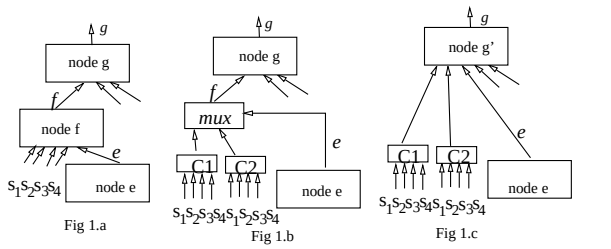

Fig 1.a Fig 1.b Fig 1.c

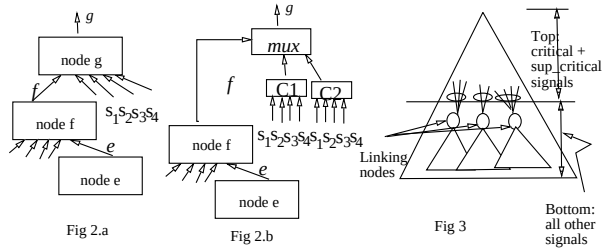

Fig 2.a Fig 2.b Fig 3

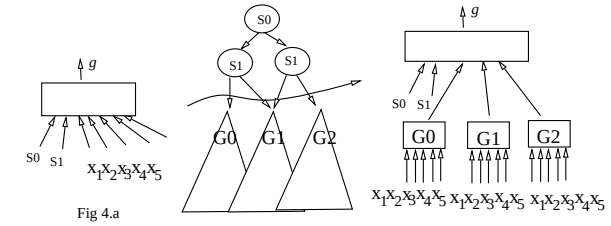

Fig 4.a Fig 4.b Fig 4.c

length of critical paths. We think that good results obtained by us can be attributed to the fact that the partitioning approach employed by us is tightly coupled with the size of the target TLU blocks. Different values of k result in different partition.

Acknowledgment

This work was supported in part by the National Science Foundation under Grant MIP 917328 and in part by AT&T Bell Laboratories through the California MICRO Program

References

[1] 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.

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

[3] R. K. Brayton, C. McMullen, G.D. Hachtel and A.Sangiovanni-Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, 1984.

[4] S.C. Chang M. Marek-Sadowska, "Technology Mapping via Transformations of Function Graphs", Proc. ICCD 1992 pp. 159-162

[5] J. Cong and Y.Ding, "An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs", Proc. Int. Conf. on Computer Aided Design, 1991, pp. 48-53.

[6] 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.

	our result depth(size)	mispga timing depth(size)	Chortle delay depth(size)
1 5xp1	2(19)	2(21)	4(29)
2 C880	8(147)	9(259)	7(383)
3 vg2	4(33)	4(39)	3(54)
4 misex1	2(18)	2(17)	3(25)
5 sao2	4(59)	5(45)	4(58)
6 rd73	2(8)	2(8)	4(52)
7 z4ml	2(6)	2(10)	3(20)
8 b9	3(46)	3(47)	3(62)
9 misex2	2(41)	3(37)	2(52)
10 bw	1(28)	1(28)	1(28)
11 9sym	3(6)	3(7)	4(76)
12 9symml	3(6)	3(7)	5(130)
13 apex7	4(93)	4(95)	4(131)
14 C499	5(68)	8(199)	6(436)
15 apex6	5(236)	5(274)	5(361)
16 duke2	5(218)	6(164)	4(248)
17 count	4(80)	4(81)	3(102)
18 f51m	3(81)	4(23)	5(65)
Total	62(1193)	70(1361)	70(2312)

Table 1:

[7] 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.

[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] 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.

[10] 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.

[11] R. Murgai, N. Shenoy, R. K. Brayton and A. Sangiovanni-Vincentelli, "Performance Directed Synthesis for Table Look Up Programmable gate Arrays", Proc. Int. Conf. on Computer Aided Design, 1991, pp. 572-575.

[12] J. P. Roth and R. M. Karp, " Minimization over Boolean Graphs", IBM Journal, April 1962, pp. 227-238. Conf. on Comp. Aided Des., 1991, pp. 560-563.

[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 User's Guide, 1988 Xilinx, Inc.

all inputs of f(x), Shannon expansion on f(X) with respect to x_1 can effectively move x_1 closer to the output and delay the other signals. Based on this observation, we apply Shannon expansion to compress the critical path.

In our implementation, we assumed the number of inputs in the lookup table to be 5.

Definition 6 Input depth of a signal(edge in the network graph), denoted as **LI**, is equal to the number of nodes in the longest path from the original primary inputs to the node whose fanout this signal is.

Definition 7 Output depth of a signal, denoted as **LO**, is equal to the number of nodes in the longest path from the node whose fanin this signal is to the POs.

Definition 8 Max_level of a network is the max LI over all the nodes in the network.

Definition 9 A **sup_critical** signal is a signal whose $LI + LO \leq Max_level - 1$

Let f(A) be the function realized by the node A.

Definition 10 The **Shannon-Decomposition** of a node A with respect to an input signal s is defined as follows. Node A is decomposed into 3 nodes (m, C1, C2). C1 realizes Boolean function equal to $f(A)_s$, the cofactor of f(A) with respect to s. C2 realizes Boolean function $f(A)_{\bar{s}}$, the cofactor of f(A) with respect to $\bar{s}$. M is a multiplexer with a selecting input s and two other inputs are the outputs of nodes C1 and C2.

The set of sup_critical paths represents all the paths that we can not afford to increment. In the following discussion, let e, f, and g be three consecutive nodes in a critical path and mux be a 2-to-1 multiplexer. We have three very effective rules:

Rule 1: (refer to Fig. 1)
If node g has no more than 3 (k-2) fanins (Fig. 1a), then

1. Shannon Decompose the node f with respect to the node e, (Fig. 1b)
2. collapse the node g with the multiplexer mux in Fig. 1b (Fig. 1c).

Rule 2: (refer to Fig. 2) If the node g has more than 3 fanins and all of them except the node f are not in the sup_critical path (Fig. 2b), then

1. Shannon Decompose the node g with respect to the node f, (Fig 2b),
2. apply Rule 1 (multiplexer *mux* has three fanins now).

The above two rules apply the Shannon expansion on every three nodes on the critical paths. When either of the above rules applies, the length of critical path is reduced by one. To apply the Rule 3 below, we collapse two nodes in a critical path into a node g; then, we build a BDD for the node g. The ordering of the variables in the BDD is such that all the variables corresponding to the nodes in the *sup_critical* paths appear before the other variables. We define a BDD node to be a **linking node** if this BDD node is labeled with the variable not present in the sup_critical paths and this is the first such a variable in the ordering. (see Fig 3)

Rule 3 For every two consecutive nodes in a critical path, we collapse them into a node g (refer to Fig. 4a) and build a BDD as described above for the node g (Fig. 4b). If

1. the number of fanin nodes that are not in the sup_critical paths is less than or equal to 5, AND
2. the number of fanin nodes that are in the sup_critical paths plus the number of linking nodes in the BDD is less than or equal to 5;

then:

1. for each linking node in the BDD, construct TLU that implements the function represented by the sub-BDD rooted at the linking node,
2. construct a TLU that realizes the same function as the node g. This TLU takes the TLUs constructed in (1) and the signals from the sup_critical path as fanins. (Fig. 4c).

4 Experimental Results

We have implemented the algorithms discussed in the previous sections. To compare our approach to these published previously, we have run our program on a number of benchmark examples. In our experiments k =5. The input network to our program is supplied by the standard misII script. Table I shows the results. In Columns 2, 3 and 4 are listed the experimental results of our program, misII timing [?] and Chortle delay [?] respectively. Each entry in the table has the format of i(j), where i refers to the length of critical path and j refers to the number of TLUs after technology mapping. In all examples, we obtained the same or less levels than the critical paths in misII; furthermore, the total number of TLUs is 14% less than that obtained from misII. In comparision with Chortle delay, our results are on average better in depth and only use half of the total number of TLUs used by Chortle. The cpu run time of our program is also on the average about 10 times faster than misII.

5 Conclusion

We have proposed a technology mapping algorithm for table lookup FPGA aiming at minimizing the

increasing the circuit depth unnecessarily.

A **slice** of a Boolean network is a set of nodes whose removal separate the primary inputs and the primary outputs. In our algorithm, we iteratively choose a "good" slice that is "close" to the primary inputs, then a new network is built by truncating all the nodes in between the primary inputs and the slice. In the next iteration, the fanouts of the slice in the previous iteration serve as the "primary inputs" for the newly built network.

We choose a slice in such a way that each node in the slice can be implemented by a TLU. In other words, we finish the technology mapping of the truncated nodes immediately; therefore, for each of the "primary inputs" of the next iteration, we know the exact depth information counting from the true primary inputs. Note that we also have the depth information counting towards the primary outputs. In this way, at each iteration when we choose a slice, we have the freedom of "balancing" all potentially critical paths based on the depths information.

Before explaining the way to choose a "good" slice, we need some definitions. Let k be the number of inputs in a lookup table, $|s|$ denote the size of a set s, and i_cov(v) denote the set of the primary inputs in the transitive fanins of a node v.

Definition 1 A node v is a **type-I** node if $|i_cov(v)| \leq$ k.

Definition 2 The **border set** of a network are those type-I nodes which have in their fanout at least one non-type-I node.

Definition 3 Dependent set of a node v, denoted by dep(v), are those nodes of the border set which are in the transitive fanin of v.

Definition 4 A node v is **type-III** if $|i_cov(v)| >$k and $|dep(v)| >$ k.

Definition 5 A node v is **type-II** if it is neither type-I nor type-III node.

Type-I nodes are those nodes that can be directly implemented by a lookup table. Type-II and type-III nodes, on the other hand, can not be implemented without implementing first some type-I nodes; i.e., the legitimacy of implementation of a type-II or type-III nodes depends on the earlier implementation of some type-I nodes. The difference between type-II and type-III nodes is as follows. When we select some appropriate type-I nodes and mark them as primary inputs, a type-II node has a possibility of having primary input cover not exceeding k and having among them at least one of the primary inputs.

Intuitively, type-III nodes are deep enough (measuring from the primary inputs) so that we can defer choosing them into a slice. Only type-I and type-II nodes are considered at every iteration. By distinguishing type-III from type-I and type-II nodes, we can reduce greatly the search space. In the first step of choosing a slice, we classify the nodes in the input network into three types. Next we focus on the subnetwork consisting of type-I and type-II nodes only and make decisions on the selections of the nodes within this subnetwork. The nodes in between slice and primary inputs will be truncated from the current network and the nodes in a slice will serve as primary inputs in the next iteration. We make an assumption that a good slice should be able to truncate as many nodes as possible. Let the weight of a node be equal to the number of nodes that can be truncated if the node is selected into the slice. We calculate the weights of the type-I and type-II nodes. Then, nodes are sequentially selected into the slice based on their weights and such that the selection of a node does not increase the estimated critical length. The entire algorithm works as follows.

1. Read in a network. Mark PIs as original inputs; set slice to an empty set.

2. Identify the 3 types of nodes and form a set P of type-I and type-II nodes. Calculate the weights of nodes in set P.

3. Choose the highest weight value type-I node. Find the one whose union with the slice blocks off the largest number of nodes and would not increase the estimated critical depth. Mark the node as primary inputs and add them into the slice.

4. Update the node type and remove those nodes from P which are blocked by the members of the slice. Remove from P the members of the slice if P is not empty, go to (3)

5. If there are primary outputs not included in any slice, truncate the nodes blocked by the present slice. Mark the nodes in the slice as original primary inputs; set slice to empty and go to 2; otherwise exit.

3 Level compression for TLU network

During the technology mapping process described in the previous section, we have determined a network with minimized number of TLU blocks. But it is possible that the number of blocks along the critical paths is large. In the next step of our algorithm, we try to compress the critical paths using a rule-based approach.

The first two rules are based on Shannon expansion of Boolean function. $f(x) = x_1 * f(1, x_2, ..., x_n) + \bar{x_1} * f(0, x_2, ..., x_n)$. If x_1 is the last arriving signal among

Technology Mapping and Circuit Depth Optimization for Field Programmable Gate Arrays

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. In this paper, we propose a two step technology mapping algorithm for lookup table type FPGAs. In the first step, the technology mapper attempts to minimize the total number of TLUs used and at the same time to keep the length of the critical path short. Then, it is followed by a rule based postprocessor which maximally decreases the depth of a circuit.

1 Introduction

Field Programmable Gate Arrays(FPGA) are playing a very important role among the ASICs. Due to large scale of integration, user programmability and full testability, they allow for low cost rapid system prototyping. To achieve quick turn around time and a good quality of design, efficient computerized tools are essential.

Typically, automatic design of Table Look-up(TLU) FPGA follows a sequence of three steps. A multi level Boolean network realizing the desired function is created and optimized without taking into account the implementation technology. Then, technology mapping is performed on the resulting network yielding a realization composed entirely of k input look up tables. Finally the mapped circuit is placed and routed on the chip.

This paper deals with the technology mapping step of the TLU FPGAs automatic design process. Technology mapping algorithms proposed earlier for TLU architecture used as an objective function the total number of TLUs needed by the realization [?], [?], [?], [?], [?]. More recently, work on timing driven technology mapping for TLU FPGAs was reported [?] [?], [?]. Even though the total delay of a chip is layout dependent because of delays introduced by interconnects, it is important to minimize the number of TLUs along the critical paths. Such an objective is a reasonable first order approximation of the timing driven technology mapping for FPGAs. It is based on an assumption that all TLUs have similar intrinsic delays and that routing delay may be accounted for by adding to a block delay some fixed constant [?].

The problem addressed in this paper can be formulated as follows:

Circuit Depth Minimizing Technology Mapping Problem: Given a Boolean network composed of nodes each of which has no more than k inputs, and optimized by a technology independent process, find another Boolean network which describes the same Boolean function, can be realized by k-input TLUs and such that the number of blocks along any critical path is as small as possible.

We propose a two-step algorithm to solve the problem. In the first step, a technology mapper produces a feasible solution optimizing the total number of TLUs used. At the same time, we attempt not to increase the circuit depth unnecessarily. It is then followed by a second step whose objective is to maximally decrease the depth of the circuit adding as few TLUs as possible.

The paper is organized as follows. In section 2, we describe the first step of our technology mapper, in section 3, we explain the level compression strategy and finally in section 4, we present results followed by conclusions.

2 Slicing the network

We will make use of the following terminology and definitions. A *Boolean network* is a Directed Acyclic Graph(DAG) where each node is associated with a Boolean function f_i and a Boolean variable x_i. **Input Cover** of a node are those primary inputs of a network which are in transitive fanin of that node.

In the first step, our technology mapper takes as input an optimized network G = (V, E) which is composed of nodes with less than k fanins. The goal is to find a new network G' = (V', E') such that V' $\subseteq V$and also each node in V' has less than k fanins. The algorithm attempts to minimize the size of |V'| without