3. Apply maximum matching on a graph whose nodes correspond to mergeble functions and edges show which pairs of nodes can be merged.

Table 1 illustrates the results obtained by our program on the benchmark examples. We also have listed there the coverage obtained by mis-pga, mis-pga-improved, Xmap and chortle-crf. On the average, our results are better than chortle-crf by 12%, Xmap by 22% and mis-pga by 25% and worst then mis-pga-improved by approximately 9%. Note that out of 20 examples, we did worse than mis-pga-improved only on 8 of them.

7 Conclusion.

In this paper we have proposed an algorithm for technology mapping for table look-up FPGAs. Our approach is based on Boolean manipulations of logic functions which are generally more powerful than the algebraic transformations. We have implemented our algorithm on top of misII [?]. Our program takes as an input a network with nodes of up to 9 inputs and transforms it into a network whose nodes support does not excede 5. Our results depend on the size of the network handed to us by misII. Obviously, trying various scripts in misII and picking the best one for each example would improve the quality of our results. We have decided to run all the examples using only two different scripts. We feel that in this way we are showing better the potential of our method. We need to mention also, that the quality of our results depends on variable ordering heuristics used to build ROBDDs which represent our functions. In our implementation we are using the algorithm described in [?].

8 Actnowledgement.

This work was supported by the National Science Foundation under Grant MIP 917328 and partially by AT&T Bell Labs through the MICRO program.

	BDD syn	mispga (new)	Chortle	Xmap	mispga (old)
z4ml	5	5	7	9	8
misex1	11	11	11	11	11
5xp1	12	18	28	31	31
9sym	6	7	59	73	56
9symml	6	7	59	73	56
sao2	46	28	27	37	37
rd73	7	6	16	21	19
f51m	12	127	27	37	129
count	32	31	31	31	31
apex7	56	60	60	65	64
apex6	204	182	198	231	243
bw	27	28	39	43	39
b9	35	39	41	43	39
rd84	12	10	35	36	40
C499	88	66	89	75	66
alu2	54	109	116	126	129
rot	243	181	188	212	200
frg1	38	41	40	40	41
misex2	29	28	28	28	35
C880	124	82	88	103	103
Total	1047	956	1172	1312	1286

Table 1:

References

[1] R. L. Ashenhurst, "The Decomposition of Switching Functions", The Annals of the Harvard Computation Lab., XXIX, Harvard University Press, Cambridge, MA, 1959, pp. 74-116.

[2] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A. R. Wang, "MIS: A Multiple-Level Logic Optimization System", IEEE Transactions on Comp. Aided Des. , Vol-CAD-6. , No. 6, November 1987, pp 1062-1081.

[3] R. E. Bryant, " Graph-Based Algorithms for Boolean Function Manipulation", IEEE Transactions on Computers, vol. C-35, No. 8, August 1986, pp. 677-691.

[4] R. J. Francis, J. Rose and K. Chung, "Chortle: A Technology Mapping Program for Lookup Table-Based FPGAs", 27th ACM/IEEE Design Automation Conference, 1990, pp. 613-619.

[5] R. J. Francis, J. Rose and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs", 28th ACM/IEEE Des. Automation Conf., 1991, pp. 227-233.

[6] M. Fujita and Y. Matsunaga, "Multi-level Logic Minimization Based on Minimal Support and its Application to the Minimization of Look-up Table Type FPGAs", Proc. Int. Conf. on Comp. Aided Des., 1991, pp. 560-563.

[7] K. Karplus, "Using If-then-else DAGs for Multi-Level Logic minimization", Advanced Research in VLSI, C. Seitz Editor, 1989, pp. 101-117.

[8] K. Karplus, "Xmap a Technology Mapper for Table-lookup Field-Programmable Gate Arrays", 28th ACM/IEEE Des. Automation Conf., 1991, pp. 240-243.

[9] S. Malik, A. Wang, R. K. Brayton and A. Sangiovanni-Vincentelli, "Logic Verification Using Binary Decision Diagrams", IEEE Int. Conf. on Comp. Aided Des.,1988, pp. 6-9. 52-57.

[10] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton and A. Sangiovanni-Vincentelli, "Logic Synthesis for Programmable Gate Arrays", Proc. 27th ACM/IEEE Des. Automation Conf., 1990, pp. 620-625.

[11] R. Murgai, N. Shenoy, R. K. Brayton and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures", Proc. Int. Conf. on Computer Aided Design, 1991, pp. 564-567.

[12] J. P. Roth and R. M. Karp, " Minimization over Boolean Graphs", IBM Journal, April 1962, pp. 227-238.

[13] N. S. Woo, " A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility", 28th ACM/IEEE Des. Automation Conf., 1991, pp. 248-251.

[14] Xilinx Programmable Gate Array Data Book.

5.2 The ROBDD Substitution and Reduction Process.

The substitution process brings in new variables to the original Boolean function. These new variables can be used to reduce further the ROBDD of the original function. The reductions are based on the following relationship. Let f_1, f_2, f_3, f_4 and β be Boolean functions and let a be a Boolean variable.

1.a If $f_1 = a * f_2 + a' * f_3$ and $\beta = a * f_2 + a' * f_4$ then $f_1 = a * (a * f_2 + a' * f_4) + a' * f_3 = a * \beta + a' * f_3$.

2.a If $f_1 = a * f_2 + a' * f_3$ and $\beta = a * f_2' + a' f_4$ then $f_1 = a * (a * f_2 + a * f_4)' + a' * f_3 = a * \beta' + a' * f_3$

3.a If $f_1 = a * f_2 + a' * f_3$ and $\beta = a * f_4 + a' * f_3$ then $f_1 = a' * (a * f_4 + a * f_3) + a * f_2 = a' * \beta + a * f_2$

4.a If $f_1 = a * f_2 + a' * f_3$ and $\beta = a * f_4 + a' * f_3'$ then $f_1 = a' * (a * f_4 + a' * f_3')' + a * f_2 = a' * \beta' + a * f_2$

A Boolean function of ROBDD rooted from node[i] and labeled a can be expressed as $func(node[i]) = a * f_1 + a' * f_2$ where $f_1 = func(node[i] \rightarrow T) and f_2 = func(node[i] \rightarrow E)$ For example, the relationship 1.a implies that if the Boolean function $a * f_2 + a' * f_4$ is replaced by a single variable β in the original ROBDD, then the ROBDD can be simplified if there is a node whose function $f_1 = a * f_2 + a' * f_3$. These conditions can be easily checked in graph terms.

Let node[i] be the node whose corresponding Boolean function will be substituted by a single variable function β. Node[j] is the node being checked for a possible reductions. For the case 1.a, the rules are as follows.

1.b If ID(node[i]) = ID(node[j]) and node[i]→ T = node[j] → T then node[j]→ T = new_node where new_node stands for a node in the modified ROBDD labeled β.

Similarly, for the other cases we have the following conditions.

2.b If ID(node[i]) = ID(node[j]) and node[i]→T = (node[j]→ T)' then node[j]→ T = (new_node)'

3.b If ID(node[i]) = ID(node[j]) and node[i]→E = node[j]→ E then node[j]→ E = new_node

4.b If ID(node[i]) = ID(node[j]) and node[i]→E = (node[j]→ E)' then node[j]→ E = (new_node)'.

6 Experimental Results.

We have implemented both heuristics : the ROBDD cutting and ROBDD substitution with reductions as described above. Each of them performs the following steps.

1. ROBDD cutting :

- **a.** Determine the cutting level. (In our experiments the labels were split into approximately equal cardinality sets).
- **b.** Do the cutting.

2. ROBDD substitution and reduction.

- **a.** Traverse the ROBDD and find such a node that the ROBDD rooted there has the largest number of nodes labeled with at most m labels.
- **b.** Substitute this node with a new variable.
- **c.** Perform the reduction, if possible.
- **d.** Go back to 2.a. as long as the subject ROBDD is labeled with more than m variables.

To demonstrate the efficiency of our approach we have applied these heuristics for PGA synthesis. The main steps of the PGA synthesis program are as follows.

1. In misII, read in the Boolean network. Decompose the network such that each internal node in the network will have less than nine inputs. This is the input to our heuristics.

2. Treat each internal node of the Boolean network as a Boolean function. For each of these Boolean functions do begin

2.a Construct the ROBDD for it.

2.b Estimate the number of PGAs for the ROBDD using the cutting heuristics.

2.c Calculate the exact PGA cover for the ROBDD using substitution with reduction approach.

2.d If size(cutting) $\leq$ size (substitution and reduction) then begin

2.d.a Cut the ROBDD.

2.d.b For each new $ROBDD^*$ obtained during cutting go to the step 2.b. end
else

2.d.c do ROBDD substitution and reduction.
end;

mized any further, i.e. we do not apply collapsing, partial collapsing, etc. The last step we invoke is matching of mergeble function pairs in order to compare our results to those published in the literature.

4 The Cutting Heuristic.

4.1 Boolean Function Decomposition.

Boolean function decomposition was studied in [?] and [?], to mention a few. A decomposition of a function f : $Z^n \rightarrow Z^m$ is a representation of the form z = f(x_L, x_R) = $G(w, x_R)$, where w = $\alpha(x_L)$, $x_L = (x_1, x_2, ..., x_k), x_R = (x_{k+1}, ...x_n)$.

The functions G and α for a given function f and an assumed input partitions x_L and x_R can be found using a ROBDD representation of f. Consider an ROBDD(f) such that all the labels included in x_L occur before all the labels included in x_R . Partition the ROBDD(f) such that all the nodes labeled x_L are on one side of the partition and the reminimg nodes labeled x_R are on the other side. Let c be the number of edges in ROBDD(f) cut by the partition, i.e. c is the number of communication links for that partition [?]. Let l be the least integer such that $c \leq 2^l$. We assign a binary number to each link. l digits are enough to encode each of the links. An ROBDD of the function α (α can be a multi output function) is determined as follows. We make as many copies of the top portion of the original ROBDD as there are digits in the encoding of the links. The i-th copy has values assigned to its leaf nodes equal to the i-th digit of the encoding.

An ROBDD for the function G is determined as follows. First, we construct a maximal size unreduced OBDD g labeled with variables $g_1, g_2, ...g_l$. Next, the x_R labeled part of the original ROBDD is connected to the leafs of g. The decision diagram obtained is an OBDD.

4.2 Encoding of Communication Links.

Partition of the inputs does not determine G and α functions uniquely. Different encodings of communication links of a partition yield different functions. There is more room in manipulating the α functions and those functions have greater impact on the overall result than G does. Our goal is to produce α functions with small ROBDDs, where small means having as few nodes as possible. Our ultimate goal is to break the function into a set of functions with variable support not greater than 5. However, since it is difficult to predict the number of different labels in ROBDD, we will apply actions which reduce the total number of nodes what indirectly reduce the total number of different labels.

Suppose that we have an OBDD which is obtained from partitioning some ROBDD and that values of its leaf nodes have not been decided yet. We select pairs of nodes with the same labels in OBDD such that functions f_1 and f_2 rooted at these nodes we wish to make equivalent. Each such chosen pair of functions induces a set of equality type constraints to be imposed on the leaf values. Ideally, we would like to fulfill all these constraints. It may not always be possible. We try to fulfill as many of them as possible.

To handle the problem we build a graph G^* in which nodes correspond to the leaf nodes of the OBDD we are trying to reduce. Each equality type constraint of a form $a_1 = a_2 = ...a_k$ induces a clique between vertices $a_1, a_2, ...a_k$, in G^*. In general, there might be multiple edges in G^*. We assign weights to the edges of G^* as follows. Let k equality constraints describe the equivalence of functions f^i and f^j. An edge of a clique of cardinality p is assigned a weight $\frac{1}{k*p}$ of the original edges.

One way to decide the encoding is to apply graph partitioning. First, we construct the min-cost balanced partition for G^*. The resulting graphs are G_0^* and G_1^*. All nodes included in G_0^* have the first digit of their binary encoding set to 0, the nodes in G_1^* have it set to 1. Next, we bipartition the previously obtained graph G_0^* into G_{00}^* and G_{01}^* and the graph G_1^*becomes split into G_{10}^* and G_{11}^*. The nodes of the graphs G_{00}^* and G_{10} have their second digit in the encoding set to 0 and the nodes of the graphs G_{11}^* and G_{01}^* have the second bit in encoding equal to 1. The process continues until the resulting graphs have a single node each.

5 The Substitution and Reduction Heuristics.

5.1 The Subgraph Substitution Process.

Consider an ROBDD(f) graph G with ordered node labels $x_1, x_2, ...x_n$. Let v be a node in G labeled x_k. ROBDD[v] is a subgraph of G rooted at v with nodes labeled $x_k, x_{k+1}, ..., x_n$. We can replace the ROBDD[v] in G by an ROBDD of a function $f_1(\beta)$ of the original function f; i.e. $f(x_1, ..., x_n)$ = F ($x_1, ...x_n, \beta(x_k, ..., x_n)$) , where $\beta = func(ROBDD[v])$.

We will call an internal node v of an ROBDD graph G a *substitutable* node if the transitive fanout nodes of v in G are labeled with m-1 or less distinct variables. The substitution process can be continued until the original ROBDD is represented as a family of ROBDDs, each with at most m labels. This results in a Boolean network with nodes of at most m fanins. The process terminates if $m \geq 3$.

Technology Mapping via Transformations of Function Graphs

Shih-Chieh Chang and Malgorzata Marek-Sadowska
Electrical and Computer Engineering Department,
University of California
Santa Barbara, CA 93106.

Abstract

Programmable Gate Arrays (PGAs) are important media for rapid system prototyping. The most popular architectures for PGAs are the table look up (TLU) and the multiplexor based (MB) architectures. In this paper we address the problem of technology mapping for the TLU style PGAs.

1 Introduction.

A Table Look Up (TLU) based Programmable Gate Arrays (PGA) chip is composed of an array of identical logic blocks and wires which can be programmed to realize particular interconnects. Each of the logic blocks can implement any function with $m \geq 2$ inputs. For a given architecture, m is a fixed number. For example, the Xilinx(3000) architecture [?] sets m to 5. Additionally, a logic block can realize a two output function if each of the functions has less than m inputs and the total number of different inputs for both functions is not larger than m. Such two single output functions which can be housed by a two output block are called *mergeble.* Here, we are addressing the problem of how to realize a given combinational circuit described by means of Boolean equations using the minimum number of blocks of the target TLU architecture.

2 Previous Work.

The work on minimizing block count technology mapping for PGAs can be classified as follows.

1. mis-pga [?] and *mis-pga improved* [?] : The original mis-pga has the two main phases. First, a possibly infeasible (i.e. having intermediate nodes with more than m fanins) network is transformed into a feasible one. In this step, infeasible nodes are recursively split by kernel extraction [?] or by classical Boolean function decomposition (Roth - Karp method [?]). Next, the feasible network is optimized by collapsing pairs of nodes (partition) or by collapsing clusters of nodes (covering) .

The improved mis-pga is a successor of mis-pga. It contains several additional decomposition schemes namely: cube packing, cofactoring, AND-OR decomposition and disjoint decomposition. They are invoked to handle different situations. Additionally, local optimization of covering process and partial collapse are performed. The resulting mapping is geared to increase the number of mergeble node functions.

2. Chortle [?] and *chortle-crf* [?] : Chortle divides the input Boolean network into a forest of trees and finds the optimum solution for each tree. No use of sharing among trees is made. Chortle-crf makes use of reconvergent fanouts and replicated multiple fanout nodes. It was first to use bin packing for gate level decomposition.

3. Xmap [?] : It builds an if-then-else DAG [?] representation of a Boolean network and uses covering method to obtain a feasible mapping.

3 Overview of Our Approach.

The main contribution of this paper is an efficient Boolean decomposition scheme. It works directly on a reduced ordered binary decision diagram (ROBDD) of a subject function. We use two techniques: the first one referred to as *cutting* is an efficient implementation of the Roth-Karp [?] decomposition. Here, we emphasize on the structure of the resulting functions. We achieve this by a proper encoding of communication links between the two parts of ROBDD. The heuristics involved calls for creating the functions with small size ROBDDs. The second technique used is referred to as a *substitution.* The idea is to replace subgraphs of ROBDD by new variables. The substitution process is accompanied by certain reductions of the resulting ROBDD graph which further decrease its size.

In order to demonstrate the usefulness of our Boolean decomposition heuristics, we have implemented it on top of the misII [?] program. We start with a Boolean network optimized by misII with internal nodes having at most 9 inputs. We apply our decomposition algorithm to such a network and find a network with feasible nodes. In our experiments the number of inputs to a feasible node is 5, the same as in Xlinx architectures. The feasible network is not opti-