BDD-based Logic Synthesis for LUT-based FPGAs

Navin Vemuri
University of Massachusetts, Amherst
and
Priyank Kalla
University of Massachusetts, Amherst
and
Russell Tessier
University of Massachusetts, Amherst

Contemporary FPGA synthesis is a multi-phase process in which conventional technology-independent logic optimization techniques are applied followed by mapping the decomposed network onto FPGA technology. Conventional technology-independent transformations traditionally target standard cells and are unable to optimize circuits with constraints and goals consistent with designing for FPGA architectures. This paper proposes a logic synthesis approach, *specific to FPGA technology*, which attempts to unify various multi-level logic transformation, decomposition and optimization techniques in a novel synthesis framework. Keeping in mind that the optimized network is to be mapped onto LUT-based FPGAs, we make decisions earlier in the synthesis process to guide the network transformation, decomposition and optimization steps and generate a network which can be *directly* mapped onto FPGAs.

The techniques described in this paper are built upon a BDD-based logic decomposition system. Using an efficient BDD-based decomposition approach, we can identify not only AND-OR decompositions, but also AND-XOR decompositions, resulting in large area savings while synthesizing XOR-intensive circuits. To induce good decompositions, a *maximum fanout free cone* (MFFC) based partial clustering and subsequent collapsing technique is applied. Subsequently, a novel area-driven (minimum LUTs) variable partitioning heuristic is used to decompose collapsed nodes into sub-functions which can be mapped onto individual LUTs. As a post-processing step, a performance driven re-synthesis phase is proposed to alleviate increased delay caused by excessive logic sharing. This delay can be improved by reducing the levels topologically using i) node clustering and collapsing, ii) logic simplification, followed by iii) re-decomposition.

The efficiency and robustness of the proposed techniques are demonstrated experimentally over a large set of MCNC and ISCAS benchmarks. We compare the quality of results obtained using our techniques with those of conventional FPGA synthesis tools used in both academia (Boolmap, MIS-pga, etc.) as well as in industry (Altera's Quartus FPGA synthesis tool). The experimental results demonstrate that the circuits generated by our technique are not only smaller but also significantly faster than those synthesized by using conventional FPGA synthesis tools. Furthermore, computation times required by our technique are significantly lower as compared with other techniques, especially for the larger circuits of the ISCAS benchmark suite.

Submitted to The ACM Trans. on Design Automation, Special issue on Programmable Logic
Designated Contact Author*: Prof. Russell Tessier, Dept. of Electrical and Computer Engineering, University of Massachusetts at Amherst, Email: tessier@ecs.umass.edu, Ph: (413)-545-0160; Fax: (413)-545-1993*

Categories and Subject Descriptors: B.6.3 [**Hardware**]: Logic Design

General Terms: Algorithms, Design

Additional Key Words and Phrases: FPGA, logic synthesis, BDD, decomposition

---

1. INTRODUCTION

FPGAs are a vital component of rapid prototyping systems which are often used as technology demonstrators and for the functional validation of ASIC designs. In this paper, a logic synthesis approach dedicated to optimizing circuits for k-LUT-based FPGA architectures is presented. This approach encompasses a complete FPGA-specific logic synthesis system, which includes various network transformations, *technology dependent* logic decomposition, subsequent optimization and technology mapping.

Earlier attempts at FPGA synthesis [Murgai et al. 1990] [Sicard and *et al.*d 1991] [Babba and Crastes 1992] proposed the idea of contemporary technology-independent logic synthesis followed by technology mapping onto FPGA architectures. The premise behind this approach was that multi-level logic optimization techniques were mature enough to generate minimized multi-level logic nodes, and technology mapping onto LUTs could be carried out as a *post-processing* step [Murgai et al. 1995]. A number of FPGA technology mapping approaches have been proposed in literature [Murgai et al. 1990][Francis et al. 1990] [Francis et al. 1995] [Filo and *et al.* 1991] [Sawkar and Thomas 1992] [Cong and Ding 1994]. While many techniques demonstrated robustness [Chen et al. 1992] and optimality [Cong and Ding 1994] in mapping onto FPGAs, they were used subsequent to, and in isolation with, the logic optimization steps. Hence, the resulting area/delay characteristics of the circuits were not satisfactory, especially for large designs. Subsequently, a number of attempts have been made to restructure/optimize the logic to facilitate mapping onto FPGA architectures.

Karplus [Karplus 1991] proposed Xmap, which uses an **if-then-else** (ITE) DAG for representing functions and uses cofactoring for functional decomposition. Subsequent bin-packing techniques are used for technology mapping. Recent techniques [Legl et al. 1996] [Eckl et al. 1996][Jiang et al. 1997] [Stanion and Sechen 1995] follow a three step approach: First, *optimization* of circuits using *conventional technology independent optimization* tools such as SIS [Sentovich and *et al.* 1992] is carried out. Architecture specific network transformation heuristics are then used to reduce the logic depth in order to create k-feasible *supernodes*. Finally, mapping algorithms are used to realize the circuits in the desired FPGA architecture.

One notices that a common feature of the above techniques is the initial technology independent optimization step. The implications of performing conventional technology independent optimization (*e.g.* SIS-type) prior to, and in isolation with, FPGA-specific restructuring and subsequent mapping are as follows:

(1) Conventional synthesis tools perform a variety of network transformations/optimizations which are geared specifically towards *standard cell* architectures. Knowledge of FPGA architectures is not used in the optimization process. The goal of multi-level logic optimization, with standard cell technology as the target, is to minimize the number of literals. For LUT-based FPGAs, resource consumption depends on the the number of inputs to the look-up table. A literal-minimized network when mapped onto LUTs may lead to a suboptimal implementation.

(2) Larger nodes need to be decomposed, into smaller, less complex, sub-functions. Logic optimization techniques rely extensively on algebraic factoring/decomposition. Conventional AND-OR intensive factoring heuristics normally applied are un-

able to efficiently generate sub-functions that can be directly mapped onto LUTs.

(3) Conventional logic synthesis tools resort to aggressive factoring (kernel extraction) in order to reduce the number of literals, often resulting in high-fanout nodes. While this may be acceptable for standard cell implementations (if library cells with sufficient drive capabilities exist), FPGAs however suffer from limited resources; often resulting in routing difficulties. Also, aggressive extraction creates too many nodes which results in resource consumption and degrades the area/delay characteristics.

Recently, Chen and Cong [Chen and Cong 2001] proposed that since the impact of logic decomposition on delay and area cannot be forecasted accurately, mapping could be performed simultaneously over a *set of decompositions*, while choosing the one combination that provides the best solution. They effectually combined the *technology decomposition* phase with *technology mapping*. However, the initial *logic optimization* used was still conventional (say, SIS-type). Our approach differs from [Chen and Cong 2001] in the sense that we take into consideration the target technology (k-input LUTs) during the technology-independent logic optimization/decomposition phase.

1.1 Research Contributions

To overcome the limitations of the above conventional FPGA synthesis techniques, this paper proposes a novel LUT-based FPGA synthesis approach that unifies technology-independent logic optimization with LUT-based logic restructuring. We guide the network transformation and subsequent logic decomposition and optimization steps to generate a network which can be efficiently mapped to LUT-based FPGAs.

BDDs have been exploited by CAD engineers for FPGA synthesis [Legl et al. 1996] [Chang et al. 996] [Sawada et al. 1995] [Lai et al. 1993] [Jiang et al. 1997], most significantly for the logic decomposition phase. Recently, a generic BDD-based functional decomposition approach called **BDS** [Yang 2000] was presented which can efficiently identify both *algebraic* and *Boolean* (also, AND-OR and AND-XOR) decompositions. Using a *functional decomposition engine* similar to that of BDS, we present a comprehensive logic synthesis system, to specifically target logic decompositions for k-feasible LUTs. We use Binary Decision Diagrams [Bryant 1986] to represent and manipulate Boolean functions in order to perform logic decompositions. To induce good decompositions, a *maximum fanout free cone* (MFFC) based partial collapse technique, in the Boolean domain, is applied. Efficient BDD-based variable partitioning heuristics are subsequently used to decompose all collapsed nodes into k-LUT feasible sub-functions. The above transformations are targeted to realize the circuit with a minimum number of LUTs (minimum area). These enhanced FPGA-specific optimization capabilities, form the heart of our synthesis framework, named **BDS-pga** (BDD-based Decomposition system for FPGAs). The synthesis process is completed with technology mapping using the popular **Flowmap** tool [Cong and Ding 1994].

The technology mapping tool Flowmap uses a Max-flow Min-cut algorithm for depth optimal mapping of k-feasible nodes on k-input LUTs. It also incorporates

heuristics to perform area minimal mapping. Flowmap pre-processes the optimized network by performing a two input AND-OR decomposition on each node in the network. Such a restructuring can potentially undo the decompositions performed by BDS-pga. The authors wish to highlight that since BDS-pga already creates a k-feasible network, this two-input AND-OR decomposition feature of Flowmap is disabled so that BDS-pga's logic decomposition is not destroyed.

To reduce critical-path delay, we incorporate a performance driven re-synthesis step. This delay can be improved by reducing the levels topologically using i) controlled clustering and collapsing, ii) re-decomposition, iii) further logic simplification and iv) subsequent technology re-mapping. Experiments using area minimization and subsequent delay re-synthesis indicate that circuits generated using our technique are not only smaller but also significantly faster than those synthesized by using conventional FPGA synthesis approaches.

The remainder of the paper describes our logic synthesis infrastructure. Section 2 introduces the terminology used in this paper and reviews the logic decomposition using BDS. We also present the motivation for the techniques derived in this paper by means of preliminary experiments, and present an overview of the CAD methodology targeted for LUTs. In Section 3, we present an MFFC based iterative partial collapse approach. Section 4 describes two decomposition algorithms, which are based on variable partitioning of BDD graphs. Section 5 focuses on the strategies devised to tackle delay reduction issues. Experimental results are presented in Section 6 and the achievements and limitations of **BDS-pga** are analyzed. Section 7 discusses possible future work and concludes the paper.

2. PRELIMINARIES

A combinational Boolean network η can be represented by a directed acyclic graph $\eta = (V, E)$ where each node $v \in V$ represents an arbitrary logic gate and each directed edge $(u, v) \in E$ represents a connection from the output of the node u to the input of node v. The *depth* of a node v is the number of edges on the longest path from any primary input (PI) to v. In this paper, we define the depth of each PI as one. The depth of a network is the largest depth for nodes in the network. Let $input(v)$ and $fanout(v)$ represent the set of fanins and the set of fanouts of node v, respectively. The **support** of a Boolean expression F is the set of variables, $\mathcal{S}_{\mathcal{F}}$ that F explicitly depends on, *i.e.* fanin of that node. For example, if $F = xy + x'y'$, $\mathcal{S}_{\mathcal{F}} = \|(x, y)\| = 2$. A **Binary Decision Diagram (BDD)** is a rooted, directed acyclic graph (DAG) representing a switching function [DeMicheli 1994], with an unconstrained number of in-edges and two out-edges, one for each of the one and zero decision paths of any given variable.

A *fanin cone* C_v rooted at v is a connected subnetwork consisting of v and its predecessors. A *fanout free cone* (FFC) is a subnetwork where *no* node in the cone is connected to nodes *not* in the *cone*. A *maximum fanout free cone* (MFFC) consists of the maximum allowable nodes without violating the fanout condition. Let k be the number of inputs to a LUT (k-LUT). A *k-feasible* node has no more than k inputs. All nodes in a k-feasible network are k-feasible.

In a network η, the topologically longest path(s) are considered to be critical. Nodes on the critical path(s) are critical nodes. In this paper, we refer to ϵ-critical paths as those paths whose topological lengths are within ϵ of the longest path. $\mathcal{L}$

refers to the longest critical path, while $\mathcal{L}_\Upsilon$ refers to the set of all longest paths. $\mathcal{L}_\epsilon$ is the set of paths that are ϵ-critical.

2.1 Review of BDD-based Logic Optimization

In this subsection, we briefly review the basic BDD-based logic decomposition theory presented in [Yang 2000], as it appears in the context of our work. For a detailed description of the **BDS** decomposition system, the reader is referred to [Yang et al. 1999] [Yang 2000].

BDD-based Functional Decomposition: In our synthesis framework, BDDs [Bryant 1986] are the functional representation of choice. First, a Reduced Ordered Binary Decision Diagram (ROBDD) is built for a function. BDDs for certain classes of functions are *exponential* in the number of variables and cannot be constructed. To overcome this problem, *partitioned-ROBDDs* with intermediate variables [Narayan and *et al.* 1996] are used to further partition the functions. Since each partition of the function (a sub-function in itself) is represented by a BDD, partitioned ROBDDs present an initial circuit partitioning over which the decompositions are carried out.

The theory of *Dominators* [Yang et al. 1999] forms the basis for BDD-based decomposition. A BDD traversal (scan) is performed using a Depth-first Search (DFS) to identify these *dominators*. Dominators essentially correspond to structural features that indicate the convergence of positive and negative edges of the BDD at a particular node. After identifying these dominators, a *cut* is performed on the BDD at the appropriate levels which divides the BDDs into two parts and hence leads to a decomposition.

Table 1. Dominators and their corresponding decompositions

Type	BDD Structure	Decomposition
1	*1-dominator*	algebraic AND
2	*0-dominator*	algebraic OR
3	*x-dominator*	algebraic XNOR
4	*generalized dominator*	Boolean AND/OR
5	*generalized x-dominator*	Boolean XNOR
6	*cof. wrt. single node*	simple MUX
7	*cof. wrt. super node*	functional MUX

The decomposition engine performs a search for efficient BDD decompositions, from the most efficient (algebraic), to less efficient decompositions (Boolean). The engine first searches for a simple disjunctive (algebraic) decomposition. These decompositions are based on 1,0 and x-dominators, which are critical points on the BDD where simple decompositions (AND, OR and XOR decompositions, respectively) can be performed. A 1-dominator is a node v lies on all paths from the root node to the **constant 1** node, and so on. When this fails, the BDDs are decomposed using generalized dominators. As a last resort, the BDD is decomposed by cofactoring with respect to the top variable.

While decomposing, (refer to Fig. 1), the bound set variables (the variables above the cut) correspond to the *divisor* **D**. The *free set* constitutes the variables below the cut. **D** is referred to as a generalized dominator as it does not necessarily lead to an algebraic (disjoint) decomposition. All dangling solid edges are then tied to leaf node **1** in Fig. 1(b). The *quotient* of this division is obtained from **F** by setting the off-set of the divisor **D** as a don't care, and performing don't care minimization. The *restrict* [Coudert and Madre 1990] algorithm is used for this purpose. **F** is then minimized with this don't care in 1(c). The minimized function is the quotient **Q** of the division. This is a case of non-disjoint **Boolean** decomposition.

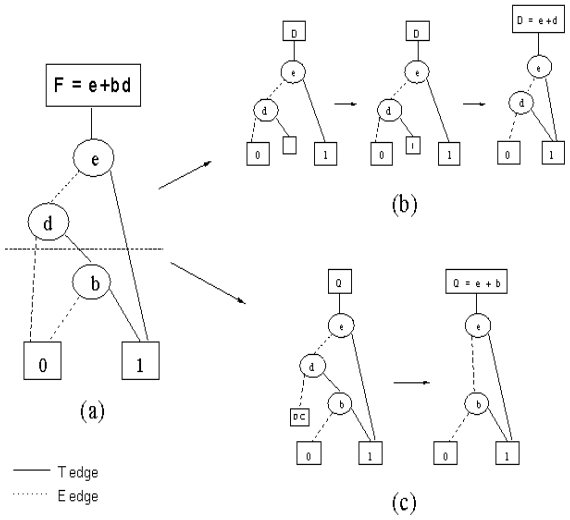

Fig. 1. Simple example of Boolean division

Prior to decomposition, network transformation procedures such as *sweep*, *eliminate* and *re-substitution* are incorporated as operations on BDDs. These operations are an essential part of logic synthesis systems, and help in restructuring and optimizing the Boolean network.

2.2 A case for using BDD-based decomposition for FPGA synthesis

An important feature of BDS is that the decomposition engine can identify both AND/OR and AND/XOR decompositions efficiently. XOR identification can significantly reduce implementation resources if efficient XOR implementations are available. Unlike standard cell designs, where decomposition significantly depends upon the type (functionality) of cells present in the technology library, decomposition for LUT-based FPGAs depends upon the number of inputs to, and not the functionality of, the decomposed nodes. These features, combined with the fact that BDD-based decomposition is very fast (by virtue of it being a by-product of simple graph traversal), makes it a natural choice for LUT-based FPGA synthesis.

To understand the capabilities of the BDD-based optimization system and its extension to LUT-FPGAs, we carried out comparative experiments for SIS and BDS and produced optimized networks to be mapped onto FPGAs. For technology mapping, we used the RASP [Cong et al. 1996] suite of tools (FlowMap, FlowSyn). The area measures of the circuits synthesized using SIS (using script.rugged) and Flowmap were compared against those generated by using BDS and Flowmap. These results in the last column of Table 2 indicate a 31% improvement in area over those obtained by using SIS and Flowmap. Furthermore, the circuits generated were on average, 8% faster than those obtained by SIS and Flowmap.

Table 2. Preliminary Experiments: # of LUTs

Ckt.	sc.rug+Flowmap	BDS+Flowmap
9sym	135	8
C499	66	70
C880	110	139
alu2	153	94
alu4	236	193
apex6	235	200
b9	51	42
clip	103	62
des	1291	983
duke2	163	187
f51m	32	14
rot	302	259
t481	401	5

It becomes clear that using the BDD-based logic optimization system (BDS), we were able to generate good decompositions that could be used for FPGA mapping. This experiment motivated us to further examine the following issue: How do we exploit BDD-based functional decomposition to directly and efficiently target k-input LUT-based FPGAs? We present a BDD-based FPGA logic synthesis system, called BDS-pga, that addresses the above issue. The synthesis flow of the BDS-pga framework is presented in Fig. 2, with the shaded blocks indicating the modifications/enhancements to the contemporary logic synthesis techniques for FPGA. The new paradigm incorporates the following steps, indicated in Fig. 2:

MFFC-based Eliminate. We introduce an eliminate procedure which collapses the network within its maximum fanout-free cone to create a cluster of variable partition size. This is done to cluster/collapse nodes into supernodes so as to induce good decompositions.

Decomposition. Two decomposition schemes which generate area-minimal k-feasible networks are presented: A greedy variable partitioning based iterative decomposition step has been implemented. A more intelligent area-driven variable partitioning based decomposition algorithm is also implemented.

Performance directed delay optimization. A fallout of area-driven decompositions is the increased circuit delay. Delay re-synthesis techniques are applied on the delay critical paths to reduce the circuit delay. Critical paths are collapsed into nodes, which are simplified and then re-decomposed.

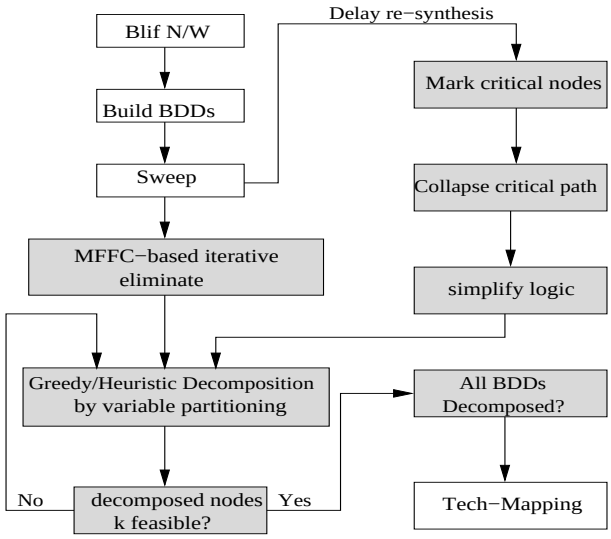

Fig. 2. Synthesis flow for BDS-pga

3. PARTIAL COLLAPSE

Due to the high complexity of logic minimization, a large circuit has to be clustered into blocks, where each block consists of a number of collapsed functions, or a *super-node*, such that the individual clusters can be decomposed relatively easily. In our paradigm, partitioning is used to reduce the number of nodes in the network. While this may increase the size of the node, care is taken so that this increase is managable.

Maximum fanout free cone (MFFC) [Chen et al. 1992] based partitioning [Cong et al. 1994], is a natural way of clustering nodes in a combinational Boolean network. Figure 3(b) illustrates the construction of a MFFC (from outputs to inputs), with the collapsed network shown in Fig. 3(c). We incorporate and improve this method by using the number of inputs to the cone as a collapsing constraint. Moreover, the number of nodes in the BDD representing the collapsed node can be used to limit its complexity. Furthermore, the collapsing is performed iteratively until the process converges.

3.1 MFFC based Iterative Eliminate Procedure

ALGORITHM 1. *MFFC-based eliminate*
require *network (logic network)*
begin
topologically order network nodes from PIs to POs;
*build BDDs for each node; /*partitioned ROBDD for the network*/*
identify MFFCs and identify eliminatable nodes;
while *number of collapsible nodes* $\neq$ *0*
while *traversed until end of linked list* $\neq$ *TRUE*
if *(node == collapsible)*
collapse node into its immediate fanout;
end while
update network and re-identify all eliminatable nodes;
end while
end

Algorithm 1 lists the main operations in the identification and subsequent collapse of nodes. We topologically order the network, traversing from primary inputs to primary outputs, and construct a BDD for each node in the network (*i.e.*, partitioned ROBDDs for the network are constructed). We identify MFFCs and mark the network nodes that can be collapsed into their immediate fanouts. These steps are iterated upon until no more collapsible nodes are found in the network.

We traverse a topologically ordered list of nodes from primary inputs to primary outputs to identify collapsible nodes. The arrow to the right of Fig. 3(a) indicates that this operation is performed from the primary inputs to the primary outputs. MFFCs are identified and collapsed, by allowing only the nodes with one fanout to be merged into their fanouts. No gate duplication is performed.

Notice that the node collapsing is performed on a Boolean level. Since a BDD is built for each node in the network, the node collapsing reduces to a variable substitute operation (the *bdd_compose* operator is used for this purpose). This collapsing operation can be controlled by user defined limits on the BDD size (number of nodes in the BDD-manager), and also on the number of inputs to a cluster.

Once the collapsible nodes have been collapsed into their respective fanouts, BDDs for the composite functions are constructed. The *collapsible* node is then erased from the network and the network is updated. The above iteration is performed again, and so on, as shown in Fig. 3(c).

Using an iterative MFFC-based iterative eliminate procedure further enhances BDS-pga's optimization capability for large circuits. Table 3 compares results of the new MFFC based eliminate algorithm with those of the original non-iterative implementation in BDS [Yang 2000]. A significant reduction, upto **40%**, in the number of network nodes (= no. of BDDs) to be decomposed, has been achieved. While the number of BDDs reduces, the size of BDDs increases, though not significantly. Moreover, the growth of the BDD size (because of collapsing) is controlled by user defined limits. As a result, we are able to induce good decompositions by searching a larger space (corresponding to more variables in the BDDs), and also

have to handle fewer BDDs for decomposition. There is no significant increase in execution time for the new *eliminate* algorithm.

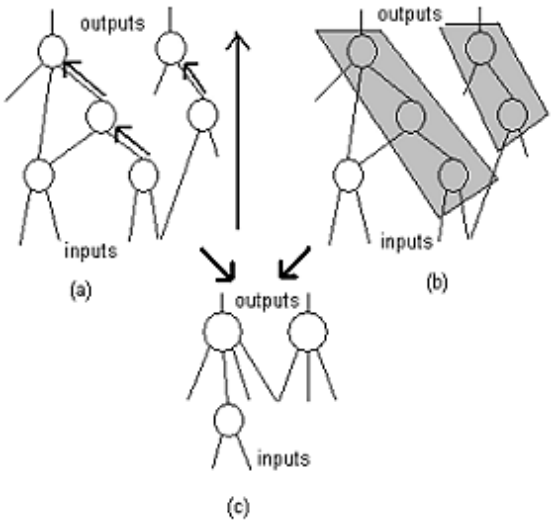

Fig. 3. (a) new iterative collapse routine and (b), non-iterative MFFC-based collapse, (c) the final network

Table 3. Number of Local BDDs to be decomposed for BDS & BDS-pga

Circuit	BDS #BDDs	BDS-pga #BDDs	% Reduction
C1355	98	78	20.4
C1908	108	58	46.3
C3540	343	200	41.7
C432	65	41	37
C499	60	46	23.3
C5315	393	326	17
C6288	727	524	28
C7552	506	487	4
C880	136	80	41.2
dalu	254	223	12.2
des	539	294	45.5
mult32	2494	1066	57.3
pair	446	274	38.6
Total	6169	3697	40

4. GREEDY AND HEURISTIC VARIABLE PARTITIONING BASED DECOMPOSITION

The *MFFC-based eliminate* heuristic clusters the nodes in order to induce good decompositions. We aim to decompose these collapsed *supernodes* into a minimum

number of k-feasible nodes. First, a greedy variable partitioning technique is introduced. A variable swapping based heuristic decomposition technique which uses an area cost function, is described later.

Decomposition is the process of breaking a function and representing it using sub-functions with smaller fanin. Given a function, f(X), which depends on n variables, X = $x_1, x_2, \ldots x_n$, we can represent f(X) as a new composition function $g(g_1(X_b), \ldots g_m(X_b), X_f)$. X_b is the **bound set** (**BS** henceforth) and X_f is the **free set** (**FS**). Variable partitioning involves computation of the bound and the free sets. This is equivalent to performing a cut on the BDD, using the portion *above* the cut as the bound set and those variables *below* the cut as the free set. The number of edges intersecting the cut line represents the number of equivalence classes in Roth-Karp decomposition.

Ashenhurst [Ashenhurst 1959] detailed a procedure for finding a set of variables which caused a simple disjunctive decomposition of a function. Roth and Karp [Roth and Karp 1962] proposed a memory efficient algorithm to perform Ashenhurst decompositions. Lai et. al. [Lai et al. 1993] described a faster Roth-Karp implementation, using the EVBDD. Stanion and Sechen's method [Stanion and Sechen 1995] implicitly enumerates all of the cut sets in the BDD to determine the bound set and free set. They demonstrate that any cut in a BDD can induce a certain decomposition of the function. Using BDDs, Jiang [Jiang et al. 1997] proposed to solve the variable partitioning problem by selecting lambda set variables in the Roth-Karp decomposition for better LUT utilization. SIS [Sentovich and *et al.* 1992] employs a SOP-based functional decomposition method which greedily selects a non-trivial decomposition of *bound set* size k. In contrast, Eckl [Legl et al. 1996] proposed a variable partitioning heuristic which selects a **good** *bound set* size, from a number of bound set sizes.

An efficient variable partitioning heuristic must choose a bound and free set such that the smallest number of equivalent classes and hence sub-functions, result from the decomposition. In BDD terms, this is equivalent to enumerating all of the cuts possible on the DAG. There are however, 2^n-1 cuts possible for a function with $S_f = n$. Thus, for computational efficiency, only a sub-set of cuts, say, a cut with $\leq k$ variables in the bound set, are considered. We describe by illustration, the decomposition of a function using BDS, the greedy heuristic (BDS-pga) and, the area-minimal heuristic (BDS-pga) in sub-sections 4.1, 4.2 and 4.3 respectively. We also experimentally compare the area results using the greedy and area-minimal heuristic decomposition algorithms in sub-section 4.4.

4.1 Functional Decomposition using BDS

Consider the ROBDD of the function $f(a, b, c, d, e, f) = ab + cd + ad + be + e + f$, shown in Fig. 4(a). Assume a 3 input LUT ($k = 3$) target technology. BDS decomposes BDDs by *searching* for the most *efficient* decomposition. First, it attempts to perform the algebraic decompositions closest to the center. In the absence of algebraic decompositions, Boolean decompositions are performed at that level which yields the least cost, according to the following cost function [Yang 2000],

$$cost = \alpha N + (1 - \alpha)V \tag{1}$$

where N is the ratio of the sum of BDD nodes in the decomposed functions to number of BDD nodes in the original function; V is the ratio of number of shared variables to the number of variables in the original function; $\alpha = 0.5$.

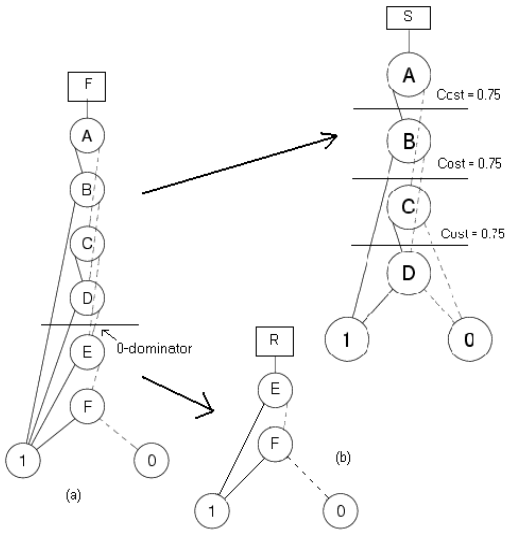

Fig. 4. Decomposition with BDS: Algebraic decomposition (disjunctive) (a), subtrahend (S) and remainder (R) after 1^{st} cut (b)

The BDS decomposition engine scans the BDD and finds a *0-dominator* located at node E, resulting in an algebraic disjunctive decomposition. The resulting sub-functions are $R = e + f$ and $S = ab + cd + ad$, where $F = S + R$. BDS now decomposes $S = ab + cd + ad$ (Fig. 4(b)). In Fig. 4(b), the decomposition cost (eqn. 1) for each level on S, has been marked. One notices that all decompositions invoke equal cost, i.e. resulting sub-functions from each decomposition would be either $\mathcal{S}_{\mathcal{R}} = 2$ and $\mathcal{S}_{\mathcal{S}} = 3$, or vice-versa. BDS decomposes S at the center, below B. The resulting circuit, when mapped (using Flowmap) consists of 4 3-LUTs, with a depth of 2.

4.2 Greedy Variable Partitioning Algorithm

The greedy decomposition engine decomposes the BDD for each internal node into two sub-functions. k-infeasible sub-functions are recursively decomposed until k-feasibility is achieved. In order to carry out a greedy k-feasible decomposition, the algorithm marks the levels at which decomposition can be performed by visiting each BDD node in a depth first manner. The greedy algorithm allows decompositions at only those levels which are multiples of k. If $k = 5$, decompositions can be performed at levels 0,5,.... Furthermore, the support size of the BDD (number of variables) determines the level at which the *greedy* decomposition is performed. For a BDD with support size $\leq k$, no decomposition is performed and it is preserved. When BDD support size $\leq 2k$, $2k + 1$, it is decomposed with *bound set* size $= k$. For a BDD with support size $\leq 3k$, $3k + 1$, it can be decomposed with *bound set*

size = $\frac{no\_levels}{2}$ (*cutting* at the center) or $n \times k$, where n = 1,2.... For BDDs with support $\geq 3k+2$, decomposition is performed at the center i.e. $\frac{no\_levels}{2}$.

The next step involves the decomposition of a function (say $\mathcal{F}$) into two sub-functions $\mathcal{G}$ and $\mathcal{H}$. Boolean decompositions result in sub-functions with shared variables, i.e. non-disjoint decomposition. The *sum* of the fanin of each decomposed sub-function may greatly exceed the original fanin, leading to an inefficient decomposition. The cost function,

$$cost = S_{\mathcal{G}} + S_{\mathcal{H}} \leq S_{\mathcal{F}} + S_{\mathcal{F}} \times B \tag{2}$$

where $\mathcal{B} = 0.5$, limits such inefficient decompositions. If a decomposition fails this test, the function is decomposed by cofactoring w.r.t the top variable. Experimentation has shown that Boolean decompositions which result in large sub-function fanin lead to a larger number of subsequent decompositions, and hence, area.

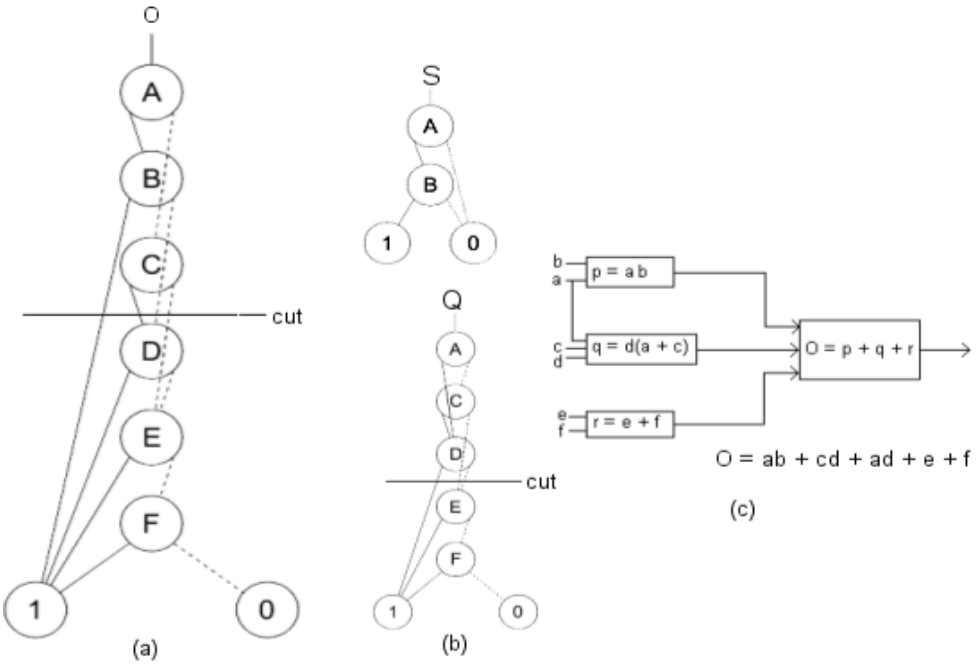

Fig. 5. Greedy decomposition with $k = 3$: Initial *cut* for $S_O = 6$ (a), subtrahend (S) and remainder (Q) after 1^{st} cut (b) and, final decomposition (c)

Decomposition example

Referring to Fig. 5(a) and assuming $k = 3$, a cut is **greedily** performed at level 3. The *bound set* size is 3, ensuring that at least one sub-function of the decomposition is 3-feasible. The decomposition, shown in Fig. 5(b), is Boolean, resulting in a subtrahend $S = ab$ ($\mathcal{S}_S$=$\|(a,b)\| = 2$), and a remainder $Q = cd + ad + e + f$ ($\mathcal{S}_Q$ =$\|(a,c,d,e,f)\| = 5$). The 3-infeasible function (Q) is further decomposed

algebraically with the *0-dominator* at node e, and the final decomposition is shown in Fig. 5(c). The result is a network with 4 3-LUTs and a topological depth of 2.

4.3 Heuristic Variable Partitioning for Area

The goal of the heuristic variable partitioning decomposition scheme is to decompose a k-infeasible function by re-ordering the variables around a fixed *partition* or *cut* k (i.e. a fixed *bound set*), so that a minimum number of k-feasible sub-functions result from the decomposition. This is achieved by *selectively swapping* a pair of variables, one each from the *bound set* and the *free set*. After each swap, an Area Cost Function (ACF) is evaluated to determine the number of sub-functions that would be required for a decomposition at k, with the current variable order. The variable ordering which evaluates to the smallest ACF is chosen.

A FS variable directly connected to an edge in the cut is called a *cut-node*. In Fig. 6, the BS variables are A, B, and the FS variables are C, D. C and *0* are *cut-nodes*. The cardinality of the *cut-node-set*, n, is equal to the number of distinct columns in the (Ashenhurst/Roth-Karp) decomposition chart, and hence the number of encoding bits (variables) required. Thus, the number of sub-functions resulting from a decomposition can be determined by identifying the *cut-node-set* of the BDD. The number of variables (factored sub-functions) required to encode a given function is $\lceil log_2(n) \rceil$. For Fig. 6, the ACF is 1.

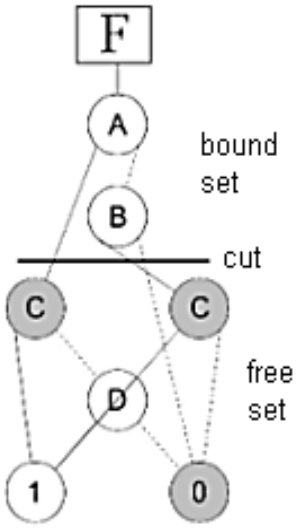

Fig. 6. Illustration of a cut, cut-nodes, free sets and bound sets

Each iteration of the heuristic involves two steps. First, the BDD is scanned using a depth first search, in order to identify those variables which will most likely lead to a reduction in n when swapped. The second step involves the calculation of the Area Cost Function (ACF) after the swap. The swapped variables are *locked*, i.e. they cannot be swapped until all other variables have been swapped in succeeding iterations. This process continues until all variables are locked or until n improves.

If n does not decrease by the time all variables are locked, we re-initiate the swapping process for one more set of iterations. Thus, a maximum of $\| BS \| + \| FS \|$ swaps are required to get a good decomposition. After the swaps have taken place, the variable order that has the least ACF is selected.

Identifying the best swap variables: The choice of the FS and BS variable to be swapped is determined by the effect that this swap would have on the cardinality of the *cut-node-set*. Intuitively, a FS variable which lies on a large number of paths from the root to the terminals has a lot of incoming edges. Clearly, this variable appears in a large number of minterms of the function. Swapping this variable with any BS variable would, in effect, lead to a reduction in the size of the *cut-set*.

DEFINITION 1. *A* **FS swap variable** *is an unswapped variable with the most number of incident edges which resides closest to the cut.*

Which variable in the BS should this FS variable in question be swapped with? It can be inferred that a BS variable that has fewer incident edges appears in fewer minterms of the function and would be a good candidate for a swap. This is because the dependence of the function on this variable is not significant. Exchanging a node with a large number of incident edges results in a smaller *cut-set*. This corresponds to a lower column multiplicity in the (Ashenhurst/Roth-Karp) decomposition chart and leads to fewer encoding variables in the decomposition.

DEFINITION 2. *A* **BS swap variable** *is an unswapped variable with the least number of incident edges which resides closest to the cut.*

$log_2(n)$ is used in Roth-Karp decomposition to determine the number of encoding variables. This measure is also found to be highly suitable for a BDD-based decomposition environment. The Area Cost Function can then be defined thus:

$$ACF = \lceil log_2(n) \rceil \tag{3}$$

where n is the cardinality of the *cut-set*. This cost function is an indication of the number of sub-functions that may result from the decomposition.

According to the heuristic, a cut is placed after the third variable (level) in the BDD (Fig. 7(a)), so that BS = *a, b, c* and FS = *d, e, f*. The BDD in Fig. 7(a) represents the same function as that in Fig. 4. It has not been reduced, to reflect the swapping mechanism of the decomposition engine. The shaded nodes are the *cut_set* nodes, and in Fig. 7(a), $\|cut\_set\| = 3$. Thus, ACF = $\lceil log_2(3) \rceil = 2$, which indicates that a minimum of 2 sub-functions would result from this decomposition. Now the best BS and FS variables to be swapped are determined. The FS variable with the most number of incident edges and the BS variable with the least number of incident edges are swapped. In the event that more than one variable has the same number of incident edges, the BS and FS variables closest to the cut are swapped. As shown in Table 4, node *c* appears in the fewest number of minterms (10) of the function in Fig. 7(a), of all the BS variables, and has the fewest number of incident edges. Furthermore, the FS node with the most incident edges, *e*, appears in the most minterms of all **FS** variables. In Fig. 7(a), *e* in the FS and *c* in the BS are chosen to be swapped.

After swapping, the BDD is shown in Fig. 7(b), and ACF is calculated to be 1. Nodes *e* and *c* are now locked and cannot be swapped. Similarly, nodes *f* and *b*

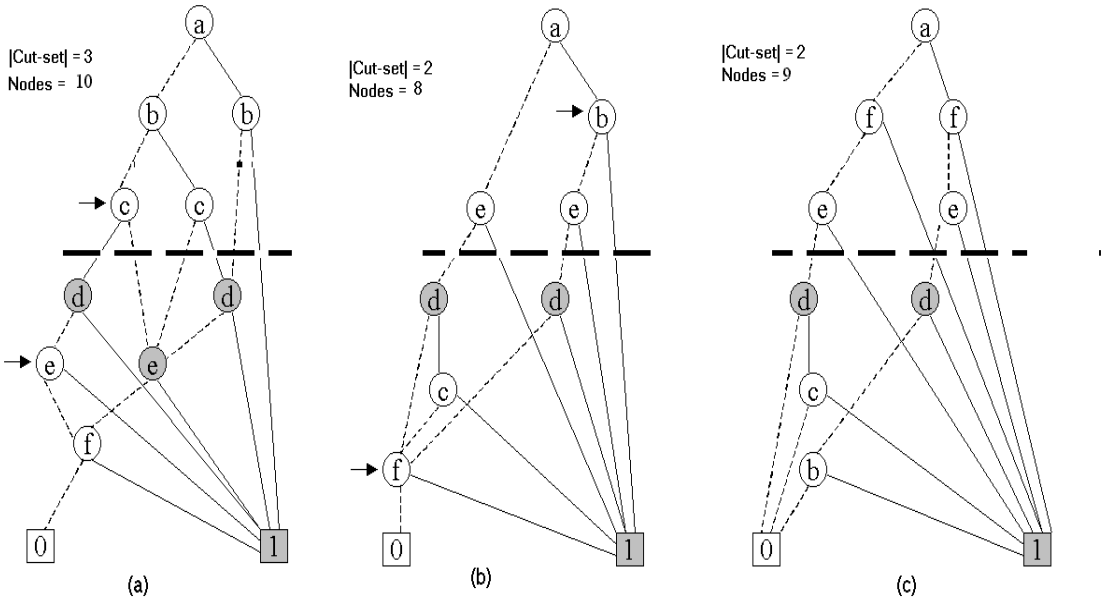

Fig. 7. Variable swapping illustration. Variables indicated by arrows are swapped

Table 4. Minterm table for variables before 1^{st} swap

Variable	Minterms	Incident edges
a	14	–
b	14	2
c	**10**	**2**
d	9	3
e	**9**	**4**
f	5	2

are swapped in the next iteration. Here, referring to Table 5, BS node b appears in the fewest minterms, while FS node f appears in the most minterms while having the most incident edges. The new BDD is shown in Fig. 7(c), where ACF = 1. The first two variable-pair swaps have helped identify a sub-function $e + f$. We choose Fig. 7(c) as the decomposition candidate and perform the decomposition as described in Section 2, resulting in: $f(a, b, c, d, e, f) = D + H$, where $D = f + f^{'}e$ is the Boolean divisor, and $H = ab + ad + cd$ is the quotient. D can be implemented in a 3-LUT, while H needs to be decomposed further. Using the above method, H is further decomposed and the final decomposition is shown in Fig. 8.

Using the area-driven variable partitioning heuristic, we were able to map this function onto 3 3-LUTs with a depth of 3. The ACF in this case was reduced from 2 to 1 in two iterations. The SIS [Sentovich and *et al.* 1992] platform using the *xl_k_decomp -n 3* decomposition scheme mapped this function onto 7 LUTs with a depth of 3. The algorithm is formally presented in 2.

Table 5. Minterm table for variables after 1^{st} swap

Variable	Minterms	Incident edges
a	8	–
b	**4**	**1**
e	6	2
d	2	2
c	1	1
f	**2**	**3**

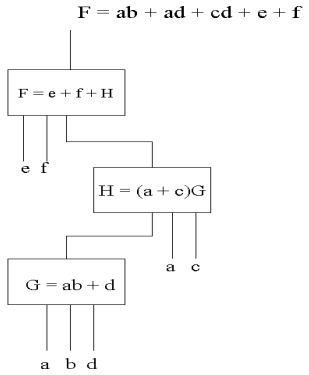

Fig. 8. Final result: Decomposition using swapping

ALGORITHM 2. *Heuristic variable partitioning (decomposition) for Area*
require *logic network, feasibility factor k*
produce *k-feasible optimized, decomposed network*
begin
MFFC-based iterative eliminate;
for *each BDD in the network* **do**
place a cut across a BDD such that bound set size = k or (no. of levels)/2;
repeat
compute n = cardinality of cut-node-set;
Select FS swap var as one with most incident edges;
Select BS swap var as one with least incident edges;
Swap the variables and lock them;
compute $ACF = \lceil log_2(n) \rceil$;
until *(all vars swapped or n improves)*
if *n does not decrease* **then**
repeat the above process at a different cut across the BDD;
end if
end for
end

4.4 Comparative Analysis of BDS, Greedy and Area-Minimal Decomposition

We applied three decomposition heuristics to the function in Fig. 4(a). In the *first* experiment, BDS decomposition is followed by technology mapping with Flowmap, wherein we achieved a circuit with 4 3-LUTs. *Secondly*, our greedy decomposition followed by technology mapping generated a circuit of 4 3-LUTs. *Lastly*, our area-minimal heuristic variable partitioning algorithm followed by technology mapping with Flowmap, uses only 3 3-LUTs. Evaluating eqn. 1 for the decompositions performed on Fig. 4(a), Fig. 5(a) and Fig. 7(c) yields the following decomposition costs: $cost_{BDS} = 0.5$, $cost_{greedyBDS-pga} = 0.67$ and, $cost_{area-minimalBDS-pga} = 0.33$. To perform a thorough comparison of the greedy and area-minimal algorithms, 14 MCNC benchmarks were first optimized using the greedy and area-minimal algorithms and then mapped onto 5-LUT based architectures using Flowmap. The results of the synthesis are shown in Table 6.

Table 6. Comparison of Greedy & Area-Minimal Heuristic

Ckt.	Greedy		Area-minimal Heuristic	
	LUT	delay	LUT	delay
5xp1	16	2	14	2
9sym	7	3	7	3
9symml	7	3	7	3
C499	154	6	64	4
C880	114	9	108	8
alu2	50	4	41	4
apex6	217	6	186	4
apex7	86	4	71	3
b9	47	5	40	3
count	34	4	26	5
misex1	14	2	14	2
rot	263	9	218	9
C1908	204	12	119	7
C5315	460	11	447	7
Total	**1673**	80	**1362**	64

From the above discussion, and from results shown in Table 6, we arrive at the following conclusions:

(1) The BDS decomposition scheme has no provision to account for an eventual implementation of the decomposed functions on k-LUTs. Our greedy implementation, on the other hand, restricts the decompositions to those with a bound set size of k. While this may lead to inefficient Boolean decompositions, the operation is very fast due to the small search space for a decomposition. This method generated more area-efficient circuits than those synthesized by BDS. In the area-driven variable partitioning heuristic, we restrict the bound set size to k. Using a novel variable swapping method coupled with an efficient

cost function (ACF), we generate a BDD which results in smaller k-LUT based implementations than the previous methods.

(2) The BDS implementation cost (Eqn.1) is measured on an optimized BDD [Yang 2000]. However, when performing the area-minimal decomposition, it is noticed that while the BDD size increases from Fig. 7(b) to Fig. 7(c), the implementation cost (using eqn.1) is reduced from 0.521 to 0.33. This indicates that a non-reduced BDD (initial representation or final candidate) with more *nodes* may **not necessarily** result in a poorer decomposition, especially for FPGAs.

(3) Evaluation of the cost function in BDS is carried out on optimized BDDs (achieved by time-intensive variable reordering) and involves decompositions which have to be carried out at each level, since it depends on the relative sizes of the resulting sub-functions. This is a very time consuming procedure. On the other hand, the ACF determines the dependence of variables in the BS and FS, and the eventual decomposition cost, by a simple BDD traversal. It has a time complexity of O(n), where n is the number of levels in the BDD.

(4) The results in Table 6 indicate that synthesis with the area-minimal heuristic consistently resulted in smaller circuits (on average, 19% smaller), and were on average 20% faster. This is due in large part to the refinement provided by the area-minimal heuristic *vis-a-vis* the greedy decomposition. By using an intelligent variable swapping technique, the heuristic takes the greedy algorithm a step further. All results shown henceforth use the area-minimal heuristic decomposition technique.

5. DELAY RE-SYNTHESIS

This section presents delay re-synthesis techniques. After the first optimization pass, re-synthesis techniques can be applied to improve the circuit delay. Speed-up [Singh et al. 1988], was one of the first attempts at reducing delay in combinational networks by collapsing critical sub-networks and re-decomposing them. Subsequently, a number of similar performance directed re-synthesis techniques for FPGAs [Murgai et al. 1995][Lai et al. 1993][Legl et al. 1996] [Cong and Ding 1993] have been studied.

5.1 The delay algorithm

The input to our algorithm is a circuit that has been optimized for area in the first pass of BDS-pga. Starting at the primary outputs, it identifies critical paths, depending on the depth of the transitive fanin.[1] Starting from the primary inputs, nodes on the critical path(s) are collapsed only into corresponding nodes on the critical path (This is shown in Fig. 9(a) and (b)). This is done because otherwise indiscriminate collapsing can potentially increase the fanin of the resulting nodes and increase the final implementation cost after technology mapping.

The collapse operation reduces the depth of the primary output node. In achieving this, the collapsed node may become k-infeasible. To generate k-feasible nodes, we first simplify the collapsed node using the heuristic two-level logic minimizer

[1]While false path identification has not yet been programmed for **BDS-pga**, it can be easily incorporated into our synthesis framework.

Espresso [Brayton and *et al.* 1984]. The simplified node is then re-decomposed, based on the heuristic variable partitioning-based decomposition algorithm.

Determining Critical Path(s)

After the area optimized network is parsed and assigned depths, $\mathcal{L}$ is determined. Starting at the primary output(s) with the largest depth(s), we recursively traverse back towards the primary inputs. Figure 9(a) illustrates the critical path from primary output *PO1*. Nodes are assigned character names; integers represent the depths at inputs/outputs. To determine $\mathcal{L}_\Upsilon$, *all* primary outputs with largest depth are considered.

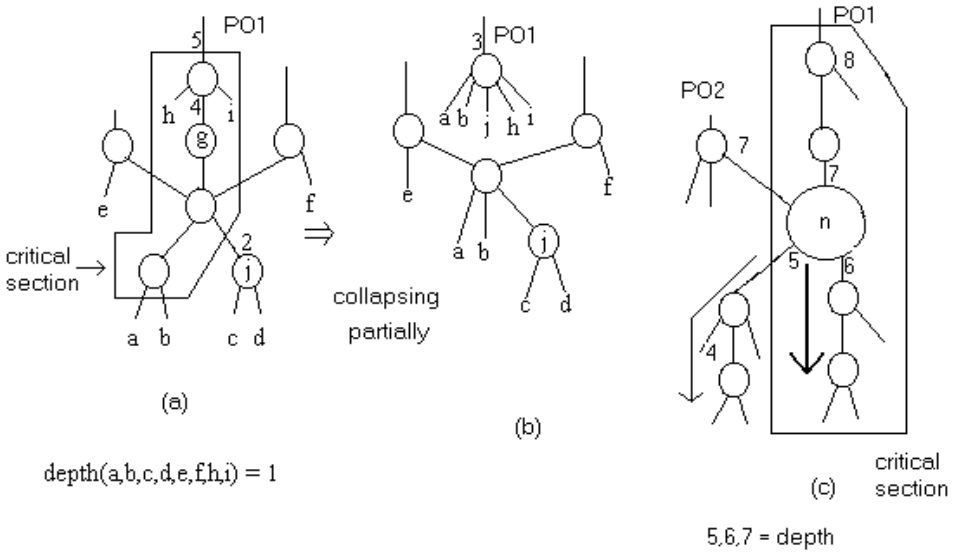

Fig. 9. Determining critical path(s) (a), collapsing (b) and (c) finding sub-critical paths.

All ϵ-critical Primary Outputs (PO) are used to determine $\mathcal{L}_\epsilon$. While determining $\mathcal{L}_\Upsilon$, we may encounter a node n that has been marked as critical from a previous iteration of the algorithm. Instead of traversing the ϵ-critical paths from this node, we follow the critical path $\mathcal{L}$, thus avoiding the *sub-critical* paths in proximity to the critical ones in question. For example, Fig. 9(c) depicts, with the dark arrow, the critical path marked in a *previous* iteration. The thinner arrow in Fig. 9(c) indicates the sub-critical path that must be ideally traversed and marked. We resolve this issue either by traversing the next more critical path $L_{\epsilon-eps}$ from node n, or by exiting from the routine at this point.

Level reduction by Partial Collapse

After determining critical paths, the nodes are partially collapsed into their critical fanouts. The partially collapsed circuit is shown in Fig. 9(b). Nodes are collapsed into those that only lie on the critical path. This necessitates the duplication of

multiple-fanout nodes. In Fig. 9(a), the depth of node **PO1** is 5. After partial collapse, in Fig. 9(b), the largest depth at the input of PO1 is now 1. The depth at the output of node **PO1** is 2.

Simplification of collapsed node using Espresso
After the collapse phase, the *supernode(s)* are simplified using the two-level minimizer, Espresso. Logic simplification aims to reduce the fanin (support size) of the *supernode*, which aids in inducing a more delay efficient decomposition when using the variable partitioning decomposition described earlier. BDS does not have an efficient logic minimization capability using don't-cares. While the *BDD_restrict* [Bryant 1986] algorithm has been proposed for this purpose [Yang et al. 1999], the results have not been satisfactory. This necessitates the use of an alternative logic simplification solution. Due to the relatively small size of the *supernodes*, we can use two-level logic minimization techniques, like Espresso, to generate a simplified *supernode* with fewer minterms. The resulting node, if k-feasible, is preserved, resulting in a decrease in the depth of the critical path.

Re-decomposition
After simplification, the *simplified* k-infeasible nodes are re-decomposed using the heuristic variable partitioning approach. The aim is to produce a decomposed subnetwork with the least possible delay **constrained** by the number of k-feasible sub-functions.

6. RESULTS

We implemented our new approach and compared our experimental results with other synthesis approaches, SIS and BoolMap [Legl et al. 1996]. Experiments have also been carried out with a commercial tool, *Quartus*, by *Altera Corp.* on some of the larger circuits of the ISCAS benchmark suite. In this section, we will first compare BDS-pga's area/delay results with those of Boolmap and SIS, and then present the comparative results of BDS-pga with those of Quartus.

6.1 Experimental Results: Area Optimization

The FPGA synthesis experiments were conducted in two phases. The **first** phase is multi-level logic optimization. For this purpose, we experimented with the i) the optimization scripts of Boolmap, ii) optimization scripts of SIS, and iii) the proposed techniques of **BDS-pga**. The **second** phase is FPGA technology mapping. The RASP script [Cong et al. 1996], which includes Flowmap [Cong and Ding 1994] and FlowSYN [Cong and Ding 1993], has been utilized for this purpose. The value of k for these experiments was taken to be 5.

The first set of experiments were conducted using the synthesis system TOS (Boolmap) [Legl et al. 1996]. The area was measured in terms of the number of LUTs, and the delay in terms of the depth of the critical path. The CPU time required to synthesize the designs was recorded. The "smaller" MCNC benchmarks were first fully collapsed and then the optimization script *mmap_h_a_5.scr* [Eckl et al. 1996] [Legl et al. 1996] was used to generate decomposed k-feasible networks. The "larger" ISCAS'89 benchmarks which could not be fully collapsed, were initially decomposed using the specified *smap_h_a_5.scr* script, followed by the

reduce_depth procedure. Then the multi-output decomposition and mapping script *mmap_h_a_5.scr* was used to generate an optimized mapped network. The results are depicted in Table 7 under the "Boolmap-Area" column.

The second set of experiments were conducted with SIS as follows. The benchmark circuits were first optimized using SIS *script.rugged*. These optimized networks were re-synthesized for FPGAs using *script.fpga*. SIS *script.fpga* performs partial collapsing, simplification, logic decomposition, and technology mapping using a binate covering algorithm. The results are shown in Table 7, in the "script.rugged+script.fpga" column.[2].

Table 7. Area Synthesis Results for BoolMap-Area, SIS & BDS-pga

Ckt.	BoolMap-Area			script.rugged + script.fpga			BDS-pga+Flowmap		
	LUT	delay	time (s)	LUT	delay	time (s)	LUT	delay	time (s)
5xp1	13	2	0.59	23	4	4.4	14	3	1.06
9sym	8	3	0.41	7	3	147	7	3	1.2
9symm	8	3	0.28	7	3	32	7	3	1.28
C499	98	5	40.4	70	6	50	70	5	1.2
C880	121	11	5.4	88	13	42	103	8	1.99
alu2	46	5	126	96	18	71	41	4	4.09
alu4	150	11	101	236	11	1762	190	7	25.2
apex6	152	6	12.5	166	8	20	186	4	7.81
apex7	61	5	8	58	6	35	71	3	3.28
b9	43	3	0.4	34	4	2	40	3	2.31
clip	15	2	0.6	58	9	40	30	4	12.54
count	31	7	1.6	31	5	1.4	26	5	1.9
des	1462	9	86.8	949	16	340	909	4	55.7
duke2	187	8	199	144	7	33	173	8	6.9
misex	13	2	.21	13	4	0.7	14	2	7.1
rd84	10	2	2	10	3	85	13	3	3.32
rot	347	19	86.2	180	13	28.2	223	10	10.61
vg2	31	4	9.32	24	5	4.4	12	3	5
z4ml	5	2	0.2	5	2	0.9	5	2	1.39
t481	5	3	30.11	121	13	63	5	2	2.6
C1355	80	6	4.85	70	6	50.2	64	4	5.19
C1908	130	12	40.49	95	15	30	123	7	7.26
C5315	545	13	37.22	374	18	73	435	7	23
Total	3561	143	1098.4	2859	163	2915.2	**2761**	**95**	191.75
Norm	1.24	0.88	0.38	1	1	1	**0.97**	**0.58**	**0.066**

Finally, we used **BDS-pga** to generate optimized circuits for FPGAs using the

[2]The use of script.rugged before script.fpga has been suggested to the authors. Also, the authors have noted significant area improvements using "script.rugged+script.fpga" over those obtained solely by "script.fpga". The authors also wish to mention that, to the best of their knowledge, these results have not been referred to in literature.

techniques described in this paper. The circuits were *partially collapsed* to produce clusters of supernodes using the proposed MFFC-based approach. Logic decomposition was carried out using the **heuristic variable partitioning technique** to produce a k-feasible network. Following this logic optimization phase, technology mapping was carried out using the RASP [Cong et al. 1996] suite of tools. The results are shown in Table 7 under the "BDS-pga+Flowmap" column.

The RASP script has been modified to preserve the nature of the decompositions obtained by **BDS-pga**. RASP, as a pre-processing step, decomposes the nodes of the network into 2-feasible AND/OR gates. This could conceivably undo the optimization characteristics of **BDS-pga**. Furthermore, **BDS-pga** has the capability of identifying AND/XNOR decompositions, which would be undone by RASP, as it decomposes all complex gates into a network of AND-OR gates. To preserve these decompositions, the technology decomposition routine has been eliminated.

Table 7 shows results for experiments conducted for 25 MCNC and ISCAS benchmarks. Comparison of Area results for BDS-pga in Table 7 to those of BDS and SIS (script.rugged) in Table 1, indicates a 11.8 % area improvement of BDS-pga over BDS. The results demonstrate that the optimization provided by BDS-pga is significant. Overall **BDS-pga** generates circuits with fewer LUTs that SIS and Boolmap. Also, note that for almost all benchmarks, the delay (topological depth) of the circuit is smallest when using **BDS-pga**. CPU times reported for **BDS-pga** are an order of magnitude smaller than both SIS and Boolmap for most benchmarks. SIS was unable to optimize C7552, while Boolmap failed to synthesize C3450 in acceptable time (almost 5 hrs). **BDS-pga** was able to synthesize both circuits within a matter of seconds (refer to Table 9, under the Area heading).

6.2 Experimental Results: Delay Optimization

To compare the delay optimization power of **BDS-pga** with that of SIS and Boolmap, we synthesized the circuits within the respective delay-synthesis paradigms of corresponding tool suites. First, using SIS, we used *script.rugged* followed by *script.delay* to generate delay-optimal circuits. These circuits were then decomposed using *script.fpga* to generate delay-optimized mapped circuits. Similarly, for Boolmap, we used the prescribed delay-optimization scripts [Eckl et al. 1996] to generate delay-optimized circuits. For **BDS-pga**, we used the area-optimized circuits from the previous experiments and performed the proposed re-synthesis for the delay critical path(s). The delay reduction algorithm proposed in this paper was invoked in the second optimization pass of BDS-pga. This routine partially collapses the longest path, simplifies the collapsed nodes with don't cares and re-decomposes the whole circuit. The results are presented in Table 8.

It can be noticed that **BDS-pga** outperforms SIS for delay-optimization. It also compares favourably with Boolmap. Moreover, both SIS and Boolmap were unable to optimize a few circuits for delay. Refer to Table 9, under the delay heading, for a list of benchmarks un-synthesizable by Boolmap and SIS. However, *BDS-pga* was able to perform delay re-synthesis for all the examined benchmarks. It may be further noticed that comparing between Table 7 and Table 8, the delay resynthesis step in **BDS-pga** almost always improves the delay. In no case did delay resynthesis increase the delay of the network. Also, it can be noted that often the delay improvement is achieved not at the expense of area. Clearly, the circuits

Table 8. Delay Re-synthesis for BoolMap-Delay, SIS & BDS-pga

Ckt.	BoolMap-d			script.rugged + script.delay + script.fpga			BDS-pga+Flowmap	
	LUT	delay	CPU(s)	LUT	delay	CPU(s)	LUT	delay
5xp1	13	2	0.86	23	4	6	15	2
9sym	7	3	0.44	7	3	21	7	3
9symml	7	3	0.32	7	3	18.2	7	3
C499	102	4	107.1	62	6	54	64	4
C880	134	8	13.3	129	13	69	108	8
alu2	50	5	62.7	112	14	61	41	4
apex6	188	4	12.5	191	3	36	186	4
apex7	78	3	8	62	5	7.2	71	3
b9	41	3	0.4	34	4	2	40	3
clip	15	2	0.6	58	9	40	30	4
count	42	2	1.6	38	5	4.7	26	5
duke2	192	5	199	172	8	33	169	7
misex1	15	2	0.2	15	4	2.1	14	2
rd84	10	2	2	10	3	27	13	3
rot	244	6	-	232	11	62	218	9
vg2	30	4	9.32	29	6	1.3	12	3
z4ml	5	2	0.2	5	2	1.3	5	2
t481	5	3	30.11	119	12	432.7	5	2
C1355	98	5	10.7	62	6	53	65	4
C1908	137	7	97	135	12	85	119	7
C5315	672	9	154	546	17	226.3	447	7
C7552	729	9	102.6	544	20	200	631	12
Total	**2814**	103	812.95	2592	170	1442.8	**2293**	93
Norm	**1.09**	.61	.56	1	1	1	**0.88**	.55

synthesized by our approach deliver a better area/performance trade-off.

6.3 Comparative Results for BDS-pga and Quartus

From our previous experiments, we have noticed that for the larger circuits of the MCNC and ISCAS benchmark suite, BDS-pga outperforms Boolmap and SIS for area and delay synthesis in most cases. This motivated us to compare BDS-pga's results with a commercial tool, *Quartus*, by *Altera Corp.* The comparative results for area minimization for FPGA and Quartus are presented in Table 10. For these experiments the value of k, the feasibility factor, was kept to 4, consistent with *Altera's* Apex series of devices. As before, in order to preserve BDS-pga's k-feasible decomposition, we had disabled the feature of Flowmap that decomposes the network into two-input AND/OR gates.

From the above experiments, it can be noted that for benchmarks *C1355, C1908, C5315* and *C7522*, BDS-pga produces better area optimized net-lists as compared with Quartus. Moreover, the delay of all the the circuits synthesized by using

Table 9. Un-synthesizable circuits for BoolMap and SIS

Ckt.	BoolMap-Area		script.rugged + script.fpga		BDS-pga+Flowmap	
	LUT	delay	LUT	delay	LUT	delay
Area						
C3540	-	-	278	19	311	15
C7552	696	13	-	-	631	12
Delay						
alu4*	264	7	-	-	190	7
des*	594	3	-	-	909	4
C3540*	-	-	-	-	324	13

Table 10. Area Synthesis for Quartus & BDS-pga. K=4.

Ckt.	Quartus			BDS-pga+Flowmap		
	LUT	delay	CPU time	LUT	delay	CPU time
des	1501	9	7.26 mins	1592	5	50 sec
C1355	98	6	44 sec	94	4	9.1 sec
C1908	223	8	1.15 mins	162	8	3.25 sec
C5315	640	14	4.51 mins	582	8	14 sec
C3540	426	18	2.56 mins	556	11	28.1 sec
C7522	788	15	6.20 mins	769	11	19 sec

BDS-pga and flowmap is better as compared with those synthesized by Quartus. Furthermore, the CPU times for "BDS-pga + flowmap" combine are orders of magnitude smaller than those required by Quartus[3].

For delay resynthesis results shown in Table 11, for no benchmark circuit does "BDS-pga + flowmap" produce worse delay than Quartus. Other than for benchmark C1908, BDS-pga's delay optimization is far better than that of Quartus. Moreover, for benchmark C1908, while both BDS-pga and Quartus produce the same delay, the area overhead incurred by Quartus is far inferior to that incurred by BDS-pga. From these results it is clear that our approach: i) presents a far superior area-performance tradeoff than the others; ii) is, especially for large circuits, often orders of magnitude faster than Boolmap, SIS, and Quartus.

7. CONCLUSIONS AND FUTURE WORK

This paper has presented **BDS-pga** a comprehensive, BDD-based, FPGA-specific logic synthesis system. Our approach has attempted to unify various multi-level logic transformation techniques together with FPGA-specific logic decomposition algorithms in a novel synthesis framework. We make decisions earlier in the synthe-

[3]Time utilized by Quartus consists of time required for building the database for the circuit, time required for logic synthesis and mapping, and time required for various other operations. for our experiments, only the time required for *logic synthesis and mapping* is reported to make a fair comparison with BDS-pga.

Table 11. Delay Synthesis for Quartus & BDS-pga. K=4.

Ckt.	Quartus			BDS-pga+Flowmap		
	LUT	delay	CPU time	LUT	delay	CPU time
des	2055	8	8.36 mins	1590	5	53 sec
C1355	86	5	45 sec	90	4	4.3 sec
C1908	294	7	1.21 mins	163	7	3.25 sec
C5315	833	12	4.22 mins	588	7	14 sec
C3540	575	18	3.30 mins	556	11	28.1 sec
C7522	1013	14	6.56 mins	770	10	19 sec

sis process, keeping in mind that the resulting circuits are to be mapped onto LUT-based FPGAs. The resulting synthesized circuits exhibit superior area/performance characteristics than those synthesized by conventional FPGA-synthesis tools. Moreover, using an efficient BDD-decomposition engine, we are able to decompose large designs quickly, without sacrificing the quality of the resulting implementation.

One of the limitations of our tool is the lack of efficient techniques for logic simplifications with don't care sets using BDDs. While the *restrict* and *constrain* operators have been proposed to simplify a BDD with respect to another, no satisfactory solutions have been found. This limitation has forced us to use Espresso which, though robustly handles relatively large designs, often requires significant compute times. We are investigating various *implicit* logic minimization schemes to incorporate within our synthesis framework.

The decomposition in **BDS-pga** is guided by the fact that the resulting network would be mapped directly onto a k-LUT FPGA. We use the Flowmap tool to perform the mapping. Flowmap, as a pre-processing "decomposition" step, decomposes the network into two-input AND/OR gates. We disable this feature of Flowmap so as to preserve our k-feasible decompositions. While our approach attempts to decompose logic targeting the FPGA technology, mapping is still carried out as a post-processing step. Analogous to the techniques presented in [Chen and Cong 2001], it would be desirable to analyze a set of k-feasible decompositions during technology mapping in order to derive a better mapped solution.

REFERENCES

Ashenhurst, R. L. 1959. The decomposition of switching functions. In *Proc. Int'l Symp. Theory of Switching Functions* (1959), pp. 74–116.

Babba, B. and Crastes, M. 1992. Automatic Synthesis on Table Lookup-based FPGAs. In *Proc. Euro-ASIC* (May 1992), pp. 25–31.

Brayton, R. K. and *et al.* 1984. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers.

Brayton, R. K., Rudell, R., Sangiovanni-Vincentelli, A., and Wang, A. R. 1987. Mis: A multiple-level logic optimization system. *IEEE Transactions on Computer-aided design*, 1062–1081.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers C-35*, 677–691.

Chang, S. C., Marek-Sadowska, M., and Hwang, T. T. 996. Technology mapping for tlu fpga's based on decomposition of binary decision diagrams. *Proc. IEEE Trans. Computer*

Aided Des. Integrated Circuits & Syst., 1226–1236.

CHEN, G. AND CONG, J. 2001. Simultaneous Logic Decomposition with Technology Mapping in FPGA Designs. In *Proc.International Symposium on Field Programmable Gate Arrays* (Feb 2001), pp. 48–55.

CHEN, K. C., CONG, J., DING, Y., KAHNG, A. B., AND TRAJMAR, P. 1992. Dag-map: Graph-based fpga technology mapping for delay optimization. In *IEEE Design and Test of Computers* (Sept. 1992), pp. 7–20.

CONG, J. AND DING, Y. 1993. Beyond the combinatorial limit in depth minimization for lut-based fpga designs. In *Proc. 1993 IEEE/ACM Int'l. Conf. on CAD* (Santa Clara, CA, Nov. 1993), pp. 110–114.

CONG, J. AND DING, Y. 1994. 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 13*, 1–12.

CONG, J., LI, Z., AND BAGRODIA, R. 1994. Acyclic multi-way partitioning of boolean networks. In *Proc. DAC* (1994), pp. 670–675.

CONG, J., PECK, J., AND DING, Y. 1996. Rasp: A general logic synthesis system for sram-based fpgas. In *Proc. ACM 4th Int. Symp. on FPGAs* (1996), pp. 137–143.

COUDERT, O. AND MADRE, J. 1990. A Unified Framework for the Formal Verification of Sequential Circuits. In *Proc. ICCAD* (1990), pp. 126–129.

DEMICHELI, G. 1994. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, Inc., Hightstown, NJ 08520.

ECKL, K., LEGL, C., AND WURTH, B. 1996. *TOS Version 2.2: User Manual*. Institute of Elec. Design Automation, Tech. Univ. of Munich.

FILO, D. AND *et al.* 1991. Technology Mapping for Two Output based FPGAs. In *Proc. EDAC* (1991), pp. 534–538.

FRANCIS, R., ROSE, J., AND CHUNG, K. 1990. Chortle:a technology mapping for lookup table-based field programmable gate arrays. In *Proc. of the Design Automation Conference* (1990), pp. 613–619.

FRANCIS, R., ROSE, J., AND VRANESIC, Z. 1995. Chortle-crf:fast technology mapping for lookup table-based fpgas. In *Proc. Int'l Symp. Theory of Switching Functions* (1995), pp. 74–116.

HACTEL, G. D. AND SOMENZI, F. 1996. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, Dordrecht, THE NETHERLANDS.

JIANG, J.-H., JOUT, J.-Y., HUANG, J.-D., AND WEI, J.-S. 1997. A variable partitioning algorithm of bdd for fpga technology mapping. *IEEE Trans. Fundamentals E80-Ad*, 10 (October), 1813–1819.

KARPLUS, K. 1991. XMAP: A Technology Mapper for Table-Lookup based FPGAs. In *Proc. DAC* (1991), pp. 240–243.

LAI, Y. T., PEDRAM, M., AND VRUDHULA, S. 1993. Bdd based decomposition of logic functions with application to fpga synthesis. In *Proc. 30th Design Automation Conf.* (June 1993), pp. 642–647.

LEGL, C., WURTH, B., AND ECKL, K. 1996. A boolean approach to performance -directed technology mapping for lut-based fpga designs. In *Proc. Design Automation Conference* (1996), pp. 74–116.

MURGAI, R., BRAYTON, R. K., AND SANGIOVANNI-VENCENTELLI, A. 1995. *Logic Synthesis for Field-Programmable Gate Arrays*. Kluwer Academic Publishersc, Dordrecht, THE NETHERLANDS.

MURGAI, R., NISHIZAKI, Y., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 1990. Logic Synthesis for Programmable Gate Arrays. In *Proc. DAC* (1990), pp. 620–625.

NARAYAN, A. AND *et al.* 1996. Partitioned-ROBDDs: A Compact Canonical and Efficient Representation for Boolean Functions. In *Proc. ICCAD* (1996), pp. 547–554.

ROTH, J. P. AND KARP, R. M. 1962. Minimization over boolean graphs. *IBM J. of Research and Development 6*, 227–238.

SAVAGE, J. E. 1976. *The Complexity of Computing.* Wiley Interscience Publication.

SAWADA, H., SUYAMA, T., AND NAGOYA, A. 1995. Logic synthesis for look-up table based fpgas using functional decomposition and support minimzation. In *Proc. Int. Conf. Computer-Aided Design* (1995), pp. 54–59.

SAWKAR, P. AND THOMAS, D. 1992. Area and delay mapping for table-look up based field programmable gate arrays. In *Proceedings of the Design Automation Conference* (June 1992), pp. 368–373.

SENTOVICH, E. M. AND *et al.* 1992. SIS: A System for Sequential Circuit Synthesis. Technical report ucb/erl m92/41 (March), Dept. of EECS, Univ. of California, Berkeley.

SICARD, P. AND *et al.*D. 1991. Automatic Synthesis of Boolean Functions on Xilinx and Alcatel Programmable Devices. In *Euro ASIC* (1991), pp. 142–145.

SINGH, K. J., WANG, A. R., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 1988. Timing optimzation of combinational logic. *IEEE Trans. Computer Aided Design*, 282–285.

STANION, T. AND SECHEN, C. 1995. A method for finding good ashenhurst decompositions and its application to fpga synthesis. In *Proc. 32nd ACM/IEEE Design Automation Conference* (1995), pp. 74–116.

TRIMBERGER, S. 1994. *Field Programmable Gate Array Technology.* Kluwer Academic Publishers, Boston.

YANG, C. 2000. *BDD-Based Logic Synthesis System.* Ph. D. thesis, Univ. of Massachusetts, Amherst.

YANG, C., SINGHAL, V., AND CIESIELSKI, M. J. 1999. BDD Decomposition for Efficient Logic Synthesis. In *Proc. ICCD* (1999).

YANG, C., SINGHAL, V., AND CIESIELSKI, M. J. 2000. BDS: A BDD-based Logic Synthesis System. In *Proc. DAC* (2000).