KL-Cuts:

A New Approach for Logic Synthesis Targeting Multiple Output Blocks

Osvaldo Martinello Jr, Felipe S. Marques, Renato P. Ribas, André I. Reis
Institute of Informatics
Federal University of Rio Grande do Sul – UFRGS
Porto Alegre, Brazil
{omjunior, felipem, rpribas, andreis}@inf.ufrgs.br

Abstract — This paper introduces the concept of *kl*-feasible cuts, by controlling both the number *k* of inputs and the number *l* of outputs in a circuit cut. To provide scalability, the concept of factor cuts is extended to *kl*-cuts. Algorithms for computing this kind of cuts, including *kl*-cuts with unbounded *k*, are presented and results are shown. As a practical application, a covering algorithm using these cuts is presented.

Keywords – AIG; cut enumeration; technology mapping.

I. INTRODUCTION

Recent advances on logic synthesis are based on And-Inverter-Graphs – AIGs, for scalability reasons [1, 2]. Part of these advances is based on the concept of *k*-feasible cuts [3, 4], including algorithms for re-synthesis based on AIG rewriting [5]. Scalability is obtained by keeping the value of *k* small so that logic functions can be manipulated as vectors of integers. For instance, in [6] scalability is achieved by using functions of 16 or less inputs represented as binary truth-tables.

Algorithms for efficient cut computation are well known for single output cuts. Particularly, algorithms for exhaustive computation of *k*-feasible cuts were introduced by Cong [3] and Pan [4]. Chatterjee [7] introduced the concept of factor cuts, where exhaustive enumeration is avoided by making a separation between dag nodes and tree nodes in the AIG. The computation of factor cuts enables to work with cuts up to 16-inputs, which is not possible with the previous algorithms of exhaustive enumeration [3, 4]. All these algorithms for cut enumeration are only able to take the number *k* of inputs into account, not contemplating the benefits of multiple output reasoning. For example, in technology mapping using *k*-feasible cuts, logic duplication may occur during the step of covering, which is likely a problem on a design flow.

In this paper, we introduce the idea of controlling the number of outputs *l* in *k*-feasible cuts. Applications of *kl*-cuts may include peephole optimization [8], regularity extraction [9] and technology mapping. The use of *kl*-feasible cuts in peephole optimization is justified as an arbitrary portion of the circuit, potentially having multiple outputs, can be exchanged by another one by taking into account all signals which it affects at once. Its use in regularity extraction can be justified as many regular (logic) patterns are composed of multiple output circuits. This is especially true for arithmetic circuits, e.g. full adder and half adder library cells.

Research partially funded by Nangate Inc under a Nangate/UFRGS research agreement, by CNPq Brazilian funding agency, and by the European Community's Seventh Framework Programme under grant 248538 - Synaptic.

The remainder of this paper is organized as follows. Section II presents a state-of-the-art review. Section III discusses the concept of backcuts and *l*-feasible backcuts. Section IV shows the *kl*-cuts definition, and algorithms to enumerate *kl*-cuts on an AIG. Section V presents the results of the implemented algorithm. Section VI discusses some possible applications for the proposed *kl*-cuts, and Section VII concludes the paper.

II. BACKGROUND

This section provides a review of the state-of-the-art concerning AIG, *k*-feasible cuts [3, 4] and factor cuts [7].

A. AIG

And-Inverter-Graph (AIG), $\mathcal{G}$, is a specific type of Directed Acyclic Graph (DAG), where each node has either 0 incoming edges – *primary inputs* (PI) – or 2 incoming edges – AND nodes. Each edge can be complemented or not. Some nodes are marked as *primary outputs* (PO).

B. K-Feasible Cuts

A *cut* of a node *n* is a set of nodes *c* such that every path between a PI and *n* contains a node in *c*. A cut of *n* is *irredundant* if no subset of it is a cut. A *k*-feasible cut is an irredundant cut containing *k* or fewer nodes.

Let A and B to be two sets of cuts. Let the auxiliary operation $\bowtie$ to be:

$$A \bowtie B \equiv \{a \cup b | a \in A, b \in B, |a \cup b| < k\}$$

Let $\Phi_{\mathcal{K}}(n)$ to be the set of *k*-feasible cuts of $n \in \mathcal{G}$, and if *n* is an AND node, let n_1 and n_2 to be its inputs. Then, $\Phi_{\mathcal{K}}(n)$ is defined recursively as follows:

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

The $\bowtie$ operation can also easily remove the redundant cuts, by comparing the cuts with one another, or by making use of signatures [10].

C. Dag Nodes and Tree Nodes

A *dag node* is defined as a node with more than one node connected to its output. The nodes that are not dag ones are *tree nodes*. The set of all dag nodes in $\mathcal{G}$ is represented by $\mathcal{F}$, and the set of tree nodes by $\mathcal{T}$.

978-3-9810801-6-2/DATE10 © 2010 EDAA

A sub-graph of $\mathcal{G}$, $\mathcal{G}_{\mathcal{T}}$, induced by the nodes in $\mathcal{T}$ is a forest of trees. The root node of a tree in $\mathcal{G}_{\mathcal{T}}$ is either an input of a dag node or a PO. Consider a sub-graph T_n induced by a dag node n and the trees in $\mathcal{G}_{\mathcal{T}}$ that are inputs to it. T_n is a factor tree. In addition, when the root node of a tree in $\mathcal{G}_{\mathcal{T}}$ is a PO, the tree itself is also a factor tree. This way, each node n in $\mathcal{G}$ is contained by a single factor tree. In Fig. 1, the dag nodes are shaded, and its factor trees are delimited.

A leaf of a factor tree that is not a PI has dag nodes as its inputs. A factor tree along with the dag nodes that are inputs of its leaves is called a *factor leaf-DAG*. In Fig. 1, the factor leaf-DAG for the node x is its factor tree in conjunction with the nodes a and b.

D. Factor Cuts

Factor cuts are a collection of k-feasible cuts, grouped in two categories, *local cuts* and *global cuts*. The definition of these groups can vary according to the factorization scheme, and are detailed in [7]. However, the idea is to construct these two sets of cuts, and then expand them to generate a (possibly complete) set of k-factor cuts. The expansion process is explained as follows.

Let c to be a global cut of a node n, and let c_i to be a local cut of a node i belonging to c. If e is a cut defined as $e = \bigcup_i c_i$, and e is k-feasible, than e is a *1-step expansion* of n. The set of cuts obtained by expansion of the cut c is defined as 1-step(c).

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

In Fig. 1, expanding the node a in the cut $\{a, b, z\}$ by its local cut $\{p, q\}$, we get the cut $\{p, q, b, z\}$, which is a 1-step expansion of $\{a, b, z\}$.

Factor cuts allow algorithms that were conceived to use k-feasible cuts to work without the need of enumerating all cuts of every node. This calculation can be made on-the-fly as the data are needed.

III. L-Feasible Backcuts

The algorithms for computing cuts work from inputs to outputs. Computing kl-feasible cuts involves computing backward cuts – or *backcuts* – from outputs to inputs. The proposed backcuts are quite similar to cuts. However, instead of representing a set of nodes that can generate n, they represent a set of nodes that are influenced by n.

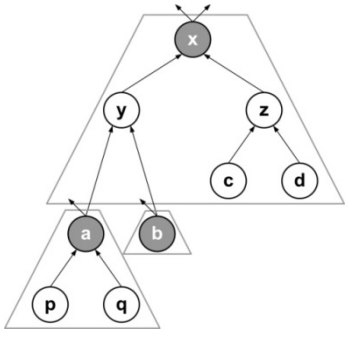

Figure 1. AIG illustrating cut factorization [7]. Nodes p, q, b, c and d are primary inputs. Node x is a primary output.

A backcut of a node n is a set of nodes c such that every path between n and a PO contains a node in c. A backcut is irredundant if no subset of it is a backcut. An l-feasible backcut is an irredundant backcut containing l or lesser nodes.

For convenience, let us define another operator, the operation $\bowtie_{i=m}^{n} x_i \equiv x_m \bowtie \cdots \bowtie x_n$. This attribution can be made since the $\bowtie$ operation is commutative.

Let $\Phi_{\mathcal{L}}(n)$ to be the set of l-feasible backcuts of n, and let n_i to be the i-th node connected to its output. We define $\Phi_{\mathcal{L}}(n)$ as:

$$\Phi_{\mathcal{L}}(n) \equiv \begin{cases} \{\{n\}\} & : n \text{ is a PO} \\ \{\{n\}\} \cup \left(\bowtie_{\mathrm{i}} \Phi_{\mathcal{L}}(n_{\mathrm{i}})\right) & : otherwise \end{cases}$$

As an example, in Fig. 2, $\Phi_{\mathcal{L}}(p) = \{\{p\}, \{r, s\}, \{s, t\}\}$.

A. Factor Backcuts

As it can be done when dealing with cuts, backcuts can be factored into two groups: global and local backcuts. A factorization very close to the partial cuts factorization scheme is proposed for backcuts, and similarly to the precursor scheme, the proposed algorithm cannot generate every l-feasible backcut by 1-step expansion. The definition is shown as follows.

Let $\Phi^{\dagger}_{\mathcal{LL}}(n)$ to be an auxiliary function:

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

Let $\Phi_{\mathcal{LL}}(n)$ define the set of local backcuts of the node n:

$$\Phi_{\mathcal{LL}}(n) \equiv \begin{cases} \{\{n\}\} & : n \text{ is a PO} \\ \{\{n\}\} \cup \left(\bowtie_{\mathrm{i}} \Phi^{\dagger}_{\mathcal{LL}}(n_{\mathrm{i}})\right) & : otherwise \end{cases}$$

For example, in Fig. 2, $\Phi_{\mathcal{LL}}(c) = \{\{c\}, \{p, q\}, \{p, s\}\}$.

And let $\Phi_{\mathcal{LD}}(n)$ denote the set of global backcuts of the node n:

$$\Phi_{\mathcal{LD}}(n) \equiv \begin{cases} \{\{n\}\} & : n \text{ is a PO} \\ \{\{n\}\} \cup \left(\bowtie_{\mathrm{i}} \Phi_{\mathcal{LD}}(n_{\mathrm{i}})\right) & : n \in \mathcal{F} \\ \bowtie_{\mathrm{i}} \Phi_{\mathcal{LD}}(n_{\mathrm{i}}) & : otherwise \end{cases}$$

In Fig. 2, $\Phi_{\mathcal{LD}}(c) = \{\{c\}, \{p, s\}, \{s, t\}\}$.

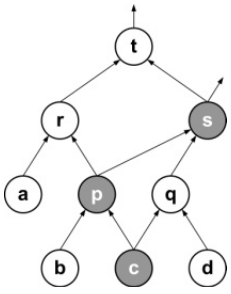

Figure 2. AIG demonstrating backcut factorization. Nodes a, b, c and d are primary inputs. Nodes s and t are primary outputs.

This definition allows the local backcuts to contain only nodes belonging to its factor leaf-DAG, and let the global backcuts to transgress the factor leaf-DAG barriers, allowing the reconstruction of many of the *l*-feasible backcuts by 1-step expansion. The expansion of backcuts works in the same way as for cuts. The global backcuts have their nodes expanded by the local backcuts.

As an example of 1-step expansion, consider the backcut $\{p, s\}$ in Fig. 2. Expanding the node p by its local backcut $\{r, s\}$, we get to the backcut $\{r, s\}$. Thus, $\{r, s\}$ is a 1-step expansion of $\{p, s\}$.

IV. KL-FEASIBLE CUTS

Cuts are an efficient way of representing a region of an AIG regarding one signal generation. However, when it comes to multiple output regions multiple cuts would be needed.

A *kl-cut* defines a sub-graph $\mathcal{G}_{kl}$ of $\mathcal{G}$ which has no more than k inputs and no more than l outputs. It is represented as two sets of nodes $\{\mathcal{G}_k, \mathcal{G}_l\}$: being $\mathcal{G}_k$ the inputs set and $\mathcal{G}_l$ the outputs set. If a node n belongs to a path between $n_k \in \mathcal{G}_k$ and $n_l \in \mathcal{G}_l$, and $n \notin \mathcal{G}_k$, then n is contained in $\mathcal{G}_{kl}$. Notice that all nodes in $\mathcal{G}_l$ are contained in $\mathcal{G}_{kl}$. However, $\mathcal{G}_{kl}$ does not contain any node of $\mathcal{G}_k$.

A *kl*-cut is said to be complete when all the following conditions are met:

c1: Every path between a PI and a node $n_l \in \mathcal{G}_l$ contains a node in $\mathcal{G}_k$;

c2: Every path between a node contained in $\mathcal{G}_{kl}$ and a PO contains a node in $\mathcal{G}_l$;

c3: No *kl*-cut defined by a subset of $\mathcal{G}_k$ and the same $\mathcal{G}_l$ is complete;

c4: No *kl*-cut defined by the same $\mathcal{G}_k$ and a subset of $\mathcal{G}_l$ is complete.

A. KL-Cuts Generation Algorithm

The objective of this algorithm is to find *kl*-cuts that have shared nodes on the generation of more than one output, that is, nodes that belong to *k*-feasible cuts of more than one output.

Fig. 3 shows a pseudo-code for *kl*-cuts enumeration. It starts enumerating all *k*-feasible cuts and all *l*-feasible backcuts on the circuit. Each computed backcut generates a set of *kl*-cuts. The function on line 5 combines the *k*-feasible cuts of the nodes belonging to the current backcut *d*. Let d_i to be a node of *d*. Let *p* to be $p = \bowtie_i \Phi_{\mathcal{K}}(d_i)$. This way, *p* is a set of input groups p_i, and each one defines a *kl*-cut $\{p_i, d\}$. Although, not every resulting *kl*-cut is complete, because condition c2 is not assured. So, the function on line 8 adds nodes to the outputs set in order to make the *kl*-cut complete, or else discards the *kl*-cut. If a node n_{kl} belonging to $\mathcal{G}_{kl}$ has as output a node that does not belong to $\mathcal{G}_{kl}$, then n_{kl} must be added to $\mathcal{G}_l$. If $\mathcal{G}_l$ still have no more than l nodes, $\mathcal{G}_{kl}$ is a complete *kl*-cut. On this implementation the *check_and_fix()* function also discards *kl*-cuts that are not connected.

```
1. compute_klcuts(k, l, aig) {
2.   kcuts = compute_kcuts(aig, k)
3.   lcuts = compute_lcuts(aig, l)
4.   for (each lcut in lcuts) {
5.     p = combine_kcuts(lcut)
6.     for (each pi in p) {
7.       klcut = create_klcut(pi, lcut)
8.       if (check_and_fix(klcut))
9.         klcuts.add(klcut)
10.    }
11.  }
12.  return klcut
13.}
```

Figure 3. Pseudo-code for *kl*-cuts calculation.

For instance, in Fig. 4 (a), starting by the backcut $\{u, v\}$ generated by the node *s*, the cuts $\{a, b, s\}$ from *u* and $\{s, g, h\}$ from *v* are combined, generating the incomplete *kl*-cut $\{\{a, b, s, g, h\}, \{u, v\}\}$. The last step adds the node *r* to the outputs set, resulting on the complete *kl*-cut $\{\{a, b, s, g, h\}, \{r, u, v\}\}$ containing the nodes *u*, *v*, *r* and *t*.

To reduce the number of calculated *kl*-cuts, one can use only global cuts and global backcuts in the process, which generate fewer and larger *kl*-cuts. However, the covering of the circuit may get compromised. To ensure the covering, an additional round of *kl*-cuts generation could be done, this time using only local cuts and backcuts, possibly only over the previously uncovered portion of the AIG.

As an example, let us consider the AIG shown in Fig. 4. If the *kl*-cuts, with $k = 5$ and $l = 3$ (or simply 5-3-cuts), are computed based only on global cuts and backcuts one possible covering for the circuit is shown in Fig. 4 (a), which is composed by the *kl*-cuts $\{\{a, b, s, g, h\}, \{r, u, v\}\}$ and $\{\{c, d, e, f\}, \{s\}\}$. On the other hand, the circuit cannot be covered by using only 3-2-cuts generated by global cuts and backcuts. Under these conditions, only the nodes *r*, *t*, *u* and *v* can be covered. To complete the covering, local cuts and backcuts must be used, and a possible covering is shown in Fig. 4 (b).

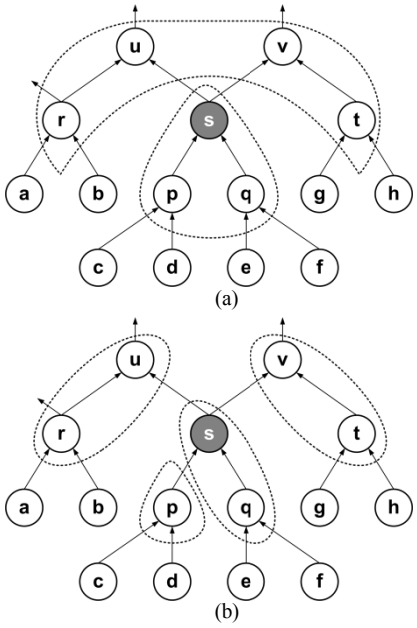

Figure 4. AIG illustrating covering. Nodes *a*, *b*, *c*, *d*, *e*, *f*, *g* and *h* are primary inputs. Nodes *u* and *v* are primary outputs. (a) A covering using 5-2-cuts. (b) A covering using 3-1-cuts.

```
1.  compute_unbounded_klcuts(l, aig) {
2.    lcuts = compute_lcuts(aig, l)
3.    for (each lcut in lcuts) {
4.      inputs = {}
5.      for (each node in lcut)
6.        add_node(node, aig, inputs)
7.      klcuts.add(create_klcut(inputs, lcut))
8.    }
9.    return klcuts
10. }
11. add_node(node, aig, inputs) {
12.   if (lcuts_ok(node) || node is PI) {
13.     add_node(input1, aig, inputs)
14.     add_node(input2, aig, inputs)
15.   } else
16.     inputs.add(node)
17. }
```

Figure 5. Pseudo-code for unbounded *kl*-cuts calculation.

B. Unbounded KL-Cuts

Another use for the *l*-feasible backcuts is the construction of *unbounded kl-cuts*, that are very similar to regular *kl*-cuts, although they have no restriction on *k*, i.e. the number of inputs.

Fig. 5 shows the pseudo-code for the algorithm. First of all, the *l*-feasible backcuts are calculated. Each backcut is the outputs set of a *kl*-cut, so the inputs set needs to be found. The graph is recursively run in depth first order (function *add_node*). Each node that either is a PI, or does not have a backcut composed exclusively by nodes on the outputs set (function *lcuts_ok*), is added to the inputs group.

Notice that by applying this algorithm, any *l*-feasible backcut in the AIG leads to exactly one unbounded *kl*-cut, and this computation is performed in linear time with respect to the number of nodes. See also that the value of *k* is self adjusted by the topology and convergence of the graph.

Instead of computing all backcuts, it is possible to use only global backcuts, speeding up the process while not compromising the circuit covering. Moreover, when using only global backcuts for calculating unbounded *kl*-cuts, the generated sub-graphs have factor trees as its elementary blocks. In other words, these *kl*-cuts are constituted by one or more complete factor trees.

It is of particular interest the unbounded *kl*-cuts with $l = 1$, and based on global backcuts. In this case the sub-graph may contain more than one factor trees that are reconvergent to the sole output that it presents. For instance, in Fig. 6, the unbounded *kl*-cut for the global backcut $\{t\}$ is $\{\{a, b, c, d\}, \{t\}\}$. Observe the incorporation of 3 factor trees on this *kl*-cut. Moreover, there is only one possible covering of a circuit by *kl*-cuts of this specific type, and its computation is done in linear time with respect to the number of nodes of the AIG. This performs a full clusterization of the circuit.

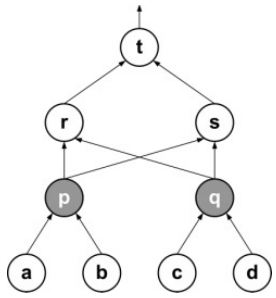

Figure 6. AIG exemplifying unbounded *kl*-cuts. The *kl*-cut r = $\{\{a, b, c, d\}, \{t\}\}$ is an unbounded *kl*-cut.

An unbounded *kl*-cut with more than one output will have all nodes contained on the unbounded *kl*-cuts for each single output along with the nodes that are not used for other outputs. For instance, in Fig. 2, the backcut $\{r\}$ leads to the unbounded *kl*-cut $\{\{a, p\}, \{r\}\}$, the backcut $\{s\}$ to $\{\{p, c, d\}, \{s\}\}$, and the backcut $\{r, s\}$ to $\{\{a, b, c, d\}, \{r, s\}\}$.

Observe that on this process no *k*-feasible cuts need to be calculated. Also, the use of global backcuts instead of all backcuts does not compromise the covering of the whole AIG into *kl*-cuts.

V. RESULTS

The algorithm was implemented in Java language and all results were obtained by execution on a 2.4GHz Intel Pentium IV with 1GB of RAM. The implemented program reads BAF files as inputs, which were generated from EQN files using the ABC tool [11], after running the *'dc2 –l'* command twice.

Table I shows a brief comparison between complete *l*-feasible backcuts enumeration and factor backcuts enumeration. For $l = 2$, the factor backcuts represent 89% the number of backcuts, and the run-time was 66% of the total.

TABLE I. COMPARISON BETWEEN ALL L-FEASIBLE BACKCUTS ENUMERATION AND FACTOR BACKCUTS ENUMERATION.

Name	Nodes		All *l*-feasible backcuts				Factor backcuts					
			l = 2		*l* = 4		*l* = 2			*l* = 4		
	Total	*% Dag*	*Total*	*Time (s)*	*Total*	*Time (s)*	*Local*	*Global*	*Time (s)*	*Local*	*Global*	*Time (s)*
C1355	433	38.11	1707	0.032	4457	0.344	915	1081	0.078	1109	2125	0.125
C1908	393	41.98	1805	0.015	7350	0.421	966	997	0.047	1252	2253	0.109
C2670	777	22.52	4535	0.078	26286	1.578	2645	1295	0.062	3305	1946	0.093
C3540	975	27.18	6986	0.078	64779	334.703	6176	1361	0.094	41228	3764	258.703
C5315	1522	26.87	9960	0.109	29807	4.047	4787	3248	0.094	5929	5349	0.203
C6288	1902	73.87	7682	0.046	140157	3.875	4638	5871	0.078	4669	90755	1.875
C7552	1625	38.34	16790	0.188	71183	5.391	5037	4452	0.141	6357	8735	0.250
s13207	2757	27.64	10699	0.078	20605	22.422	10279	3234	0.093	15290	3785	22.531
s15850	3374	29.02	13391	0.125	54001	4.578	10376	4712	0.125	20058	6841	1.218
s35932	10841	30.84	90148	1.250	122938	9.547	31938	23485	0.344	35818	25295	0.656
s38417	9795	25.88	41122	0.250	255430	15.485	28832	14035	0.235	42600	23873	0.797
s38584	11306	27.56	39848	0.312	135238	10.594	34007	14007	0.297	75167	18039	4.969
Avg.	3808	34.15	20389	0.213	77686	34.415	11716	6482	0.141	21065	16063	24.294

TABLE II. COMPARISON BETWEEN FULL KL-CUTS SEARCH AND GLOBAL KL-CUTS SEARCH.

Name	All *kl*-cuts						Global *kl*-cuts					
	4-2-cuts			*6-4-cuts*			*4-2-cuts*			*6-4-cuts*		
	Total	*Size*	*Time (s)*	*Total*	*Size*	*Time (s)*	*Total*	*Size*	*Time (s)*	*Total*	*Size*	*Time (s)*
C1355	2778	4.19	0.578	36186	8.76	20.969	676	5.45	0.250	4346	10.33	1.953
C1908	2300	3.59	0.422	24838	7.38	20.266	563	4.13	0.172	3330	7.65	1.468
C2670	3313	3.18	0.797	21847	6.35	35.938	221	4.34	0.141	450	7.28	0.344
C3540	5392	2.61	1.063	56782	5.47	413.547	459	2.94	0.125	1463	5.57	0.609
C5315	7680	3.00	2.297	59529	6.39	63.422	629	3.86	0.250	2984	7.30	1.641
C6288	10836	3.21	2.797	345069	9.12	1083.828	4475	3.48	1.312	89946	9.79	245.172
C7552	9624	3.31	3.296	111160	7.10	224.500	1515	3.40	0.453	7774	7.08	5.203
s13207	8862	2.41	1.438	58607	5.11	38.266	1149	2.62	0.218	3220	4.60	0.593
s15850	13928	2.62	2.563	113296	5.48	70.219	1614	2.71	0.328	5144	5.00	1.125
s35932	54015	3.60	44.907	391684	6.75	817.985	6619	4.14	6.735	11910	7.23	20.281
s38417	44322	2.61	6.062	351974	5.38	345.703	3337	3.04	0.672	8069	5.56	2.813
s38584	44425	2.40	20.515	369003	5.22	294.844	4218	2.23	1.656	13989	5.08	10.375
Avg.	17290	3.06	7.228	161665	6.54	285.791	2123	3.53	1.026	12719	6.87	24.298

For $l = 4$, the factor backcuts were 48% of the total, calculated in 71% of the time. Notice that for the circuit C3540 the time taken to compute the backcuts was particularly larger than the others. This is due to the fact that some nodes have a huge number of backcuts. The utilization of a limit factor of backcuts per node would significantly reduce this time in despite of the quality of results.

Table II shows the results for constructing *kl*-cuts from all cuts and backcuts enumeration, and from global factor cuts and backcuts enumeration. For 4-2-cuts, factored enumeration produced 12% of the number of *kl*-cuts generated from full enumeration, in 14% of the run-time. For 6-4-cuts, the number of *kl*-cuts produced by factored enumeration was about 8% of total enumeration, with 9% of the run-time. It is also noticeable that *kl*-cuts generated by factored enumeration have more nodes on average than *kl*-cuts produced by complete enumeration, which is shown on the columns labeled *Size*.

A comparison between the circuit covering by factor trees and by unbounded *kl*-cuts is established on Table III. The column labeled *Total* shows the number of sub-graphs necessary to perform the covering, the column *Size* shows the average number of nodes per sub-graph, and the column *Mean k* shows the average number of inputs of the sub-graphs. Notice that the increase in the number of nodes (2.97 to 4.32) is proportionally larger than the increase in the number of inputs (3.15 to 3.87). This indicates that the unbounded *kl*-cuts are indeed reconvergent graphs and not single trees.

Finally, Table IV compares all calculated unbounded *kl*-cuts. For $l = 1$ the cuts have on average less than 4 inputs and 4 internal nodes, and for $l = 4$ more than 14 inputs with nearly 30 nodes. But the generation of these cuts is much faster than generating *kl*-cuts by combining *k*-feasible cuts: the average time for computing unbounded *kl*-cuts for $l = 1$ was 0.1 seconds and for $l = 4$ was 1.17 seconds.

TABLE III. COMPARISON BETWEEN THE COVERING USING UNBOUNDED KL-CUTS AND USING FACTOR TREES.

Name	Factor Trees			Unbounded *kl*-cuts ($l = 1$)		
	Total	*Size*	*Mean k*	*Total*	*Size*	*Mean k*
C1355	164	2.390	2.110	68	5.765	3.676
C1908	157	2.293	2.338	103	3.495	3.019
C2670	134	4.052	3.784	59	9.203	6.390
C3540	230	4.022	4.248	192	4.818	4.865
C5315	350	3.840	3.626	241	5.577	4.581
C6288	1404	1.332	2.330	1403	1.333	2.331
C7552	607	2.336	2.590	370	3.832	3.346
s13207	658	3.125	3.585	643	3.198	3.645
s15850	941	2.935	3.325	852	3.242	3.563
s35932	3355	2.706	2.631	2320	3.913	3.303
s38417	2434	3.341	3.595	2245	3.622	3.751
s38584	2980	3.302	3.577	2538	3.877	3.996
Avg.	1118	2.973	3.145	920	4.323	3.872

VI. APPLICATIONS

A. Technology Mapping

Most of standard cell libraries have multiple output cells, e.g. full adders. Similarly, library free technology mappers could use multiple output cells to significantly reduce area of a circuit. Moreover, current FPGAs have multiple output LUTs [12, 13].

A simple algorithm to perform a full covering of a circuit by *kl*-cuts was elaborated. This algorithm has no intention of achieving the state-of-the-art on mapping, but to confirm the potential of using *kl*-cuts on technology mapping. The greedy algorithm searches for local maxima. The covering is performed from the inputs to the outputs. At each iteration the largest possible *kl*-cut is chosen, and all *kl*-cuts with overlapping nodes are eliminated from the solution space. These iterations are repeated until the circuit is fully covered. Two effort levels are defined. Using high effort level, all

TABLE IV. COMPARISON BETWEEN ALL COMPUTED UNBOUNDED KL-CUTS.

Name	$l = 1$			$l = 4$		
	Total	*Size*	*Mean k*	*Total*	*Size*	*Mean k*
C1355	164	4.439	2.988	1108	31.475	12.588
C1908	157	3.490	2.911	997	22.412	11.102
C2670	134	6.478	4.672	287	65.808	25.223
C3540	230	4.457	4.543	811	35.144	22.436
C5315	350	6.666	4.669	1718	53.702	20.162
C6288	1404	1.333	2.331	16972	19.303	11.153
C7552	607	4.699	3.634	3175	42.709	17.708
s13207	658	3.251	3.629	1349	14.746	12.459
s15850	941	3.275	3.564	3157	20.183	13.381
s35932	3355	4.017	3.030	9354	12.485	6.249
s38417	2434	3.598	3.749	5973	21.454	12.669
s38584	2980	3.500	3.735	7211	16.673	10.856
Avg.	1118	4.100	3.621	4343	29.675	14.666

TABLE V. COMPARISON BETWEEN COVERING USING KL-CUTS WITH DIFFERENT EFFORT LEVELS AND MAPPING USING ABC TOOL [11].

Name	Low effort, 5-2-cuts					High effort, 5-2-cuts					ABC
	Time (s)	*KL-Cuts*	*% MOC*	*LUTs*	*% MOL*	*Time (s)*	*KL-Cuts*	*% MOC*	*LUTs*	*% MOL*	*LUTs*
C1355	0.844	79	30.38	56	83.93	2.578	76	34.21	54	88.89	68
C1908	0.531	79	58.23	67	86.57	1.782	76	68.42	64	100.00	105
C2670	0.906	147	10.20	112	44.64	3.985	147	32.65	112	74.11	145
C3540	2.172	306	17.32	243	47.74	7.844	264	48.86	219	79.45	276
C5315	2.813	350	18.00	279	48.03	14.844	327	44.65	271	74.54	324
C6288	5.984	347	96.54	341	100.00	13.453	347	95.39	339	100.00	501
C7552	2.453	347	45.24	280	80.00	15.796	320	61.25	272	89.71	372
s13207	4.546	639	32.39	503	68.19	19.64	582	55.33	482	87.55	703
s15850	6.782	844	28.55	659	64.64	38.438	810	47.53	647	84.70	958
s35932	87.047	2470	29.15	1804	76.83	721.297	2196	56.69	1735	98.33	2493
s38417	43.188	2577	16.61	1942	54.74	351.578	2116	62.48	1781	93.04	2654
s38584	93.11	2712	27.65	2231	55.18	444.282	2574	43.40	2107	75.18	2655
Avg.	20.8647	908	34.19	710	67.54	136.293	820	54.24	674	87.12	938

kl-cuts are present on the solution space of the algorithm. Using low effort level, only global and local *kl*-cuts are available to the covering algorithm.

Table V shows some results regarding covering with 5-2-cuts. These constraint values were chosen based on the current FPGAs capabilities [12, 13].

Columns named *KL-Cuts* show the number of cuts for the resulting covering. Columns *% MOC* are the percentage of cuts that have multiple outputs. Two single output cuts can be implemented by a single multiple output LUT, as long as the sum of the number of inputs of these cuts is no larger than the LUT number of inputs. For that reason the columns labeled *LUTs*, which show the number of LUTs necessary to implement the covering, present a lower value than the columns *KL-Cuts*. The columns *% MOL* represents the percentage of used LUTs that uses both outputs. Finally, the *ABC* column shows the number of LUTs used by mapping the circuit running the *'st; dch; if –C 12; mfs –W 4 –M 5000'* commands [14] on ABC four times, and picking the best result, with a library containing LUTs with up to 5 inputs, subtracting the constants, buffers and inverters.

The algorithm finds a good fraction of *kl*-cuts which are naturally multiple output, and most of the single output *kl*-cuts found have few inputs, which allows its combination leading to a high utilization of multiple output LUTs (more than 85% for high effort), resulting on fewer LUTs (30% reduction comparing to ABC mapping).

B. Other Applications

Regularity extraction is important for improving the yield of fabrication [9]. Maintaining a hash table of *kl*-cuts structure and/or functions could help on an implementation of an algorithm to perform regularity extraction on a circuit.

Another application for *kl*-cuts is to perform peephole optimizations. By defining a sub-graph on an AIG, one can replace it by any other sub-graph that implements the same outputs, but minimizing a given cost function.

VII. CONCLUSIONS AND FUTURE WORK

The concept of *kl*-feasible cuts was introduced, which allows controlling both the number *k* of inputs and the number *l* of outputs in the computation of circuit cuts. Algorithms for computing *kl*-feasible cuts were presented and results have shown the usefulness of the method. The concept of factor cuts was also extended to *kl*-cuts. A novel algorithm for computing *kl*-feasible cuts with unbounded *k* was presented. This has shown the viability of computing *kl*-cuts with large *k*. Results about the implemented covering are encouraging, even though it is a proof of concept algorithm. Further work is necessary in the use of *kl*-feasible cuts in peephole optimization, AIG rewriting, regularity extraction, and the elaboration of an enhanced technology mapping algorithm.

REFERENCES

[1] Ling, A. C., Zhu, J., "Scalable Synthesis and Clustering Techniques Using Decision Diagrams", IEEE Trans. on CAD. 2008.

[2] Mishchenko, A., Brayton, R., "Scalable Logic Synthesis using a Simple Circuit Structure", Int'l Workshop on Logic & Synthesis, 2006.
http://www.eecs.berkeley.edu/~alanmi/publications/2006/iwls06_sls.pdf

[3] Cong J., Wu, C., Ding, Y., "Cut Ranking and Pruning: Enabling A General and Efficient FPGA Mapping Solution", Int'l Symp. on FPGA, 1999.

[4] Pan, P., Lin C., "A New Retiming-based Technology Mapping Algorithm for LUT-based FPGAs", Int'l Symp. on FPGA, 1998.

[5] Mishchenko, A., Chatterjee, S., Brayton, R., "DAG-Aware AIG Rewriting: A Fresh Look at Combinational Logic Synthesis", Design Automation Conference, 2006.

[6] Mishchenko, A., Brayton, R., Chatterjee, S., "Boolean Factoring and Decomposition of Logic Networks", Int'l Conf. on CAD, 2008.

[7] Chatterjee, S., Mishchenko, A. and Brayton, R., "Factor Cuts", Int'l Conf. on CAD, 2006.

[8] Werber, J., Rautenbach, D., Szegedy, C., "Timing Optimization by Restructuring Long Combinatorial Paths", Int'l Conf. on CAD, 2007.

[9] Rosiello, A. P. E., Ferrandi, F., Pandini, D., Sciuto, D., "A Hash-based Approach for Functional Regularity Extraction During Logic Synthesis", IEEE Comp. Soc. Annual Symp. on VLSI, 2007.

[10] Mishchenko, A., Chatterjee, S., Brayton, R., "Improvements to Technology Mapping for LUT-Based FPGAs", Int'l Symp. on FPGA, 2006.

[11] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification".
http://www.eecs.berkeley.edu/~alanmi/abc

[12] Xilinx, "Achieving Higher System Performance with the Virtex-5 Family of FPGAs", White Paper, 2006. http://www.xilinx.com/

[13] Altera, "Improving FPGA Performance and Area Using an Adaptive Logic Module", White Paper, 2004. http://www.altera.com/

[14] Mishchenko, A., Brayton, R., Jiang, J. H. R., Jang, S., "Scalable Don't-Care-Based Logic Optimization and Resynthesis", Int'l Symp. on FPGA, 2009.