Partitioning Combinational Circuits in Graph and Logic Domains

David Ihsin Cheng, Shih-Chieh Chang and Malgorzata Marek-Sadowska
Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
U.S.A

Abstract:

Circuit partitioning is the process of dividing a given circuit into subcircuits. Traditionally, this problem is done by first modeling a circuit as a graph (hypergraph) and then partitioning is performed on the graph. Up to now, almost all existing circuit partitioning techniques use only graph domain information ([4] [5] [7] [15]). Modeling a circuit partitioning problem as a graph partitioning problem, in spite of its simplicity, loses optimality. In this paper, we propose an efficient method that locally combines logic domain and graph domain information. We first perform partitioning using any traditional partitioning method on graph domain. Then we try to locally improve this initial partition by considering the logic functions performed in vertices that are around the boundary of the initial partition. We have implemented our logic domain method and combined our method with the well known Fiduccia-Mattheyses graph domain method. This combined algorithm iteratively improves a given partition by switching between logic domain and graph domain. Our method is very feasible and efficient because we only consider a small portion of a circuit when we are in logic domain. Only when we are in graph domain we perform global iterative improvement. Experimental results on MCNC combinational benchmarks generate very good final solutions. In addition, because of the switching between logic domain and graph domain, our method generates a set of solutions that have a variety of areas and cut set sizes.

1 Introduction

Circuit partitioning is the process of dividing a given circuit into subcircuits. Traditionally, this problem is done by first modeling a circuit as a graph (hypergraph) and then partitioning is performed on the graph. Graph partitioning problems, for different objective functions (e.g.: min-cut, minimum ratio cut), are known to be NP-complete ([6] [15]). Many heuristics for graph partitioning problems have been developed.

We use the term *graph domain* to refer to the information concerning only connections among nodes, and the term *logic domain* to refer to the information concerning the function performed by each node. Almost all existing circuit partitioning techniques use only graph domain information ([4] [5] [7] [15]). Modeling a circuit partitioning problem as a graph partitioning problem, in spite of its simplicity, loses optimality. Since each vertex, which models a gate in a given circuit, actually performs some logic function, partitioning on a graph unnecessarily gives up potential useful logic domain information. Take the relationship between logic synthesis and partitioning as an example. On one hand, the process of area or timing minimization in logic synthesis uses only logic information without considering the potential difficulties the subsequent graph model would encounter when the circuit needs to be partitioned. On the other hand, the graph partitioning stage partitions the modeling graph without considering the logic functions performed in vertices. In other words, the traditional separation of the logic domain information and the graph domain information actually traded simplicity of modeling for quality of partitioning.

In this paper, using the concept of *alternative wires* in the logic domain, we propose a very efficient method that is able to preserve a local optimal solution in the graph domain while a different graph, representing the same circuit, is generated. When a conventional graph partitioning technique reaches a local optimal solution, our proposed technique generates a different graph that is logically equivalent to the original circuit, and that has equal or better partitioning solution. Taking the newly generated graph and the current partitioning solution, a conventional graph partitioning technique can then continue searching for better solutions on this different graph. Essentially, instead of abiding by a strict graph model (in which case we are not allowed to change the graph, as in most existing techniques), we take advantage of the fact that a vertex is free to merge or split with other vertices and that a wire is free to be replaced by its alternative wires, as long as we keep the functionality correct.

To the best of our knowledge, the only studies that utilize logic information in the partitioning problem are [1] [8] [10]. In [1], the problem of assigning primary inputs (PIs) and primary outputs (POs) to a multi-chip environment is investigated. The proposed method essentially tries all combinations of assignments of PIs and POs to a given multi-chip environment. Calculating the minimum necessary communication lines between the partitioned blocks for each assignment, the method can then choose the optimal PIs and POs assignment. There are two weaknesses in this method. First, even though the method avoids physically exhausting all the combinations of assignments by integrating all solutions in a logic function, the computational cost is much more expensive than traditional graph domain partitioning. Second, the area estimation for each partitioned block is difficult because the proposed method starts even before any logic optimization and technology mapping processes. In [8] and [10], the authors avoid performing partitioning on graph domain by allowing vertex replication, which can be viewed as a restricted operation on the logic domain information. All of these three approaches demonstrate the effectiveness of performing partitioning by considering logic domain information.

We have implemented our technique and combined it with the well known Fiduccia-Mattheyses (FM) graph domain partitioning algorithm [5]. This combined algorithm iteratively improves a given partition by switching between logic domain and graph domain. Experimental results on MCNC benchmarks generate very good final solutions.

The remainder of this paper is organized as follows. Section 3 presents our algorithms. Section 4 shows our experimental results. Section 5 concludes this paper.

2 Definitions and problem formulation

2.1 Definitions

A combinational circuit is modeled as a directed graph $G < V, E >$, where V is the vertex set and E is the edge set. Let PI denote the set of primary inputs and PO denote the set of primary outputs. The *distance* of two vertices p and q, denoted $dist(p,q)$, is the number of vertices of the shortest path (undirected) between p and q.

Let $\pi = (L, R)$ be a partition, where $L \cap R = \phi$ and $L \cup R = V$.

The *cut edges* are those edges that are "cut" by the partition, i.e., $\{(p,q) \,|\, (p,q) \in E, (p \in L, q \in R) \;\; or \;\; (p \in R, q \in L)\}$. The *border zone* B is defined as

$$B = \{\, p \mid (p,q) \in E,\; p \notin PI \cup PO,\; (p \in L,\; q \in R) \;\; or \;\; (p \in R, q \in L)\} \tag{1}$$

Given a constant k, the *flexible zone* F of a partition π is defined as

$$F = \{\, p \mid dist(p,q) \leq k,\; p \notin PI \cup PO,\; q \in B\} \tag{2}$$

Note that the flexible zone obviously contains the border zone.

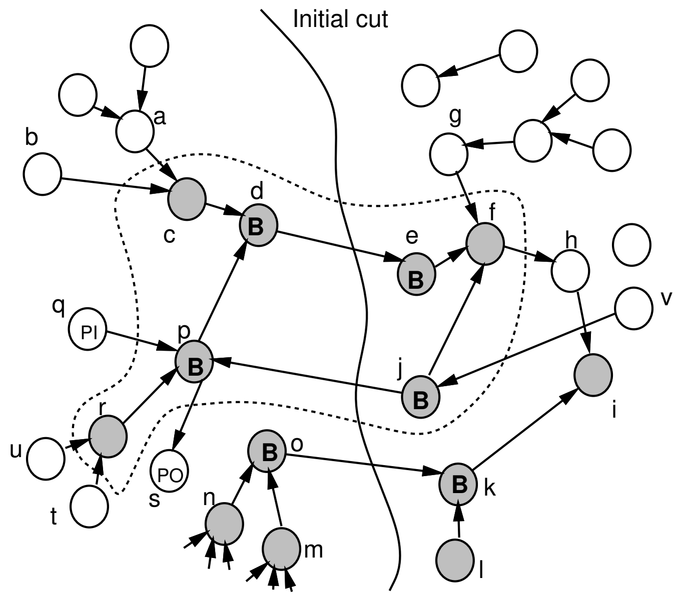

Figure 1: peephole and the related sets

A *peephole* P is a subset of vertex set V, excluding PIs and POs. That is, P is a peephole if $P \subset V$, $P \cap PI = \phi$, $P \cap PO = \phi$. [1]

The *fanin sets* FI_L, FI_R of a peephole P are defined as, respectively,

$$FI_L = \{\ p \mid (p, q) \in E,\ p \in L,\ p \notin P,\ q \in P\} \tag{3}$$

$$FI_R = \{\ p \mid (p, q) \in E,\ p \in R,\ p \notin P,\ q \in P\} \tag{4}$$

FI_L and FI_R are those vertices that fanin into the peephole P from blocks L and R, respectively. The *fanout sets* FO_L, FO_R of a peephole P are defined as, respectively,

$$FO_L = \{\ p \mid (p, q) \in E,\ q \in L, q \notin P,\ p \in P\} \tag{5}$$

$$FO_R = \{\ p \mid (p, q) \in E,\ q \in R, q \notin P,\ p \in P\} \tag{6}$$

FO_L and FO_R are those vertices in peephole P that fanout to blocks L and R, respectively. Notice that some nodes in peephole P may belong to both fanout sets, meaning that they fanout to both blocks L and R. For each vertex v in the flexible zone, the *fanin (fanout) cone* of v is the set of all the vertices and edges of the transitive fanin (fanout) of v within the flexible zone. Further, the *cut influence* of v is the number of cut edges in the fanin cone or fanout cone of v within the flexible zone. The *basic peephole* of a vertex v in the flexible zone is the set of vertices that have a path (undirected) completely within the flexible zone to v.

Fig. 1, which shows a partial view of an initial cut, illustrates the above definitions. The border zone consists of those nodes marked with "B". The shaded nodes form the flexible zone with distance $k = 1$ from the border zone. Vertices have different cut influence. For example, node e has two cut edges in its fanin cone and zero cut edges in its fanout cone, hence a cut influence of two. Node k has one cut edge in its fanin cone and zero cut edges in its fanout cone, hence a cut influence of one. The dotted line circles the basic peephole of node e. This basic peephole has FI_L as $\{a, b, q, u, t\}$, FI_R as $\{g, v\}$, FO_L as $\{p\}$ (to s), and FO_R as $\{f\}$ (to h).

Informally, the border zone contains all the vertices that have direct connections to the other side of the partition. The flexible zone contains all the vertices that are "close" to the cut boundary (within a given distance k). The flexible zone is the area where we will group certain vertices into a peephole and look into their functions in the logic domain. The cut influence

[1] In fact, a peephole doesn't have to exclude POs. We define it in this way for clarity of presentation.

of each vertex v in the flexible zone indicates the degree of influence on the cut edges. If a vertex v in the flexible zone has high cut influence, we know v in fact influences many cut edges and is close to the cut boundary. Forming the basic peephole for v, we will locally improve the quality of the peephole according to a cost function that will be described in subsequent sections. Once we find a good peephole P, the fanin and fanout sets, FI_L, FI_R, FO_L, and FO_R, will be regarded as the primary inputs and primary outputs of P. In the logic domain, the function performed in each $f \in FO_L \cup FO_R$ can be expressed in terms of the nodes in $FI_L \cup FI_R$. Additionally, as long as we can generate the same function for each of these fs, manipulating and changing signals in peephole P would not change the behavior of the original network.

2.2 Peephole partitioning problem

Now we propose an interesting *peephole partitioning problem* defined as follows. Given a peephole P and an assignment FI_L, FI_R, FO_L, and FO_R of P to two partitioned blocks, we ask what the minimal number of communication lines that have to exist between these two blocks is. Although we still seek the same goal of minimal number of cut lines crossing the cut boundary, this new problem is different from conventional partitioning problem in two aspects. First, here we do not care about the area balance of the two partitioned blocks implementation. This is because we have the freedom to choose a small enough peephole so that the result of the peephole partitioning, as far as area is concerned, does not affect too much the original partitioning. Second, here we do not allow "free" replication of the PIs. Each "replication" of a signal s actually means sending signal s from its assigned block to the complemented block.

Informally, the peephole partitioning problem can be viewed as a problem of signal encodings and decodings. From block L, certain information has to be encoded and sent to block R, which has to be able to decode the information and produce FO_R. Similarly, from block R, certain information has to be encoded and sent to block L, which has to be able to decode the information and produce FO_L.

2.3 Don't-cares in the peephole partitioning problem

We can analyze the peephole partitioning problem from the communication complexity viewpoint. Past research (e.g.: [9] [12]) on complexity analysis mainly focus on solving the following problem. Given a function $f(X)$, find the minimum m so that $f = g(\alpha^1(X_L), \alpha^2(X_L), \cdots, \alpha^m(X_L), X_R)$, where X_L and X_R form an input partition and g, α^1, α^2, $\cdots$, α^m are boolean functions. One of the major difficulties in this problem is the determination of the partitioning on the input set X into X_L and X_R. In the definition of our peephole partitioning problem, however, the input partitioning is actually given (as FI_L and FI_R). Unfortunately, optimally solving the peephole partitioning problem still appears to be difficult. This is because of two types of don't-cares: external don't-cares and the internal satisfiability don't-cares.

The external don't-cares to the peephole partitioning problem exist because of the fact that the signals in FI_L or FI_R are not independent. These don't-cares can be obtained by performing an image computation operation [14] for all FI_L and FI_R in terms of PIs. The internal satisfiability don't-cares arise when a signal $s = f(FI_L)$ is sent, say, from L side to R side, it introduces the satisfiability don't care $\overline{s \equiv f(FI_L)}$ on the R side. The use of the external and the internal satisfiability don't-cares can affect the result of communication lines reduction. It is not clear how to use these don't-cares to further reduce communication lines. It is also not clear which s should be chosen to be sent to the complemented side and in what order these signal s's should be constructed.

The basic problem formulation in [9][12] is not suitable for our peephole partitioning problem. We are currently studying methods to optimally solve the problem. In this paper, therefore, we only propose a heuristic that incorporates logic domain information into partitioning and demonstrate the effectiveness of this direction.

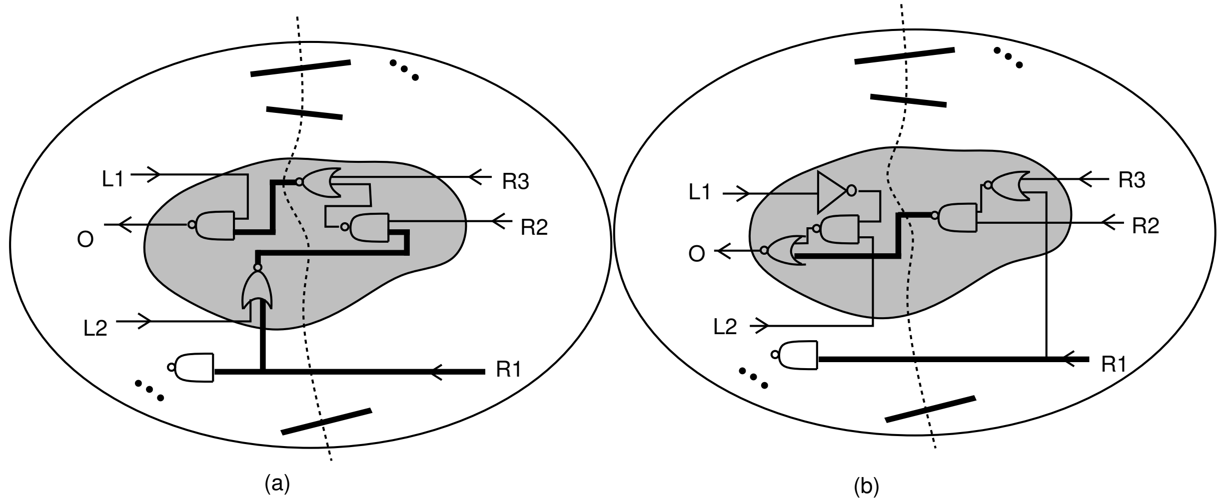

Figure 2: Using logic domain information to improve a partition

2.4 An example

We give a simple example to demonstrate the effectiveness of logic domain information. Fig. 2(a) shows a peephole that contains four nodes.[2] Because of the connections outside the peephole (not shown in Fig. 2), there is little we can do to improve the current partition in the graph domain. Now we resort to logic domain. Using the technique that will be described in the subsequent sections, we obtain the partition shown in Fig. 2(b). As a result, the number of cut lines is reduced from 6 to 5 (2 to 1 inside the peephole), and the behavior of the whole network is intact. Note that even though the function of the whole network is the same, the network has actually been changed. This gives us a good opportunity to switch back to the graph domain and explore a completely different search space in the graph domain. This example illustrates the poor cooperation between the logic synthesis process that uses only logic domain information and the partitioning process that uses only graph domain information, as we mentioned in Section 1.

3 Algorithm

In this section, we describe our algorithm, which follows the same flavor as illustrated in the previous example. Fig. 3 shows a high level description. Line 0 in Fig. 3 generates an initial partition using FM-method[3]. Line 1, which is to be described in detail in Section 3.1, picks up a "good" peephole. We then calculate, inside this selected peephole, the minimal number of communication lines that have to exist so that the behavior of the original network is intact. This is done in Line 2 and will be explained in Section 3.2. If the calculated minimal number of communication lines is smaller than the original cut lines in the peephole (Line 3), we then realize the implementation of this minimal number of communication lines by resynthesizing the peephole (Line 4, details also in Section 3.2). After resynthesizing a peephole, the new network becomes different from the original network in graph domain (but remains the same in logic domain). In other words, the graph representation of the network has been changed (Line 5). We then apply the FM-method on this new network and in fact explore a completely different

[2]This example is taken directly from our experiment. The original circuit is the MCNC benchmark *cmb.blif*. The sis script *script.boolean* is applied to perform technology independent optimization. Then we map *cmb.blif* using a library consisting of a 2-input NAND gate, a 2-input NOR gate, and an inverter. After running FM method for 500 times, we obtain a solution that has 6 cut lines, which we believe is the optimal solution in graph domain. Using the information of cut influence, we obtain the peephole as shown in Fig. 2(a). Using our heuristic to synthesize the minimal communication lines, we obtain the result as shown in Fig. 2(b).

[3]Line 0 and line 6 of Fig. 3 can be changed to any graph domain method.

```
Algorithm partition(G)
0.  generate initial partition on input graph G using FM-method.
1.  while (a good peephole P can be found) {
2.          calculate the minimum number of communication lines m necessary in P
3.          if (m < the number of cut lines in P) {
4.                  resynthesize and technology map P
5.                  replace old P with new P
6.                  perform FM-method on the new network
7.          }
8.  }
```

Figure 3: High level description of our algorithm

```
Algorithm choosing(G)
0.  Sort all nodes in flexible zone according to their cut influence.
1.  i = 0
2.  while ( i < # of nodes in flexible zone && # of ret_array element < N_PEEP {
3.      peephole = basic_peephole(node i)
4.      improve_basic_peephole(peephole)
5.      if (is_good(peephole))
6.        insert peephole into ret_array
7.      i = i + 1
8.  }
9.  sort(ret_array) according to cost function of peepholes
9.  return (ret_array)
```

Figure 4: choosing a good peephole

search space (Line 6). We switch back and forth (the while loop in Line 1) between logic domain and graph domain in this fashion until no improvement can be further done.

Note that our method is very feasible and efficient because we only consider a small portion of a circuit when we are in logic domain. Only when we are in graph domain we perform global improvement using FM-method. The next two subsections explain our method for Line 1, choosing a good peephole, and for lines 2 and 4, calculating and resynthesizing the minimal number of communication lines.

3.1 Choosing a good peephole

Fig. 4 shows our algorithm for choosing a "good" peephole. This algorithm returns a sorted array of good peepholes. The algorithm first sorts all nodes in the flexible zone according to their cut influence value (Line 0). Starting with the node that "influences" the most cut edges, we form its basic peephole (Line 3). This basic peephole should contain all the nodes that together have a good potential of improving the cut lines that are included in it. The quality of a peephole is affected by the following factors. We describe them in descending importance.

• Number of cut lines, $|cut|$, : The more cut lines are included in a peephole, the better a chance to improve the partition. So we would like to maximize $|cut|$.

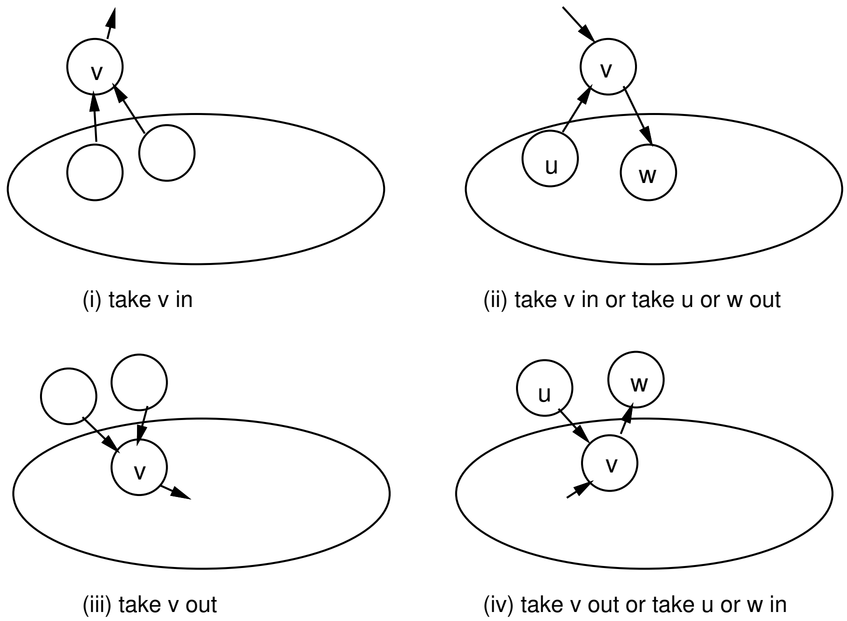

Figure 5: Example rules for improving a partition

• Number of fanouts, $|FO|$, : As mentioned earlier, we need to keep the functions of all fanouts intact. Each fanout therefore behaves like a constraint in our resynthesis process. So we would like to minimize $|FO|$.
• Number of fanins, $|FI|$, : We do not wish to spend too much CPU time doing "large-scale" logic resynthesis. Therefore, we need to limit $|FI|$ to some constant c_1 so that we can build BDDs quickly.
• Minimal number of fanins from each side, $|FI_L|$ and $|FI_R|$,: Since there is little we can do in logic domain if, for example, fanin from L block has only 1 or even zero signal, we would like to have $|FI_L|$ and $|FI_R|$ greater than some constant c_2.
Our cost function of evaluating a peephole is, therefore,

$$merit(peephole) = c_3 * |cut| - c_4 * |FO| \tag{7}$$

subject to

$$|FI| < c_1,\ |FI_L| > c_2,\ |FI_R| > c_2. \tag{8}$$ [4]

Line 4 in Fig. 4 performs local improvement on the quality of the basic peephole. This local improvement consists of several rule-based transformations. Fig. 5 depicts some example rules. Rule (i) reduces $|FO|$ by 1, rule (iii) reduces $|FI|$ by 1, and rules (ii) and (iv) reduces $|FI|+|FO|$ by 1. These rules are simple observations and have the effect on improving the quality of a peephole, thereby lessen the constraints we will face later in the peephole resynthesis stage. Line 5 in Fig. 4 determines if the peephole under consideration is good or not according to the above cost function. If it is good (satisfying Eq. 8), we insert it into the returned array. Otherwise we discard it. The main loop of our algorithm stops either when we exhaust all the nodes in the flexible zone or, to be more efficient, when we find the first N_PEEP [5] good peepholes (line 2). The remaining lines in Fig. 4 are self-explanatory.

[4] In our experiment, c_1, c_2, c_3, c_4 are set to 30, 3, 2, and 1, respectively.

[5] In our implementation, N_PEEP is set to 5, meaning that we only consider 5 good peepholes maximally. We believe that if the first few "good" peepholes that have "big influences" cannot improve the partition, we probably should stop.

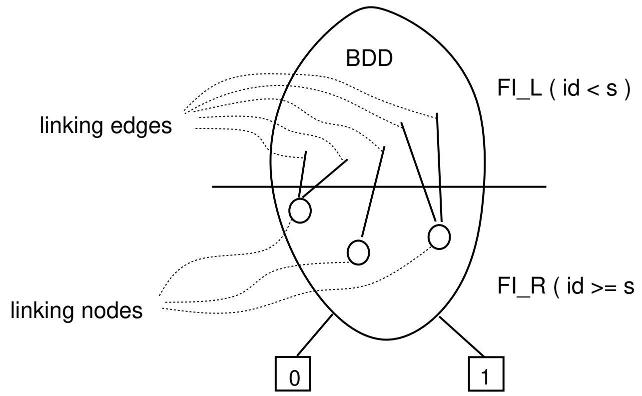

Figure 6: linking nodes and linking edges

3.2 Calculation and resynthesis of peepholes

As mentioned in Section 2.3, optimally solving the peephole partitioning problem appears difficult. In this paper, therefore, we do not take the don't-cares stated in section 2.3 into consideration. Now we will present our heuristic for the peephole partitioning problem. Instead of solving for exact minimized solution, we perform two passes of independent minimization processes. In the first pass we minimize the set of communication lines FI_L has to provide for FO_R (from side L to side R); in the second pass we minimize the set of communication lines FI_R has to provide for FO_L (from side R to side L). For simplicity, we only explain the way we do minimization from side L to side R. The other direction, from side R to side L, follows in a similar way.

We are given the set of functions that fanout to the R side (FO_R), the set of fanins from the L side (FI_L), and the set of fanins from the R side (FI_R). The goal is to find the minimum number of m such that $FO_R = g(\alpha^1(FI_L), \alpha^2(FI_L), \cdots, \alpha^m(FI_L), FI_R)$. This problem has been studied by other researchers ([1] [9] [12]), here we show an efficient implementation operating on BDDs [6].

We need a few simple definitions to present our approach. We can view a BDD as a directed graph $< V, E >$. Let $s = |FI_L|$ and $t = |FI_R|$. For each $f_i \in FO_R$, we construct a BDD by ordering the variables in FI_L as the top s variables and the variables in FI_R as the lower t variables. The *linking nodes* in BDD(f_i) are $\{q \mid (p,q) \in E,\ id(p) < s \ and\ id(q) \geq s\}$. All the edges that point to a linking node is called *linking edges* of the linking node (see Fig. 6). For each linking node j in BDD(f_i), the associated *linking function* f_{ij} is constructed by redirecting all the linking edges of linking node j to leaf node "1" and by redirecting all linking edges of other linking nodes in BDD(f_i) to leaf node "0".

Viewing FO_R as a multiple output function as in $FO_R\ :\ B^{FI_L} \times B^{FI_R} \rightarrow B^{FO_R}$, we construct the associated transition relation F :

$$F(FI_L, FI_R, z_1, z_2, \cdots, z_{|FO_R|}) = \prod_{1 \leq i \leq |FO_R|} (z_i \equiv FO_{R_i}(FI_L, FI_R)) \tag{9}$$

Let the number of linking nodes in F be k. We can prove that, without considering the don't care conditions stated in Section 2.3, the minimal number of communication lines required from side L to side R within the peephole is $log_2 k$. As stated in Section 3, once the above transition relation F is built and $log_2 k$ is obtained, we compare $log_2 k$ with the original number of cut edges in the peephole. If $log_2 k$ is smaller, we know we can implement the peephole in a different way where the number of cut edges will be reduced and the behavior of the peephole, from the viewpoint of the nodes outside the peephole, is intact. If $log_2 k$ is larger than the original number of cut edges, we discard this peephole. As to the resynthesis of the new peephole, we follow the

[6]We use the term *BDD* to refer to ROBDD (reduced, ordered, binary decision diagram) [2] throughout this paper.

similar encoding and decoding techniques in [3] [11]. [7]

4 Experimental Results

We have implemented our algorithm and experimented on MCNC benchmarks. For each circuit, we first run the script *script.boolean* provided in *sis* [13] for technology independent optimization. Then we map each circuit, using *map* command in sis, to a library consisting of a 2-input NAND gate, a 2-input NOR gate, and an inverter. To guarantee the graph domain information is exhausted, we run FM-method 500 times for each of these technology mapped circuits, each time with a random initial partition. To allow minor area unbalances, we set the parameter to FM-method to be maximally 1:2 for left-right ratio. Then we run our algorithm iteratively switching between logic domain and graph domain. The result is shown in Table 1. The first row of each circuit shows the original partition result of running FM-method for 500 times. In this way, we believe many of them have achieved the optimal solution in graph domain. If the partition has improved, we show the result starting from the second row of each circuit. All those circuits that have only one row mean that our algorithm failed to improve the partition generated by running FM-method for 500 times. As can be seen in Table 1, our algorithm was able to improve the near-optimal solution in graph domain. [8] Those that our algorithms failed to improve were mostly with small number of cut lines. In addition, during the approach to the final solution, we also generate a whole spectrum of different area-cut tradeoffs for many circuits.

To focus on improving cut lines only, we did not run any area optimization technique in our experiment for the final and intermediate partitioned blocks. As mentioned in Section 2.3, there exist a fairly large don't care conditions when we synthesize the peepholes. We strongly believe that, in practice, we can reduce the final areas of the partitioned blocks quite easily.

5 Conclusion

We have proposed an algorithm that exploits the logic domain information when partitioning is performed. Our preliminary experimental results indicate that there exists a fairly large improvement space on top of the traditional graph domain methods. Many problems in our method still remain unanswered, but the general direction of incorporating logic domain information into the partitioning problem is undoubtedly worth more research. We are currently developing more tightly-coupled algorithms between the graph domain and the logic domain.

6 Acknowledgement

This work was supported by the National Science Foundation under Grant MP917328 and partially by AT&T Bell Labs through the MICRO program.

References

[1] M. Beardslee, B. Lin, and A. Sangiovanni-Vincentelli, "Communication Based Logic Partitioning," *Proc. IEEE European Design Automation Conference, pp. 32-37,* Sep. 1992.

[2] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Tran. on Computers,* Vol C-35, no.8, August 1986.

[7] Because of space limitation, we do not provide the details.

[8] Running FM-method 500 times provides a very good, if not optimal, solution in the graph domain. We can then fairly investigate how much improvement space is left in the logic domain. Optimal solutions in graph domain may not need 500 times to be achieved in many cases. No direct comparison with the FM-method is intended.

model	#in	#out	#node	#cut	Larea	Rarea	FMcpu	OURcpu
C1355	41	32	621	31	497	991	1307.8	16.9
				30	522	1016		
C499	41	32	621	31	519	969	1340.2	29.2
				30	555	989		
				29	1030	538		
alu4	14	8	828	71	767	1527	1761.6	87.5
				66	1228	2452		
				61	1279	2531		
				58	2362	1181		
				57	2501	1296		
apex6	135	99	1010	35	1333	849	1880.4	26.0
				34	1400	965		
				32	1451	996		
				31	1543	1017		
b9	41	21	188	7	140	206	220.9	2.5
cm152a	11	1	36	4	38	31	27.5	0.9
cmb	16	4	80	6	106	58	80.5	1.9
				5	108	58		
comp	32	8	238	3	259	273	352.3	5.7
				2	245	287		
count	35	16	191	5	258	135	216.1	8.4
				4	275	176		
exmple2	85	66	486	12	616	316	713.5	17.5
				10	676	349		
f51m	8	8	151	11	236	148	204.1	5.6
				10	233	154		
k2	45	45	1131	116	981	1929	2391.5	80.3
				113	1297	2107		
				109	1423	2268		
ldd	9	19	115	9	80	159	128.8	12.4
				8	114	205		
pcler8	27	17	145	6	173	100	145.9	4.3
				4	154	191		
pm1	16	13	81	6	85	58	70.7	1.9
rot	135	107	968	48	701	1310	1864.0	30.7
				41	741	1319		
				40	748	1343		
term1	34	10	293	14	378	321	462.6	7.0
				13	459	384		
ttt2	24	21	269	8	372	258	395.9	7.0
				6	474	332		
unreg	36	16	171	4	154	168	173.6	5.6
vda	17	39	627	73	537	1054	1033.4	19.4
				65	1348	674		
				64	1346	680		
x3	135	99	1001	47	1383	778	1823.5	23.2
				45	796	1367		
				44	803	1382		
x4	94	71	573	24	397	720	816.4	15.1
				23	391	726		
				22	396	774		
				21	412	807		

Table 1: Testing Result on Benchmark

[3] S.C. Chang and M. Marek-Sadowska, "Technology Mapping via Transformations of Function Graphs", *Proc. IEEE ICCD*, 1992.

[4] J. Cong, L. Hagen, and A. Kahng, "Net Partitions Yield Better Module Partitions," *Proc. ACM/IEEE Design Automation Conference, pp. 47-52,* Jun. 1992.

[5] C.M. Fiduccia and R.M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions," *Proc. ACM/IEEE Design Automation Conference, pp. 171-181,* 1982.

[6] M. Garey and S. Johnson, "Computers and Intractability: A guide to the Theory of NP-completeness," 1979.

[7] L. Hagen and A. Kahng, "A New Approach to Effective Circuit Clustering," *Proc. ACM/IEEE International Conference on Computer-Aided Design, pp. 422-427,* Nov. 1992.

[8] J. Hwang and A. El Gamal, "Optimal Replication for Min-Cut Partitioning," *Proc. ACM/IEEE International Conference on Computer-Aided Design, pp. 432-435,* Nov. 1992.

[9] R.M. Karp, "Functional Decomposition and Switching Circuit Design," *J. of Soc. Ind. Appl. Math.* Vol. 11, June 1963.

[10] C. Kring and A.R. Newton, "A Cell-Replicating Approach to Mincut-Based Circuit Partitioning," *Proc. ACM/IEEE International Conference on Computer-Aided Design, pp. 2-5,* Nov. 1991.

[11] Y.T. Lai, M. Pedram, and S. Vrudhula, "BDD based Decomposition of Logic Functions with Application to FPGA Synthesis" *Proc. ACM/IEEE Design Automation Conference, pp. 642-647,* June. 1993.

[12] T.T. Hwang, R.M. Owens, and M.J. Irwin, "Efficiently Computing Communication Complexity for Multilevel Logic Synthesis," *IEEE Tran. on Computer-Aided Design,* Vol. 11, May 1992.

[13] E.Sentovich, K.Singh, L.Lavagno, C.Moon, R.Murgai, A.Saldanha, H.Savoj, P.Stephan, R.Brayton, and A.Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis" *Memorandum No. UCB/ERL M92/41, University of California, Berkeley.*

[14] H.Touati, H.Savoj, and B.Lin, "Implicit State Enumeration of Finite State Machines using BDD's" *Proc. ACM/IEEE International Conference on Computer-Aided Design,* Nov. 1990.

[15] Y.C. Wei and C.K. Cheng, "Ratio Cut Partitioning for Hierarchical Designs," *IEEE Tran. on Computer-Aided Design,* Vol. 10, July 1991.