Dual-Output LUT Merging during FPGA Technology Mapping

Feng Wang
Peking University
Beijing, China
yzwangfeng@pku.edu.cn

Liren Zhu
Peking University
Beijing, China
zlr_zlr@pku.edu.cn

Jiaxi Zhang
Peking University
Beijing, China
zhangjiaxi@pku.edu.cn

Lei Li
Huawei Technologies Co., Ltd.
Shenzhen, China
lilei291@huawei.com

Yang Zhang
Huawei Technologies Co., Ltd.
Shenzhen, China
zhangyang167@huawei.com

Guojie Luo
Peking University
Beijing, China
gluo@pku.edu.cn

ABSTRACT

Modern commercial Field-Programmable Gate Array (FPGA) architectures support dual-output look-up tables (LUTs). If the number of total inputs in two small LUTs do not exceed the constraint, e.g., 5 in Xilinx UltraScale+ series, we can pack them into one dual-output LUT to reduce area, i.e., the number of LUTs. However, previous works have not fully utilized this feature. They usually generate single-output LUTs in the technology mapping phase and merge LUTs in a later packing phase. In this situation, they cannot get LUT merging information during technology mapping and will generate some single-output LUTs that are not suitable for merging.

In this work, we directly generate dual-output LUTs in the technology mapping phase and propose a novel cut-based mapping flow. The mapping flow consists of several mapping passes with different cut selection metrics. In each pass, we first compute the priority single-output cuts of each node. Then, we merge dual-output cuts from the priority cuts to generate a mapped LUT netlist. Finally, we do some local refinement to further improve the merging rate and reduce area. Experimental evaluation shows that our mapping flow can merge up to 14.89% more LUTs and save up to 13.98% area on average, compared to the state-of-the-art technology mapping tool ABC, without worsening the total delay.

CCS CONCEPTS

• **Hardware → Technology-mapping**; *Circuit optimization.*

KEYWORDS

FPGA, dual-output LUT, technology mapping, cut

ACM Reference Format:
Feng Wang, Liren Zhu, Jiaxi Zhang, Lei Li, Yang Zhang, and Guojie Luo. 2020. Dual-Output LUT Merging during FPGA Technology Mapping. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20), November 2–5, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3400302.3415617

ICCAD '20, November 2–5, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415617

1 INTRODUCTION

Technology mapping is an important phase in Field-Programmable Gate Array (FPGA) design. It transforms a technology-independent Boolean circuit into a functionally equivalent netlist of look-up tables (LUTs). An FPGA comprises of a number of K-input LUTs (also denoted as LUT-K), each of which can implement an arbitrary Boolean function of no more than K inputs. K is an architecture parameter defined by the FPGA. We can implement any Boolean function on an FPGA using limited K-input LUTs.

A mainstream approach of technology mapping is based on cut enumeration [5, 21]. A cut of a node is a set of nodes in its transitive fanins that can block all of the paths from the primary inputs to it. A cut with K nodes can be implemented by a K-input LUT. The cut-based approach first selects a representative cut for each node in the Boolean circuit. Then, it selects a subset of cuts to cover the whole circuit. The corresponding LUTs of these cuts are used in the final mapping.

Technology mapping usually targets delay (the depth of the LUT netlist) and area (the number of LUTs) minimization. To achieve this goal, previous works [22] have proposed several heuristic metrics for cut selection, e.g., cut size, area flow (a global area estimation), exact flow (a local area estimation). The mapping procedure ranks cuts with different heuristic metrics in different mapping passes.

However, the cut-based approach and these heuristic metrics are designed for conventional single-output LUT architectures. We cannot apply them to dual-output LUT architectures directly. Several modern FPGAs support dual-output LUTs. A LUT can be used as a large single-output LUT or fractured into two small LUTs. Two small LUTs can be merged if both their independent input size and sharing input size do not exceed the constraint. For example, in Xilinx UltraScale+ series [3], a 6-input LUT can be fractured into two LUTs with up to 5 inputs. This feature can be utilized to further improve the mapping quality, including delay, area, and even robustness [14, 15].

To support this feature, existing tools separate single-output LUT generation and dual-output LUT merging into two design phases. The technology mapping phase remains unchanged, and a later packing phase merges LUTs. Since minimizing the number of single-output LUTs does not necessarily minimize the number of dual-output LUTs, existing technology mapping approaches usually cannot get the optimal result. Although previous works design some heuristic metrics, e.g., edge recovery in WireMap [13], to improve the LUT merging rate, the LUT merging rate remains low. Taking

13 typical cases in the EPFL benchmark and UltraScale+ series (see Section 6) as an example, only less than 10% LUTs can be merged after ABC [25] technology mapping, while the LUT merging rate is close to 20% in our work.

In this work, we address this problem by proposing a novel technology mapping flow. The mapping flow generates single-output LUTs and merges dual-output LUTs simultaneously in the technology mapping phase for the first time. The main contributions are listed as follows:

- We propose a cut-based mapping flow with several mapping passes. In each mapping pass, we perform global dual-output cut merging from priority cuts and then do some local refinement for a higher merging rate.
- We extend the heuristic metrics for single-output cut selection to support dual-output cut selection. The extended heuristic metrics prefer dual-output cuts with more sharing inputs.
- We experimentally evaluate our mapping flow using the modern EPFL benchmark and show that our flow can merge up to 14.89% more LUTs and save up to 13.98% area compared to the state-of-the-art technology mapping tool.

2 PRELIMINARIES

2.1 Combinational Circuit

A combinational circuit can be represented by a directed acyclic graph (DAG) where nodes correspond to logic gates and edges correspond to wires connecting the gates. Each node in the graph has zero or more fanins and fanouts. The primary inputs (PIs) are nodes without fanins, while the primary outputs (POs) are nodes without fanouts.

A fanin cone of a node is a subset of nodes in the graph that can be reached from the given node through the transitive fanin edges. A maximum fanout free cone (MFFC) of this node is a subset of its fanin cone, such that every path from a node in the subset to the POs passes through this node.

A graph is called a K-bounded subject graph if the number of fanins of any node does not exceed K. Any given network can be decomposed to create a K-bounded graph suitable for technology mapping. For example, we can use ABC to transform an arbitrary graph to an AND-INV graph (AIG) that only contains two-input AND gates and inverters.

2.2 Single-Output Cut

Given a single node as the root ($root(c)$), a single-output cut c [8] is a set of nodes that satisfies every path from a PI to the root passes through at least one node in the set. Nodes in this set are also called leaves. A trivial cut only contains the root itself and does not cover any nodes. A non-trivial cut covers the root and all of the nodes on the path from the leaves to the root except the leaves. A cut is K-feasible if the number of leaves in it does not exceed K. A cut is dominated if another cut rooted at the same node also covers the nodes it covers.

Let $\Phi(n)$ denote the set of K-feasible cuts rooted at the node n, we define the operation $\otimes$ as:

$$\Phi(n_1) \otimes \Phi(n_2) = \{c_1 \cup c_2 | c_1 \in \Phi(n_1), c_2 \in \Phi(n_2), |c_1 \cup c_2| \leq K\}.$$

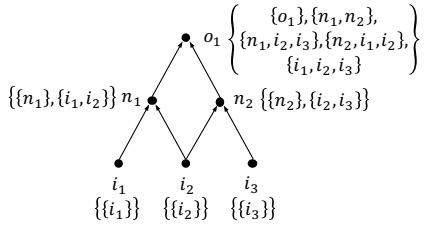


Figure 1: A cut enumeration example. We list all of the 3-feasible single-output cuts next to each node.

We can compute $\Phi(n)$ by combining the trivial cut and the cut set of its fanins:

$$\Phi(n) = \begin{cases} \{\{n\}\} & n \in PI, \\ \{\{n\}\} \cup (\Phi(n_1) \otimes \Phi(n_2) \otimes ...) & fanin(n) = \{n_1, n_2, ...\}. \end{cases}$$

For a K-bounded subject graph, we can enumerate K-feasible cuts of all nodes in a topological order, such that the fanin cuts are available when computing the cut set of the root. Figure 1 gives a cut enumeration example with $K = 3$. Three PIs i_1, i_2, i_3 only have a trivial cut each. As the depth increases, the number of cuts increases exponentially. Among the cuts rooted at o_1, $\{n_1, i_2, i_3\}$ and $\{n_2, i_1, i_2\}$ are dominated by $\{i_1, i_2, i_3\}$.

Cuts are widely used in technology mapping. Since a K-input LUT can implement an arbitrary Boolean function, we can implement a K-feasible cut with a K-input LUT. The conventional cut-based technology mapping for K-input single-output LUTs is applied to K-bounded subject graphs. Usually, a mapping procedure first enumerates K-feasible cuts in the graph. Then, it selects one cut as the representative cut for each node. Finally, it selects a subset of nodes whose representative cuts can cover all non-PI nodes. These nodes are used in this mapping.

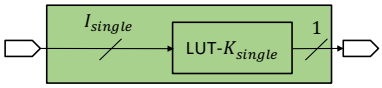
2.3 Dual-Output LUT

In several modern FPGAs, a large LUT with 6 inputs or more can be "fractured" into two small LUTs, which helps further improve delay and area. When implementing a large circuit, we can map the timing-critical part with single-output LUTs to reduce delay. We can also map the non-timing-critical part with dual-output LUTs to reduce area. Combining two small LUTs is called LUT merging.

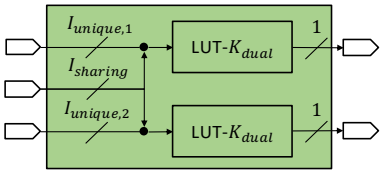
Figure 2 shows two modes of a LUT. This LUT has two usable pins in this architecture. In the single-output mode, it can implement a large LUT with no more than I_{single} (= K_{single}) inputs. In the dual-output mode, it can implement two independent small LUTs that meets three constraints:

- Two LUTs have no more than I_{dual} independent inputs in total, i.e., $I_{unique,1} + I_{sharing} + I_{unique,2} \leq I_{dual}$.
- Each LUT has no more than K_{dual} inputs, i.e., $I_{unique,i} + I_{sharing} \leq K_{dual}$ $(i = 1, 2)$.
- Two LUTs can have no more than $I_{sharing}$ sharing inputs.

Table 1 gives the parameters of three typical FPGA series that support dual-output LUTs. Intel ALM series has several configuration parameters. Here lists a typical set of parameters. We can see that K_{dual} is usually no more than K_{single}.



(a) The single-output mode.



(b) The dual-output mode. The total input bit width is $I_{unique,1} + I_{sharing} + I_{unique,2}$.

Figure 2: Two modes of a LUT. I_x represents an input bit width, and K_x represents a LUT size.

Table 1: Parameters of three typical FPGA series.

FPGA	UltraScale+ [3]	Versal [11]	Intel ALM [1, 6]
I_{single} (K_{single})	6	6	8
I_{dual}	5	6	8
$I_{sharing}$	5	6	4
K_{dual}	5	6	6

3 CUT-BASED LUT MERGING PROBLEM

3.1 Dual-output Cut Extension

To state this problem, we first extend the cut definition to dual outputs. For two single-output cuts c_1 and c_2 with different roots, we define their dual-output cut $c_1 \cup c_2$ as the union of them. The roots of them $root(c_1)$ and $root(c_2)$ become the root of the dual-output cut. $c_1 \cup c_2$ covers the nodes covered by c_1 and c_2. Similar to the single-output cut, the dual-output cut satisfies that every path from a PI to either root passes through at least one node in the cut.

Let $\Phi(n_1, n_2)$ denote the subset of dual-output cuts rooted at the node n_1 and n_2 that satisfy the constraints in Section 2.3, we can compute $\Phi(n_1, n_2)$ after $\Phi(n_1)$ and $\Phi(n_2)$ are enumerated:

$$\Phi(n_1, n_2) = \{c_1 \cup c_2 | c_1 \in \Phi(n_1), c_2 \in \Phi(n_2), |c_1 \cup c_2| \le I_{dual}, \\ |c_1| \le K_{dual}, |c_2| \le K_{dual}, |c_1 \cap c_2| \le I_{sharing}\}.$$

For example, in Figure 1, assuming that $I_{dual} = 3$ and $I_{sharing} = K_{dual} = 2$, we can get $\Phi(n_1, n_2) = \{\{n_1, n_2\}, \{n_1, i_2, i_3\}, \{n_2, i_1, i_2\}, \{i_1, i_2, i_3\}\}$.

3.2 Problem Statement

Given a K_{single}-bounded subject graph and the LUT architecture shown in Figure 2, this work aims to select a subset of single nodes and node pairs from the graph. These nodes are used in technology mapping. Each single node n has a representative single-output cut $cut(n)$, while each node pair (n_1, n_2) has a representative dual-output cut $cut(n_1, n_2)$. These cuts can cover all non-PI nodes.

This work targets delay and area minimization. The delay of the mapped netlist is simply modeled as the greatest depth of all selected nodes in this work, which can be extended to model accurate delay using weighted depth. The depth of a node is computed as:

- For a PI node n, $depth(n) = 0$.
- For a single node n, assuming that $cut(n) = \{n_1, n_2, ...\}$, $depth(n) = \max\{depth(n_1), depth(n_2), ...\}+1$. We also define the cut depth as the node depth $depth(cut(n)) = depth(n)$.
- For a node pair (n_1, n_2), they have to be computed at the same time and thus have the same depth. $depth(n_1) = depth(n_2) = \max\{depth(cut(n_1)), depth(cut(n_2))\}$.

The area of the mapped netlist is the number of used LUTs, each of which has either one output or dual outputs.

4 DUAL-OUTPUT LUT MAPPING FLOW

This section proposes our cut-based dual-output LUT mapping flow and focuses on three steps in one mapping pass.

4.1 Mapping Flow Overview

Figure 3a depicts our dual-output LUT mapping flow. For any given combinational circuit, we convert it into a 2-bounded subject graph by performing ABC single-output LUT-2 mapping. The following LUT generation and merging are based on this subject graph. Since a K-input LUT can implement any logic function with no more than K inputs, we do not restrict the logic operation type in our 2-bounded subject graph. The following steps select nodes and cuts from the graph without changing the graph structure.

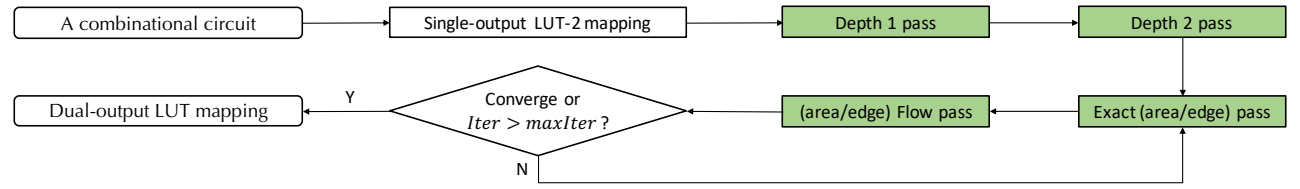
We perform several mapping passes on the subject graph to minimize the delay and area of the mapped netlist. The first two depth passes optimize the depth of all nodes with different heuristic metrics. The following exact and flow passes iteratively optimize the local area and the global area without worsening the depth until convergence or reaching the maximum number of iterations.

Figure 3b illustrates the details of a mapping pass. A mapping pass takes the original 2-bounded subject graph and the mapped netlist with dual-output LUTs (except the first pass) from the previous pass as the input. In each pass, we first compute priority single-output cuts for all nodes in the subject graph. Then, we select representative single-output cuts and dual-output cut pairs by global cut merging. Heuristic metric computation for cut selection depends on the previous mapped LUT netlist and varies in different passes. Finally, we refine the mapped netlist in every few layers to further improve the cut merging rate.

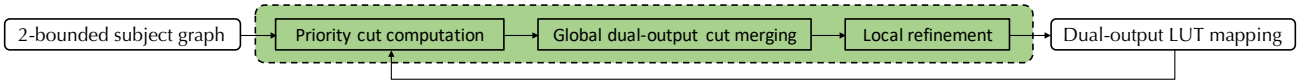
4.2 Priority Cut Computation

Similar to [22], we avoid the hurdle of enumerating all cuts by computing limited priority cuts. At each node, we store up to $2P$ priority cuts, including P K_{single}-feasible cuts and P K_{dual}-feasible cuts. The former ones are prepared for single-output cut generation while the latter ones are prepared for dual-output cut merging. Two types of cuts are selected individually using the same heuristic metrics. The best K_{single}-feasible cut is the representative single-output cut of the node. Typically, $P \le 30$ in our mapping flow.

The priority cut computation starts from PIs and is performed on the subject graph in the topological order. A node is computed after its two fanins are computed. For a non-PI node, we can generate at most $(2P + 1)^2$ candidate cuts. The best two cuts, i.e., a K_{single}-feasible one and a K_{dual}-feasible one, from the previous mapping pass are also added to the candidates. After filtering the dominated cuts and the cuts that do not satisfy the delay constraint, we sort the rest cuts to find no more than $2P$ priority cuts.



(a) Mapping flow overview.



(b) A (depth/exact/flow) mapping pass. All of the mapping passes in the flow consist of three steps in common.

Figure 3: Our dual-output LUT mapping flow.

Algorithm 1 Global dual-output cut merging algorithm

Input: A 2-bounded subject graph
Output: The cut set CUT used in the mapped netlist

1: $CUT \leftarrow \varnothing$
2: $MP \leftarrow POs$ // the mapping frontier
3: **while** $MP \neq \varnothing$ **do**
4: $n_{deep} \leftarrow \arg\max_{n \in MP}\{depth(n)\}$
5: $cut_{better} \leftarrow cut(n_{deep})$
6: $cut_{merge} \leftarrow$ the best K_{dual}-feasible cut rooted at n_{deep}
7: **for** $n \in MP - n_{deep}$ **do** // find the best dual-output cut
8: **for** $cut_{merge} \cup cut_{other} \in \Phi(n_{deep}, n)$ **do**
9: **if** $cut_{merge} \cup cut_{other}$ is better than cut_{better} **then**
10: $cut_{better} \leftarrow cut_{merge} \cup cut_{other}$
11: **end if**
12: **end for**
13: **end for**
14: $CUT \leftarrow CUT \cup cut_{better}$
15: $MP \leftarrow MP - root(cut_{better})$
16: $MP \leftarrow MP \cup cut_{better} - PIs$
17: **end while**

We do not compute dual-output cuts in this step for two reasons. On the one hand, if we enumerate dual-output cuts for all node pairs, the time complexity becomes $O(P^2N^2)$ and is unacceptable for large graphs. N is the number of nodes. On the other hand, if we only compute limited dual-output cuts for a node, it is hard to guarantee that both cut roots are used simultaneously.

4.3 Global Dual-Output Cut Merging

Algorithm 1 shows the global dual-output cut merging algorithm. We traverse the subject graph in a reverse topological order to derive a mapped netlist. During the traversal, we maintain a mapping frontier list MP to store the nodes that have to be mapped next. At the beginning, MP consists of all POs (Line 2). After a cut cut_{better} is added to the mapping result, we move the cut root(s) out of MP and add the non-PI cut leaves into MP (Line 15-16).

Each iteration from Line 4 to Line 13 selects a new cut and adds it into CUT. First, we specify the deepest node n_{deep} in MP as a cut root (Line 4). Its representative single-output cut $cut(n_{deep})$ is a candidate (Line 5) while its best K_{dual}-feasible cut cut_{merge} is one single-output cut in the dual-output cut (Line 6). Then, we

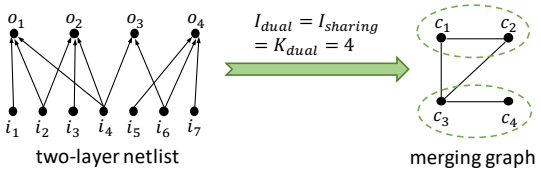


Figure 4: An example of the merging graph construction. The root of c_i is o_i. The best mapping merges two dual-output cuts, $c_1 \cup c_2$ and $c_3 \cup c_4$, from four single-output cuts. The merging rate is $\frac{4-2}{4} \times 100\% = 50\%$.

enumerate the K_{dual}-feasible cuts of the other nodes in MP to find the best dual-output cut (Line 7-13). Both roots stay in MP, which guarantees that they appear in the mapped netlist simultaneously. Finally, we add the best cut cut_{better} into CUT (Line 14).

Since n_{deep} is the deepest node, its priority cut is deeper than the rest cuts. Merging it with the other node will not worsen its depth. Furthermore, n_{deep} is on the critical path of the graph, so the cut merging operation will not worsen the total delay of the mapped netlist. We derive that, if the subject graph structure is not changed, our merging algorithm can achieve the best delay of the conventional cut-based mapping approach without considering dual-output cuts.

4.4 Local Refinement

The global cut merging step greedily selects the representative cuts and may not get the optimal area. As a result, we do some local refinement to further reduce the area by improving the cut merging rate. In this work, we define merging rate as $\frac{single\ area - dual\ area}{single\ area} \times 100\%$. A higher merging rate means smaller area for a given single-output cut set. This step does not change the used nodes but only merge more cuts.

To ensure the optimal delay during refinement, we compute the required depths, i.e., the depth upper bounds that can remain the current delay, of used nodes [5]. For two single-output cuts, if the current depths of both cuts do not exceed the required depths of the other cuts, merging two cuts will not worsen the delay of the mapped netlist. The required depths of POs are equal to their current depths. For the other nodes, we can compute their required

Table 2: Heuristic metrics for cut selection. The cut with a smaller heuristic metric is better. In this table, $fanout(x)$ means the fanout set of x or $root(x)$ in the previous mapped netlist except for the first mapping pass. It can fluctuate among different mapping passes. If a node x is unused, $fanout(x)$ is an empty set. $area(x)$ represents the number of cuts that can cover all of the nodes in x. $edge(x)$ represents the total fanin edge number of nodes in x.

Metric	Single-output cut c	Dual-output cut $c = c_1 \cup c_2$
Depth	$depth(c)$	$\max\{depth(c_1), depth(c_2)\}$
Exact area	$area(MFFC(root(c)))$	$\frac{2}{5} \cdot area(MFFC(root(c_1), root(c_2)))$
Area flow	$\frac{1}{\lvert fanout(c)\rvert} \cdot \left[area(c) + \sum_{n \in c} area_{flow}(cut(n))\right]$	$\frac{2}{\lvert fanout(c_1)+fanout(c_2)\rvert} \cdot \left[area(c) + \sum_{n \in c} area_{flow}(cut(n))\right]$
Cut size	$\lvert c\rvert$	$\lvert c_1 - c_2\rvert + \lvert c_2 - c_1\rvert$
Fanin refs	$\sum_{n \in c} \lvert fanout(n)\rvert$	$\sum_{n \in c_1 - c_2} \lvert fanout(n)\rvert + \sum_{n \in c_2 - c_1} \lvert fanout(n)\rvert$
Exact edge	$edge(MFFC(root(c)))$	$\frac{2}{5} \cdot edge(MFFC(root(c_1), root(c_2)))$
Edge flow	$\frac{1}{\lvert fanout(c)\rvert} \cdot \left[edge(c) + \sum_{n \in c} edge_{flow}(cut(n))\right]$	$\frac{2}{\lvert fanout(c_1)+fanout(c_2)\rvert} \cdot \left[edge(c) + \sum_{n \in c} edge_{flow}(cut(n))\right]$

depths in a reverse topological order:

$$req(n) = \min\{req(n_1), req(n_2), ...\} - 1 \quad fanout(n) = \{n_1, n_2, ...\}.$$

We refine the mapped netlist in every few layers that contain the nodes with the same depth to reduce the time complexity. The layer number depends on the circuit size. The dual-output cuts in these layers are first decomposed to single-output cuts. Then, we construct a merging graph with each node representing a single-output cut. If merging two cuts do not worsen the total delay, we add an edge between them. Finally, we compute the maximum cardinality matching using the $O(N^3)$ blossom algorithm [10]. The matched nodes in the merging graph corresponds to the cut pairs in the mapped netlist.

Figure 4 gives an example of two layers and four cuts. Cut pairs that meet the constraints in Section 2.3 are connected. Since these cuts have the same depth, connecting them will not worsen the delay. We find two dual-output cuts from the merging graph, which is the best mapping in this small netlist.

5 CUT SELECTION HEURISTIC

This section reviews some existing heuristic metrics for single-output cuts and then extends them to support dual-output cuts.

5.1 Single-output Cut Selection

We list some commonly used heuristic metrics [13, 21] for single-output cut selection in the second column of Table 2. They are:

- depth. Minimizing the depth of all cuts can minimize the total delay.
- exact area. This is a local area estimation. The MFFC of a node can only be covered by the cut rooted at the node or its precursor.
- area flow. This is a global area estimation, which equals to the least number of cuts needed to map the current cut and all of its precursors.
- cut size. Without affecting the delay and area of the mapped netlist, a smaller cut is more likely to be merged in a later packing phase.
- fanin refs. A smaller fanin reference counter means a smaller overlap between cuts. Thus, these cuts can cover more nodes.

Table 3: Heuristic metric combination in different passes.

Mapping pass	Depth 1	Depth 2	Exact	Flow
Primary metric	depth	depth	exact area	area flow
Tie-breaker 1	cut size	area flow	fanin refs	fanin refs
Tie-breaker 2	fanin refs	cut size	depth	depth
Tie-breaker 3	–	–	exact edge	edge flow

- exact edge. This is a local edge estimation. Minimizing edges can reduce the wire length in the physical implementation and the average cut size.
- edge flow. This is a global edge estimation.

Among them, we can get cut size and fanin refs directly. Depth, area flow and edge flow can be computed in a topological order during priority cut computation. The MFFC set is typically small, so we can compute exact area and exact edge with a fast local depth-first traversal. We still use these heuristic metrics in the priority cut computation step of our mapping pass.

5.2 Dual-output Cut Selection

In Algorithm 1, Line 9 compares a single-output cut with a dual-output cut or compares two dual-output cuts. We extend the heuristic metrics to support dual-output cut selection. Our extension follows two cut selection principles for a given cut root:

- We prefer dual-output cuts with more sharing inputs rather than its representative single-output cut.
- We prefer its representative single-output cut rather than dual-output cuts with less or no sharing inputs.

We can interpret these two principles from the perspective of the covered node number. The number of non-PI nodes in the subject graph is fixed. Without considering the overlap between cuts, in order to cover the non-PI nodes with fewer cuts, we have to increase the number of nodes that each cut can cover.

For a single-output cut, the number of covered nodes is usually proportional to the input number. In particular, a single-output K-input cut in a 2-bounded subject graph, e.g., $\{n_1, n_2\}$ and $\{n_1, i_2, i_3\}$ in Figure 1, can cover at least $(K - 1)$ nodes. For a dual-output cut, with the increase of the sharing input number, the total input number and the average covered node number also increases. A dual-output cut with enough sharing inputs can cover more nodes than the representative single-output cut.

The extension is listed in the third column of Table 2. The extended metrics are:

- depth. The depth of a dual-output cut is defined in Section 3.2.
- exact area. The MFFC of two roots is defined as a subset of their fanin cones, such that every path from a node in the subset to the POs passes through one of the roots. It contains a little more nodes whose fanin reference counters equal to 2 than two independent MFFCs. It is still typically small. For fair comparison with single-output cuts, we add a coefficient $\frac{2}{5}$, which is slightly smaller than $\frac{1}{2}$.
- area flow. The coefficient 2 is for fair comparison. Both exact area and area flow represent the number of cuts that can cover some given nodes, so they follow two cut selection principles.
- cut size. We only consider the inputs that are not shared. When compared to a single-output cut, the dual-output cuts with more sharing inputs will have advantages.
- fanin refs. Similar to cut size, it encourages more sharing inputs.
- exact edge. Its extended formula is similar to that of exact area.
- edge flow. Its extended formula is similar to that of area flow.

These extended metrics can be computed in a similar way to the original ones. We can also get them in a reasonably short time.

5.3 Heuristic Metric Combination

Table 3 lists different heuristic metrics used in each mapping pass. We use the same combination as that in the single-output mapping [22]. For example, in the "Depth 1" pass, we prefer cuts with smaller depths. If there is a tie, we prefer cuts of smaller size; If there still is a tie, we prefer cuts with smaller area flow. Comparison of dual-output cuts (Line 9 in Algorithm 1) has a slight difference. We do not need to compare their depths because all dual-output cuts are as deep as the single-output cut cut_{merge}.

6 EXPERIMENTAL EVALUATION

We implement our technology mapping algorithm in C++ and run it on an Intel Xeon 2-CPU 10-core computer with 60GB RAM. We select 13 large circuits from the EPFL benchmark suite [2] to evaluate its performance. These circuits are synthesized with ABC commands "*resyn; resyn2;*" before technology mapping. All mapped networks have been verified using ABC command "*cec*", a combinational equivalence checker.

6.1 Delay and Area Evaluation

We compare our mapping flow with ABC using FPGA parameters in Table 1. The results are listed in Table 4. For ABC, we get its "single area" with the command "*if -C 8*" that stores at most 8 cuts at a node. Then, we construct the global merging graph similar to that in the local refinement step of our flow. We only connect LUTs satisfying the delay constraint such that the total delay also remains unchanged. We merge dual-output LUTs using this graph to get the best "dual area" of ABC. For our flow, we get "single area" by directly using the representative single-output cuts in the global cut merging step without the local refinement step. We also set $P = 8$ for a fair comparison.

We can see that our mapping flow achieves the same delay after merging dual-output cuts. These delay results verify the effectiveness of our algorithm that maps the deepest node first (Line 4 in Algorithm 1). Our flow also achieves the same delay as ABC. ABC can generate delay-optimal mapped netlists with single-output LUTs, so we infer that our flow can achieve the optimal delay as the conventional single-output mapping flow.

Despite of the disadvantage on single area, our mapping flow saves 9.14%, 13.98% and 5.68% dual area on average on the three FPGA series, respectively. This is mainly due to our higher merging rate. Since our flow merges dual-output cuts during cut generation, we can discover more possibilities for cut merging. Compared to ABC, our flow achieves a 12.16%, a 14.89% and a 10.45% higher merging rate on average on the three FPGA series, respectively.

Our mapping flow needs more area without considering dual outputs, mainly because we do not reconstruct LUT networks after the cut-based mapping. Previous reconstruction algorithms [20] are designed for single-output cut structures and cannot be applied to dual-output ones directly. We will develop new reconstruction algorithms for dual-output cut structures in our future work.

Our flow performs better on UltraScale+ and Versal series than Intel ALM series, because of their smaller LUT sizes. There are $O(N^K)$ [7] K-feasible cuts in a graph with N nodes. The number of priority cuts for node pairs is even greater. For a given priority cut number P, our flow is more likely to store the best K_{single}-feasible and K_{dual}-feasible cut with a smaller LUT size. Our flow has a higher merging rate on Versal series due to its more flexible architecture. Its dual-output LUT can implement more complex logic.

6.2 Detailed Analysis

6.2.1 Effect of local refinement. Table 5 lists the data without local refinement. In general, local refinement can additionally improve 1.81% merging rate and save 2.10% area. Local refinement works on more than half of the cases, especially the ones that have relatively low merging rate after global cut merging. It additionally improves 2.94% merging rate and saves 3.41% area on these cases.

6.2.2 Effect of maxIter. Figure 5 shows the effect of different *maxIter*s. The dual area of our flow converges after 3 iterations for most of the cases. But for some special cases, e.g., sin and square, the dual area continues to change until 5 iterations. This may be due to their special subject graph structures. The iterative area optimization using both area flow and exact area heuristic metrics brings in about 16% additional dual area improvement. 14% improvement comes from the first iteration, and the rest 2% comes from the following iterations.

6.2.3 Effect of P. Figure 6 shows the effect of different Ps. The area does not change much (<3%) when $P \geq 5$ for most of the cases. It means that the priority cut set already contains the best representative cuts and cut pairs. Continuing to increase P will only lead to useless cut enumeration. But for some special cases, e.g., cavlc, sin and int2float, our dual area gradually decreases with the increase of P. Setting $P = 30$ helps save 3% area on these three cases compared to $P = 5$.

Table 4: Comparison of ABC and our LUT mapping flow. Delay remains unchanged after considering dual-outputs in both ABC and our flow, so each flow only contains one delay column. We set $maxIter = 5$ and $P = 8$ in our mapping flow.

FPGA	Case	PI	PO	Size	ABC				Ours				Area saving	
					single area	dual area	merging rate	delay	single area	dual area	merging rate	delay	single	dual
UltraScale+	voter	1001	1	8869	1736	1621	6.62%	13	1807	1520	15.88%	13	-4.09%	6.23%
	cavlc	10	11	705	118	115	2.54%	4	119	108	9.24%	4	-0.85%	6.09%
	dec	8	256	321	287	157	45.30%	2	273	140	48.72%	2	4.88%	10.83%
	priority	128	8	803	179	175	2.23%	26	219	160	26.94%	26	-22.35%	8.57%
	adder	256	129	1020	257	236	8.17%	51	254	189	25.59%	51	1.17%	19.92%
	arbiter	256	129	12351	2722	2700	0.81%	18	2725	2469	9.39%	18	-0.11%	8.56%
	sin	24	25	4398	1473	1389	5.70%	36	1680	1426	15.12%	36	-14.05%	-2.66%
	bar	135	128	3323	512	501	2.15%	4	512	448	12.50%	4	0.00%	10.58%
	max	512	130	3669	793	775	2.27%	39	837	822	1.79%	39	-5.55%	-6.06%
	i2c	147	142	1414	315	288	8.57%	4	382	300	21.47%	4	-21.27%	-4.17%
	square	64	128	13388	3948	3725	5.65%	50	3902	3241	16.94%	50	1.17%	12.99%
	ctrl	7	26	129	28	27	3.57%	2	28	17	39.29%	2	0.00%	37.04%
	int2float	11	7	243	47	46	2.13%	3	46	41	10.87%	3	2.13%	10.87%
	average						**7.36%**				**19.52%**		**-4.53%**	**9.14%**
Versal	voter	1001	1	8869	1736	1558	10.25%	13	1807	1264	30.05%	13	-4.09%	18.87%
	cavlc	10	11	705	118	109	7.63%	4	119	80	32.77%	4	-0.85%	26.61%
	dec	8	256	321	287	157	45.30%	2	273	137	49.82%	2	4.88%	12.74%
	priority	128	8	803	179	164	8.38%	26	219	150	31.51%	26	-22.35%	8.54%
	adder	256	129	1020	257	223	13.23%	51	254	140	44.88%	51	1.17%	37.22%
	arbiter	256	129	12351	2722	2615	3.93%	18	2725	2457	9.83%	18	-0.11%	6.04%
	sin	24	25	4398	1473	1210	17.85%	36	1680	1109	33.99%	36	-14.05%	8.35%
	bar	135	128	3323	512	448	12.50%	4	512	448	12.50%	4	0.00%	0.00%
	max	512	130	3669	793	650	18.03%	39	837	690	17.56%	39	-5.55%	-6.15%
	i2c	147	142	1414	315	266	15.56%	4	382	268	29.84%	4	-21.27%	-0.75%
	square	64	128	13388	3948	2642	33.08%	50	3902	2127	45.49%	50	1.17%	19.49%
	ctrl	7	26	129	28	21	25.00%	2	28	15	46.43%	2	0.00%	28.57%
	int2float	11	7	243	47	45	4.26%	3	46	35	23.91%	3	2.13%	22.22%
	average						**16.54%**				**31.43%**		**-4.53%**	**13.98%**
Intel ALM	voter	1001	1	8869	1405	1357	3.42%	10	1606	1221	23.97%	10	-14.31%	10.02%
	cavlc	10	11	705	42	41	2.38%	2	42	38	9.52%	2	0.00%	7.32%
	dec	8	256	321	256	256	0.00%	1	256	256	0.00%	1	0.00%	0.00%
	priority	128	8	803	139	135	2.88%	18	185	134	27.57%	18	-33.09%	0.74%
	adder	256	129	1020	221	191	13.57%	37	218	155	28.90%	37	1.36%	18.85%
	arbiter	256	129	12351	2069	2003	3.19%	13	2067	1876	9.24%	13	0.10%	6.34%
	sin	24	25	4398	1275	1187	6.90%	26	1359	1126	17.14%	26	-6.59%	5.14%
	bar	135	128	3323	512	449	12.30%	4	512	439	14.26%	4	0.00%	2.23%
	max	512	130	3669	643	504	21.62%	28	668	498	25.45%	28	-3.89%	1.19%
	i2c	147	142	1414	259	226	12.74%	3	304	235	22.70%	3	-17.37%	-3.98%
	square	64	128	13388	3441	3202	6.95%	36	3640	2583	29.04%	36	-5.78%	19.33%
	ctrl	7	26	129	25	15	40.00%	1	25	14	44.00%	1	0.00%	6.67%
	int2float	11	7	243	26	25	3.85%	3	29	25	13.79%	3	-11.54%	0.00%
	average						**9.98%**				**20.43%**		**-7.01%**	**5.68%**

6.2.4 LUT Distribution. Figure 7 lists the ratio of LUTs with different inputs. We first focus on the dual-output LUTs in our flow. LUT-5 and LUT-4 occupy 64.08% and 23.54% of decomposed dual-output LUTs. On average, a dual-output LUT is composed of two single-output LUT-4.51 and has 4.02 sharing inputs. The LUT distribution is consistent with our dual-output cut selection principles in Section 5.2, which verifies the effectiveness of our extension.

Our flow also has advantages when compared to ABC. The dual-output LUTs generated by our flow have advantages not only in quantity, i.e., merging rate, but also in quality. In ABC, LUT-5 and LUT-4 only occupy 61.44% (2.64% less) and 15.59% (7.95% less) of decomposed dual-output LUTs. On average, a dual-output LUT is composed of two single-output LUT-4.30 (0.21 less) and has 3.60 (0.42 less) sharing inputs.

6.2.5 Runtime. As shown in Figure 8, our mapping flow consumes 3.50×, 3.54× and 2.90× runtime on average on the three series, respectively. Despite this, our flow successfully filters out most of the node pairs from $O(N^2)$ pairs in total and saves a lot of computation. The runtime of our flow is still linear in subject graph size and is affordable for large industrial circuits. Our flow additionally traverses the subject graph in global cut merging and local refinement step to merge cuts, which leads to a larger coefficient in time complexity.

Table 5: Merging rate and area saving of our mapping flow without local refinement using UltraScale+ parameters.

FPGA	Case	Dual area	Merging rate	Area saving
UltraScale+	voter	1597	11.62%	1.48%
	cavlc	110	7.56%	4.35%
	dec	140	48.72%	10.83%
	priority	171	21.92%	2.29%
	adder	189	25.59%	19.92%
	arbiter	2502	8.18%	7.33%
	sin	1519	9.58%	-9.36%
	bar	448	12.50%	10.58%
	max	826	1.31%	-6.58%
	i2c	307	19.63%	-6.60%
	square	3376	13.48%	9.37%
	ctrl	17	39.29%	37.04%
	int2float	41	10.87%	10.87%
	average		**17.71%**	**7.04%**

7 RELATED WORK

There are mature technology mapping flows for single-output LUTs. ABC integrates some mapping commands. For example, "*fpga*" [21] and "*if*" [22] implement cut-based mapping approaches. Based on the results of these two commands, "*mfs*" [4] further optimizes don't cares by a SAT solver while "*lutpack*" [20] decomposes and rewrites LUT networks. Besides ABC, DAOmap [5] closely monitors various node duplication scenarios and designs novel heuristic metrics. PIMap [17] couples logic transformations and technology mapping using a parallelized iterative improvement approach.

Some works are aware of the dual-output LUTs and attempt to consider LUT merging during technology mapping. Murgal et al. [23] first transform the LUT merging problem to the maximum cardinality matching problem. Huang et al. [12, 16] exploit the feature of the two-output LUT architecture with Roth-Karp decomposition. WireMap [13] uses an edge flow heuristic to reduce the average LUT size such that the VTR packing phase [24] can merge more LUTs. Dickin et al. [9] improve WireMap by considering LUT balancing. However, these works still separate single-output LUT generation and dual-output LUT merging into two design phases, which leads to a low LUT merging rate. Besides, Jr et al. [18, 19] confirm the potential of using KL-cuts on multi-output technology mapping, but they do not achieve the state-of-the-art result on mapping.

8 CONCLUSION

In this work, we propose a novel cut-based mapping flow that can merge dual-output LUTs during technology mapping. In each mapping pass, we perform global dual-output cut merging and local refinement. We also extend conventional heuristic metrics to make it applicable to dual-output cut selection. Experimental evaluation shows that our mapping flow achieves a higher LUT merging rate and smaller area without worsening the delay.

ACKNOWLEDGMENTS

This work is partly supported by Beijing Municipal Science and Technology Program under Grant No. Z201100004220007, Key Area R&D Program of Guangdong Province with Grant No. 2018B030338001, and Beijing Academy of Artificial Intelligence (BAAI).

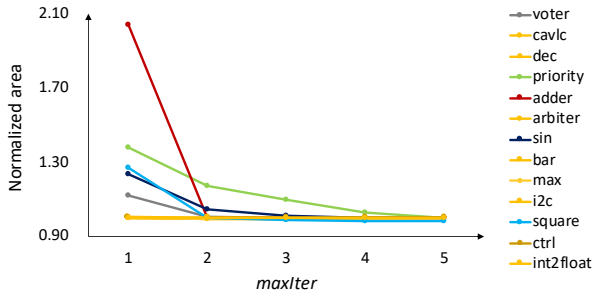


Figure 5: The effect of different *maxIter*s using UltraScale+ parameters. Our dual area in Table 4 is normalized to 1.

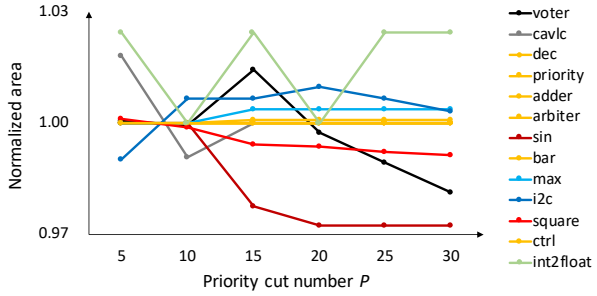


Figure 6: The effect of different *P*s using UltraScale+ parameters. Our dual area in Table 4 is normalized to 1.

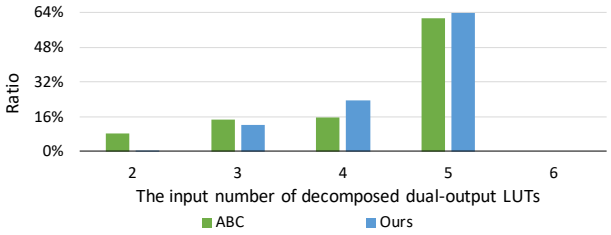


Figure 7: LUT distribution of two flows using UltraScale+ parameters. We decompose a dual-output LUT to two independent single-output LUTs so there are no 6-input LUTs.

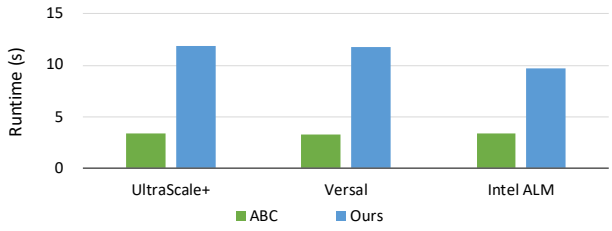


Figure 8: Average runtime of two flows.

REFERENCES

[1] I Altera. 2018. Intel Arria 10 Device Overview. (2018).

[2] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL combinational benchmark suite. In *Int'l Workshop on Logic & Synthesis (IWLS)*. IEEE.

[3] Vamsi Boppana, Sagheer Ahmad, Ilya Ganusov, Vinod Kathail, Vidya Rajagopalan, and Ralph Wittig. 2015. UltraScale+ MPSoC and FPGA families. In *Hot Chips Symposium (HCS)*. IEEE, 1–37.

[4] RK Brayton, Jie-Hong Roland Jiang, and Stephen Jang. 2007. SAT-based logic optimization and resynthesis. In *Int'l Workshop on Logic & Synthesis (IWLS)*. IEEE, 358–364.

[5] Deming Chen and Jason Cong. 2004. DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs. In *Int'l Conf. on Computer-Aided Design (ICCAD)*. IEEE, 752–759.

[6] Jeffrey Chromczak, Mark Wheeler, Charles Chiasson, Dana How, Martin Langhammer, Tim Vanderhoek, Grace Zgheib, and Ilya Ganusov. 2020. Architectural Enhancements in Intel® Agilex™ FPGAs. In *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*. 140–149.

[7] Jason Cong and Yuzheng Ding. 1994. On area/depth trade-off in LUT-based FPGA technology mapping. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 2, 2 (1994), 137–148.

[8] Jason Cong, Chang Wu, and Yuzheng Ding. 1999. Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution. In *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*. ACM, 29–35.

[9] David Dickin and Lesley Shannon. 2011. Exploring FPGA technology mapping for fracturable LUT minimization. In *Int'l Conf. on Field-Programmable Technology (FPT)*. IEEE, 1–8.

[10] Jack Edmonds. 1965. Paths, trees, and flowers. *Canadian Journal of mathematics* 17 (1965), 449–467.

[11] Brian Gaide, Dinesh Gaitonde, Chirag Ravishankar, and Trevor Bauer. 2019. Xilinx adaptive compute acceleration platform: VersalTM architecture. In *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*. 84–93.

[12] Juinn-Dar Huang, Jing-Yang Jou, and Wen-Zen Shen. 1995. Compatible class encoding in Roth-Karp decomposition for two-output LUT architecture. In *Int'l Conf. on Computer-Aided Design (ICCAD)*. IEEE, 359–363.

[13] Stephen Jang, Billy Chan, Kevin Chung, and Alan Mishchenko. 2008. WireMap: FPGA technology mapping for improved routability. In *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*. ACM, 47–55.

[14] Ju-Yueh Lee, Zhe Feng, and Lei He. 2010. In-place decomposition for robustness in FPGA. In *Int'l Conf. on Computer-Aided Design (ICCAD)*. IEEE, 143–148.

[15] Ju-Yueh Lee, Yu Hu, Rupak Majumdar, Lei He, and Minming Li. 2010. Fault-tolerant resynthesis with dual-output LUTs. In *Asia and South Pacific Design Automation Conf. (ASP-DAC)*. IEEE, 325–330.

[16] Christian Legl, Bernd Wurth, and Klaus Eckl. 1998. Computing support-minimal subfunctions during functional decomposition. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 6, 3 (1998), 354–363.

[17] Gai Liu and Zhiru Zhang. 2017. A parallelized iterative improvement approach to area optimization for lut-based technology mapping. In *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*. ACM, 147–156.

[18] Lucas Machado, Mayler Martins, Vinicius Callegaro, Renato P Ribas, and André I Reis. 2012. KL-cut based digital circuit remapping. In *NORCHIP*. IEEE, 1–4.

[19] Osvaldo Martinello, Felipe S Marques, Renato P Ribas, and André I Reis. 2010. KL-cuts: a new approach for logic synthesis targeting multiple output blocks. In *Design, Automation, and Test in Europe (DATE)*. IEEE, 777–782.

[20] Alan Mishchenko, Robert Brayton, and Satrajit Chatterjee. 2008. Boolean factoring and decomposition of logic networks. In *Int'l Conf. on Computer-Aided Design (ICCAD)*. IEEE, 38–44.

[21] Alan Mishchenko, Satrajit Chatterjee, and Robert K Brayton. 2007. Improvements to technology mapping for LUT-based FPGAs. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 26, 2 (2007), 240–253.

[22] Alan Mishchenko, Sungmin Cho, Satrajit Chatterjee, and Robert Brayton. 2007. Combinational and sequential mapping with priority cuts. In *Int'l Conf. on Computer-Aided Design (ICCAD)*. IEEE, 354–361.

[23] Rajeev Murgai, Yoshihito Nishizaki, Narendra Shenoy, Robert K Brayton, and Alberto Sangiovanni-Vincentelli. 1990. Logic synthesis for programmable gate arrays. In *Design Automation Conf. (DAC)*. IEEE, 620–625.

[24] Jonathan Rose, Jason Luu, Chi Wai Yu, Opal Densmore, Jeffrey Goeders, Andrew Somerville, Kenneth B Kent, Peter Jamieson, and Jason Anderson. 2012. The VTR project: architecture and CAD for FPGAs from verilog to routing. In *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*. ACM, 77–86.

[25] Berkeley Logic Synthesis. 2007. ABC: a system for sequential synthesis and verification, release 70930.