

A Boolean Approach to Performance-Directed Technology Mapping for LUT-Based FPGA Designs

Christian Legl* Bernd Wurth** Klaus Eckl*

*Institute of Electronic Design Automation, Technical University of Munich, 80290 Munich, Germany

**Synopsys, Inc., 700 E. Middlefield Rd., Mountain View, CA 94043

Abstract — This paper presents a novel, Boolean approach to LUT-based FPGA technology mapping targeting high performance. As the core of the approach, we have developed a powerful functional decomposition algorithm. The impact of decomposition is enhanced by a preceding collapsing step. To decompose functions for small depth and area, we present an iterative, BDD-based variable partitioning procedure. The procedure optimizes the variable partition for each bound set size by iteratively exchanging variables between bound set and free set, and finally selects a good bound set size. Our decomposition algorithm extracts common subfunctions of multiple-output functions, and thus further reduces area and the maximum interconnect lengths. Experimental results show that our new algorithm produces circuits with significantly smaller depths than other performance-oriented mappers. This advantage also holds for the actual delays after placement and routing.

1 INTRODUCTION

An important class of FPGAs is based on the lookup-table (LUT) as the basic programmable logic block. A k -LUT implements any Boolean function of up to k variables. The LUTs are wired by various kinds of programmable interconnects [1]. Minimizing the delay of LUT-based FPGA designs is an important task because the programmable interconnects introduce extra delay compared with conventional gate array or standard cell technologies. The performance of an FPGA design is determined by the number of LUTs and the interconnect delays on the critical path.

Performance-driven technology mapping for LUT-based FPGAs is to transform a Boolean network, which has been produced in the technology independent logic optimization phase, into a functionally equivalent LUT network with minimum circuit delay. Technology mapping of a Boolean network is usually performed in two steps: The first step is the decomposition of nodes with more than k inputs into smaller nodes with k or less inputs. The resulting network is called *k-bounded*. A subsequent covering step finds a circuit of LUTs covering the k -bounded network.

A variety of technology mapping algorithms tackle performance optimization by minimizing the depth of k -bounded networks. Chortle-d, which is based on tree decomposition and bin-packing, is depth-optimal for trees [2]. DAG-Map heuristically minimizes the depth of a general k -bounded network [3]. The FlowMap algorithm is a significant advance since it guarantees a depth-optimal covering for general k -bounded networks [4].

However, minimizing the network depth does not consider the interconnect delays. Since the maximum interconnect length on the chip is correlated with the circuit area, minimizing the area also contributes to delay minimization. This is considered in the FlowMap-r [5] algorithm, which achieves depth-optimal mappings as FlowMap but reduces the number of LUTs. Recent algorithms improve on the FlowMap algorithm by modeling interconnect delay more accurately. This is done by assigning different delays to different nets (*nominal delay model* [6]) or even to the different interconnections of the same net [7]. The combination of the technology mapping and layout synthesis phases has been proposed to achieve small interconnect delays and improve routability [8, 9].

Another class of performance-directed algorithms performs Boolean operations during the covering step. Collapsing of criti-

cal nodes and re-decomposition are employed in MIS-pga(delay) [8], TechMap-D [10] and FlowSYN [11]. The FlowMap-based FlowSYN algorithm, which uses an efficient functional decomposition method, outperforms FlowMap-r in terms of circuit depth and area.

All these performance-directed mapping algorithms concentrate on the covering step. To obtain a k -bounded network in the decomposition step, the DMIG algorithm [3] and SIS-algorithms [12] like *xl_k_decomp*, *speed_up*, and *tech_decomp* are used. The decomposition step typically yields a network in which each gate has at most two inputs. This maximizes the flexibility during the covering step. A reason for the little attention given to the decomposition step is the fact that technology independent logic optimization generates Boolean networks with relatively small nodes. Thus, decomposition has only local effect, whereas a state-of-the-art covering step can deal with the entire network and therefore dominates the final result.

In this paper, we present a novel, Boolean approach to performance-directed technology mapping. As a first step, delay-driven collapsing is performed. In contrast to previous mapping approaches, large network portions are collapsed such that the decomposition step can have a significant impact. As the core of our approach, we have developed a powerful, functional decomposition algorithm that creates k -bounded networks with small depth and area. The main components of the decomposition algorithm are: First, a BDD-based, iterative variable partitioning procedure that efficiently evaluates a large number of variable partitions for their effect on circuit depth and area. Second, we have developed cost functions that estimate the delay and area of Boolean functions after decomposition and determine the effectiveness of the variable partitioning procedure. Third, we use a multiple-output decomposition approach which extracts common subfunctions.

The rest of the paper is organized as follows. Section 2 reviews the state of the art in functional decomposition. In Section 3, we present our performance-directed variable partitioning procedure. Section 4 describes the overall approach. We show experimental results in Section 5, and conclude the paper in Section 6.

2 REVIEW OF FUNCTIONAL DECOMPOSITION

Single-output decomposition. We first review the functional single-output decomposition which is based on the theory of Curtis [13], Roth and Karp [14]. Functional single-output decomposition breaks a function $f(\mathbf{x}, \mathbf{y})$ into the composition function $g(\mathbf{v}, \mathbf{y})$ and the subfunction vector $\mathbf{d}(\mathbf{x}) = (d_1(\mathbf{x}), \dots, d_c(\mathbf{x}))$ such that $f(\mathbf{x}, \mathbf{y}) = g(\mathbf{d}(\mathbf{x}), \mathbf{y})$. We deal with disjoint decompositions, where the bound set $BS = \{x_1, \dots, x_b\}$ and the free set $FS = \{y_1, \dots, y_{n-b}\}$ are disjoint sets where b is the size of the bound set and n is the number of inputs of f . In non-trivial decompositions, composition function g as well as the subfunctions d_i have fewer inputs than the original function f . Therefore, functional decomposition can be recursively used to compute k -bounded networks.

Two problems must be solved to perform functional decomposition. First, the input variables of f must be partitioned into the bound set and the free set. This is the *variable partitioning problem*. Second, given a variable partition, a minimum number c of subfunctions d_i must be computed. This number c depends on the variable partition.

We deal with the second problem first. To compute subfunctions d_i , the notion of *compatible* BS-vertices was introduced [14]. Two BS-vertices $\hat{\mathbf{x}}_v \in \{0, 1\}^b$ and $\hat{\mathbf{x}}_w \in \{0, 1\}^b$ are *compatible*, denoted by $\hat{\mathbf{x}}_v \sim \hat{\mathbf{x}}_w$, if and only if $\forall \hat{\mathbf{y}} \in \{0, 1\}^{n-b} : f(\hat{\mathbf{x}}_v, \hat{\mathbf{y}}) = f(\hat{\mathbf{x}}_w, \hat{\mathbf{y}})$. For completely specified functions, compatibility is an equivalence relation, which partitions the set of BS-vertices into *compatible classes*. The number of compatible classes is denoted by ℓ . The *decomposition condition* states that a decomposition with the subfunction vector \mathbf{d} exists if and only if $\forall \hat{\mathbf{x}}_v, \hat{\mathbf{x}}_w \in \{0, 1\}^b : \hat{\mathbf{x}}_v \not\sim \hat{\mathbf{x}}_w \implies \mathbf{d}(\hat{\mathbf{x}}_v) \neq \mathbf{d}(\hat{\mathbf{x}}_w)$, i.e., different

codes $\mathbf{d}(\hat{\mathbf{x}})$ must be assigned to incompatible BS-vertices. Thus, the minimum number c of subfunctions is $c = \lceil \log_2 \ell \rceil$. A simple method to compute subfunctions and to fulfill the decomposition condition is to assign a unique code of length c to each compatible class.

It is obvious that the computation of ℓ is an important subtask of functional decomposition. Roth and Karp [14] described how ℓ is computed using a SOP representation of the function f . Lai et al. showed that the computation is significantly sped up if the function f is represented by a BDD in which the bound set variables are ordered before the free set variables [15]. Lai introduced a set of BDD nodes called $cut_set(f, b)$ comprising all BDD nodes that have a level greater than b and a predecessor with a level less than or equal to b . Each node $v \in cut_set(f, b)$ is in a one-to-one correspondence to a compatible class. Thus, the number of compatible classes is given by the cardinality of $cut_set(f, b)$.

We now give a brief review of previous approaches to the variable partitioning problem. Note that all these approaches target area. During technology mapping for k -LUT architectures, usually BS cardinality k is chosen. The SOP-based functional decomposition method implemented in SIS either selects the first variable partition with BS size k that yields a non-trivial decomposition [16], or enumerates all partitions of fixed size [17]. The enumerative approach was adapted for BDD-based functional decomposition by Lai et al. [15] and Sasao [18]. Since enumeration is very expensive, it is not applicable for functions with many variables. Recently, a heuristic was proposed which directly constructs a BS of fixed size from the SOP representation of f [19]. In contrast to the approaches mentioned above, Schlichtmann proposed a BDD-based variable partitioning approach that also selects a good BS size [20]. Our new performance-directed variable partitioning presented in Section 3 is based on this approach.

Multiple-output decomposition. We briefly summarize functional multiple-output decomposition. Its goal is to compute subfunctions d_i that can be used for several outputs. Functional multiple-output decomposition breaks a multiple-output function $\mathbf{f}(\mathbf{x}, \mathbf{y}) = (f_1, \dots, f_m)$ into the composition function vector $\mathbf{g}(\mathbf{v}, \mathbf{y}) = (g_1, \dots, g_m)$ and the subfunction vector $\mathbf{d}(\mathbf{x}) = (d_1, \dots, d_q)$ such that $\mathbf{f}(\mathbf{x}, \mathbf{y}) = \mathbf{g}(\mathbf{d}(\mathbf{x}), \mathbf{y})$. Each composition function output g_k depends on a subset of the q subfunctions d_i . Precisely, g_k depends on $c_k = \lceil \log_2 \ell_k \rceil$ subfunctions d_i , where ℓ_k is the number of compatible classes of function $f_k(\mathbf{x}, \mathbf{y})$. This guarantees that multiple-output decomposition of a vector \mathbf{f} is at least as good as single-output decomposition of each output of \mathbf{f} with respect to a given variable partition.

Multiple-output functional decomposition has the advantage that by extracting common subfunctions it performs a task that is typically confined to the logic optimization stage before technology mapping.

The problem faced during multiple-output decomposition is the usually very large number of possible subfunctions for each output. To cope with this problem, we use the multiple-output decomposition approach as described in [21]. Additionally, we compute subfunctions with minimal support [22].

3 PERFORMANCE-DIRECTED VARIABLE PARTITIONING

For ease of explanation, we first describe the variable partitioning procedure for a single-output function f . Our goal is to partition the variables of f into bound set variables \mathbf{x} and free set variables \mathbf{y} such that the arrival time $at(g)$ at the output of composition function g is minimal. We use the unit delay model, i.e., each LUT has a delay of 1 unit. The second goal of our variable partitioning procedure is to reduce the LUT count needed to implement function f . Minimizing the LUT count reduces the maximum interconnect length on the FPGA chip and thus also affects performance.

We illustrate the variable partitioning problem with an example. To obtain a small arrival time $at(g)$, one might intuitively assign early arriving inputs to the bound set, and inputs with large arrival times to the free set.

Example 1 Figure 1 shows an example with the bound set size 3. Please assume that we want to achieve a 3-LUT implementation. The numbers at the inputs denote arrival times. Each of the resulting subfunctions d_1 and d_2 has 3 inputs and can be implemented by a single 3-LUT, thus we have propagation delays $dt(d_i) = 1$ and $at(d_i) = 3$. Since the composition function g has 4 inputs, it has to be decomposed further, and we have an estimated propagation delay $dt(g) = 2$

and arrival time $at(g) = 6$. Note that we must assume a propagation delay of 2 for each path through g since we do not know at this point how g will be decomposed.

It is easily recognized that using the variable with arrival time 3 in the bound set and one of the inputs with arrival time 2 in the free set would not increase the maximum arrival time of the inputs of g , which is 4 anyway. Thus, there are several variable partitions of bound set size 3, all of which should be evaluated to possibly reduce the number c of subfunctions d_i . Note that a variable partition for which $c = 1$ reduces the number of inputs of g to 3, thus decreasing the arrival time $at(g)$ to 5. \square

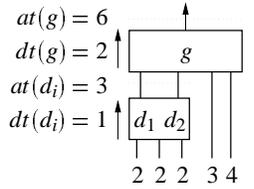


Figure 1: Delay oriented decomposition.

The example shows that the variable partition must take into account the arrival times of the input variables, the number $c = \lceil \log_2 \ell \rceil$ of subfunctions, and the estimated propagation delays of the resulting functions d_i and g , which depend on the BS size and c , respectively.

We separate the variable partitioning problem into two subtasks. First, we determine an optimal variable partition VP_i for each bound set size i . Then we select the best partition among all VP_i . A solution of the two subtasks requires proper cost functions.

To solve the first subtask, we propose an iterative heuristic. Each iteration step involves a cost-reducing exchange of a BS and a FS variable. Note that the BS size is given and is not allowed to change. A greedy method would require $|BS| \cdot |FS|$ tentative variable exchanges in an iteration step to find the best exchange of a BS and a FS variable. This is too expensive if the number of variables is large. Therefore, we compute the BS variable that yields the lowest costs if moved to the FS, and similarly we compute the FS variable that yields the lowest costs if moved to the BS. The *best* BS and the *best* FS variables found in such a way are exchanged if this reduces the costs. An iteration step then requires only $|BS| + |FS|$ tentative variable exchanges to find a good (and possibly suboptimal) exchange.

We employ cost functions for delay and area to evaluate tentative variable exchanges and to solve the second subtask mentioned above. The cost functions estimate the arrival time of the composition function g and the total LUT count of the resulting functions d_i and g . The only information used as input of the delay and area cost functions is the BS size b , the number of compatible classes ℓ , the maximum arrival time among the BS variables, denoted by $at_{max}(\mathbf{x})$, and the maximum arrival time among the FS variables, denoted by $at_{max}(\mathbf{y})$. Let us first introduce the *function propagation delay estimate FDE* and the *function area estimate FAE* of a Boolean function $h(\mathbf{x})$.

Definition 1 The *function propagation delay estimate FDE* is a measure of the propagation delay of a Boolean function $h(\mathbf{x})$ in the unit delay model. It is defined by

$$FDE(h(\mathbf{x})) = \begin{cases} 1 & : |\mathbf{x}| \leq k \\ |\mathbf{x}| - k + 1 & : |\mathbf{x}| > k \end{cases} \quad (1)$$

where $|\mathbf{x}|$ denotes the number of variables that h depends on, and k is the number of inputs of a LUT.

The *FDE* is the depth of a LUT network obtained by the Shannon decomposition of h and thus is a worst-case estimate of the function's propagation delay in the unit delay model.

For area we have examined various estimates that are linear, quadratic or exponential in the number of variables. Our experiments have shown that the best results are obtained using the same linear estimate as for the function propagation delay estimate.

Definition 2 The *function area estimate FAE* is a measure of the area of a Boolean function $h(\mathbf{x})$ in terms of LUTs. It is defined by

$$FAE(h(\mathbf{x})) = \begin{cases} 1 & : |\mathbf{x}| \leq k \\ |\mathbf{x}| - k + 1 & : |\mathbf{x}| > k \end{cases} \quad (2)$$

We define the **delay cost** function as

$$DC = \max[(at_{max}(\mathbf{x}) + FDE(\mathbf{d})), at_{max}(\mathbf{y})] + FDE(g), \quad (3)$$

which is a simple computation of the arrival time $at(g)$ as described in standard literature.

The **area cost** function

$$AC = \sum_{j=1}^c FAE(d_j) + FAE(g) \quad (4)$$

sums up the area estimates for the resulting functions. Note that we must know the number $c = \lceil \log_2 \ell \rceil$ of subfunctions to compute the delay and area estimates $FDE(g)$ and $FAE(g)$.

It has been shown in Section 2 that computing the number ℓ of compatible classes is simple if f is represented by a BDD and the BS variables are ordered before the FS variables. In this case, we have $\ell = |\text{cut_set}(f, b)|$. Thus, the delay and area cost functions can be evaluated very fast for a given variable partition. Representing function f by a BDD additionally has the advantage that variable moves and thus variable exchanges can be performed rapidly. If, e.g., a variable on level i shall be moved to level j , $j - i$ adjacent variable swaps must be performed. Adjacent variable swaps modify the BDD only on the levels of the swapped variables and are therefore carried out rapidly.

Let us resume the discussion of our variable partitioning procedure. We now describe in more detail the computation of the *best* BS and the *best* FS variable for a bound set size b . To find the best BS variable, the topmost BS variable is tentatively moved to the FS. This is done in the BDD by a variable move (a sequence of adjacent variable swaps) from level 1 to level b . Thus, the variable previously on level 2 is on level 1 now, the variable previously on level b is on level $b - 1$ and the variable previously on level 1 is now on level b . The BDD is then traversed to compute $\text{cut_set}(f, b - 1)$; delay and area costs are evaluated and stored. The variable move from level 1 to level b is repeated for the remaining $b - 1$ BS variables. The *best* BS variable is the variable with minimal area cost among all BS variables with minimal delay cost. The *best* FS variable is computed similarly by a sequence of variable moves (from level n to level $b + 1$) and BDD traversals. After exchanging the best BS and FS variable, the BDD is traversed once again to compute $\text{cut_set}(f, b)$ and to check if the costs have actually been reduced by the exchange.

Example 2 We illustrate the computation of the best BS and FS variable for the function $f(\mathbf{z}) = z_1 z_2 z_3 z_4 + \bar{z}_1 \bar{z}_5$ and a bound set size of 3. Figure 2 a) shows the initial BDD of function f where the variables z_i are ordered according to their arrival times $at(z_i)$, which are indicated by the numbers next to the corresponding BDD nodes. First, we have to compute the costs for the initial variable partition $BS = \{z_1, z_2, z_3\}$ and $FS = \{z_4, z_5\}$. For ease of explanation, we only consider delay costs. In Figure 2 a), all nodes that have a level greater than 3 and a predecessor with a level less than or equal to 3 are shaded. These nodes comprise the $\text{cut_set}(f, 3)$. We need $c = 2$ subfunctions, as $l = |\text{cut_set}(f, 3)| = 3$. Note that we have the same decomposition structure and the same arrival times of the BS and the FS variables as in Figure 1 of Example 1. Therefore, we have a delay cost of $DC = 6$. Now, we have to compute the best BS variable. We move z_1 from level 1 to level 3. The obtained BDD is shown in Figure 2 b). The delay cost after moving z_1 to the free set is determined by evaluating the delay cost function DC for the $BS = \{z_2, z_3\}$. As there are 2 nodes in $\text{cut_set}(f, 2)$, only one subfunction is needed, which can be implemented by a single 3-LUT. Therefore, the estimated propagation delay of this subfunction is $FDE(d) = 1$. The resulting composition function g depends on 4 variables. Thus, $FDE(g)$ evaluates to 2. Using Equation (3) and the maximum arrival times of the BS and FS variables, we obtain $DC = \max[2 + 1, 4] + 2 = 6$. If z_2 and z_3 , respectively, are moved to the free set, we get delay costs of 7. Thus, the best BS variable is z_1 .

The computation of the best FS variable is done similarly. Moving z_5 to the bound set yields a delay cost of 7, whereas moving z_4 to the bound set yields a delay cost of 6. Thus, the best FS variable is z_4 .

Now, the best BS and FS variables are exchanged as shown in the BDD of Figure 2 c) in order to check if a cost reduction is achieved. The resulting BDD has 2 nodes in $\text{cut_set}(f, 3)$ so that we need one subfunction. The number of inputs to the composition function g is 3. Thus, $FDE(d)$ and $FDE(g)$ evaluate to 1. We obtain $DC = 5$. Thus, the delay costs are reduced from 6 for the initial partition to 5 for the new variable partition $BS = \{z_2, z_3, z_4\}$ and $FS = \{z_1, z_5\}$. \square

Iterative variable exchanges are performed for a given BS size b to find an optimal variable partition VP_b . However, a closer look reveals that we can gather information that is useful for other BS sizes during the iterative procedure. Note that the BDD is completely traversed $|\text{BS}| + |\text{FS}| + 1$ times for each variable exchange. Although each BDD traversal is carried out to compute the cut set for a specific BS size b , we can gather *all* cut sets $\text{cut_set}(f, i)$, $i = 2, \dots, n - 1$, with only small additional computational effort. For each BS size i , we com-

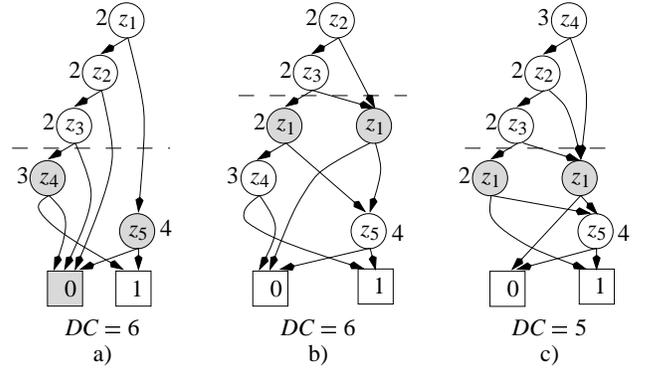


Figure 2: BDDs of Example 2.

pare the delay and area costs, as computed using the current variable order, with delay and area costs stored for VP_i . If a cost reduction is achieved, then VP_i is replaced by the variable partition determined by the current variable order. This method yields a coupling between the iterative procedures performed for each BS size.

Multiple-output decomposition of function vector $\mathbf{f}(\mathbf{x}, \mathbf{y}) = (f_1, \dots, f_m)$ requires a slight modification of this algorithm. First, we have a BDD with several roots, one for each output f_j . Second, the delay and area cost functions must be modified:

We use the **multiple-output delay cost** function

$$MDC = \max[DC_1, \dots, DC_m], \quad (5)$$

where DC_j denotes the delay cost for output f_j . The **multiple-output area cost** function sums up the area estimates AC_j for each output f_j :

$$MAC = \sum_{j=1}^m AC_j. \quad (6)$$

4 ALGORITHM OVERVIEW

In this section we describe the overall algorithm for performance-directed technology mapping. The algorithm consists of three steps, i.e., collapsing, decomposition, and covering.

We first try to completely collapse the circuit within a given limit of CPU time. If collapsing is possible, the decomposition step starts from the obtained flattened circuit. Otherwise, a depth-oriented partial collapsing is performed by applying the SIS-command *reduce_depth -r -d d_value* [23]. This command, which is based on Lawler's clustering algorithm, first clusters nodes and then collapses each cluster such that the resulting network has the specified depth d_value and the cluster size is minimal. Since this command should only be used on a network with nodes of comparable complexity, we first decompose the nodes of the original network into nodes with at most k inputs using our performance-directed decomposition approach. We then apply *reduce_depth* to the decomposed network to obtain a network with depth 3 or 4. These networks are used as the starting point for the final performance-directed decomposition.

For the decomposition step, functional multiple-output decomposition as described in Section 2 and the variable partitioning algorithm of Section 3 are used. Only nodes of the network with more than k inputs are decomposed. As candidates to be decomposed we select only these nodes for which all nodes in the transitive fanin have at most k inputs. This guarantees accurate arrival times at the inputs of the considered nodes.

After all nodes in the network have been decomposed into nodes with at most k inputs, we do a simple covering step. A node is collapsed into its successors if each successor does not have more than k inputs after collapsing.

5 EXPERIMENTS

Depth and Area Results. We implemented our new approach called *BoolMap-D* and integrated it into the synthesis tool TOS^{nm}. We compared *BoolMap-D* with two other performance-directed technology mappers, i.e., *FlowMap-r* [5] and *FlowSYN* [11]. We used the same set of benchmark circuits as in [5, 11], which are given in the first column of Table 1. In columns 2 to 5, we repeat the LUT count and depth results for *FlowMap-r* and *FlowSYN* from [5, 11]. The columns titled *BoolMap-D* show the results of our algorithm as described in Section 4. CPU times of column 8 are measured on a

Table 1: TECHNOLOGY MAPPING FOR 5-LUT BASED FPGAs

circuit	FlowMap-r[5]		FlowSYN[11]		BoolMap-D		
	#LUT	depth	#LUT	depth	#LUT	depth	CPU/s
5xp1	23	3	20	2	13	2	3.6
9sym	61	5	7	3	7	3	1.4
9symml	58	5	7	3	7	3	1.4
C499*	151	5	133	5	101	4	627.8
C880*	211	8	232	8	146	7	62.3
alu2	148	8	113	6	43	4	38.5
alu4	245	10	249	9	268	7	950.0
apex6	232	4	257	4	189	4	139.1
apex7	80	4	89	4	78	3	70.3
count	73	3	75	3	42	2	30.2
des*	1087	5	893	4	594	3	1787.1
duke2	187	4	187	4	193	5	346.9
misex1	15	2	15	2	15	2	1.1
rd84	43	4	13	3	10	2	3.6
rot*	243	6	262	6	228	6	99.0
vg2	38	4	45	4	30	4	22.4
z4ml	13	3	6	2	5	2	0.6
sum	2908	83	2603	72	1969	63	-
perc.	-	-	100%	100%	-24.4%	-12.5%	-

DEC AlphaStation 250 4/266. All circuits that have been partially collapsed are marked with an asterisk in Table 1. For the marked circuits, the CPU times include initial decomposition, partial collapsing (*reduce_depth*), the final performance-directed decomposition, and covering. For the other circuits, CPU time is spent for collapsing, performance-directed decomposition, and covering.

BoolMap-D outperforms *FlowMap-r* and *FlowSYN* with respect to the circuit depth by 24.1% and 12.5%, respectively. Furthermore, a reduction in LUT count of 32.3% and 24.4% compared to *FlowMap-r* and *FlowSYN* is achieved. There is only one circuit, *duke2*, for which *BoolMap-D* produces a larger depth than *FlowMap-r* or *FlowSYN*. Compared to *FlowMap-r*, *BoolMap-D* produces 12 circuits with smaller depth and 4 circuits with equal depth. Compared to *FlowSYN*, *BoolMap-D* produces 8 circuits with smaller depth and 8 circuits with equal depth. For all circuits except for *alu4* and *duke2*, *BoolMap-D* produces circuits with fewer LUTs than *FlowMap-r* and *FlowSYN*, respectively.

Delay after Placement and Routing. To show the effectiveness of *BoolMap-D* in reducing the circuit delay after placement and routing, we implemented all designs (except *apex6*, *des*, and *rot*) obtained with *BoolMap-D* on Xilinx XC3000 FPGAs. The circuits *apex6* and *rot* have too many I/O pins to be implemented on a single XC3000 FPGA, and circuit *des* has too many CLB's [1]. We used the Xilinx tool *apr* for placement and routing. For each design, we selected a Xilinx XC3000 chip which yields about 80% cell utilization as proposed in [1]. The circuits obtained with *BoolMap-D* could be routed easily. In fact, all designs were routed in the first routing attempt.

We compare our results with the results of a FlowMap-type algorithm that aims at minimizing the nominal delay of a circuit [6]. We used the same set of benchmark circuits as in [6]. Column 2 of Table 2 shows the type of the used Xilinx XC3000 chip. Columns 3 and 4 repeat the number of CLB's and the actual delay after placement and routing for the *Nominal Delay Algorithm* [6]. The columns titled *BoolMap-D* show the results for our approach. As in [6], we measured the actual circuit delays using the Xilinx tool *xdelay*. The last column gives the relative reduction of the circuit delay achieved by *BoolMap-D*. The circuit delay is reduced by 27.8% on average.

6 CONCLUSION AND FUTURE WORK

In this paper, we have presented *BoolMap-D*, a Boolean approach to simultaneously minimize depth and area during LUT-based FPGA technology mapping. In the first step, large network portions are collapsed. Collapsing is motivated by the idea that decomposition has then a greater potential to determine a new network structure with small depth and area.

Functional decomposition is applied to the collapsed networks. We have presented an effective heuristic solution of the variable partitioning problem targeting small circuit depth in the first place and small area in the second place.

Compared to the mapping algorithms *FlowMap-r* and *FlowSYN*, *BoolMap-D* reduces the depth of LUT networks by 24.1% and 12.5% on average, and the number of LUTs by 32.3% and 24.4%, respectively. We also placed and routed Xilinx FPGA designs, and achieved

Table 2: DELAYS AFTER PLACEMENT AND ROUTING

Circuit	XC3000 part#	Nom.Del.Alg [6] #CLB	actual delay	BoolMap-D	
				#CLB	actual delay red.
9sym	3020PC68	50	94.3	6	59.0 37.4%
C880	3090PQ208	195	208.3	117	143.8 31.0%
alu2	3064PC84	149	200.1	37	85.0 57.5%
apex7	3042PP132	65	93.2	59	80.7 13.4%
count	3020PC68	60	79.8	31	65.6 17.8%
vg2	3020PC68	39	78.1	25	70.7 9.5%
aver.					27.8%

delay reductions of 27.8% on average compared with a FlowMap-type nominal delay algorithm.

ACKNOWLEDGMENT

The authors are very grateful to Prof. Kurt J. Antreich and Dr. Ulf Schlichtmann for many valuable discussions.

REFERENCES

- [1] Xilinx Inc., San Jose, CA-95125, *The Programmable Logic Data Book*, 1994.
- [2] R. J. Francis, J. Rose, and Z. Vranesic, "Technology mapping of lookup table-based FPGAs for performance," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 568–571, 1991.
- [3] K.-C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, "DAG-Map: Graph-based FPGA technology mapping for delay optimization," *IEEE Design & Test of Computers*, pp. 7–20, Sept. 1992.
- [4] J. Cong and Y. Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems CAD*, vol. 13, pp. 1–12, Jan. 1994.
- [5] J. Cong and Y. Ding, "On area/depth trade-off in LUT-based FPGA technology mapping," in *30th ACM/IEEE Design Automation Conference (DAC)*, pp. 213–218, 1993.
- [6] J. Cong and Y. Ding, "On nominal delay minimization in LUT-based FPGA technology mapping," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 82–88, Feb. 1995.
- [7] H. Yang and D. F. Wong, "Edge-Map: Optimal performance driven technology mapping for iterative LUT based FPGA designs," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 150–155, 1994.
- [8] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance directed synthesis for table look up programmable gate arrays," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 572–575, 1991.
- [9] N. Togawa, M. Sato, and T. Ohtsuki, "Maple: A simultaneous technology mapping, placement, and global routing algorithm for field-programmable gate arrays," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 156–163, Nov. 1994.
- [10] P. Sawkar and D. Thomas, "Performance directed technology mapping for look-up table based FPGAs," in *30th ACM/IEEE Design Automation Conference (DAC)*, pp. 208–212, 1993.
- [11] J. Cong and Y. Ding, "Beyond the combinatorial limit in depth minimization for LUT-based FPGA designs," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 110–114, Nov. 1993.
- [12] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," in *IEEE International Conference on Computer Design (ICCD)*, pp. 328–333, Oct. 1992.
- [13] H. A. Curtis, "A generalized tree circuit," *Journal of the ACM*, vol. 8, pp. 484–496, 1961.
- [14] J. P. Roth and R. M. Karp, "Minimization over boolean graphs," *IBM Journal of Research and Development*, pp. 227–238, 1962.
- [15] Y. Lai, M. Pedram, and S. Vrudhula, "BDD based decomposition of logic functions with application to FPGA synthesis," in *30th ACM/IEEE Design Automation Conference (DAC)*, pp. 642–647, June 1993.
- [16] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Logic synthesis for programmable gate arrays," in *27th ACM/IEEE Design Automation Conference (DAC)*, pp. 620–625, June 1990.
- [17] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved logic synthesis algorithms for table look up architectures," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 564–567, Nov. 1991.
- [18] T. Sasao, *Logic Synthesis and Optimization*. Kluwer Academic Publishers, Boston/London/Dordrecht, 1993.
- [19] W.-Z. Shen, J.-D. Huang, and S.-M. Chao, "Lambda set selection in Roth-Karp decomposition for LUT-based FPGA technology mapping," in *32nd ACM/IEEE Design Automation Conference (DAC)*, pp. 65–69, 1995.
- [20] U. Schlichtmann, "Boolean matching and disjoint decomposition," in *IFIP Workshop on Logic and Architecture Synthesis*, pp. 83–102, Dec. 1993.
- [21] B. Wurth, K. Eckl, and K. Antreich, "Functional multiple-output decomposition: Theory and an implicit algorithm," in *32nd ACM/IEEE Design Automation Conference (DAC)*, pp. 54–59, June 1995.
- [22] C. Legl, B. Wurth, and K. Eckl, "An implicit algorithm for support minimization during functional decomposition," in *European Design and Test Conference (EDAC/ETC/EUROASIC)*, March 1996.
- [23] H. Touati, H. Savoj, and R. K. Brayton, "Delay optimization of combinational logic circuits by clustering and partial collapsing," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 188–191, 1991.