schematic representation of a sequential machine.

2. PRELIMINARIES

2.1. Sequential Machine and Transition Relation

Let $f : B^n \to B^m$ denote a (multiple-output) Boolean function with n inputs and m outputs, where $B = \{0, 1\}$.

Definition 1. Given a function $f : B^n \to B^m$, its *characteristic function* is a single-output function $F : B^n \times B^m \to B$ defined by $F(x, z) = 1$ iff $f(x) = z$.

A sequential machine consists of a combinational logic and memory elements. It can be represented schematically as in Fig. 2 and defined by Boolean variables and functions.

Definition 2. A *sequential machine M* is a 6-tuple $(X, Y, Y', Z, \lambda, \delta)$ where $X = \{x_1,\ldots, x_n\}$ is the set of input variables, $Y = \{y_1,\ldots, y_k\}$ is the set of present state variables, $Y' = \{y_1',\ldots, y_k'\}$ is the set of next state variables, $Z = \{z_1,\ldots, z_m\}$ is the set of output variables, $\lambda = (\lambda_1,\ldots,\lambda_m)$ is the output function, and $\delta = (\delta_1,\ldots,\delta_k)$ the next state function. The output function $\lambda : B^n \times B^k \to B^m$ and the next state function $\delta : B^n \times B^k \to B^k$ describe the behavior of the sequential machine and are defined by the following logic equations:[1]

$$z_i = \lambda_i(X, Y) \qquad (i = 1,\ldots,m) \qquad (1)$$

$$y_i' = \delta_i(X, Y) \qquad (i = 1,\ldots,k). \qquad (2)$$

A 4-tuple (X, Y, Y', Z) that satisfies (1) and (2) denotes a valid transition of the sequential machine in which the machine produces output Z and changes its state from state Y to state Y' on input X. Combining Equations (1) and (2) for the multiple-output functions λ and δ, we can derive a single-output function called the *transition relation* that defines all the valid transitions of a sequential machine.

Definition 3. Given a sequential machine $M = (X, Y, Y', Z, \lambda, \delta)$, its *transition relation* is the characteristic function $T : B^n \times B^k \times B^k \times B^m \to B$ of (λ, δ), i.e.

$$T(X, Y, Y', Z) = \prod_{i=1}^{m}(z_i \equiv \lambda_i(X, Y)) \wedge \prod_{i=1}^{k}(y_i' \equiv \delta_i(X, Y)). \quad (3)$$

1. For the purposes of this paper, we use the same symbol X to denote the variable set $\{x_1, \ldots, x_n\}$ and the vector $(x_1, \ldots, x_n)$.

From the definition, we see that $T(X, Y, Y', Z) = 1$ iff (X, Y, Y', Z) is a valid transition. Thus, we can use the transition relation of the sequential machine to describe its behavior.

2.2. BDD-based Function Decomposition

Definition 4. An α-g decomposition of function $f(X)$ with *bound set U* and *free set V*, $U \cup V = X$, is a transformation of f to $g(\alpha(U), V)$. If $U \cap V = \varnothing$ then the function decomposition is *disjunctive*; otherwise, it is *nondisjunctive*.

For the purpose of this paper, we consider only the disjunctive form for α-g decomposition. The following discussion is based on the results of [4][8] and we extend their concepts to decompose a multiple-output function by its characteristic function.

Let $\pi_{U\to V}$ denote a variable ordering of $X = U \cup V$ where $x_i < x_j$ for any $x_i \in U$ and $x_j \in V$ and $BDD(f, \pi_{U\to V})$ denote a BDD representing a function $f(X)$ with variable ordering $\pi_{U\to V}$.

Definition 5. Given $BDD(f, \pi_{U\to V})$, we define $cutset(BDD(f, \pi_{U\to V}), U)$ as the set of nodes in the BDD with the following properties:

(i) it is a sink node or a node corresponding to a variable in V.

(ii) it is linked by an edge from a node corresponding to a variable in U.

For example, Fig. 3 shows the BDD of function $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ with variable ordering $x_1 < x_2 < x_3 < x_4 < x_5 < x_6 < x_7$. With $U = \{x_1, x_2, x_3, x_4\}$ and $V = \{x_5, x_6, x_7\}$, the cutset of $BDD(f, \pi_{U\to V})$ is $\{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$. Actually, the ordering of variables within U and the ordering of variables with V will not affect the size of the cutset [4].

Lemma 1 [4]. Given $BDD(f, \pi_{U\to V})$ with $cutset(BDD(f, \pi_{U\to V}), U)$, the decomposition $f(X) = g(\alpha(U), V)$ where $\alpha = (\alpha_1,\ldots,\alpha_k)$ exists iff $\lceil \log_2 |cutset(BDD(f, \pi_{U\to V}))| \rceil \le k$.

Lemma 1 states that the minimum number of outputs in the α function is at least $\lceil \log_2 |cutset(BDD(f, \pi_{U\to V}))| \rceil$ since we need so many bits (or variables) to encode all the

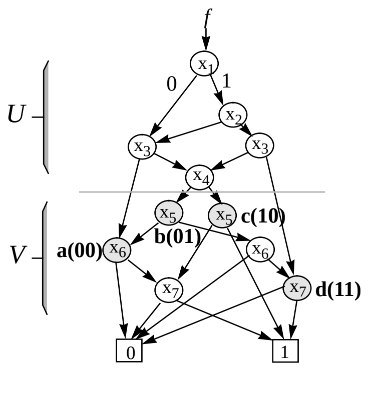

Fig. 3. An example BDD, $BDD(f, \pi_{U\to V})$, of a function f and its cutset for a bound set U.

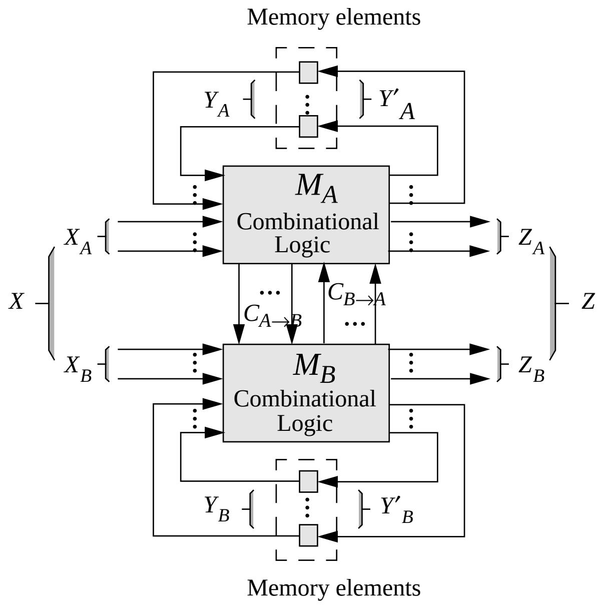

Fig. 4. A two-way partitioning of a sequential machine.

nodes in the cutset. For example, the BDD in Fig. 3 has a cutset of four nodes and each node can be encoded by a unique code of two bits.

The results in Lemma 1 can be generalized for multiple-output functions if the functions are represented by an Edge-Valued Binary Decision Diagrams (EVBDD) [4] or Multiple-Output Binary Decision Diagrams (MOBDDs) [8]. For logic partitioning, we propose to use the BDD that represents the single-output characteristic function $F(X, Z)$ instead of Z's multiple-output function $f(X)$ itself. A similar result can be derived from Lemma 1 when we use this representation.

Lemma 2 Given a function $f(X)$ and its characteristic function $F(X, Z)$, the decomposition $f(X) = g(\alpha(U), V)$ where $\alpha = (\alpha_1,\ldots,\alpha_k)$ exists iff there exists a bound set $U \subset X$ so that $\lceil \log_2 |cutset(BDD(F, \pi_{U \to (X\backslash U) \cup Z}), U)| \rceil \le k$.

3. Sequential Machine Partitioning

3.1. Problem Statement

We consider the problem of partitioning a sequential machine M into two communicating submachines M_A and M_B with a topology as shown in Fig. 4. Each set of primary inputs, primary outputs and memory elements is partitioned into two disjoint subsets. In terms of variables, the each set of variable is partitioned disjointedly and the state variables have to satisfy

$$(y_i \in Y_A \Leftrightarrow y_i' \in Y'_A) \wedge (y_i \in Y_B \Leftrightarrow y_i' \in Y'_B) \qquad (4)$$

to form a valid partition of the memory elements. Hereafter, we refer to such a partition simply as a *variable partition* of the sequential machine.

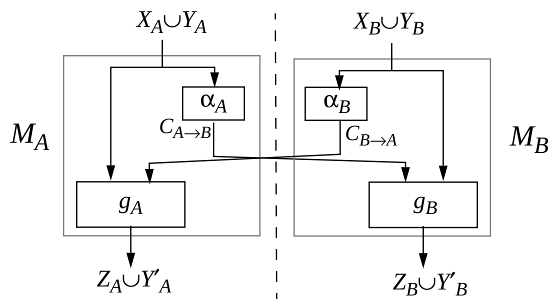

Fig. 5. Logic functions of the submachines in sequential machine partitioning.

The behavior of the original machine has to be preserved in terms of the outputs and the internal state observed, i.e. each output or memory element in M_A and M_B has an equivalent function of its corresponding output or memory element in M. To meet this requirement, in addition to the primary inputs, M_A and M_B have external inputs $C_{B\to A}$ and $C_{A\to B}$, respectively, from the combinational circuit of the other machine. We call $C_{B\to A}$ and $C_{A\to B}$ the *communication inputs/outputs* in terms of signals and *communication variables* in terms of Boolean variables that represent them. The total number of pins $|C_{B\to A}| + |C_{A\to B}|$ required for communication in each submachine is called the *communication width*.

Let PIN_A denote the number of external inputs/outputs of M_A and PIN_B denote the number of external inputs/outputs M_B, i.e.

$$PIN_A = |X_A| + |C_{B\to A}| + |Z_A| + |C_{A\to B}| \qquad (5)$$

$$PIN_B = |X_B| + |C_{A\to B}| + |Z_B| + |C_{B\to A}|. \qquad (6)$$

The objective of our partitioning problem is to minimize $\max(PIN_A, PIN_B)$ to prevent partitions with unbalanced number of input/output pins.

3.2. Submachine Construction with Minimum Communication Width

Suppose we are given a variable partition of the sequential machine. The number of pins for primary inputs/outputs in each submachine has been determined by the partition. With the objective of minimizing $\max(PIN_A, PIN_B)$, we now show how to construct the submachines with minimum communication width for a given variable partition.

For submachine M_A, its combinational logic has outputs $Z_A \cup Y'_A$ and the communication outputs $C_{A\to B}$. Let g_A and α_A be the logic functions for $Z_A \cup Y'_A$ and $C_{A\to B}$ in M_A, respectively. Then, the combinational logic function of M_A can be partitioned into α_A and g_A as shown in the left part of Fig. 5. Similarly, the combinational logic function of M_B can partitioned into α_B and g_B.

From Fig. 5, we can also observe an α-g decomposition $g_A(\alpha_B(X_B \cup Y_B), X_A \cup Y_A)$ of the multiple-output function for outputs $Z_A \cup Y'_A$ in the original machine M, i.e.

$$\left(\bigcup_{z_i \in Z_A} \lambda_i(X, Y)\right) \cup \left(\bigcup_{y'_i \in Y'_A} \delta_i(X, Y)\right)$$
$$= g_A(\alpha_B(X_B \cup Y_B), X_A \cup Y_A) \qquad (7)$$

Similarly, $g_B(\alpha_A(X_A \cup Y_A), X_B \cup Y_B)$ is an α-g decomposition of the function for outputs $Y'_B \cup Z_B$ in M, i.e.

$$\left(\bigcup_{z_i \in Z_B} \lambda_i(X, Y)\right) \cup \left(\bigcup_{y'_i \in Y'_B} \delta_i(X, Y)\right)$$
$$= g_B(\alpha_A(X_A \cup Y_A), X_B \cup Y_B)\,. \qquad (8)$$

Using BDD-based function decomposition, we can construct the submachines with minimum communication, $C_{B\to A}$ and $C_{A\to B}$, by decomposing the function for $Z_A \cup Y'_A$ in M with bound set $X_B \cup Y_B$ and the function for $Z_B \cup Y'_B$ in M with bound set $X_A \cup Y_A$. After decomposition, α_A and g_A are composed to the combinational function of M_A; α_B and g_B are composed to the combinational function of M_B.

4. BDD-based Sequential Machine Partitioning Algorithm

In the previous section, we have shown how to minimize the communication width if a variable partition of the sequential machine is given. The communication width, $|C_{A\to B}| + |C_{B\to A}|$, can be calculated by identifying the cutsets in the BDDs. In this section, we propose a BDD-based logic partitioning algorithm for sequential machines. The algorithm uses the BDD representation of the single-output transition relation of a machine to represent its multiple-output logic function. Since an exact method of enumerating all variable partitions is impractical, we use an iterative improvement approach. The idea is to locally change a variable partition by relocating a variable and the cutline in the BDD. For each new partition, the cutset of the BDD is computed to evaluate the quality of the variable partition.

4.1. Transition Relation Approach

In Section 2.1, we have shown that the behavior of a sequential machine $M = (X, Y, Y', Z, \lambda, \delta)$ can be described by its transition relation T where $T(X, Y, Y', Z) = 1$ iff $(\lambda(X, Y), \delta(X, Y)) = (Z, Y')$. There are two reasons for using the transition relation instead of the logic function of a sequential machine. First, the transition relation is a single-output function and the cutset of the BDD in function decomposition can be easily identified by Lemma 2. Second, a variable partition in a sequential machine involves input and output partitioning simultaneously. Since the transition relation $T(X, Y, Y', Z)$ is a function of the input and output variables, we can apply variable ordering operations on its BDD representation to locally change a variable partition. Although there are EVBDDs [4] and MOBDDs [8] proposed to represent multiple-output function, we can only use their representations to partition input variables but not output variables.

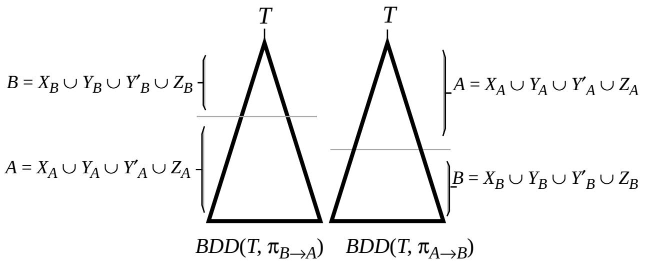
T
T
$B = X_B \cup Y_B \cup Y'_B \cup Z_B$
$A = X_A \cup Y_A \cup Y'_A \cup Z_A$
$A = X_A \cup Y_A \cup Y'_A \cup Z_A$
$B = X_B \cup Y_B \cup Y'_B \cup Z_B$
$BDD(T, \pi_{B\to A})$
$BDD(T, \pi_{A\to B})$

Fig. 6. BDD representations of the transition relation.

4.2. Evaluating a Variable Partition

Our algorithm searches different variable partitions using the BDD representation of the transition relation. For each partition, the pin requirement, $\max(PIN_A, PIN_B)$, should be calculated to evaluate the solution. The BDD-based approach has made the calculation for communication width, $|C_{A\to B}| + |C_{B\to A}|$, efficient by identifying the cutsets of two α-g decompositions (Section 3.2) in the BDDs. There are two cutsets to be identified, one with bound set $X_B \cup Y_B$ and one with bound set $X_A \cup Y_A$. Therefore, we need to maintain two BDDs for the transition relation simultaneously, where each variable ordering is reverse to the other.

Fig. 6 depicts the two BDDs, $BDD(T, \pi_{B\to A})$ and $BDD(T, \pi_{A\to B})$, where $A = X_A \cup Y_A \cup Y'_A \cup Z_A$ and $B = X_B \cup Y_B \cup Y'_B \cup Z_B$. The cutlines in the two BDDs define a variable partition. We can then compute the cutset size with respect to the cutline in each BDD to calculate $|C_{B\to A}|$ and $|C_{A\to B}|$. Note that the exact cutset sizes should be computed from the BDDs representing the characteristic function for outputs $Z_A \cup Y'_A$, i.e.

$$\prod_{z_i \in Z_A} (z_i \equiv \lambda_i(X, Y)) \wedge \prod_{y'_i \in Y'_A} (y_i' \equiv \delta_i(X, Y)) \qquad (9)$$

derived from the Equation (7) and the characteristic function for outputs $Z_B \cup Y'_B$, i.e.

$$\prod_{z_i \in Z_B} (z_i \equiv \lambda_i(X, Y)) \wedge \prod_{y'_i \in Y'_B} (y_i' \equiv \delta_i(X, Y)) \qquad (10)$$

derived from Equation (8), respectively. Although these BDDs for the characteristic functions can be constructed respectively from $BDD(T, \pi_{B\to A})$ and $BDD(T, \pi_{A\to B})$, we use $cutset(BDD(T, \pi_{B\to A}), B)$ and $cutset(BDD(T, \pi_{A\to B}), A)$ computed from the two BDDs maintained by the algorithm for efficiency. This means that we approximate the communication width by

$$\lceil \log_2 |cutset(BDD(T, \pi_{B\to A}), B)| \rceil$$
$$+ \lceil \log_2 |cutset(BDD(T, \pi_{A\to B}), A)| \rceil\,. \qquad (11)$$

Since the communication width is logarithmic in the sizes of the cutsets, the amount of over-estimation should be insignificant.

4.3. *Heuristic Algorithm by Partition Enumeration*

Fig. 7 shows our BDD-based partitioning algorithm to find a variable partition in sequential machines. The heuristic algorithm locally changes a variable partition and improves the solution iteratively.

Given an initial variable partition (A, B) and the transition relation T, two BDDs are first constructed with variable orderings $\pi_{B \to A}$ and $\pi_{A \to B}$, respectively. Each variable ordering of the two BDDs is kept totally as the reverse to the other in the algorithm. Then, each variable is subsequently considered and relocated in the BDD to a new position, if the new variable partition after the relocation is an improved solution. The new position also serves as the new cutline for defining the variable partition. To avoid extremely unbalanced partition, we set a *cutline position window* (*CPW*) for the new position of each variable and the cutline. The *CPW* is centered in the middle of the variable ordering and is defined by a width parameter ω.

```
Algorithm

INPUT: initial variable partition ($A$, $B$) and the transition relation $T$ for
          the sequential machine.
OUTPUT: final variable partition ($A$, $B$).
{
/* $N$: the total number of input/output variables or state variable pairs */
/* CPW: the cutline position window */
/* $\omega$: the width parameter of CPW */
   CPW = [$\lceil N/2 \rceil - \omega$, $\lceil N/2 \rceil + \omega$];
   construct $BDD(T, \pi_{B \to A})$ and $BDD(T, \pi_{A \to B})$;
   min_pin_num = pin requirement for variable partition ($A$, $B$);
   repeat {
      improved = false;
      for each variable or variable pair $v$ in $A \cup B$ {
         old_position = $v$'s current position;
         sift_and_search($v$, new_position, pin_num);
         if (pin_num < min_pin_num) {
            improved = true;
            min_pin_num = pin_num;
            cutline = new_position;
            sift $v$ to new_position in $BDD(T, \pi_{B \to A})$ and $BDD(T, \pi_{A \to B})$;
         }
         else sift $v$ back to old_position;
   } until (improved == false);
   return ($A$, $B$) defined by cutline;
}

sift_and_search($v$, new_position, pin_num)
{
/* $p$ : position in variable ordering with indices in $BDD(T, \pi_{B \to A})$
      and $BDD(T, \pi_{A \to B})$ reverse to each other. */
   pin_num = $\infty$;
   for each position $p \in$ CPW {
      sift $v$ to position $p$ in $BDD(T, \pi_{B \to A})$ and $BDD(T, \pi_{A \to B})$;
      temp_cutline = $p$;
      compute the cut sets defined by temp_cutline in $BDD(T, \pi_{B \to A})$
            and in $BDD(T, \pi_{A \to B})$;
      compute $PIN_A$ and $PIN_B$;
      if (max($PIN_A$, $PIN_B$) < pin_num) {
         pin_num = max($PIN_A$, $PIN_B$);
         new_position = $p$;
      }
   }
   return new_position, pin_num;
}
```

Fig. 7. A BDD-based algorithm of finding a variable partition in sequential machine.

The searching step *sift_and_search* for a variable's new position is based on a variable *sifting* operation which was initially proposed in [6] to reduce the size of a BDD. The sifting operation searches the optimum position for a variable in the *CPW* assuming the order of all other variables remains fixed. For each v being considered, we first 'sift' v (exchanging v with its next neighbor iteratively) to the first position of the *CPW*. Then v is sifted towards the end of the *CPW* and the best position with minimum pin requirement is returned. For example, with a variable ordering $<x_1, x_2, x_3, x_4, x_5>$, we can repeatedly 'sift' variable x_1 'downwards' in the *CPW* = [2, 4] from position 2 to position 4 to obtain variable orderings $<x_2, x_1; x_3, x_4, x_5>$, $<x_2, x_3, x_1; x_4, x_5>$ and $<x_2, x_3, x_4, x_1; x_5>$ in sequence. The position of the current variable v being sifted defines the cutline that separates v from the variable in front of it as shown by the semicolon (;) in the example. Note that the variable partition in the transition relation requires each pair of present state variable y_i and next state variable y'_i to stay in the same block. Therefore, each (y_i, y'_i) pair is considered as an entity whenever the sifting operation is performed, i.e. y_i and y'_i are always adjacent in the ordering and the cutline can never separate them.

After the new position is returned by *sift_and_search*, v is sifted to the best position returned if the partition defined by the new cutline is a better solution. Otherwise, v is sifted back to its original position since it has been sifted to the last position in the *CPW*. Note that the each variable ordering of the two BDDs is the reverse or the other. Thus, all the sifting operations in the algorithm should be in opposite directions on the two BDDs.

5. EXPERIMENTAL RESULTS

We have implemented the BDD-based heuristic algorithm and applied it to sequential circuit partitioning. Experiments on a set of MCNC benchmarks were conducted to demonstrate the effectiveness of the algorithm. For each test case, we extract the function of the circuit from its netlist at the gate level and construct the BDD representation of its transition relation by calling routines of BDD operations in SIS [7]. Then the BDD is used in our heuristic algorithm to find a variable partition with minimum pin requirement for partitioning the sequential machine. For comparison, the same netlist of each test case is also partitioned by the two-way Ratio Cut II netlist partitioning program [9] to minimize the input/output pins. The Ratio Cut II program is an improved implementation of the original ratio cut algorithm.We choose it as the netlist partitioning algorithm

TABLE 1. RESULTS OF SEQUENTIAL CIRCUIT PARTITIONING

Test Cases			Ratio Cut			BDD-based Partitioning			
Name	**FF**	**I/O**	**Max I/O**	**Pin1/Pin2**	**Total**	**Max I/O**	**Pin1/Pin2**	**Total**	**FF1/FF2**
s27	3	5	5	4 / 5	9	4	3 / 4	7	1 / 2
s344	15	20	22	14 / 22	36	19	17 / 19	36	6 / 9
s382	21	9	13	12 / 13	25	8	8 / 7	15	12 / 9
s386	6	14	20	20 / 18	38	11	11 / 9	20	3 / 3
s400	21	9	11	4 / 11	15	8	8 / 7	15	10 / 11
s420	16	19	20	17 / 20	37	12	12 / 11	23	9 / 7
s444	21	9	11	8 / 11	19	8	7 / 8	15	12 / 9
s510	6	26	51	33 / 51	84	23	23 / 23	46	4 / 2
s820	5	37	53	41 / 53	94	24	23 / 24	49	2 / 3
s832	5	37	49	40 / 49	89	24	23 / 24	49	2 / 3
s953	29	39	82	82 / 71	153	29	26 / 29	57	16 / 13
s1488	6	27	70	65 / 70	135	21	21 / 20	41	3 / 3
s1494	6	27	75	58 / 75	133	21	18 / 21	39	3 / 3
s208.1	8	11	11	10 / 11	21	10	10 / 7	19	4 / 4
scf	7	83	152	152 / 123	275	54	53 / 54	107	5 / 2

because it is efficient, and yet still generates comparable results against state-of-the-art netlist partitioning algorithms. To prevent partitions with extremely unbalanced sizes, we set the total node size ratio of the two subcircuits between 1/2 and 2.

Table 1 shows the characteristics of the benchmarks and the partitioning results. The numbers of flip-flops and input/output pins are listed for each benchmark circuit. In the last two columns, we compare the numbers of I/O pins of the results produced by our algorithm against the results by Ratio Cut II. In terms of total I/O pins or the maximum I/O pins of the two subcircuits, our BDD-based partitioning algorithm outperforms the netlist-based Ratio Cut II partitioning algorithm in all the test cases. In particular, the results of our algorithm for test cases **s953**, **s1488** and **s1494** require only 1/3 to 1/4 of the total number I/O pins by Ratio Cut II. The last column in Table 1 also lists the numbers of flip-flops in the submachines. We can see that our algorithm produces partitioned sequential machines with balanced number of flip-flops as well as number of I/O pins. It is an encouraging result for obtaining balanced circuit sizes after the submachines are synthesized.

6. Conclusions

We proposed a BDD-based algorithm to perform logic partitioning for sequential machine. Our approach is to use the BDD that represents the transition relation of the sequential machine to find a good variable partition. It is not difficult to see that our algorithm also can be applied to partition combinational logics (with no feed-back loop) by using the BDD representation of the characteristic function. Unlike other combinatorial logic partitioning algorithms which merely based on the α-g decomposition model, our approach can produce partitions with bidirectional communication, instead of unidirectional. Currently, we are generalizing the sequential machine partitioning to allow non-disjoint partition of inputs for possible improvements.

REFERENCES

[1] M. Beardslee, B. Lin, and A. Sangiovanni-Vincentelli, "Communication based logic partitioning," *Int. Conf. on CAD*, 1992, pp. 32-37.

[2] J. R. Burch, E. M. Clarke, and D. E. Long, "Symbolic model checking with partitioned transition relations," *Int. Conf. on VLSI*, Edinburg, Scotland, Aug. 1991.

[3] D. Geist and I. Beer, "Efficient model checking by automated ordering of transition relation partitions," *Int. Conf. on Computer Aided Verification*, Jun. 1994, pp. 299-310.

[4] Y.-T. Lai, M. Pedram, and S. B. K. Vrudhula, "BDD based decomposition of logic functions with application to FPGA synthesis," *30th Design Automation Conference*, 1993, pp. 642-647.

[5] Y.-T. Lai, M. Pedram, and S. B. K. Vrudhula, "EVBDD-based algorithms for integer linear programming, special transformation, and function decomposition," *IEEE Trans. on CAD*, Aug. 1994, pp. 959-975.

[6] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *Int. Conf. on CAD*, 1993, pp. 42-47.

[7] E. M. Sentivich *et al.*, "SIS: a system for sequential circuit synthesis," *Mem. UCB/ERLM92/41*, U. C. Berkeley, May 1992.

[8] M. Shih, "Delay optimization based on BDD and communication complexity," *Mem. UCB/ERLM92/117*, U. C. Berkeley, 1992.

[9] C.-W. Yeh and C.-K. Cheng, "Ratio cut program II," *Technical Report CS92-250*, U. C. San Diego, Jul. 1992.