A Technology Mapper for Depth-Constrained FPGA Logic Cells

Zhenghong Jiang*, Grace Zgheib†, Colin Yu Lin*, David Novo†,
Zhihong Huang*, Liqun Yang*, Haigang Yang* and Paolo Ienne†

*System on Programmable Chip Research Department
Institute of Electronics, Chinese Academy of Sciences, Beijing, China
yanghg@mail.ie.ac.cn

†Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences, 1015 Lausanne, Switzerland
{grace.zgheib, david.novobruna, paolo.ienne}@epfl.ch

***Abstract*—In the last decade, progress in logic synthesis has brought about new advantageous circuit representations. These representations, such as And-Inverter Graphs in the ubiquitous open-source synthesizer ABC, have inspired new designs of *Field Programmable Gate Arrays (FPGAs)*, which, instead of using *Look-Up Tables (LUTs)*, mimic the topology of the circuit representation in the basic logic cells. More recent examples are Majority-Inverter Graphs, another uniform representation which has triggered considerable interest in synthesis and which naturally suggests new logic cells. Yet, in this paper we observe how naïvely adapting technology mapping solutions for classic LUT-based FPGAs to these new architectures incurs severe shortcomings. The key issue is that LUTs are inherently input-constrained (the logic function they implement is irrelevant) and have generally a single output; on the other hand, logic cells made of uniform networks of some fundamental logic function (e.g., And-Invert) are constrained in terms of logic depth and multiple outputs are an integral feature. We introduce novel and effective solutions to address these differences; the result is a highly versatile mapper—thus enabling further research in these new architectures—with a significantly better performance than what is described in literature for one such architecture. Specifically, when we compare with the state of the art on one sample architecture, we obtain a significant decrease in area (on average 18% over several benchmarks) while also improving slightly the critical path (a reduction of 3%).**

I. Introduction

Field Programmable Gate Arrays (FPGAs), and in particular commercial ones, are relatively uniform in terms of architecture: a small group of *Look-Up Tables (LUTs)* connected together through a crossbar create a logic cluster; a myriad of such blocks are interconnected through some statically configurable routing network. Differences and innovation are in how exactly these LUTs are built or interconnected, in specialized logic and connectivity such as carry chains, in the blend of FPGA proper with other hardwired components such as arithmetic units. Yet, the last decade has seen a resurgence of interest in circuit representations used to improve the capabilities of logic synthesis. The most ubiquitous example is the academic synthesis and verification tool ABC [8] which uses extensively *And-Inverter Graphs (AIGs)* to represent the circuits under optimization. Although ABC is the most mature and established example, it is not the only one: more recent is the introduction of *Majority-Inverter Graphs (MIGs)* [1] which appear to have some specific strengths in the optimization process. There is a natural link between the representation used in synthesis and the basic logic cell used in an FPGA: if the results of synthesis have a particular form, logic cells somehow mimicking such form may be equally effective to map circuits and yet inherently simpler than the universal, and thus expensive, LUTs. Hence, architectures based on *And-Inverter Cones (AICs)* [10], [12] are a natural structure to map circuits synthesized from ABC, for instance.

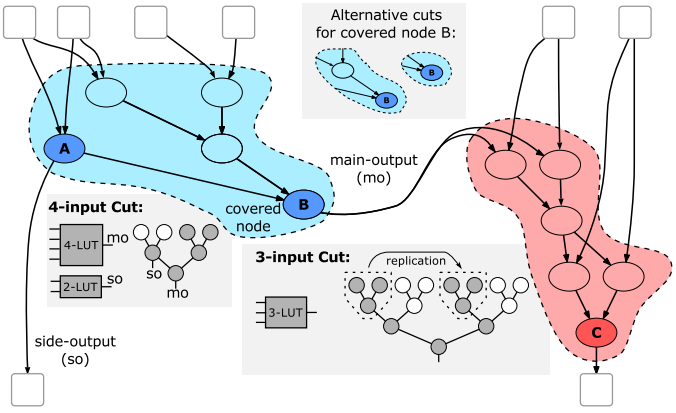


Fig. 1: Differences in technology mapping. Mapping a circuit composed of uniform components (e.g., AND-Inverter nodes) on corresponding basic logic cells is quite different from mapping the circuit on LUTs. Firstly, a feasible mapping does not depend on the number of inputs but on the logic depth of the cut, and this has an important effect on the mapping process as will be discussed later in the paper. Secondly, an efficient assignment of the side outputs, which do not exist in LUTs, is not self-evident.

This research is partially funded by the National Natural Science Foundation of China (61404140, 61271149).

Despite such mapping being "natural" and notwithstanding the possibility of a straightforward adaptation of the state-of-the-art mapping algorithms to such architectures, the technology mapping problem is here significantly different and the algorithms susceptible of tangible improvements. Informally, *technology mapping* is the process (i) of identifying subgraphs of the circuit which the basic logic cells of the FPGA can implement—called *cuts*—and (ii) of selecting a particular set of subgraphs which cover the entire circuit and minimize some metric of interest, such as critical path delay or number of logic cells used. Figure 1 suggests the two main differences between the problem of mapping a circuit on LUTs and of mapping it on specific subgraphs of the used representation. The first and foremost difference is that K-input LUTs can implement any function of at most K variables whereas predefined subgraphs of the uniform circuit representation used in synthesis (for instance, AICs, described in more detail in Section IV) can self-evidently only implement fairly specific logic functions. This is suggested in the rightmost shaded cut of Figure 1 where a simple 3-input LUT can accommodate a fairly large portion of the circuit. The point, as it will be clear in the coming sections, is that the only constraint limiting what is a *feasible cut* for an LUT is the number of inputs, and mapping algorithms are somehow built with this in mind; in stark contrast, logic cells such as AICs are limited by functionality and, equivalently, by the number of logic levels—or the *depth*—of the cut. As Section III-A will explain in detail, if this difference is not algorithmically properly accounted for, the mapping on depth-constrained logic cells may be significantly inefficient. The second important peculiarity, visible in the leftmost shaded cut of Figure 1, is that LUTs are inherently single output structures (hence two-output cuts need two LUTs to implement the functionality) whereas other logic cells may have naturally even a large number of outputs. Again, a naïve opportunistic use of these outputs is almost obvious but leads to poor mapping results and begs for algorithmic modifications.

Without loss of generality, the rest of this paper uses the most common of such uniform representations (AIGs [8]) and the corresponding architecture (based on AICs [10]) to build and improve a technology mapper for depth-constrained logic cells. Although for clarity we restrict ourselves to AIGs and AICs, our results could be easily extended to any other uniform representation where all nodes are n-ary operations arbitrarily interconnected (e.g., as already mentioned, MIGs composed of n-input majority functions and some hypothetical Majority-Inverter Cones).

The rest of the paper is organized as follows: We start by naïvely adapting a state-of-the-art mapping algorithm to AICs in Section II, while following the spirit of the first AIC mapper [10]. Then in Section III, we quantitatively investigate the modest quality of the initial results, relating it to the fundamental differences between depth-constrained and input-constrained logic cells. Consequently, we analyze in detail the effect of these differences on the technology mapping and thus rethink some of its aspects while introducing novel solutions. After defining more precisely the AICs and the cluster architecture of an AIC-based FPGA in Section IV, we present the experimental results in Section V, showing that, besides being versatile, our mapper achieves better results than those reported in the past. We then discuss the related work in Section VI and conclude with Section VII.

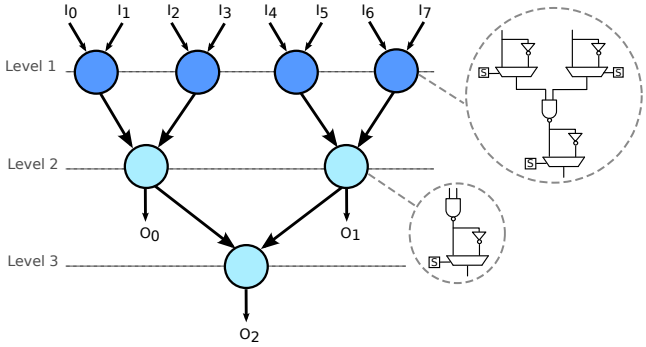


Fig. 2: A 3-level And-Inverter Cone (3-AIC).

II. Technology mapping problem

The mapping process takes a technology-independent Boolean network represented as a *Direct Acyclic Graph (DAG)*, typically in the AIG format, and transforms it into a network of logic cells. In the state-of-the-art FPGAs, the basic logic cells are K-input Look-Up Tables. In this section, we revisit the different phases of the most common technology mapping algorithm designed for LUTs and naïvely adapt it to the multi-output depth-constrained logic cells. To this extent, we re-use the open-source academic synthesis and verification tool ABC and modify its existing algorithms to map on AND-Inverter Cones while benefiting from its existing synthesis and verification features.

A. Problem formulation and definitions

In the most recent and common CAD tools, the technology mapping process is based on the representation of the logic circuit in the AIG format, which is a DAG with 2-input AND gates as nodes. The *Combinatorial Inputs (CIs)* and *Combinatorial Outputs (COs)* are also nodes in the DAG. These CIs consist of the circuit's *Primary Inputs* (PIs, nodes with no fanin) and the registers' outputs, while the COs are the *Primary Outputs* (POs, nodes with no fanout) and the registers' inputs.

For a *root* node v, a *cut* C_v is a set of nodes (referred to as *leaves*) where each path from the CIs to v passes through at least one of these nodes.The depth of a cut, represented by $depth(C_v)$, is the length of the longest path within the cut from its leaves to the root node.

A cut is said to be *K-feasible* if the number of its leaves does not exceed K. Moreover, a cut is D-feasible if its depth does not exceed a preset value D. Mapping on LUTs requires K-feasible cuts while the AIC, being a depth-constrained block requires D-feasible cuts.

The mapper consists of three main phases: the forward traversal, backward traversal and area recovery. In the remaining of this section, we describe each of these phases in details.

B. Forward traversal

In the first mapping phase, the graph is traversed from the CIs to the COs and cuts are generated for each node. The *priority cut* algorithm [9] is typically used to bound runtime where, instead of enumerating all feasible cuts for a node v, only a subset C of cuts is computed. The cuts of node v are generated by combining the different cuts of the leaves of v while discarding any non-feasible cut. Every generated cut is evaluated according to specific optimization objective functions, detailed in Section II-C, and the list of cuts is sorted, prioritizing the most optimal cuts. Typically, mapping algorithms tend to optimize for the critical path delay but other objective functions can be considered. Saving only C cuts reduces the runtime and memory requirements of the algorithm. Using the priority cut for depth-constrained logic cells and in this example, AICs, requires mainly a new definition of a feasible cut. Instead of using K-feasible cuts, only D-feasible (or depth-feasible) cuts are computed during the cut generation.

Once all the cuts are generated, the most optimal cut is selected for each node. This best cut might be updated throughout the different mapping phases and iterations, depending on the objective function of each iteration. Dynamically updating the best cut during the different mapping phases compensates for not initially computing all candidate cuts. Algorithm 1 shows the pseudocode of the forward traversal.

Algorithm 1 Forward Traversal

```
1:  for each node pNode in DAG do
2:    pNode.AicCutList = CreateEmptyCandidateList()
3:    AddAndSort(pNode.AicCutList, pNode.BestCut)
4:    pFanin0 = pNode.fanin0
5:    pFanin1 = pNode.fanin1
6:    for each Cuts c0 in pFanin0.AicCutList do
7:      for each Cuts c1 in pFanin1.AicCutList do
8:        newCut = CutGeneration(c0, c1)
9:        if !CutIsDFeasible(newCut) then
10:         continue
11:       end if
12:       newCut.Delay = CalculateCutDelay(newCut)
13:       if newCut.Delay >pNode.RequiredTime then
14:         continue
15:       end if
16:       newCut.Area = CalculateCutArea(newCut)
17:       AddAndSort(pNode.AicCutList, newCut)
18:     end for
19:   end for
20:   pNode.BestCut = FetchBestCut(pNode.AicCutList)
21: end for
```

C. Cost functions and mapping library

The priority cut algorithm ranks all the generated cuts of a node according to certain objective functions. The AIC mapper, similar to the traditional LUT mappers, aims at reducing the critical path delay as a primary objective function and then the area as a secondary objective function. Given that the delay and area of the AIC varies with its depth D, a library is used to keep track of these values for every D-AIC. Having such a library allows the user to tune the mapper by updating the delay and area values of the AICs depending on the design of the FPGA architecture. These numbers can be estimated or obtained by simulating the actual logic cluster. Table I shows a sample library of estimated metrics, which will be used in the subsequent sections to test the efficiency of the implemented mapper.

AIC depth	Area	Delay
1	1	38
2	3	42
3	7	46
4	15	50
5	31	54
6	63	58

TABLE I: Sample mapping library for AICs. The delay and area of every D-AIC are stored in a library and used in the cost function computations during the cut selection.

Having the unit delay (*UnitDelay*) and unit area (*UnitArea*) of every D-AIC, the cost function of a candidate cut C_v of node v is computed using the equation

$$Delay(C_v) = \max_{\forall vi \in inputs(C_v)} Delay(BC_{vi}) + UnitDelay(C_v), \quad (1)$$

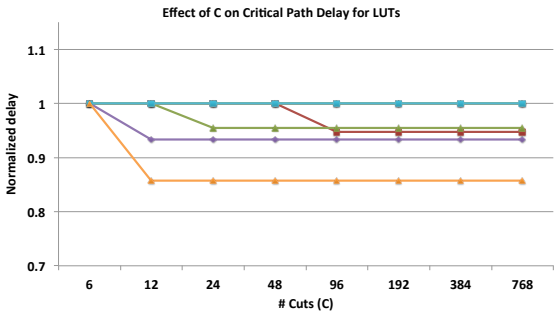
where vi are the leaf nodes of C_v and BC_v is the best cut of v. The area flow [7] is used as an area measure and is computed using the equation

$$AF(C_v) = \sum_{\forall v_i \in inputs(C_v)} [AF(C_{vi})/fanout(vi)] + UnitArea(C_v). \quad (2)$$

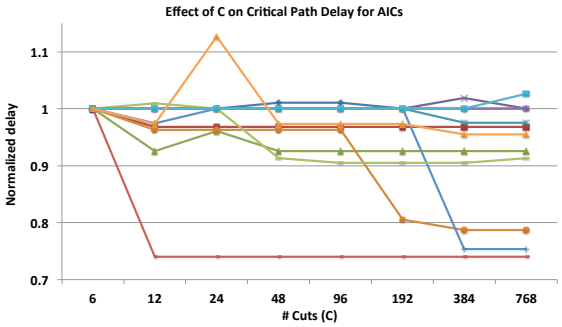
D. Backward traversal

In the typical backward traversal phase, the DAG is traversed from the COs to the CIs and the circuit is covered with a selection of best cuts. Initially, the outputs are set as *visible* and their best cuts selected as part of the final mapping solution. Then the inputs of these selected cuts become visible and are covered by their best cuts. The process is repeated until the entire DAG is covered.

However, each D-AIC has $2^{D-1} - 1$ outputs, as detailed in Section IV-A. Figure 2 shows a 3-AIC with 3 outputs: one main output at the third level (O_2) and two side outputs (O_0 and O_1). The LUT is a single-output logic block, so the assignment of side outputs is a new problem in FPGA technology mapping. A side output allows the reuse of a sub-AIC as an input to some other functions, making it a free logic; this encourages the use of side outputs whenever possible. Intuitively, a greedy approach is adopted for the assignment of these outputs. If a visible node can be assigned to a side



(a) The change in the LUTs' critical path delay as C increases.



(b) The change in the AICs' critical path delay as C increases.

Fig. 3: Effect of the number of cuts C on the critical path delay. Figure 3a shows, for a selection of benchmarks, how the critical path delay changes after the *forward traversal* as C increases for the LUT mapping, while Figure 3b shows the same data for the AIC mapping. In general, for LUTs, the delay tends to improve first but then stabilizes quickly. However, the AICs' critical path is highly sensitive to the number of cuts and tends to be unstable.

output, its best cut is discarded and that output is used. As more side outputs are utilized, more area is saved since the circuit would be mapped on fewer AICs.

E. Area recovery

Multiple iterations of area recovery can be added to the forward traversal phase. The initial execution of the forward traversal aims primarily at optimizing the critical path delay. At the end of this iteration, the critical path delay is know and the slack of the non-critical paths is computed. Any node that is not critical can be assigned a cut that is non-optimal for delay if it reduces the area of the circuit as long as the critical path delay is maintained. As such, several iteration of the area recovery can be performed with the sole objective of optimizing the circuit's area without affecting its critical path delay.

III. Multi-output and depth-specific issues

The transition from mapping on an input-constrained logic cell to a depth-constrained one, such as the AIC, involves theoretically fundamental but, in practice, fairly basic modifications. Redefining the feasibility and prioritization of the cuts, and assigning side outputs seem to introduce a natural extension of the existing algorithms customized for these new elementary blocks. However, basic testing revealed major shortcomings in this simplistic approach and raised doubts on (1) whether priority cuts can be applied to depth-constrained logic blocks and (2) whether the multiple outputs and, with it, the side output assignment can jeopardize the delay optimization. In order to identify the different deficiencies in the algorithm, each phase is tested separately and incrementally in this section.

Node	Cut list 1	Cut list 2	Cut list 3
A	$\{i_1, i_2\}$	-	-
B	$\{i_3, i_4\}$	-	-
C	$\{i_0, A\}$	$\{i_0, i_1, i_2\}$	-
D	$\{A, B\}$	$\{i_1, i_2, i_3, i_4\}$	-
E	$\{i_0, B\}$	$\{i_0, i_3, i_4\}$	-
F	$\{C, D\}$	$\{i_0, A, B\}$	$\{i_0, i_1, i_2, i_3, i_4\}$
G	$\{D, E\}$	$\{i_0, A, B\}$	$\{i_0, i_1, i_2, i_3, i_4\}$
H	$\{F, G\}$	$\{C, D, E\}$	$\{i_0, A, B\}$

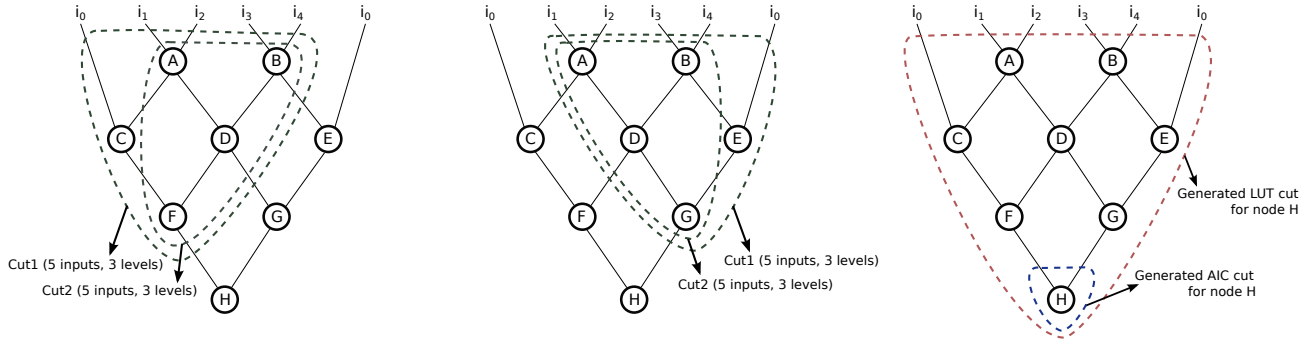
TABLE II: The generated cuts for each node (example of Figure 5) using the new method. With the new cut generation method, each node is guaranteed a cut with the maximum depth (if it is sufficiently distanced from the primary inputs). The cuts are expressed in terms of their inputs.

A. Cut generation for AICs

The currently used cut generation algorithm, described in Section II-B, generates, for every node of the AIG graph, a list of C feasible cuts. These cuts are prioritized according to the specified delay and area objective functions. It was shown that, for 6-input LUTs, 8 priority cuts are sufficient for near depth-optimal mapping while avoiding area penalties (for not enumerating all cuts) and dramatically improving the runtime and memory usage of the mapper [9]. As the number of priority cuts C increases, the results would improve but quickly stabilize once the optimal solution is reached.

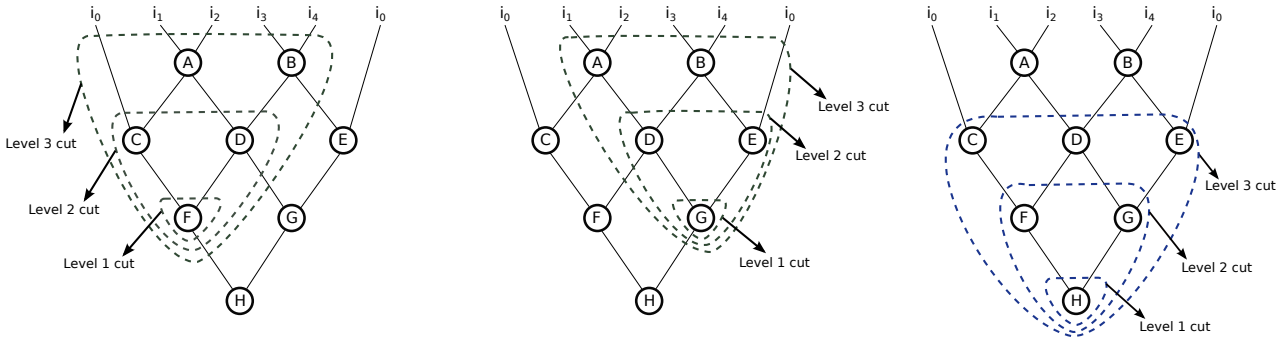
In order to verify that the priority cut algorithm is also efficient for depth-constrained cells, AICs in this case, we measure the critical path delay after the forward traversal phase. The experiment is repeated for both LUT and AIC mapping, over multiple values of C, and the results are reported in Figure 3. As expected, the number of LUT priority cuts has minimal effect on the critical path delay. As C increases, the delay improves at first and then stabilizes at its optimal value. However, the AIC results lack this stability and the delay varies with the number of cuts, without any predictable trend. Even for a very large number of cuts, the critical path delay varies and, in some cases, remains far from the optimal solution.

Mapping on AICs requires depth-feasible cuts, so the effect of cut generation might be completely different for AICs than it is for LUTs. Figure 4 shows this difference by applying priority cut generation on an AIG sub-circuit. Only 2 cuts are saved per node and mapping is done on AICs with maximum depth 3 (3-AICs) or LUTs with maximum 5 inputs (5-LUTs). The cuts generated for nodes F and G, given in Figures 4a and 4b respectively, have 5 inputs and a depth of three. The priority cuts of node H are generated by combining the different cuts



(a) Generated cuts for node F (b) Generated cuts for node G (c) AIC/LUT cut generation for node H

Fig. 4: An example on the difference between input-feasible and depth-feasible cut generation. The cut generation for AICs (depth-constrained) and LUTs (input-constrained) can have completely different outcomes. This example shows the cut generation for node H, given that two cuts are kept per node and that mapping is done on maximum 3-level AICs or 5-input LUTs. The leaf nodes (F & G) have cuts with maximum depth (3) and maximum allowed inputs (5). Mapping node H on AICs, constrains it with only a minimum depth cut while mapping it on LUTs can still generate an optimal cut (with 5 inputs).



(a) Generated cuts for node F (b) Generated cuts for node G (c) Generated cuts for node H

Fig. 5: New cut generation method. In this example, the same circuit of Figure 4 is mapped on 3-AICs but 3 cuts per node are kept, one cut for every allowed AIC depth. Given the cuts of F and G, the new method generates 3 cuts for H, one of which has the maximum depth of 3. Keeping one cut for every depth D guarantees the generation of cuts with maximum depth for every node (granted it is feasible).

of its leaves F and G. Since the cuts of F and G have a depth of 3, the cuts generated for H require a 4-level cone and are thus not feasible. So, H is mapped on a minimum depth cone which generally tends to be sub-optimal when optimizing for delay. On the other hand, LUT cuts are input-feasible and do not depend on the candidates' depth. Covering nodes with larger depth does not necessarily require a bigger LUT. Figure 4c shows, for the same sub-circuit, that even when the leaves have maximum-input cuts, an optimal LUT cut can still be generated for node H.

This example highlights how the problem of cut generation can be fundamentally different for AICs. A new priority cut algorithm is implemented to avoid this problem. Instead on keeping a single list of C prioritized cuts, multiple lists are now used, one for each depth. Thus the number of lists is now equivalent to the maximum allowed depth (*MaxDepth*) with $C/MaxDepth$ cuts per list. In each list, only cuts having the same depth are stored. Taking the same example of Figure 4, Table II lists the cuts generated for every node, using the new approach, and how they are stored in each list, assuming that 3 cuts are kept ($C = 3$) with a maximum depth of 3 ($MaxDepth = 3$). Figure 5 shows the generated cuts for nodes F, G and H, using the new approach. Since F and G have cuts with different (even minimum) depths, the generation of *MaxDepth* cuts for H is now possible. In general, the new approach guarantees, for every node, at least one cut with *MaxDepth* (as long as the AIG node has sufficient depth, greater than or equal to *MaxDepth*).

B. Side outputs assignment

The backward traversal phase, presented in Section II-D, consists basically of covering the AIG graph with selected nodes and assigning the side outputs of the AICs. The side outputs are considered free and assigned whenever possible, which limits the number of AICs covering the circuit and thus its overall area. Although side outputs are appealing and seem to come as a free lunch, they can actually have a large impact on the critical path delay. Using a side output means assigning

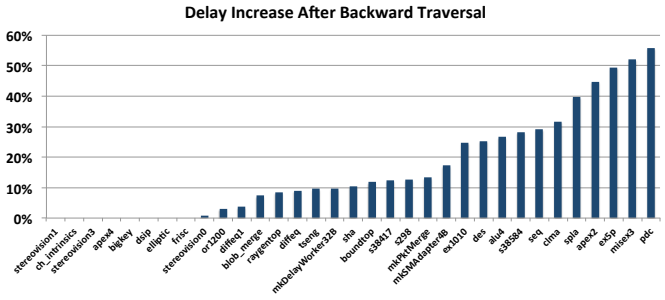


Fig. 6: Increase in critical path delay after the backward traversal. When the side outputs are greedily assigned, new critical paths can be created.

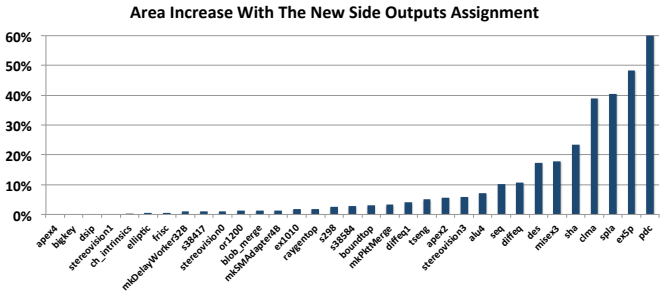


Fig. 7: Increase in area with the new side output assignment. This area increase is the cost paid for removing all the delay overhead of Figure 6. Limiting the side outputs to the ones that do not add to the critical path delay can increase the number of used AICs and as such the overall area.

a sub-AIC (since it is part of a bigger AIC) to an AIG node. However, this cone might not be the delay optimal candidate for that node. So the delay added by these sub-AICs can exceed the available slack in the circuit and create new critical paths that did not exist in the forward traversal. Figure 6 proves this phenomenon by showing the relative increase in critical path delay after the backward traversal, with respect to the delay computed at the end of the forward traversal. The increase in delay varies depending on the benchmark, exceeding 50% in some cases.

To solve this problem, side outputs must only be used if they do not create new critical paths. Thus, whenever a node can be assigned to a side output, its new delay is computed and the side output is selected if and only if the delay does not exceed the nodes' required time. Otherwise, the selected cut of the node (in the forward traversal) is used. With this new approach, all the delay increase of Figure 6 is avoided. However, as expected, this comes at the expense of area. Figure 7 represents the relative area increase between the naïve version described in Section II-D and the new side output assignment approach (the benchmarks are re-ordered in increasing area overhead). The naïve version presents a lower bound on the logic area needed to map the circuit. Whenever the side output is not used in order to avoid delay overhead, one more D-AIC is added to cover the entire graph. Despite the area increase, this enhanced side outputs selection guaranties the preservation of the critical path selected in the forward traversal according to the defined optimization objectives.

IV. Architectural Background

The mapping algorithm presented in the previous sections targets any depth-constrained FPGA logic blocks. But for convenience, the And-Inverter Cones are used to help explain the different concepts throughout the paper. This section provides detailed description of the AICs and their characteristics as well as the FPGA architecture used in the experimental setup.

A. And-Inverter Cone

The And-Inverter Cone is a tree structure of 2-input AND gates with programmable output inversion. An AIC is a depth-constrained logic block, characterized by its depth D, its 2^D inputs, $2^D - 1$ nodes and $2^{D-1} - 1$ outputs, as shown in Figure 2. The first level nodes have additional programmable input inversion [11], which improves the flexibility to the AIC allowing it to map a wider range of functions. The main AIC output is the one located at level D while the remaining outputs are called *side outputs*. Having multiple outputs enables the AIC to inherit a fracturable structure where every D-AIC can be split into two $(D-1)$-AICs. For instance, a 6-AIC can be divided into two 5-AICs, four 4-AICs, eight 3-AICs or sixteen 2-AICs. In general, and despite these characteristics, the AICs remain less flexible than the Look-Up Tables.

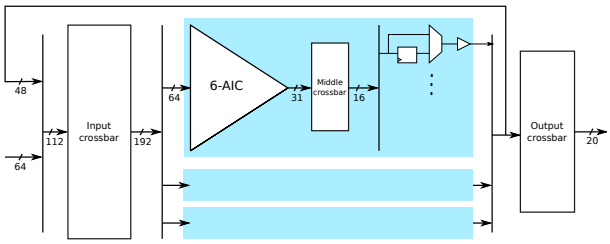


Fig. 8: AIC-based cluster with three 6-AICs.

B. AIC-based FPGA architecture

The AIC-based FPGA is composed of tiles known as logic clusters with AICs as the main logic blocks [10], [12]. The tiles are connected through global and programmable routing interconnects. The logic cluster, shown in Figure 8, consists of three AICs, each having 6 levels, 64 inputs and 31 outputs. Three crossbars, namely *input*, *middle* and *output crossbars*, are used to distribute the inputs or constrain the outputs. For instance, the middle crossbar, is used to select 16 out of the 31 outputs of the 6-AIC. In general, the cluster has 64 inputs, 48 local feedbacks and 20 outputs. This AIC-cluster tends to be at best about 18% larger than the Stratix-IV cluster with the crossbars contributing to about 80% of its area [12].

The VTR benchmark set used in the experimental setup requires some memory and multiplication blocks, so columns of these blocks are added to the FPGA architecture for every 8 columns of AIC clusters.

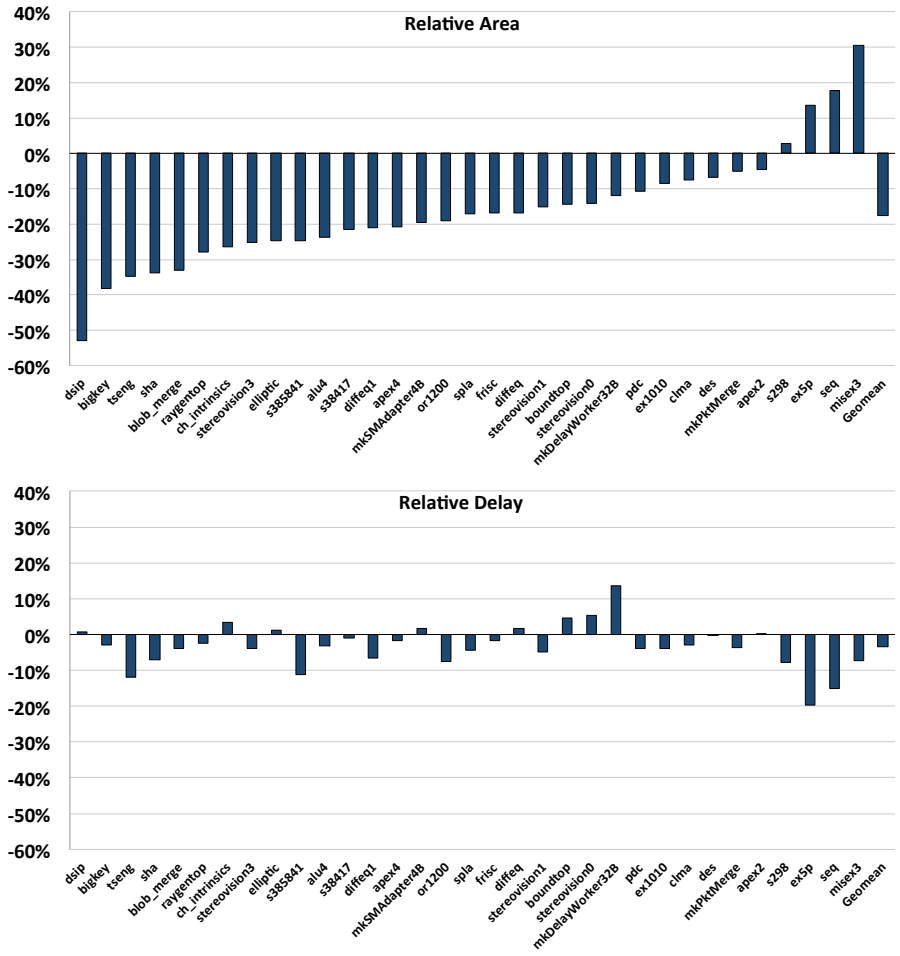


Fig. 9: Delay and area comparison between the new mapper and the existing AIC mapper [12].

V. Experimental results

To evaluate the quality of the new mapper, presented throughout this paper, we ran the complete FPGA tool flow on common benchmarks. The MCNC and VTR benchmark sets are first synthesized and mapped on D-AICs ($2 \leq D \leq 6$) using the latest ABC version extended with the new mapping algorithm ($C = 6$). Then, the VPR tool (VTR 7.0) [6] is used to pack, place and route the mapped benchmarks. VPR dynamically allocates routing channels during the placement and routing phase. Thus, after an initial run, the experiments are repeated for a fixed channel width 30% wider than the one needed in the first run.

We noticed that the critical path delay after placement and routing is highly sensitive to the used placement seed. So, to filter out the placement noise, we ran experiments with thirty different seeds and reported the average results over these seeds. To be able to compare our results to the most recently published ones on AICs, we use the same architecture files proposed in the last AIC paper [12]. We ran the experiments on the AIC-based FPGA architecture described in Section IV-B.

Figure 9 shows the relative difference in both delay and area between our and the reference results. Given that the comparison of the two mapping solutions is done at the end of the complete flow (after place and routing), this comparison is also affected by the variations introduced by VPR, which include mainly the global routing delay. Having, on average, nearly the same critical path delay suggests that both mappers present nearly-optimal solutions, with some noise due the delay of the global interconnects. However, while maintaining this average delay, our approach improves the overall circuit area by an average of 18%. More importantly, all the circuits generated by our mapper are Pareto optimal: Even in the rare cases where the mapper results in larger circuits, these are always faster than in the reference mapper showing a clear delay to area trade off.

Although the goal of this paper is to address the challenges of the technology mapping for depth-constrained logic cells, the reader may be interested in the overall result when one compares AIC-based architectures to LUT-based ones. Thus, we also modeled the Altera Stratix-IV architecture while properly accounting for its respective timing arcs and area costs (using again the architecture files of the most recent AIC paper [12]). The results reveal an average decrease in the delay and area by 24% and 15% respectively. Figure 10 shows

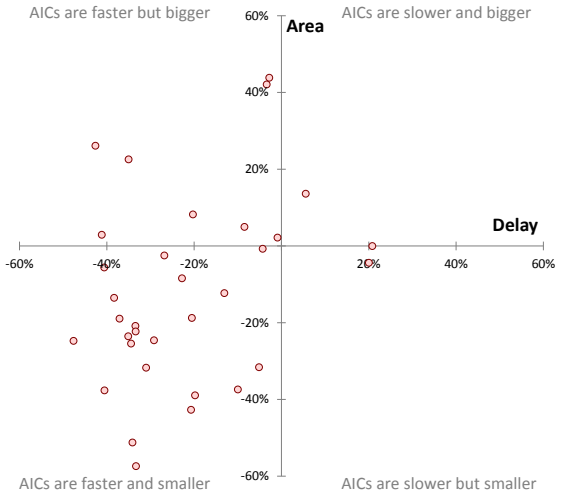


Fig. 10: Relative delay and area of AICs vs. LUTs. On average mapping on AICs improves both delay and area by 24% and 15% respectively.

the individual benchmarks: it is remarkable that two thirds of our benchmarks are in the 3rd quadrant and thus improve in *both* area and delay. Another quarter of the benchmarks land in the second and fourth quadrant of the diagram, showing a Pareto optimal solution with a different trade off of area and delay. Only two benchmarks have a negative result in the first quadrant, but, besides being a minority, suffer mostly a relatively marginal penalty compared to the significant advantages which other benchmarks enjoy (delay reduced up to half and area by more than half in extreme cases).

VI. Related Work

Mapping is one of the most researched topics in CAD for FPGAs [2] and researchers have explored several aspects such as circuit representations, optimization objectives (timing, area, power and routability), or logic transformations. Our work is based on the Priority Cuts algorithm [9] and is built on the latest version of ABC, which is used by almost all researchers in the domain.

Our naïve implementation of Section II is fairly similar to that of the first AIC mapper designed by Parandeh-Afshar et al. [10] but already improves in a couple of aspects: (1) only a subset of the highest-priority cuts is kept for each node during cut generation which has a substantial impact on runtime and (2) side outputs are assigned during the backward traversal, as opposed to directly assigning them during cut generation, allowing for a wider selection of free cones for each output.

In LUT-based architectures, fracturable LUTs are a common architectural feature; of course, these can be seen as LUTs with side outputs, much as in the case of AICs and uniform subgraphs. The main difference is that in the case of LUTs, most approaches simply treat fracturable LUTs as, as the name says, "fracturable", that is separate smaller LUTs packed together at a later stage of the FPGA toolchain [5], [6]. Our goal here is to exploit, whenever possible, the side outputs to implement real circuit fanout. Researchers also tried to enhance the existing mapping algorithms, introducing some awareness on the fracturable nature of the LUTs [4], [3], which improved the circuit density by using fewer LUTs and shorter wirelength. Nevertheless, the fractured LUTs were still considered as two separate units and not as one cell with multiple outputs like the uniform graphs targeted in this paper.

VII. Conclusions

In this paper, we introduce a new technology mapper for depth-constrained logic cells. In the process, we have shown that naïve adaptations of current LUT mappers are insufficient and that the typical mapping algorithms fail to properly handle the particularities of such cells. Detailed analysis of the different mapping stages inspired alternative solutions, tailored, as an example, to the AICs. The results are comparable to a reference AIC mapper in terms of delay but with 18% area reduction on average.

References

[1] L. Amarù, P.-E. Gaillardon, and G. De Micheli. Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization. In *Proceedings of the 51st Design Automation Conference*, pages 1–6, San Francisco, Calif., June 2014.

[2] D. Chen, J. Cong, and P. Pan. FPGA Design Automation: A Survey. *Foundations and Trends in Electronic Design Automation*, 1(3):139–169, Jan. 2006.

[3] D. Dickin and L. Shannon. Exploring FPGA technology mapping for fracturable LUT minimization. In *Field-Programmable Technology, 2011 International Conference on*, pages 1–8, Dec 2011.

[4] S. Jang, B. Chan, K. Chung, and A. Mishchenko. WireMap: FPGA Technology Mapping for Improved Routability and Enhanced LUT Merging. *ACM Transactions on Reconfigurable Technology and Systems*, 2(2):14:1–14:24, June 2009.

[5] J. Luu, J. H. Anderson, and J. Rose. Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 227–36, Monterey, Calif., Feb. 2011.

[6] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz. VTR 7.0: Next Generation Architecture and CAD System for FPGAs. *ACM Transactions on Reconfigurable Technology and Systems*, 7(2):6:1–6:30, July 2014.

[7] V. Manohararajah and S. Brown. Heuristics for area minimization in LUT-based FPGA technology mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-25(11):2331–40, Nov. 2006.

[8] A. Mishchenko, S. Chatterjee, and R. Brayton. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In *Proceedings of the 43rd Design Automation Conference*, pages 532–36, San Francisco, Calif., July 2006.

[9] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton. Combinational and sequential mapping with priority cuts. In *Proceedings of the International Conference on Computer Aided Design*, pages 354–61, San Jose, Calif., Nov. 2007.

[10] H. Parandeh-Afshar, H. Benbihi, D. Novo, and P. Ienne. Rethinking FPGAs: Elude the flexibility excess of LUTs with And-Inverter Cones. In *Proceedings of the 20th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 119–28, Monterey, Calif., Feb. 2012.

[11] H. Parandeh-Afshar, G. Zgheib, D. Novo, M. Purnaprajna, and P. Ienne. Shadow And-Inverter Cones. In *Proceedings of the 23rd International Conference on Field-Programmable Logic and Applications*, pages 1–4, Porto, Portugal, Sept. 2013.

[12] G. Zgheib, L. Yang, Z. Huang, D. Novo Bruna, H. Parandeh-Afshar, H. Yang, and P. Ienne. Revisiting And-Inverter Cones. In *Proceedings of the 22nd ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 45–54, Monterey, Calif., Feb. 2014.