Factor Cuts

Satrajit Chatterjee Alan Mishchenko Robert Brayton

Department of EECS
U. C. Berkeley
{satrajit, alanmi, brayton}@eecs.berkeley.edu

ABSTRACT

Enumeration of bounded size cuts is an important step in several logic synthesis algorithms such as technology mapping and re-writing. The standard algorithm does not scale beyond 6 or 7 inputs because it enumerates all cuts and there are too many of them. We address the enumeration problem by introducing the notion of *cut factorization*. In cut factorization, one enumerates *global* and *local* cuts (collectively called the *factor* cuts) of the network, and uses these to generate other cuts. Depending on how global and local cuts are defined, one obtains different factorization schemes. In the first scheme, complete factorization, it is possible to generate any cut from factor cuts. However, complete factorization is expensive though less expensive than exhaustive enumeration. In the second scheme, partial factorization, there is no guarantee of generating all cuts from factor cuts. However, it is much faster, and produces good results. In this paper we also present two applications of factor cuts: LUT mapping and macrocell mapping. In LUT mapping, we find that considering only factor cuts guarantees depth optimality for most nodes in the network. For the remaining nodes, other cuts need to be generated from factor cuts and examined. In macrocell mapping, we focus on a particular 9-input macrocell, and use factor cuts as a heuristic method to improve depth by reducing structural bias. Factor cuts are used to map the macrocell as a whole whenever possible instead of mapping its parts separately. In this context factor cuts enable a new quality–run-time tradeoff between mapping parts of the macrocell separately (poor quality), and mapping using all 9-input cuts (long run-time).

1. INTRODUCTION

Cut enumeration is an important step in many synthesis algorithms. In particular, technology mapping and resynthesis through re-writing require the ability to generate all cuts (of a certain size). For example, when mapping a circuit into k-input look up tables (k-LUTs) for FPGAs, one would like to enumerate all cuts of size k or less and pick the best ones. In the specific case of depth optimal LUT mapping, there is an elegant algorithm based on network flow [5] due to Cong and Ding that avoids the need for enumeration. However, as their later work indicates [6], for other objectives such as minimizing area under delay constraints, it may be necessary to enumerate cuts.

In a network of size n, the number of cuts of size k is $O(n^k)$. For small cuts (i.e. $k = 6$ or 7), one can use an enumerative procedure to compute all (or most) cuts [6, 11]. But for larger cuts, the enumerative procedure fails simply because there are too many cuts. Of course, only a small fraction (say 1%) of those cuts is useful in an FPGA mapping but *a priori* we don't know which 1%.

In this work we address the enumeration problem by trying to factor the cut space. Just as the algebraic expression $(ab + ac)$ can be factored as $a(b + c)$, the set of all cuts of a node can be factored using two sets of cuts called *global* and *local*. Collectively they are called *factor cuts*. By expanding factor cuts w.r.t. local cuts, a larger set of cuts can be obtained. During the cut computation only factor cuts are enumerated. Later on in the application (such as depth optimal LUT mapping), other cuts are generated from factor cuts as necessary.

Depending on how global and local cuts are defined, there can be different factorization schemes. In this paper, we present two schemes: complete and partial. In complete factorization, every cut can be obtained by expanding a factor cut w.r.t. a local cut. However, complete factorization is expensive since there may be a large number of global cuts.

Partial factorization is an alternative approach where there are much fewer global cuts, but there is no guarantee that all cuts can be generated by expanding factor cuts. However, in practice, good cuts are obtained with partial factorization in a fraction of the run-time required for complete enumeration.

We demonstrate the utility of factor cuts by presenting two applications of factor cuts to mapping. In the first application, we modify the algorithm for depth optimal LUT mapping to use factor cuts. For most nodes in a network, just examining factor cuts is sufficient to get optimum depth. The full set of cuts of a node has to be examined for relatively few nodes. Although LUTs with many inputs have only limited application (such as in structured ASICs where embedded memory blocks may be used to implement logic, or in future programmable architectures based on nano-devices) the depth optimal mapping problem has an elegant algorithm when factor cuts are used. It is interesting theoretically since it is suggestive of how algorithms may be

ICCAD'06, November 5-9, 2006, San Jose, CA

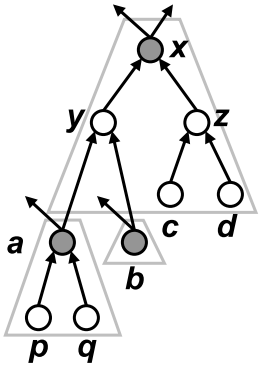

Figure 1: An AIG fragment to illustrate cut factorization. Nodes p, q, b, c, and d are primary inputs.

modified to use factor cuts.

In the second application, we use factor cuts to improve the quality of results in macrocell mapping (for a particular architecture). In this application factor cuts are used heuristically to improve the quality of the mapping obtained, over that of simple mapping which does not fully exploit macrocells. In this framework, factor cuts may be seen as an efficient method of reducing structural bias [4], by allowing deeper matches that match the macrocell as a whole, instead of breaking it into its constituent parts.

Other applications of factor cuts would be to improve the quality of standard cell mapping [4] and of logic synthesis using re-writing [10]. The cut size correlates with the capability of a mapper (or re-writing algorithm) to overcome structural bias. The larger the cut, the larger is the scope of Boolean matching (or Boolean transform) and the smaller the structural bias.

Section 2 reviews the conventional cut enumeration procedure and introduces notation used the paper. Section 3 presents the theory and experimental results for complete factorization. Section 4 describes partial factorization. Sections 5 and 6 present LUT- and macrocell- mapping algorithms based on factor cuts.

2. PRELIMINARIES AND REVIEW

AIG

An *And-Inverter Graph* (AIG), $\mathcal{G}$, is a directed acylic graph (DAG) where a node has either 0 or 2 incoming edges. A node with no incoming edges is a primary input (PI). A node with 2 incoming edges is a (2-input) And gate. An edge is either complemented or uncomplemented. A complemented edge indicates the inversion of the signal. Figure 1 shows an example of an AIG.

k-Feasible Cuts

A *cut* of a node n is a set of nodes c in its transitive fan-in such that every path from a PI to n includes a node in c. A cut is said to be *irredundant* if no subset of the cut is a cut. A *k-feasible cut* is an irredundant cut of size k or less. A *trivial cut* is a cut of size 1.

Let A and B be two sets of cuts. For convenience we define the operation $A \bowtie B$ as follows:

$$A \bowtie B \equiv \{u \cup v \mid u \in A,\ v \in B,\ |u \cup v| \leq k\}$$

($A \bowtie B$ is empty if either A or B is empty.)

Let $\Phi(n)$ denote the set of k-feasible cuts of n. If n is an And node, let n_1 and n_2 denote its inputs. We define $\Phi(n)$ as follows:

$$\Phi(n) \equiv \begin{cases} \{\{n\}\} & : \ n \text{ is a PI} \\ \{\{n\}\} \cup (\Phi(n_1) \bowtie \Phi(n_2)) & : \ \text{otherwise} \end{cases}$$

That this formula enumerates all k-feasible cuts can be easily checked by induction. If there are reconvergent paths in the network this procedure may additionally generate redundant cuts which can be removed efficiently using signatures [9]. In what follows we assume that this removal is done as part of the $\bowtie$ operation.

Dag and Tree Nodes

A *dag node* is a node in the AIG which has two or more outgoing edges. A node of $\mathcal{G}$ that is not a dag node is called a *tree node*. The set of dag nodes is denoted by $\mathcal{F}$, and that of tree nodes by $\mathcal{T}$. In Figure 1, the factor nodes are shown shaded.

The sub-graph $\mathcal{G}_\mathcal{T}$ of $\mathcal{G}$ induced by the tree nodes is a forest of trees. Each tree T in $\mathcal{G}_\mathcal{T}$ has an outgoing edge (i.e. feeds in) to exactly one dag node n_f in $\mathcal{G}$.

Consider the sub-graph T_{n_f} of $\mathcal{G}$ induced by a dag node n_f in $\mathcal{G}$ and the nodes belonging to the trees in $\mathcal{G}_\mathcal{T}$ that feed into it. T_{n_f} is a (possibly trivial) tree. T_{n_f} is called the *factor tree* of a node n in T_{n_f}. Clearly every node in $\mathcal{G}$ has a factor tree. The dag node n_f is called the root of T_{n_f}

Figure 1 shows the decomposition of the AIG into factor trees. The factor tree for node b is trivial. The factor tree of node x consists of x, y, z, c, and d.

The inputs n_i of the leaves of a factor tree are dag nodes. The factor tree along with the inputs n_i is a leaf-DAG, and is called the *factor leaf-DAG*. Every node n in G has a unique factor leaf-DAG (via its unique factor tree). The root of a factor leaf-DAG is the root of the corresponding tree.

In Figure 1, the factor leaf-dag of x contains the nodes in the factor tree of x along with nodes a and b.

Local Cuts, Global Cuts and Expansion

In the following sections we will identify some k-feasible cuts in the network as local cuts, and some others as global cuts. We refer to them collectively as factor cuts. The precise definitions of local and global will depend on the factorization scheme (complete or partial), but the general idea is to "expand" factor cuts by local cuts to obtain other k-feasible cuts. In the case of complete factorization, this expansion will produce all k-feasible cuts.

Let c be a factor cut of node $n \in \mathcal{G}$. Let c_i be a local cut of a node $i \in c$. Consider $l = \bigcup_{i \in c} c_i$. The set l is a cut of n though not necessarily k-feasible. If l is k-feasible, then l is called a *1-step expansion* of c. Define 1-step(c) as the set of cuts obtained from c by 1-step expansion, i.e.

$$\text{1-step}(c) = \{l \mid l \text{ is a 1-step expansion of } c\}$$

We ensure that $c \in$ 1-step(c) by requiring that the set of local cuts of a node always include the trivial cut.

In Figure 1, consider the cut $\{a, b, z\}$ of x. By expanding node a with its local cut $\{p, q\}$, we get the cut $\{p, q, b, z\}$ of x. Thus $\{p, q, b, z\} \in$ 1-step($\{a, b, z\}$).

Name	Nodes		Number of cuts			Run-time(sec)	
	Total	% Dag	All	Reduced	Tree	All	CF
C1355	541	50.09	47245	25811	657	0.16	0.06
C1908	435	38.85	16546	10153	558	0.04	0.01
C2670	920	17.50	27734	14828	1583	0.07	0.02
C3540	1081	22.94	46392	29356	2064	0.10	0.05
C5315	1827	22.22	67118	35852	2677	0.13	0.07
C6288	2369	60.11	279197	209807	2818	0.82	0.53
C7552	2248	29.40	129161	78543	3251	0.28	0.16
b14	6296	23.60	397951	226401	11300	0.69	0.32
b15	8869	22.01	416804	265870	18102	0.65	0.35
clma	24387	11.20	739582	658314	27419	0.52	0.51
pj1	17675	19.11	781566	465554	32725	1.20	0.61
pj2	4055	14.38	110103	84839	5889	0.13	0.10
pj3	11178	22.45	523448	373960	22659	0.77	0.50
s15850	4127	25.66	108433	75509	6066	0.18	0.11
s35932	13711	35.89	341330	223743	16106	0.52	0.30
s38417	10820	25.00	294832	164264	16895	0.47	0.23
Ratio	1.00	0.27	1.00	0.64	0.04	1.00	0.55

Table 1: Comparison of conventional enumeration (All) and complete factorization (CF) for $k = 6$. The run-times for this and other experiments are on a 3 GHz Intel Pentium 4 with 1GB of RAM.

3. COMPLETE FACTORIZATION

In complete factorization, we enumerate *tree* cuts and *reduced* cuts (defined below) which are subsets of the set of all k-feasible cuts. Tree cuts are the local cuts and reduced cuts are the global cuts. We use the term *complete factor cuts* to refer to tree cuts and reduced cuts collectively. Complete factorization has the property that any k-feasible cut can be obtained by 1-step expansion.

3.1 Theory

Tree Cuts (Local Cuts)

Let $\Phi_{\mathcal{T}}(n)$ denote the set of all tree cuts of node n. Define the auxiliary function $\Phi^{\dagger}_{\mathcal{T}}(n)$ as follows:

$$\Phi^{\dagger}_{\mathcal{T}}(n) \equiv \begin{cases} \phi & : \ n \in \mathcal{F} \\ \Phi_{\mathcal{T}}(n) & : \ \text{otherwise} \end{cases}$$

$\Phi_{\mathcal{T}}(n)$ is defined recursively as,

$$\Phi_{\mathcal{T}}(n) \equiv \begin{cases} \{\{n\}\} & : \ n \text{ is a PI} \\ \{\{n\}\} \cup (\Phi^{\dagger}_{\mathcal{T}}(n_1) \bowtie \Phi^{\dagger}_{\mathcal{T}}(n_2)) & : \ \text{otherwise} \end{cases}$$

$\Phi_{\mathcal{T}}(n)$ represents the subset of k-feasible cuts of n that only involve nodes from the factor tree of n.

For example, in Figure 1, $\Phi_{\mathcal{T}}(x) = \{\{x\}, \{y, z\}, \{y, c, d\}\}$.

Reduced Cuts (Global Cuts)

We define $\Phi_{\mathcal{R}}(n)$, the set of reduced cuts of a node n, as follows:

$$\Phi_{\mathcal{R}}(n) \equiv \begin{cases} \{\{n\}\} & : n \text{ is a PI} \\ \{\{n\}\} \cup ((\Phi_{\mathcal{R}}(n_1) \bowtie \Phi_{\mathcal{R}}(n_2)) \setminus \Phi_{\mathcal{T}}(n)) & : \text{otherwise} \end{cases}$$

The formula for $\Phi_{\mathcal{R}}(n)$ is very similar to that for $\Phi(n)$, except that non-trivial tree cuts are removed. Since this removal is done recursively $\Phi_{\mathcal{R}}(n)$ is significantly smaller than $\Phi(n)$.

For example, in Figure 1, $\Phi_{\mathcal{R}}(x) = \{\{x\}, \{a, b, z\}\}$. Note that $\{a, b, c, d\}$ is not a reduced cut of x, since $\{c, d\}$ is removed when computing $\Phi_{\mathcal{R}}(z)$.

Cut Decomposition Theorem

With local and global cuts being tree and reduced cuts respectively, a cut decomposition theorem holds. The proof of the theorem is omitted due to lack of space.

THEOREM 1 (CUT DECOMPOSITION). *Every k-feasible cut of a node n in $\mathcal{G}$ is a 1-step expansion of a k-feasible complete factor cut of n.*

3.2 Experimental Results

Table 1 shows the number of complete factor cuts for $k = 6$ for a set of benchmarks. The column labeled "dag" shows the percentage of nodes that are dag nodes. On average about 27% of the nodes are dag nodes. The number of reduced cuts is about 64% of the total number of cuts. Enumerating complete factor cuts is about 2 times faster than enumerating all cuts.

Table 2 shows the number of complete factor cuts for $k = 9$ for the same set of benchmarks. (Ignore the columns labeled "PF" for now.) For $k = 9$, not all cuts could be computed since there were too many. The columns labeled "Over" indicate the fraction of nodes at which the maximum limit of 2000 cuts was exceeded. When enumerating all cuts, the limit was exceeded in about 16% of the nodes on average. However, the reduced cut enumeration exceeded the limit in only 6.5% of the nodes. (The tree cut enumeration never exceeded the limit.) The number of complete factor cut cuts is about 68% and the enumeration runs about 34% faster.

4. PARTIAL FACTORIZATION

Although complete factorization causes a reduction in the number of cuts that need to be enumerated, further reduction is possible by sacrificing the ability to generate all k-feasible cuts by 1-step expansion. This leads to the notion of partial factorization. Partial factorization is much faster than complete factorization, and produces a "good" set of cuts in practice, especially for large k (say $k = 9$).

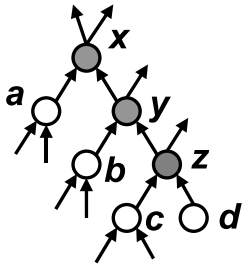

Figure 2: An AIG fragment to illustrate the limitations of partial factorization. The cut $\{a, b, c, d\}$ is a 4-feasible cut of x, but cannot be generated from a 1-step expansion of a partial factor cut of x.

4.1 Definition

In partial factorization *leaf-dag* cuts play the role of local cuts and *dag* cuts play the role of global cuts.

Leaf-dag Cuts (Local Cuts)

Let $\Phi_{\mathcal{L}}(n)$ denote the set of all k-feasible leaf-dag cuts of node n. Define the auxiliary function $\Phi^{\dagger}_{\mathcal{L}}(n)$ as follows:

$$\Phi^{\dagger}_{\mathcal{L}}(n) \equiv \begin{cases} \{\{n\}\} & : \quad n \in \mathcal{F} \\ \Phi_{\mathcal{L}}(n) & : \quad \text{otherwise} \end{cases}$$

Leaf-dag cuts are defined recursively as follows.

$$\Phi_{\mathcal{L}}(n) \equiv \begin{cases} \{\{n\}\} & : \quad n \text{ is a PI} \\ \{\{n\}\} \cup (\Phi^{\dagger}_{\mathcal{L}}(n_1) \bowtie \Phi^{\dagger}_{\mathcal{L}}(n_2)) & : \quad \text{otherwise} \end{cases}$$

Conceptually, leaf-dag cuts are similar to tree cuts. Unlike tree cuts, leaf-dag cuts also include the dag nodes that feed into the factor tree of a node. This allows more cuts to be generated by 1-step expansion at the cost of a slight increase in run-time for local cut enumeration.

In the example of Figure 1, the cuts $\{a, b, z\}$ and $\{a, b, c, d\}$ are examples of leaf-dag cuts of node x. (They are not tree cuts of x.)

Dag Cuts (Global Cuts)

Let $\Phi_{\mathcal{D}}(n)$ denote the set of k-feasible dag cuts of n. We define $\Phi_{\mathcal{D}}(n)$ as follows:

$$\Phi_{\mathcal{D}}(n) \equiv \begin{cases} \{\{n\}\} & : \quad n \text{ is a PI} \\ \Phi_{\mathcal{D}}(n_1) \bowtie \Phi_{\mathcal{D}}(n_2) & : \quad n \in \mathcal{T} \\ \{\{n\}\} \cup (\Phi_{\mathcal{D}}(n_1) \bowtie \Phi_{\mathcal{D}}(n_2)) & : \quad \text{otherwise} \end{cases}$$

In Figure 1, for $k = 4$, $\{x\}$ and $\{a, b, c, d\}$ are the only dag cuts of x.

This definition of dag cuts is motivated by a need to reduce the number of global cuts seen in complete factorization. Defining dag cuts in this manner allows us to capture much of the reconvergence in the network without having to enumerate the large number of reduced cuts (as in complete factorization).

However, by computing global cuts this way, some cuts cannot be generated by 1-step expansion, as shown in Figure 2. The 4-feasible cut $\{a, b, c, d\}$ of x cannot be generated from a 1-step expansion of a partial factor cut of x.

4.2 Experimental Results

In Table 2 the columns under "PF" show the number of partial factor cuts for $k = 9$. It is seen from the table that the number of partial factor cuts is a small fraction of the total number of cuts (15%) and the time for enumerating these cuts is less than 10% of the time required to enumerate all cuts. During enumeration only a small fraction of nodes (less than 0.5%) exceeded the limit of 2000 when computing dag cuts and hence those data are not shown in Table 2.

We note here that the multiplier (C6288) is a particularly interesting benchmark. In comparison with other benchmarks, it has many – about 60% – dag nodes. This negates the advantage of computing partial factor cuts as the factor trees are small. Hence the factor cut enumeration takes unusually long.

5. DEPTH OPTIMAL K-LUT MAPPING

In this section we present an application of factor cuts to technology mapping for LUT-based FPGAs. We review the conventional algorithm that enumerates all k-feasible cuts in Section 5.1. In Section 5.2, the conventional algorithm is modified to work with factor cuts. Section 5.3 presents experimental results using factor cuts for LUT mapping on a set of benchmarks.

5.1 Conventional Algorithm

The conventional algorithm for depth optimal k-LUT mapping enumerates all k-feasible cuts and chooses the best set of cuts using dynamic programming on the AIG. The algorithm proceeds in two passes over the nodes of the AIG. The first pass, called the forward pass, is done in topological order from PIs to POs. For each node, all k-feasible cuts are enumerated (using the k-feasible cuts of its children), and the cut with earliest arrival time is selected.

The arrival time of a cut c, denoted by arrival(c) is defined as follows:

$$\text{arrival}(c) \equiv 1 + \max_{n \in c} \text{arrival}(n)$$

where arrival(n) is the best arrival time for node n (from among all its k-feasible cuts). This recursion is well defined since when the cuts for a node n are being processed, the nodes in the transitive fan-in of n have already been processed. Thus the best arrival time of any node in a k-feasible cut of n has already been computed.

The second pass of the algorithm is done in reverse topological order from the POs to the PIs. For each PO the best k-feasible cut is chosen and a LUT constructed in the mapped netlist to implement it. Then recursively for each node in the cut, this procedure is repeated.

5.2 Depth Optimal Mapping with Factor Cuts

The main limitation of the conventional algorithm is that it explicitly enumerates a large number of all k-feasible cuts during the forward pass. The idea behind using factor cuts is to avoid enumerating all k-feasible cuts. Ideally, one would like to enumerate only factor cuts (which are fewer than k-feasible cuts) for mapping.

The problem with enumerating only factor cuts is that there is no guarantee that the best k-feasible cut is a factor cut. This means that one may have to expand factor cuts exhaustively in order to get the best k-feasible cut. But this could be as bad as enumerating all k-feasible cuts.

Name	Number of cuts and % of nodes overflowing							Run-time (sec)		
	All		CF			PF		All	CF	PF
	Total	Over	Reduced	Over	Tree	Dag	Leaf-dag			
C1355	433995	19.96	398226	16.82	657	103255	1457	10.77	9.86	1.25
C1908	247474	9.20	136755	2.30	562	53094	2582	4.06	1.94	0.51
C2670	363647	10.54	227604	2.61	3086	8412	11239	7.77	4.04	0.11
C3540	723405	18.69	531281	6.29	2818	29858	23391	16.37	12.3	0.27
C5315	822569	6.95	402009	0.71	2827	53299	12290	17.72	6.06	0.32
C6288	3232621	43.56	3220602	43.14	2818	2273878	6072	126.38	139.99	70.34
C7552	1632538	20.60	1100358	4.89	5214	186769	14341	38.64	24.93	1.52
b14	8937035	54.57	5734487	7.91	15607	507448	96713	182.01	144.51	2.78
b15	7498534	16.24	4484041	3.33	27812	995259	150682	148.60	74.91	11.86
clma	8870652	0.85	7688879	0.67	27419	364384	955803	60.29	50.3	4.13
pj1	12695024	18.91	7806133	3.55	63466	775824	443649	196.31	130.31	11.57
pj2	1633480	6.98	1315608	2.98	5944	48941	45664	24.29	20.13	0.16
pj3	8644521	18.00	5900954	5.47	69542	861144	291336	142.95	106.91	12.07
s15850	1323074	6.47	966661	3.46	8542	40838	52282	23.91	18.54	0.29
s35932	1623042	0.00	928175	0.00	16106	60029	54860	10.42	3.95	0.13
s38417	3678001	5.15	1587491	0.63	25547	69623	113578	61.24	19.1	0.69
Ratio	1.00	(16.04)	0.68	(6.55)	0.00	0.12	0.03	1.00	0.66	0.08

Table 2: Comparison of conventional enumeration (All), complete factorization (CF), and partial factorization (PF) for $k = 9$. The number of all 9-feasible cuts is an underestimate. See Section 3.2 for details.

To avoid exhaustively expanding all factor cuts, we use the following result [5, Lemma 2]:

THEOREM 2 (CONG AND DING). *If n is an And node with inputs n_1 and n_2, then*

$$\text{arrival}(n) = p \text{ or } \text{arrival}(n) = p + 1$$

where $p = \max(\text{arrival}(n_1), \text{arrival}(n_2))$.

Theorem 2 provides a lower bound on the best arrival time. If a factor cut attains the lower bound, then there is no need for expansion. If no factor cut attains the lower bound, then the factor cuts are 1-step expanded one by one. During this process if the lower bound is attained, there is no need to expand the remaining factor cuts.

Optimality. If complete factorization is used then this algorithm produces the optimal depth since 1-step expansion will produce all k-feasible cuts (by the Cut Decomposition Theorem). In the case of partial factorization, there is no guarantee of optimality. However experiments show that for large k there is no loss of optimality for the set of benchmarks considered (see Section 5.3).

Expansion. In complete factorization, 1-step expansion need not be exhaustive. It suffices to expand the late arriving inputs of the cut, one node at a time. This is because the expansions are independent – two nodes in the cut do not have to be expanded simultaneously with their tree cuts, since the tree cuts of two nodes never overlap.

In partial factorization, the leaf-dag cuts of two nodes may overlap, and so the expansions are not independent. However, in our experiments, the nodes were expanded one late-arriving node at a time since that did not degrade the quality significantly.

It is instructive to see why the conventional algorithm cannot be easily modified to exploit the lower bound. Although one need not scan all of $\Phi(n)$ to find the best cut (one can stop as soon as the lower bound is attained), one still needs to construct $\Phi(n)$ completely. This is because a cut $c \in \Phi(n)$ that does not lead to the best arrival time for n may lead to the best cut for some node n' in the transitive fanout of n.

5.3 Experimental Results

We implemented a prototype FPGA mapper based on factor cuts in the *ABC* system [2]. Table 3 shows the depth and run-times of the various modes of this mapper for $k = 9$.

The first set of columns (under the heading "Lim = 2000") show that complete factorization (CF) produces better results than enumerating all cuts and is faster. These columns directly correspond to the "All" and "CF" cut data shown in Table 2. Note that the sub-optimality of enumerating all cuts is due to the fact that not all cuts could be computed for the nodes – there was an overflow of 16%. Also by comparing the cut computation run-times in Table 2 with the overall mapping run-times in Table 3 we can see that the mapping run-time is dominated by cut computation. Expansion takes a small fraction of the total run-time and on average about 25% of the nodes needed to be expanded.

The second set of columns (under the heading "Lim = 1000") show the effect of reducing the limit on the maximum number of cuts stored at a node. Although the cut computation is more than twice as fast, the depth is 15% worse when enumerating all cuts. Complete factorization continues to produce better depths and has shorter run-times.

The final set of columns (under the heading "PF") shows the depth and run-time obtained with partial factorization. Although 1-step expansion from partial factor cuts may not generate all k-feasible cuts, the cuts that it does generate are competitive with those enumerated by the conventional procedure under the limit. Furthermore, partial factorization is about 6X faster than conventional enumeration.

We also experimented with partial factorization for different values of k. For $k = 6$ we found that partial factorization produces about 5% *worse* results than enumerating all cuts

Name	Lim = 2000				Lim = 1000				PF	
	Depth		Run-time		Depth		Run-time			
	All	CF	All	CF	All	CF	All	CF	Depth	Run-time
C1355	4	3	10.82	9.93	5	3	3.02	3.46	3	5.21
C1908	4	4	4.1	1.96	5	5	1.49	0.94	4	0.59
C2670	4	4	7.82	4.1	4	4	3.21	2.33	4	1.93
C3540	6	6	16.46	12.43	7	6	4.98	5.35	6	0.95
C5315	5	5	17.82	6.15	5	5	8.06	3.92	5	0.94
C6288	12	12	126.71	140.41	20	15	35.59	33.99	11	72.07
C7552	5	4	38.82	25.18	5	5	14.9	11.17	4	2.00
b14	10	10	183	147.17	12	10	43.86	63.21	10	6.99
b15	12	10	149.45	76.38	13	11	46.67	39.5	13	20.85
clma	9	9	61.13	51.38	9	9	40.28	36.76	10	8.36
pj1	8	8	197.64	132.27	8	8	61.83	62.99	9	22.30
pj2	4	4	24.48	20.39	5	4	9.16	8.81	4	0.27
pj3	6	6	143.89	108.18	8	7	48.76	49.99	7	24.37
s15850	8	7	24.06	18.86	9	8	10.54	8.33	7	2.12
s35932	2	2	10.59	4.15	2	2	7.46	4.14	2	0.21
s38417	5	4	61.64	19.73	6	5	24.97	12.79	4	2.90
Ratio	1.00	0.94	1.00	0.67	1.15	1.09	0.40	0.32	0.97	0.15

Table 3: Comparison of conventional mapping (All) and complete factorization (CF) with different limits, and partial factorization (PF) with a limit of 2000 cuts per node. $k = 9$. The run-times are in seconds.

and runs about 3X faster. For $k = 12$, we found that trying to enumerate all cuts leads to poor results since more than 40% of the nodes exceed the cut limit. Partial factorization works better, producing 50% smaller depth on average than exhaustive enumeration.

6. MACROCELL MAPPING

In this section, we apply factor cuts to improve the quality of macrocell mapping. Rather than providing large LUTs, FPGA manufacturers prefer to provide configurable logic blocks, or macrocells. Unlike LUTs, macrocells cannot directly realize the full set of functions of their inputs, but only a (large) subset. This restriction allows a macrocell (and the surrounding routing network) to be implemented with less area than a LUT. It is hoped that most functions appearing in practical circuits can be realized by a well designed macrocell.

Due to the wide variety of macrocells possible, *efficient* general algorithms are not known for macrocell mapping.[1] Therefore, we focus here on a specific macrocell architecture used in a commercial FPGA and illustrate how factor cuts may be used to significantly improve the quality of mapping. Given a different macrocell architecture, the matching step (i.e. deciding if a cut can be realized by the macrocell) would be different, but the basic idea of using factor cuts to improve the mapping is still applicable.

6.1 Macrocell Architecture

We work with the macrocell shown in Figure 3. The architecture is based on Actel's ProASIC3 family of FPGAs [1]. The FPGA consists of a sea of tiles, where each tile can implement any function of 3 inputs i.e. is equivalent to a 3-LUT. Each tile is fully connected to its 8 nearest neighbours using fast local interconnect. This naturally leads to the 9-input–1-output macrocell comprised of three 3-LUTs feeding into another 3-LUT. Note that this macrocell cannot implement an arbitrary 9-input function, but only a 9-input function that has this particular disjoint-support decomposition. Some of the inputs could be the same signal (or constants), and so functions with less than 9 inputs can also be implemented. Clearly, these functions need not have disjoint support decompositions.

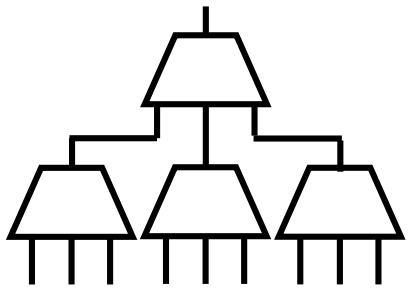

Figure 3: The 9-input macrocell (consisting of four 3-LUTs) used for mapping in Section 6. It is based on Actel's ProASIC3 family of FPGAs.

Observe that 9-input functions that are not realizable with the macrocell (call them *hard*) can still be implemented in the FPGA since the FPGA provides 3-LUTs which are universal. However, these hard functions will require more than 2 levels of 3-LUTs for their implementation.

Generalization. This observation generalizes to other macrocell architectures used in practice. Although the macrocell cannot implement certain functions directly, by using multiple macrocells (each possibly "under-utilized"), one can

[1]In fact, Ling et al. [8] provide a general algorithm for macrocell matching, but it is not scalable to large macrocells. Since they enumerate all cuts, it is possible that their approach might benefit from the use of factor cuts; but that remains to be studied.

Algorithm 1 Forward pass of macrocell depth-oriented mapping

INPUT: AIG $\mathcal{G}$
OUTPUT: bestcut(n) for every $n \in \mathcal{G}$
for each node $n \in \mathcal{G}$ in topological order **do**
  compute $\Phi^3(n)$, $\Phi^9_{\mathcal{L}}(n)$, $\Phi^9_{\mathcal{D}}(n)$
  $\chi(n) \leftarrow \{c \,|\, c \in \Phi^3(n) \cup \Phi^9_{\mathcal{L}}(n) \cup \Phi^9_{\mathcal{D}}(n), c \text{ is valid}\}$
  if n is And node **then**
    bestcut(n) $\leftarrow \arg\min_{c \in \chi(n)} \text{arrival}(c)$
    arrival(n) $\leftarrow$ arrival(bestcut(n))
  else
    arrival(n) $\leftarrow$ arrival time of PI n
  end if
end for

implement arbitrary functions. For example, the adaptive logic module in the Altera Stratix II architecture [7] can implement certain functions of 7-inputs, but can also be used as a general 6-LUT.

6.2 Mapping Algorithm

The mapping consists of two steps. The first step is a pre-computation to characterize the set of functions that can be realized by the macrocell. This is only done once. The second step is the mapping proper, where the best configuration of macrocells is chosen. This is done for each design being mapped.

Pre-computation. In this step, we enumerate the truth tables of all the functions that can be realized by the macrocell. Since the number of functions is very large we exploit the phase and permutation symmetries of the functions to limit the enumeration. As a result of enumeration, there are about 20,000 NPN classes that can be realized by the macrocell. This pre-computation takes several minutes, and is done with truth-tables represented as bit-strings. The resulting truth tables of the NPN representatives are saved in a file.

Mapping. During mapping, the truth-tables of the NPN-representatives computed above are read in from the file and hashed. Then the mapping is done in two passes as usual. The first pass proceeds in topological order from the PIs to the POs. For each node n in the AIG, we compute the set of 3-feasible cuts $\Phi^3 n$, and the 9-feasible leaf-dag cuts $\Phi^9_{\mathcal{L}}(n)$ and dag cuts $\Phi^9_{\mathcal{D}}(n)$.

A cut c is *valid* if it can be realized by the macrocell. Every 3-feasible cut is valid. The validity of a 9-feasible cut is checked by computing its NPN representative (using a method similar to that of Chai and Kuehlmann [3]) and looking it up in the hash table.

Among the valid cuts, the cut with the earliest arrival time is selected. The arrival time of a cut c, denoted by arrival(c) is defined as follows:

$$\text{arrival}(c) \equiv \text{d}(c) + \max_{n \in c} \text{arrival}(n)$$

where arrival(n) is the best arrival time for node n as before, and d(c) for a valid cut is defined as:

$$\text{d}(c) \equiv \begin{cases} 1 & : \quad c \text{ is 3-feasible} \\ 2 & : \quad \text{otherwise} \end{cases}$$

Name	Depth			Run-time (sec)		
	3-Cut	PF	All	3-Cut	PF	All
C1355	9	6	9	0.00	1.26	3.32
C1908	12	11	10	0.00	0.61	1.90
C2670	10	10	9	0.00	0.06	3.53
C3540	17	16	15	0.00	0.29	5.94
C5315	14	12	11	0.01	0.44	8.65
C6288	31	27	27	0.00	34.71	33.44
C7552	11	10	10	0.01	1.88	15.26
b14	30	29	25	0.01	4.12	57.36
b15	31	30	26	0.02	8.24	60.86
clma	36	31	26	0.05	2.45	64.87
pj1	20	19	17	0.03	7.01	85.31
pj2	11	9	8	0.01	0.32	12.01
pj3	18	16	15	0.03	9.04	62.79
s15850	18	17	17	0.01	0.26	11.64
s35932	5	4	4	0.03	0.30	9.49
s38417	12	12	11	0.02	0.58	29.74
average	1.00	0.90	0.85		1.00	29.13

Table 4: Experimental results for mapping to the macrocell architecture with a limit of 1000.

After the first pass, the second pass is done, as usual, in reverse topological order from POs to PIs to choose the best cover.

6.3 Discussion

There is an important difference in the way factor cuts are used in k-LUT mapping and in macrocell mapping. For LUT mapping, depth optimality can be guaranteed by using complete factor cuts and expanding when necessary. However for macrocells, factor cuts are only used in a heuristic manner to improve the solution; and no guarantee of depth optimality can be made.

The main problem is that there is no analog of Cong and Ding's theorem (Theorem 2) for macrocells. This is because the validity (in the sense defined above) of a cut of an And node does not imply the validity of cuts of its children. Thus the optimal depth of a node is not bounded by the depths of its children.

In terms of the algorithm, this means that there is no way to know if the depth obtained by just looking at factor cuts is sufficient. One can extend the macrocell mapping algorithm above to try a few expansions of factor cuts to see if the depth can be improved but we have not implemented this yet.

6.4 Experimental Results

We implemented Algorithm 1 in our FPGA mapper, and Table 4 shows the experimental results on the set of benchmarks. We compare the optimal depth and run-time of the mapper in three different modes. In the first mode (columns labeled "3-Cut"), the mapping is done using only 3-feasible cuts. (Macrocells are not directly used, but obtained by grouping 3-LUTs.) The second mode (columns labeled "PF"), corresponds to Algorithm 1 above where partial factor cuts are used. The final mode (columns labeled "All"), all 9-feasible cuts are enumerated (up to a limit of 1000 cuts per node), and the valid 9-feasible cuts are used for mapping. (In this process we ensure that all 3-feasible

cuts are computed.)

The best results are obtained when all 9-feasible cuts are used, giving on average 15% better depth than when just 3-feasible cuts are used. However, the run-time is long, taking minutes for the larger benchmarks. Factor cuts provide an intermediate quality-run-time trade-off between these two extremes. With factor cuts, much of the gain of using all 9-feasible cuts is seen (10% better depth), with a run-time of a few seconds. (Again, C6288 is an exception, since it has many partial factor cuts.)

6.5 Area

In practice, it is important to control the area during mapping. Since Algorithm 1 makes no attempt to control area, the area could be significantly worse than when only 3-feasible cuts are used.

We propose to control area by using the idea of choice networks [9]. We first obtain a mapping with good depth using Algorithm 3. The mapped netlist is then "unmapped" by decomposing it into an AIG. This AIG is combined with the original AIG to create a choice network on which a conventional depth-optimal 3-LUT mapping is done followed by area recovery as in [9]. This would guarantee that the superior depth of Algorithm 3 is obtained at a reasonable cost in area. We have not implemented this yet and leave it to future work.

7. CONCLUSIONS

In this paper we presented two schemes for factoring the cut space of a network. In the first scheme, complete factorization, the factor cuts are sufficient to generate all cuts, though generating the factor cuts themselves is somewhat expensive. This property can be used to design optimal algorithms by generating other cuts on demand. In the second scheme, partial factorization, there is no such guarantee. However, much fewer cuts need to be enumerated which makes it very fast.

We also looked at two applications of factor cuts. The first was depth optimal technology mapping for large LUTs. Although this has limited practical application at present, it is interesting theoretically, since it is suggestive of how algorithms may be modified to work with factor cuts. The key observation enabling this application is Cong and Ding's lower bound theorem. Using that theorem, in most cases it is sufficient to just examine factor cuts in order to guarantee depth optimality.

Furthermore, our experiments show that using factor cuts is better than using conventional enumeration but limiting the number of cuts per node. Mapping using partial factor cuts is faster and produces better depth, especially for $k = 9$ and above.

The second application was mapping into large macrocells. Although we do not have a clean theoretical result (as in the k-LUT mapping case), in practice, factor cuts enable a new quality–run-time tradeoff between the poor quality of mapping macrocells by mapping its parts separately and the slow run-time of mapping the entire macrocell by enumerating all cuts. Although we focussed on a specific macrocell architecture, the technique is fairly general and should work for different kinds of macrocells.

One aspect of factor cuts that we did not discuss in this paper relates to their potential for area-oriented mapping. Since global cuts contain a larger proportion of multi-fanout nodes, using a mapping solution based on global cuts (especially dag cuts) is likely to be a good starting point for area-oriented mapping. Our preliminary experiments along these lines have been encouraging.

Looking forward, we plan to further explore the use of factor cuts in technology mapping for macrocells (focusing on area, and other macrocell architectures); in reducing structural bias for standard cell mapping; and in logic synthesis using AIG re-writing. On the theoretical side, it would be interesting to explore the connection between factor cuts and disjoint support decompositions in order to improve macrocell matching.

8. ACKNOWLEDGMENTS

We thank Niklas Eén for helpful discussions. This work was supported by SRC, C2S2, the MARCO Focus Center for Circuit and System Solution; and by the California Micro Program with our industrial sponsors Altera, Intel, Magma and Synplicity.

9. REFERENCES

[1] Actel Corporation, "ProASIC3 Flash Family FPGAs Datasheet," Available from Actel website.

[2] Berkeley Logic Synthesis Group, The ABC Logic Synthesis System, UC Berkeley. `http://www.eecs.berkeley.edu/~alanmi/abc/`

[3] D. Chai and A. Kuehlmann, "Building a Better Boolean Matcher and Symmetry Detector," In DATE '06, pp. 1079-1084.

[4] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing Structural Bias in Technology Mapping," In ICCAD '05, pp. 519-526.

[5] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," IEEE Trans. CAD, Vol.13, No. 1 (January 1994), pp. 1-12.

[6] J. Cong, C. Wu and Y. Ding, "Cut Ranking and Pruning: Enabling a general and efficient FPGA mapping solution," In FPGA '99, pp. 29-35.

[7] M. Hutton et al., "Improving FPGA Performance and Area using an Adaptive Logic Module," In Field Programming Logic and Application 2004, pp. 135-144.

[8] A. Ling, D. Singh and S. Brown, "FPGA Logic Synthesis using Quantified Boolean Satisfiability," In SAT '05, Springer LNCS Vol. 3569, pp. 444-450.

[9] A. Mishchenko, S. Chatterjee and R. Brayton, "Improvements to Technology Mapping for LUT-based FPGAs," Proc. FPGA '06, pp. 41-49.

[10] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG Re-writing: A fresh look at combinational logic synthesis," In DAC '06, pp. 532-536.

[11] P. Pan and Liu, "A New Retiming-based Technology Mapping Algorithm for LUT-based FPGAs," In ACM Int'l Symp. on FPGAs, pp. 35-42, 1998.