Beyond the Combinatorial Limit in Depth Minimization for LUT-Based FPGA Designs

Jason Cong and Yuzheng Ding
Department of Computer Science
University of California, Los Angeles, CA 90024

Abstract

In this paper, we present an integrated approach to synthesis and mapping to go beyond the combinatorial limit set up by the depth-optimal FlowMap algorithm. The new algorithm, named FlowSYN, uses the global combinatorial optimization techniques to guide the Boolean synthesis process during depth minimization. The combinatorial optimization is achieved by computing a series of minimum cuts of fixed heights in a network based on fast network flow computation, and the Boolean optimization is achieved by efficient OBDD-based implementation of functional decomposition. The experimental results show that FlowSYN improves FlowMap in terms of both the depth and the number of LUTs in the mapping solutions. Moreover, FlowSYN also outperforms the existing FPGA synthesis algorithms for depth minimization.

1. Introduction

The field programmable gate array (FPGA) has become a very popular technology in VLSI ASIC design and system prototyping. An FPGA chip consists of programmable logic blocks, programmable interconnections, and programmable I/O pads. The lookup table (LUT) based FPGA architecture is produced by several FPGA manufacturers (e.g. Xilinx [12]), in which the basic programmable logic block is a K-input lookup table (K-LUT) that can implement any Boolean function of up to K variables. The synthesis and technology mapping problem for LUT-based FPGA designs is to generate a K-LUT network for a given set of Boolean functions.

Previous synthesis and mapping algorithms for LUT-based FPGA design have been focused on the minimization of the delay of the solution network (e.g. [2, 5, 8, 10]), the minimization of the number of LUTs used in the solution (e.g. [4, 7]), or the routability of the solution (e.g. [11]). Among these algorithms, many concentrated only on the mapping step. These algorithms cover a given Boolean network with LUTs using various combinatorial optimization techniques (such as bin-packing [4, 5] or flow computation [2, 3], etc.). Others tried to combine logic synthesis and technology mapping together by allowing Boolean operations (such as cofactoring and cube extraction [7, 8], and Shannon expansion [10], etc).

The FlowMap algorithm [2] results in a polynomial time depth-optimal technology mapper. It outperforms other existing LUT covering based mapping algorithms, and the optimality result sets up a limit of depth minimization using pure combinatorial optimization techniques. In fact, since FlowMap performs global structural optimization, in many cases it also outperforms existing synthesis algorithms for depth minimization, especially for large designs. It is of both theoretical and practical interest to see if we can use more powerful Boolean optimization techniques to go beyond the combinatorial limit of depth minimization set up by the FlowMap algorithm.

In this paper, we present an integrated approach to synthesis and mapping for depth minimization in LUT-based FPGA designs. It uses the global combinatorial optimization techniques to guide the Boolean synthesis process during depth minimization. Combinatorial optimization is achieved by computing a series of min-cuts of fixed heights in a network based on fast network flow computation, and Boolean optimization is achieved by efficient OBDD-based implementation of global functional decomposition. The experimental results have shown that our algorithm, named FlowSYN, improves FlowMap in terms of both the depth and the number of LUTs in the mapping solutions. It also outperforms the existing FPGA synthesis algorithms for depth minimization.

2. Problem Formulation

We assume that the input to the FPGA synthesis and mapping system is a combinational circuit specified by a general Boolean network (or equivalently, a set of Boolean equations), which can be represented by a directed acyclic graph (DAG). Each node in the DAG represents a logic gate, and a directed edge (i, j) exists if the output of gate i is an input of gate j. A primary input (PI) node has no incoming edge and a primary output (PO) node has no outgoing edge. We use $input(v)$ to denote the set of nodes which are the fanins of node v. Given a subgraph H, $input(H)$ denotes the set of *distinct* nodes outside H which supply inputs to the gates in H. The *level* (or *depth*) of a node v is the length of the longest path from any PI node to v. The *depth* of a network is the largest node level in the network. A Boolean network is *K-bounded* if $|input(v)| \le K$ for each node v.

For a node v in the network, a *cone of* v, denoted C_v, is a subgraph of logic gates (excluding PIs) consisting of v and its predecessors such that any path connecting a node in C_v and v lies entirely in C_v. We call v the *root* of C_v. A cone C_v is *K-feasible* if $|input(C_v)| \le K$.

If a K-LUT LUT_v implements a K-feasible cone C_v, we say that LUT_v *implements* node v and that v is the *root* of LUT_v. In order to cover C_v with a K-LUT, we may need to duplicate the non-root nodes in C_v that have fanouts

This research is partially supported by a grant from Xilinx Inc. under the State of California MICRO program, the NSF Young Investigator Award, and a grant from UCLA Academic Senate.

outside of C_v since each K-LUT has a single output. The *technology mapping problem* for K-LUT based FPGA designs is to cover the network with K-LUTs (possibly with node duplications). Note that according to this definition, the mapping problem is a combinatorial optimization problem. In an *integrated synthesis and mapping approach*, it is allowed to transform a given network into other functionally equivalent networks using Boolean operations during the construction of the final K-LUT network. In this paper, we shall take such an approach. Our main objective is to minimize the delay of the resulting K-LUT network. We use the simple *unit delay model* where the network delay is proportional to the network depth. Our secondary objective is to reduce the number of K-LUTs used in the technology mapping solution.

3. Review of the FlowMap Algorithm

We first briefly review the FlowMap algorithm which is the basis of the new FlowSYN algorithm.

Given a network $N = (V(N), E(N))$ with a source s and a sink t, a *cut* $(X, \bar{X})$ is a partition of the nodes in $V(N)$ such that $s \in X$ and $t \in \bar{X}$. The *node cut-size* of $(X, \bar{X})$, denoted as $n(X, \bar{X})$, is the number of nodes in X that are adjacent to some node in $\bar{X}$, i.e.

$$n(X, \bar{X}) = |\{x : (x, y) \in E(N),\ x \in X \text{ and } y \in \bar{X}\}|$$

A cut $(X, \bar{X})$ is *K-feasible* if its node cut-size is no more than K, i.e., $n(X, \bar{X}) \le K$. In the remainder of this paper, any reference to the cut-size means node cut-size, and a min-cut refers to a cut of the minimum node cut-size. Moreover, assume that there is a given label $l(v)$ associated with each node v. Then, the *height* of a cut $(X, \bar{X})$, denoted $h(X, \bar{X})$, is the maximum label in X, i.e.

$$h(X, \bar{X}) = \max\{l(x) : x \in X\}$$

The FlowMap algorithm runs in two phases. In the first phase, it computes a label for each node which reflects the level of the K-LUT implementing that node in an optimal mapping solution. In the second phase, it generates all the necessary K-LUTs starting from POs based on the node labels computed in the first phase.

Given a K-bounded Boolean network N, let N_v denote the cone of node v consisting of v and all its predecessors. The *label* of v, denoted as $l(v)$, is defined to be the depth of the optimal K-LUT mapping solution of N_v. Therefore, the level of the K-LUT containing v in the optimal mapping solution of N is at least $l(v)$, and the maximum label of all the POs of N is the depth of the optimal mapping solution of N. The first phase of FlowMap uses the dynamic programming method to compute all node labels in N according to the topological order starting from the PIs (which have label zero). Suppose t is the current node being processed. Then,

$$l(t) = \min_{(X, \bar{X}) \text{ is } K\text{-feasible}} h(X, \bar{X}) + 1.$$

That is, computing label $l(t)$ is equivalent to finding a minimum height K-feasible cut in N_t (see [2] for details).

One important contribution of FlowMap is that it has developed an $O(Km)$ time algorithm for computing a minimum height K-feasible cut in network N_t (where m is the number of edges in the network), which leads to efficient node label computation at each node t. In fact, the following results were shown in [2]:

Lemma 1 If the maximum label of the fanins of t is p, then $l(t) = p$ or $l(t) = p + 1$. □

Lemma 2 Let N_t^0 denote the network after collapsing all the nodes of label p into the sink t in N_t. Then, N_t has a K-feasible cut of height $p - 1$ if and only if N_t^0 has a K-feasible cut. □

Theorem 1 $l(t) = p$ if and only if the min-cut in N_t^0 has cut-size no more than K. □

Based on these results, FlowMap computes the label of each node t based on the following simple test: If the min-cut size in N_t^0 is more than K, $l(t) = p$; otherwise, $l(t) = p + 1$. In the former case, the min-cut in N_t^0 induces a minimum height K-feasible cut in N_t. In the latter case, the cut that separates t and its predecessors is K-feasible since the network is K-bounded. After all the node labels have been computed, FlowMap generates necessary K-LUTs using the minimum height K-feasible cuts computed in the labeling phase.

4. The FlowSYN Algorithm

The FlowSYN algorithm will inherit the combinatorial optimization techniques from FlowMap. In addition, it will also use the global structural information obtained during combinatorial optimization to selectively resynthesize parts of the given network using Boolean logic operations for further depth and area optimization.

The FlowSYN algorithm follows the two-phase process used in FlowMap: In the first phase, FlowSYN labels all the nodes according to the topological order starting from PIs. Each node label $l(v)$ is an upper bound of the depth of the optimal K-LUT implementation of N_v. Note that since Boolean synthesis is combined into the mapping process, computing the *optimal depth* among all possible K-LUT implementations of N_v becomes an NP-hard problem (based on a simple reduction from the Boolean SAT problem). Therefore, FlowSYN computes an *upper bound* of the depth of the optimal K-LUT implementation of N_v and use this upper bound as the node label of v. In the second phase, FlowSYN generates all the necessary K-LUTs in the final solution in the same way as that of FlowMap. Our discussion concentrates on the label computation in the first phase of FlowSYN.

4.1. When to Resynthesize

The key of the FlowSYN algorithm is to use Boolean optimization operations to resynthesize part of the network when combinatorial optimization techniques fail to produce a good result. In particular, assume that t is the current node being labeled. If the min-cut in N_t^0 has cut-size more than K, FlowMap will assign $l(t) = p + 1$ (where p is the maximum label of the fanins of node t). In this case, FlowSYN tries to resynthesize N_t so that $l(t)$ remains to be p. In fact, we can show the following:

Lemma 3 Suppose $(X, \bar{X})$ is a cut of height $p - h$ in N_t. If $\bar{X}$ has a K-LUT implementation of depth no more than h, then $l(t) \leq p$. □

Based on this result, when FlowMap fails to label node t by p, FlowSYN will resynthesize N_t as follows:

```
h=2;
Repeat
  Find a min-cut $(X_h, \bar{X}_h)$ of height $p - h$;
  If $\bar{X}_h$ has a K-LUT implementation
        of height $h$ using Boolean resynthesis
  Then $l(t) = p$ and Exit-Loop
  Else $h = h + 1$
Until $h > p$;
```

There are two reasons for choosing a min-cut $(X_h, \bar{X}_h)$ instead of an arbitrary cut of height $p - h$ in N_t for resynthesis: (i) A min-cut yields the minimum number of inputs to $\bar{X}_h$. In general fewer inputs to $\bar{X}_h$ gives a better chance to successfully resynthesize $\bar{X}_h$ to get a K-LUT implementation of height $\leq h$. (ii) A min-cut leads to the most efficient resynthesis procedure, since the complexity of our Boolean synthesis operations (to be described later) depends on the number of inputs to $\bar{X}_h$.

To compute a min-cut $(X_h, \bar{X}_h)$ of fixed height $p - h$, let H_h denote the set of nodes with labels larger than $p - h$. We collapse all the nodes in H_h into the sink t in N_t and name the resulting network N_t^h. It is not difficult to show the following lemma:

Lemma 4 $(X', \bar{X}')$ is a min-cut in N_t^h if and only if $(X', \overline{X' \cup H_h})$ is a min-cut of height $\leq p - h$ in N_t. □

Since a min-cut in $(X', \bar{X}')$ can be computed efficiently using network flow computation according to the max-flow min-cut theorem, a min-cut $(X_h, \bar{X}_h)$ of fixed height $p - h$ in N_t can be obtained efficiently.

4.2. How to Resynthesize

Note that $\bar{X}_h$ is a single-output subnetwork rooted at t which implements a single-output Boolean function $f(x_1, x_2, ..., x_r)$ where $\{x_1, x_2, ..., x_r\} = input(\bar{X}_h)$. We shall apply functional decomposition techniques to compute a K-LUT implementation of f with small depth.

Clearly, we have $r > K$, otherwise FlowMap would have found a K-feasible cut of height $p - 1$. Without loss of generality, we assume that $x_1, ..., x_r$ are ordered according to the increasing order of their node labels, i.e. $l(x_1) \leq l(x_2) \leq \cdots \leq l(x_r)$. We apply the functional decomposition [9] to the inputs $x_1, x_2, ..., x_K$ to see if we can decompose f into the following form

$$f(x_1, x_2, ..., x_r) = f'(y_1, ..., y_J, x_{K+1}, ..., x_r) \qquad (1)$$

where $y_1, ..., y_J$ are functions of $x_1, x_2, ..., x_K$ and $J < K$. There are two cases:

(i) If such a decomposition succeeds, we can implement each of $y_1, ..., y_J$ using a K-LUT and each of $y_1, ..., y_J$ will have node label $l(x_K) + 1$. Then, we reorder $y_1, y_2, ..., y_J, x_{K+1}, ..., x_r$ to f' according to the increasing order of there labels, and recursively decompose f' into K-LUT implementation.

(ii) If the decomposition in (1) fails, we try to apply the functional decomposition to the inputs $x_1, ..., x_{K-1}, x_{K+1}$ and so on. In the worst case, we may try $\binom{r}{K}$ combinations of input variables. In practice, however, the decomposition often succeeds for the first a few combinations.

If the decomposition in (1) succeeds, f' will have at most $r - 1$ inputs. The recursive decomposition stops when the remaining function has no more than K inputs. Since each step reduces the number of inputs by at least one, the depth of recursion is bounded by $r - K$. Moreover, each successful decomposition uses no more than $(K - 1)$ K-LUTs. Therefore, the total number of K-LUTs used for implementing $\bar{X}_h$ is bounded by $(K - 1)\cdot(r - K)$ and the depth of the K-LUT implementation is no more than $r - K$. Based on these discussions, we have

Theorem 2 If the recursive decomposition of $\bar{X}_h$ succeeds, $\bar{X}_h$ can be implemented using a network of no more than $(K - 1)\cdot(r - K)$ K-LUTs with depth no more than $r - K$. Moreover, if the root K-LUT in this implementation gets label d_h, then the label of node t is

$$l(t) = \min(p + 1, d_h). \qquad □$$

We choose to apply the functional decomposition first to the inputs $x_1, x_2, ..., x_K$ since these nodes have the smallest node labels. As a result, we are more likely to get a resynthesis solution of $\bar{X}_h$ with small root node label.

If FlowSYN successfully generates a K-LUT implementation of some $\bar{X}_h$ with root node labeling $d_h < p + 1$ by the recursive decomposition, we can label node t with d_h, which goes beyond the combinatorial limit $p + 1$ set up by the FlowMap algorithm. If the functional decomposition fails for all possible input combinations during a recursive decomposition step, or the label has exceeded $p + 1$, FlowSYN gives up generating a K-LUT implementation of $\bar{X}_h$ based on functional decomposition and move on to process the next min-cut of the fixed height and try to find a K-LUT implementation of $\bar{X}_{h+1}$ using functional decomposition. If the resynthesis fails to generate a K-LUT implementation of $\bar{X}_h$ with height $\leq h$ for any $2 \leq h \leq p$, we will give t the label $p + 1$ which represents the combinatorial solution.

It is clear that the efficiency of the FlowSYN algorithm depends on the computational complexity of the functional decomposition formulated in (1). We use an efficient implementation based on the OBDD representation. A similar method was used in [6].

4.3. OBDD Based Functional Decomposition

The functional decomposition problem formulated in (1) was studied by Roth and Karp in [9]. Their solution is based on *clique covering* of the *compatibility graph*, which requires a *cube* representation of the function.

In practice, the input to FPGA synthesis and mapping algorithm is usually a multi-level network optimized by technology independent synthesis tools. The original Roth-Karp algorithm needs to convert a multi-level network into a two-level one to get the cube representation. However, such a conversion will undo the minimization effort in the synthesis of the input network, and often result in prohibitively large cube representation at the root node, which requires large amount of memory, and long computation time in finding a disjoint clique covering on the compatibility graph. Therefore, we seek more efficient functional decomposition method that takes the advantage of the given multi-level network representation.

An *ordered binary decision diagram* (OBDD) [1] is a BDD where a total ordering among the variables is imposed. Given an OBDD, it can be reduced to a canonical form by eliminating duplicated vertices and redundant tests. We will denote a canonical form of the OBDD of function f as $OBDD_f$.

Given an OBDD of n variables, a *k-partition* is a partition $(D, \overline{D})$ of the OBDD such that all the nonterminal vertices corresponding to the first k variables in the variable ordering are in D, while all the other vertices are in $\overline{D}$. The *size* of the partition $(D, \overline{D})$ is defined to be the number of vertices in $\overline{D}$ that have incoming arcs from the vertices in D. The following result relates the OBDD representation to functional decomposition.

Theorem 3 Let $f(x_1, x_2, ..., x_n)$ be a Boolean function of n variables. Then, f can be decomposed into the form $f(x_1, x_2, ..., x_n) = f'(y_1(x_1,...,x_k),..., y_m(x_1,...,x_k), x_{k+1}, ..., x_n)$, if and only if the size of the k-partition of $OBDD_f$ with the variable ordering $x_1 < x_2 < \cdots < x_n$ is no more than 2^m. □

Intuitively this theorem says that the decomposition is possible if and only if we can encode the first k variables into m variables in the function. The decomposition of f is easily obtained from $OBDD_f$. Specifically, let $(D, \overline{D})$ be the k-partition of size no more than 2^m, and the boundary nodes of the k-partition in $\overline{D}$ are $v_0, ..., v_s$, where $s < 2^m$. To obtain $OBDD_{f'}$, we first create an unreduced OBDD of m variables. Clearly, there is a terminal of the OBDD corresponding to each assignment of the m variables. Let terminal t_i corresponds to the assignment that gives the jth variable the jth bit value of i. Then, $OBDD_{f'}$ is constructed by replacing t_i with v_i for $0 \le i \le s$ in the new OBDD, and then performing canonical conversion. To obtain $OBDD_{y_j}$, we replace in $OBDD_f$ each node v_i with the terminal of the value equal to the jth bit of i, and remove the rest of the nodes in $\overline{D}$.

The efficiency of this algorithm depends on the efficiency of the OBDD construction. Although in the worst case the size of OBDD can be exponential to the number of variables, in practice the size of OBDD is usually much smaller. In our implementation, the OBDD is constructed using the Boolean operation based construction method [1] which allows immediate size reduction during the construction procedure, and supports effective sharing of sub-OBDDs.

4.4. Area Minimization

The main objective of the FlowSYN algorithm is to minimize the depth of the mapping solution. Under this restriction, it also tries to minimize the number of K-LUTs that may be used in the mapping solution. We adopt several measurements to achieve this objective.

When the combinatorial method fails to give a node t the label p, the resynthesis is invoked, and if it gives t a smaller label than $p+1$, we accept this solution. If the resulting label from the resynthesis is also $p+1$, we compare the two solutions, and accept the one that results in an implementation of the fanin cone of t using fewer K-LUTs.

Since the FlowSYN algorithm tries to minimize the depth of every node, the depths of some nodes on non-critical paths may also be minimized by the resynthesis procedure, which may result in the use of a larger number of K-LUTs. In a mapping solution, if a node is implemented by an K-LUT using a resynthesis solution, and by switching to its combinatorial solution the depth of the entire network will not increase (although the depth of the node will increase), the node is said *non-essential*. The *cost* of a non-essential node is the number of K-LUTs that can be saved (in the mapping solution of the entire network) by switching its implementation to the combinatorial one. Note that a non-essential node may have negative cost. After the labeling phase is done, we will examine the solution, and select among the non-essential nodes the one with the maximum cost and temporarily switch its implementation to the combinatorial one to get a new solution. We repeat this on the new solutions until there is no non-essential node. Then, we find the prefix of the above switching sequence that results in the maximum area reduction (which can be zero), make those switchings permanent, and undo the remaining ones.

Finally, before the actual mapping, we will also compare the solution with the pure combinatorial solution, and the better one is adopted. This guarantees that FlowSYN will not result in a worse solution than FlowMap.

5. Implementation and Experimental Results

The FlowSYN algorithm has been implemented on SUN Sparc workstations. For practical purposes, several options are provided in our implementation to control the functional decomposition procedure, including the maximum number of variables to be considered, the maximum number of input combinations to be considered. and whether the first decomposition or the best decomposition to be taken. These choices control the trade-off between the efficiency and solution quality.

we tested FlowSYN on a set of MCNC benchmark circuits that were used by several previous algorithms [2, 5, 8]. The benchmark circuits have been synthesized for technology independent delay optimization, and are in multi-level format. After FlowSYN, we also performed the post-processing step of FlowMap [2] for area reduction. In this experiment, We only considered the first input combination for functional decomposition, since our experience showed that symmetry among the variables often exists. Also, we always accepted the first decomposition solution since the best solution is too expensive to find, and restricted the number of variables considered for functional decomposition to be no more than 12 (in some cases smaller bounds are sufficient) since the cost of the decomposition is exponential to the number of variables in the worst case. Although the program can handle larger parameters, this set of parameters gave us satisfactory solutions within very reasonable running time: the total CPU time used on the 17 benchmark circuits was less than 15 minutes on a SUN Sparc2 with 32MB memory.

Our results are compared with three previous mapping algorithms for depth minimization, namely, Chortle-d [5], MIS-pga-delay [8], and FlowMap [2], in Table 1. As can be seen from the table, FlowSYN outperformed all the other three algorithms. On two-thirds of the cases, FlowSYN improved the FlowMap solutions. Overall, the three previous algorithms used 15% to 25% more levels and 25% to 88% more 5-LUTs than FlowSYN. Comparisons with other algorithms also gave similar results. For example, Techmap-D [10] used 10% more levels and 26% more 5-LUTs than FlowSYN.

6. Conclusion

In this paper we have presented an integrated approach to synthesis and mapping for depth minimization in LUT-based FPGA designs. Our FlowSYN algorithm combines the global combinatorial optimization techniques and efficient functional decomposition process and achieves encouraging experimental results.

We are extending the FlowSYN algorithm in several directions, including the delay and area trade-off [3] using the integrated synthesis and mapping procedure, and more efficient algorithms for functional decompositions and Boolean resynthesis.

References

[1] Bryant, R. E., ‘‘Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams,’’ *ACM Computing Surveys*, Vol. **24**, pp. 293-318, Sept. 1992.

[2] Cong, J. and Y. Ding, ‘‘An Optimal Technology Mapping Algorithm fo Delay Optimization in Lookup-Table Based FPGA Designs,’’ *Proc. IEEE Int’l Conf. on Computer-Aided Design*, pp. 48-53, Nov. 1992.

[3] Cong, J. and Y. Ding, ‘‘On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping,’’ *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 213-218, June 1993.

[4] Francis, R. J., J. Rose, and K. Chung, ‘‘Chortle: A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays,’’ *Proc. 27th ACM/IEEE Design Automation Conference*, pp. 613-619, June 1990.

[5] Francis, R. J., J. Rose, and Z. Vranesic, ‘‘Technology Mapping of Lookup Table-Based FPGAs for Performance,’’ *Proc. IEEE Int’l Conf. on Computer-Aided Design*, pp. 568-571, Nov. 1991.

[6] Lai, Y.-T., M. Pedram, and S. Vrudhula, ‘‘BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis,’’ *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 642-647, June 1993.

[7] Murgai, R., Y. Nishizaki, N. Shenay, R. Brayton, and A. Sangiovanni-Vincentelli, ‘‘Logic Synthesis Algorithms for Programmable Gate Arrays,’’ *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 620-625, 1990.

[8] Murgai, R., N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, ‘‘Performance Directed Synthesis for Table Look Up Programmable Gate Arrays,’’ *Proc. IEEE Int’l Conf. on Computer-Aided Design*, pp. 572-575, Nov. 1991.

[9] Roth, J. P. and R. M. Karp, ‘‘Minimization Over Boolean Graphs,’’ *IBM Journal of Research and Development*, pp. 227-238, April 1962.

[10] Sawkar, P. and D. Thomas, ‘‘Performance Directed Technology Mapping for Look-Up Table Based FPGAs,’’ *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 208-212, June 1993.

[11] Schlag, M., J. Kong, and P. K. Chan, ‘‘Routability-Driven Technology Mapping for Lookup Table-Based FPGAs,’’ *Proc. 1992 IEEE International Conference on Computer Design*, pp. 86-90, Oct. 1992.

[12] Xilinx, *The Programmable Gate Array Data Book*, Xilinx, San Jose (1992).

Table 1 Comparison of FlowSYN mapping solutions with previous algorithms (K=5).

Circuit	FlowSYN		FlowMap		Mis-pga-delay		Chortle-d	
	#LUTs	Depth	#LUTs	Depth	#LUTs	Depth	#LUTs	Depth
5xp1	20	2	25	3	21	2	26	3
9sym	7	3	61	5	7	3	63	5
9symml	7	3	58	5	7	3	59	5
C499	133	5	154	5	199	8	382	6
C880	232	8	232	8	259	9	329	8
alu2	113	6	162	8	122	6	227	9
alu4	249	9	268	10	259	11	500	10
apex6	257	4	257	4	274	5	308	4
apex7	89	4	89	4	95	4	108	4
count	75	3	76	3	81	4	91	4
des	893	4	1308	5	1397	11	2086	6
duke2	187	4	187	4	164	6	241	4
misex1	15	2	15	2	17	2	19	2
rd84	13	3	43	4	13	3	61	4
rot	262	6	268	6	322	7	326	6
vg2	45	4	45	4	39	4	55	4
z4ml	6	2	13	3	10	2	25	3
total	2603	72	3261	83	3286	90	4906	87
comparison	1	1	+25.3%	+15.3%	+26.2%	+25.0%	+88.5%	+20.1%