Improving Technology Mapping for And-Inverter-Cones

Martin Thümmler, Shubham Rai, Akash Kumar
Chair for Processor Design, CfAED Technische Universität Dresden, Germany

***Abstract*—AND-inverter-cones (AICs), proposed in 2012, offer a suitable alternative to *Look-Up-Tables* (LUTs) as the basic building block for FPGAs. They support tapping of multiple side outputs and are intrinsically fracturable which favours reduction of logic duplication. Unlike *k*-inputs LUTs, their area scales linearly with the number of inputs. Technology mapping is one of the crucial tasks to realize the full power of AIC-based FPGAs. However, the current state-of-the-art implementations suffers two main drawbacks as they do not account for the AIC properties fully: (i) The required time set for each node is suboptimal in the context of AIC and that impairs the mapping quality; (ii) they rely on priority cuts, which are unnecessarily runtime-intensive in the context of AIC mapping. To improve the mapping quality, we propose and proof a new method to calculate the maximal required time for each node purely based on its graph depth and height. We propose an asymptotically runtime-optimal in-memory direct cut selection method which leads to similar area numbers ($\sim$ 1% area overhead) as our reference priority cut implementation. Combining these improvements with a second area recovery round leads to a final area reduction of 16.4% and 3% for the MCNC and VTR benchmarks respectively as compared to our reference implementation of the latest known technology mapper, while leaving the delay unaltered.**

I. Introduction

Traditionally, the functional flexibility offered *Look-Up-Tables* (LUTs) makes *FieldProgrammable Gate Arrays* (FPGAs) reconfigurable. A LUT is a multiplexer, where for each input signal combination the resulting output is stored in a static random-access memory (SRAM) cell. This exhaustive functional richness comes at the cost of exponential area scaling in terms of the number of inputs. In 2012, *And-Inverter-Cones* (AICs) [1] were proposed as an alternative to LUTs. A d-AIC is a cone tree-like structure of depth d consisting of And gates, whose outputs can be optionally inverted using the programmable SRAM cells. Another very similar implementation are the *Nand-Nor-Cones* (NNCs) [2], where each gate acts either as NAND or as NOR. NNCs can be a suitable implementation paradigm for emerging technologies such as spin-based [3] or reconfigurable nanotechnology devices [4, 5].

Their structure is motivated by the use of *And-Inverter-Graphs* (AIGs) in current logic synthesis algorithms. The node functionality inside an AIG is directly mapped to single AIC-gates to omit the area-heavy LUT structures. AICs are superior to LUTs in the following properties: (i) First, an AIC has an exponentially increasing number of inputs in terms of its depth, while LUTs have only a limited number of inputs, i.e. less than 10; (ii) Second, with AICs, multiple side outputs can be tapped which can be utilized to reduce logic duplication; (iii) Third, the physical AICs can be split logically into multiple smaller AICs i.e. they are fracturable. Even if, there exist fracturable LUT implementations [6], they require additional hardware effort. LUTs provide no possibility of splitting the fractured LUTs again, too. The last two properties lead to reduce logic duplication for AICs-based FPGAs.

Current AIC technology mapping algorithms have two major shortcommings, because their main ideas are transfered from LUT mapping, hence they do not account for the special properties of AICs: (i) First, the required times by choices method [7] does not assign each AIG node a useful required time, which deteriorates the flexibility of the mapper during the area recovery as argued in section III; (ii) Second, priority cuts methods with reasonable cut sizes (runtime) are infeasible to consider a significant quota of all the possible cuts. Hence the cut pruning is at choice. Therefore they lead not to an improved mapping quality compared to a fast direct cut selection method which the experiments in section V proofs.

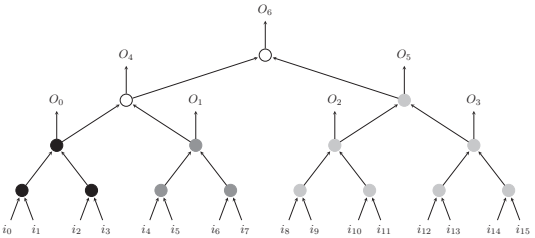


Fig. 1: And-Inverter-Cone of depth 4. The black, gray, and light gray colored gates indicate one possible use as two 2-AICs and one 3-AIC. The light gray cone has one main output (O_5) and two side outputs (O_2 and O_3). [1]

We propose two algorithmic improvements to exploit the AIC properties fully, namely

- **Optimal calculation of the latest possible required times:** The required time of each node is directly inferred by its graph height, as well as the circuit depth for unconstrained timing. Additionally it is shown, that the so calculated required times are optimal, i.e. any violation leads directly to an increased circuit delay.
- **Direct cut selection:** Instead of using priority cuts for the mapping, a new direct cut selection method based on a subgraph local view is proposed. The cut selection method performs its calculation in-memory and is faster than the priority cuts method.

These two improvements leads together with a second area recovery round to an average area reduction of 16.4% (3%) for the MCNC (VTR) benchmark, while leaving the average delay practically unchanged. Additionally, the implemented algorithms are provided open-source as an extension of the open-source system Abc [8].

II. Background

The cone-like structure of an AIC is depicted in figure 1. It can be logically split as indicated into smaller sub-AICs. Each logical AIC has exactly one main output (at highest level), the remaining outputs are called side outputs. Note, that a similar-sized AICs and LUTs implement different functions. For instance, a 3-AIC can implement the conjunction of 8 inputs signals, which is impossible for a 3-LUT. On the opposite side, an exclusive Or of three input signals can be implemented by a 3-LUT, but not by a 3-AIC. To map a subgraph to an AIC

978-3-9819263-6-1/DATE22/©2022 EDAA

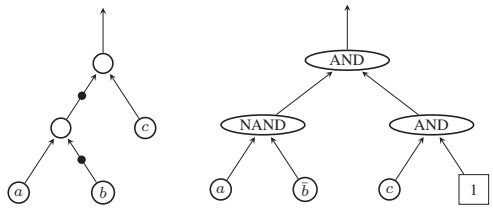


Fig. 2: Mapping of a 2-depth-feasible subgraph (shown in (a)) to to an 2-AIC with an addtional constant driver (see (b)).

of depth – also called level – d it has to be d-feasible, i.e. has a depth of at most d. An example is shown in figure 2. The mapping of the depth-feasible And-Inverter-subgraphs to AICs and NNCs requires only the correct configuration of the gates and the insertion of constant drivers for some inputs.

A. Definitions

Throughout the manuscript, various terminologies have been used. These are listed as follows. A *cut* rooted at a node n is a set of nodes, through which every path from any primary input to n traverses at least once. A cut A is *dominated* by a cut B, if $B \subset A$. The set of input nodes (leaves) of a subgraph G of a full AIG will be denoted as L_G. A mapping of an AIG for LUTs (AICs) is an equivalent representation of that graph, where each node functionality can be calculated by the given logic cells. The area and delay of a logic cell C are denoted as A_C and D_C , respectively. The area of the logic cell required to implement the functionality of a cut C is denoted as A(C). The *arrival time* of a node n in a mapping M will be denoted as $a_M(n)$. In other words $a_M(n)$ is the minimal time, after which the value of the node n is calculated in the mapping M. For sake of simplicity, the subscript M will be left out, whenever possible. The earliest possible arrival time of a node n is defined as $a_{\text{opt}}(n)$, i.e. the minimum arrival time over all possible mappings containing n as primary input or logic cell output. The *required time* $r(n)$ for a node n is the lowest upper boundary of the arrival time $a(n)$, for which it can be guaranteed, that a mapping exists, which serves for all primary outputs the required times. A node is *visible* in a mapping if it is an input or an output of an AIC or the graph. The remaining nodes are *invisible*. The *height* (*depth*) of a node n, i.e. the length of the maximal path from any primary input (output) to the node n, is denoted as h_n (d_n).

B. Related Work

The publication [1] from 2012, which introduced AICs, proposed a technology mapper (TM) to map to LUTs or AICs simultaneously. First, all (depth) feasible cuts for each node are calculated, by varying its cone depth. They claimed, that the computational effort is proportional to the maximum allowed cone depth. In the absence of an open-source implementation, according to our understanding, we expect that they generated only a single cut per cone depth. Then, a topological graph traversal is performed to calculate the initial arrival time for each node. For each node n, their consideration included not only the cuts rooted at n, but also all cuts, where n can be used as side output. If two cuts for the same node n lead to the same arrival time for n, then the area flow heuristic [9] is used as a tie-breaker. The side outputs are used regardless of the required time.

The next – and to our best knowledge – the last algorithmic improvement was done by the introduction of a depth-constrained FPGA logic cell mapper [10] in 2015. The authors highlighted, that during side output selection the required time has to be taken into account, to avoid an increase of the critical path. Additionally, they adapted the priority cuts method for LUTs to depth constrained logic cells to reduce the circuit area by performing a single area recovery round. The authors recognized, that a single priority cut list [11] for each node like for LUT mapping is not sufficient. So they proposed to maintain for each cone-depth at each node an extra cut list. Note that for this algorithm it is not explicitly reported, how the required times are calculated. So, we assume that they used the required time by choices method, as they referenced to [11].

III. Limitations of previous TMs

A. Nodes with unset required times

All of the previous known technology mappers for depth constrained logic cells do not account for the huge cut sizes (in terms of nodes) and enormous number of non-dominated cuts for nearly all nodes. The huge cut sizes, i.e. for an 6-AIC up to 64 inputs and 62 invisible nodes, lead to a significant quota of invisible nodes for each mapping. Therefore, in the current implementation [11], the required time calculation based on the required time by choices (in the current implementation within ABC) ignores the invisible nodes. The required times are set to infinity (the maximum allowed circuit delay) for each node (primary output). Then in a reverse topological order traversal for each node n the required times of the input nodes of the previously calculated representative cut rooted at n are updated to ensure that they obey the global timing constraint. This implies, that only the required times of the visible nodes are updated. Hence, the required times of the remaining nodes will remain unset, i.e. set to infinity. This impairs the flexibility of the cut selection in the following area recovery rounds as shown in the example in figure 3. The reason is, that the nodes with unset required times may get assigned cuts leading to arrival times, which are quite large to be used as a part of a cut rooted at any visible node without violating its required time. Therefore the use of nodes with unset required time in the resulting mapping can be prohibited by choice. To overcome this issue the section IV shows how a useful required time can be set for all nodes.

B. Computational overhead of priority cuts

The runtime for priority cuts approaches scales quadratically in the maximum allowed cut list length per node due the number of required cut merges. For multiple cut list, like for one per depth [10], the runtime scales quadratically in the maximum cone depth, too. However, the proposed direct cut selection method, see section IV, has a runtime proportional to the maximum depth-feasible subgraphs. Note, that this is asymtotically optimal as it is expected, that for any (reasonable good) mapping, the average size of the cuts is proportional to the maximum subgraph size.

Now, we argue that the additional computational effort of priority cuts will not lead to a significant gain of knowledge to select the cuts more consciously. For that purpose consider the maximum number of possible (non-dominated) cuts of depth d, called C_d. It increases rapidly with the maximum allowed cone

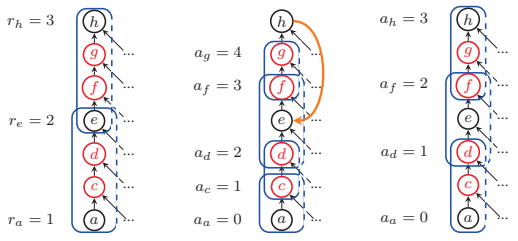


Fig. 3: Negative effect of nodes with unset required time, which are marked in red. Assuming a unit delay model, for a maximum cone depth of 3, a required time for node h is 3, and arrival time of a has to be zero. The required times (r) and arrival times (a) are depicted left of the nodes. The representative cuts are marked by the blue rectangles. Their dashed lines indicate, that more nodes will be inside the cuts. (a) Possible set of representative cuts after initial graph traversal for circuit arrival time calculation. Let the resulting required time for the node h be $r_h = 3$. (b) Example how the graph might be covered in a following traversal. Now the cut rooted at node h must contain the node e as the nodes f and g have large arrival times. (c) Example of a valid covering using the node f in a cut rooted at h.

TABLE I: Maximum possible number of non-dominated cuts C_d as a function of the subgraph depth d.

C_d	0	1	2	3	4	5	6
d	1	1	3	21	651	457,653	210,065,930,571

depth as shown in table I, and can be calculated by $C_{d+1} = C_d^2 + 2C_d \sum_{k=0}^{d-1} C_k$ with $C_0 = 1$. Note that this formula counts the number of possible subgraphs with depth d recursively based on the assumption, that all possible input nodes are different. Even if the real number of cuts might be smaller as nodes may occur multiple times in the depth-feasible fanin cone, for most nodes number of non-dominated cuts will be much larger than 10^4. Hence, any priority cut list [11, 10] based algorithm (with reasonable runtime) considers only a very small quota of all the possible cuts and performs its cut list pruning practically by choice. Note that this is contrary to LUT-mapping, where the number of possible cuts is on average in the order of 10^2.

IV. The new TM

The new technology mapper is based on a constant delay model, i.e. a cone of depth d has – regardless of its configuration – a delay of D_d. Note that D_d increases with d. Let $d_{\max}$ be the maximum cone depth. For the sake of simplicity, we restrict ourselves in the following section to timing unconstrained mapping as handling various constraints on the mapping is just a mere extension of this problem as it is a more runtime-intensive computation. In the present work, our algorithm handles a constant required time for each node in the logic graph. This is also consistent with the current implementation within ABC [8]. An overview of our technology mapper is presented in algorithm 1. First, (see section IV-A), we verify the validity of the required time calculation. Afterwards, we discuss the implementation details.

During the algorithm, first the required times are calculated. Afterwards, area recovery rounds are performed, i.e. for each node in topological order traversal, the best cuts based on the (estimated) area flow heuristic are calculated. Lastly, the side outputs are assigned.

Algorithm 1 AIC-Technology mapper

procedure MAP(AIG G, $d_{\max}$, $\{D_d\}$)
  assign depth and height for each node(G)
$arrival$ = optimalArrivalTimes($depth$, $d_{\max}$, D_d)
for *each node* v *in* G **do**
  $v_{\text{required}} = arrival[d_{\max}] - v_{\text{depth}}$
  for *each node* n *in topological order of* G **do**
    $n_{\text{af}} = \infty, n_{\text{cut}} = \{\}$
    for *each* d *in* $\{d_{\max}, ..., 2, 1\}$ **do**
      G_S = copySubgraphRootedAt(n, d)
      for *each node* n *in* G_S **do**
        $n_{\text{afe}} = n_{\text{af}}$/numberOfOccurences($n, G_S$)
      cut_d = greedyMinimize(G_S) // see figure 4
      af_d = calculateRealAreaFlow(cut_d)
      if $\text{af}_d < af_{\text{best}}$ **then**
        n_{af}, n_{cut} = af_d, cut_d
performAreaRecoveryRounds(G) // see [9]
assignSideOutputs(G) // see section IV-C
end procedure
procedure OPTIMALARRIVALTIMES($graphDepth, d_{\max}, \{D_d\}$)
  $arrival[0] = 0$
for $d \in \{1, 2, \ldots, graphDepth\}$ **do**
  $arrival[d] = \min_{k \le d_{\max}, k \le graphDepth}(arrival[d-k] + D_k)$
return $arrival$
end procedure

A. Direct required time calculation

First we show the following statement about the minimal achievable arrival time. Second, we use that statement to define a maximal and useful required time for each node.

> In the given model, the least possible arrival time depends only on the node height, i.e. $a_{\text{opt}}(n) \equiv a_{\text{opt}}(h_n)$. Additionally, $a_{\text{opt}}(h)$ is an increasing function in h.

Proof by induction over node height. For circuit inputs n, we have by definition $a_{\text{opt}}(n) = a_{\text{opt}}(h_n) = a_{\text{opt}}(0) = 0$. Now assume, that $a_{\text{opt}}(n) = a_{\text{opt}}(h_n)$ for every node n with $h_n \le h_{\text{ind}}$ holds, where h_{Ind} is a positive integer. Then for every node r of height $h_r = h_{\text{ind}} + 1$ the equation

$$a_{\text{opt}}(r) = \min_{1 \le d \le d_{\max}} \left\{ D_d + \min_{c \in \mathrm{S}_d(r)} \max_{n \in c} \{a_{\text{opt}}(n)\} \right\}, \quad (1)$$

holds, where $\mathrm{S}_d(r)$ is defined as the super set of inputs (leaves) of all subgraphs rooted at r with a depth not exceeding d. The equation (1) expresses, that the minimal arrival time is computed over all possible cell depths and all possible feasible subgraphs based on the latest arrival time of the subgraph leaves. This equation can be transformed using $a_{\text{opt}}(n) = a_{\text{opt}}(h_n)$ for all subgraph leaves as they have a height less or equal h_{ind}. Therefore

$$\min_{c \in \mathrm{S}_d(r)} \max_{n \in c} \{a(n)\} = \min_{c \in \mathrm{S}_d(r)} \max_{n \in c} \{a(h_n)\} \quad (2)$$

holds, using the monotonic property of the arrival time, the equation reduces to

$$\min_{c \in \mathrm{S}_d(r)} \max_{n \in c} \{a(n)\} = a(\min_{c \in \mathrm{S}_d(r)} \max_{n \in c} \{h_n\}). \quad (3)$$

Now the minimum function can be evaluated by considering the possible subgraphs. The maximal subgraph (in terms of size) with a depth of at most d will always lead to the minimum value. As all other subgraphs contain at least one leaf l, whose

fanins can be included in the subgraph without exceeding the maximum depth d. As the fanins of l have a smaller height than l, they will therefore never lead to an increased maximum height over all leaf heights of the subgraph.

The maximum height of all the leaves of the maximal d-depth-feasible subgraph rooted at r is given by $\max\{0, h_r - d\}$, which follows directly from the definition of the height of a node and induction over the maximum depth d. Therefore equation (1) reduces to

$$\begin{aligned} a_{\text{opt}}(r) &= \min_{1 \le d \le d_{\max}} \{D_d + a_{\text{opt}}(\max\{0, h_r - d\})\} \\ &= \min_{1 \le d \le d_{\max},\, d < h_r} \{D_d + a_{\text{opt}}(h_r - d)\}. \end{aligned} \tag{4}$$

Note that this equation has two properties: First, $a_{\text{opt}}(r)$ is determined only by its node height and the arrival time of smaller node heights. Second, $a_{\text{opt}}(h_r) \ge a_{\text{opt}}(h_r - 1)$ as can be seen directly from equation (4) and $a_{\text{opt}}(h_r - d - 1) \le a_{\text{opt}}(h_r - d)$ for all possible values of d. ■

By using equation (4) the optimal circuit arrival time a_G of an AIG can be calculated directly as a function of the logic cell delays and the circuit depth. Let us define the required time $r(n)$ for each node n as

$$r(n) = a_G - a(d_n), \tag{5}$$

where d_n is the depth of the node n. The so calculated required times are valid, i.e it suffices (and is possible) to respect the required time for each (visible) node in a topological order traversal of the graph to ensure that circuit delay is not increased. This can be seen, when considering only the subgraph G_d (of the full AIG G), which contains each node with a depth up to d. Let L_d be the input nodes (leaves) of G_d. Note that all the nodes in L_d have by definition a height of at most d. It directly follows, that for each circuit output c, the minimal achievable arrival time $a_{\text{opt}}(c)$ is bounded by

$$\begin{aligned} a_{\text{opt}}(c) &\le \max_{l \in L_d} \{a(l) + a(d_l)\} \\ &\le a(d) + \max_{l \in L_d} \{a(l)\}, \end{aligned} \tag{6}$$

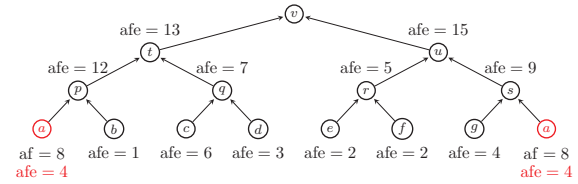
after applying Equation 4 to the subgraph G_d. Now, if equation (5) is served, then $\max_{l \in L} \{a(l)\} \le a_G - a(d)$ is guaranteed. This means, that under this condition there exists always a mapping, for which each circuit output arrival time is limited by

$$a_{\min}(c) \le a(d) + [a_G - a(d)] = a_G. \tag{7}$$

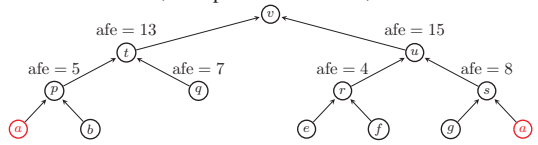
As the depth d can be chosen arbitrarily, the required times are valid for every node in the AIG. Note that the required times can not be increased further as for every depth d at least one node $n \in L_d$ on the critical path exists. For this node $h_n = D_{\text{AIG}} - d$ holds, therefore all inequalities in the equations (6) and (7) become equal.

B. Direct cut selection

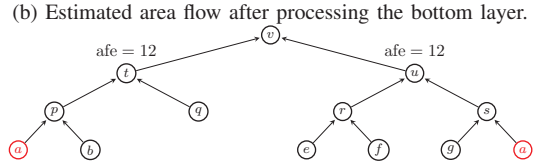
The proposed algorithm for the direct cut selection is applied during the area recovery phase. Its goal is to calculate for each node a cut with minimal area flow [9, 1], that serves the required times. The direct cut selection is applied for the subgraph depths from 1 to $d_{\max}$ separately to account for all possible logic cells. Then, the cut with the best area flow serving the required time is chosen. The core algorithm starts with the maximum possible depth-feasible subgraph G rooted at node r.



(a) Initial estimated area flow (afe), it is identical to the previously calculated area flow, except for the node a, which occurs twice.



(b) Estimated area flow after processing the bottom layer.



(c) Finally selected subgraph.

Fig. 4: Subgraph selection based on the estimated area flow heuristic. Note, the resulting cut minimzes the area flow for the node v, which would not be the case if the pure area flow was greedily minimized instead.

During a topological order traversal the subgraph is pruned at a node n if $\text{afe}(n) < \text{afe}(n_1) + \text{afe}(n_2)$, where n_1 and n_2 the two fanins of n and afe stands for *estimated area flow*. If the subgraph is not pruned, the node esimated area flow is updated as the sum of the estimated area flows of its fanins. The afe is defined as

$$\text{afe}_G(v) = \text{af}(n)/\text{numberOfOccurences}(G, v), \tag{8}$$

where af is the area flow [9] and $\text{numberOfOccurences}(G, v)$ is the number of occurrences of the node v inside the graph G. This renormalization of the area flow is introduced to make the greedy approach aware that some nodes can be used multiple times but contribute only once to the area flow. An explanatory example is given in figure 4. Afterwards, the pruned subgraph is determined, the corresponding non-dominated cut is calculated. Lastly, the area flow for the considered node r is calculated based on the selected cut.

C. Additional algorithmic details

Contrary to the previous known technology mappers an arbitrary number of area recovery rounds may be performed. The weighting average to estimate the expected fanout is done only for the main outputs of the AICs. The side output assignment is done similarly to [10] as last step of the mapping algorithm: In a first graph traversal all possible side outputs are detected and marked if they satisfy the required time. In a second topological order graph traversal the side outputs are assigned in the mapping. As this may make some previous assigned AICs obsolete, a third graph traversal is conducted to tidy up the netlist.

For reference purposes the priority cut algorithm from [10] was reimplemented except the only difference, that the cut lists are not globally stored, but locally recalculated for each node and cone level combination.

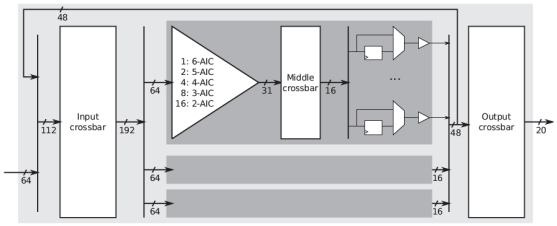


Fig. 5: Used AIC block design from [13] with 50% populated input-crossbar and a minimal full-capacity middle crossbar.

V. Experiments and Discussion

A. Experimental setup

The performance of the TM is evaluated based on emulated FPGA designs. We use VTR8 [12] with our extended Abc version and one architecture file from [13], which customizable logic block design is depicted in figure 5. Before calling the TM, the circuit is optimized by Abc with the commands `strash`, `balance`, `ifraig`, `scorr`, `dc2`, `dch` for both pure AIC and pure LUT mapping. The AIC-TM is invoked with the delay values from the architecture file for the cones and an additional wire delay of 1 ns to account for the experimental found average routing delay between two AICs. The correctness of the netlist is verified with Abc. For reference LUT mapping the default VTR8 tool flow is used with a comparable architecture file (Stratix Lab line IV emulation). The following results are the averaged over 10 runs of VPR each with an 30% increased routing channel width. This averaging is required as the quality of the placement and routing is highly sensitive to the initial seed of VPR [10].

B. Discussion

Before presenting the experiments, let us define the different mapping options:

1) A1p: One area recovery round, priority cuts, required time by choices,
2) A1pS: One area recovery round, priority cuts, direct required time
3) A1dS: One area recovery round, direct cut selection and direct required time
4) A2dS: Same as A1dS except two area recovery rounds

The priority cut lists have a maximum size of 3 per cone depth. Note, that A1p is used as reference implementation of the previous technology mapper [10]. Figure 6 depicts the area and delay of the emulated MCNC and VTR circuits. For the MCNC (VTR) benchmarks shown in figure 6a (6b) we see that the area is reduced by 10.9% (2.5%) on average, when applying the direct cut selection method (A1pS) compared to our reference (A1p). For the option A1dS we get an area reduction of 11.5% (area increase of 1.4%). The A2dS leads to an area reduction of 16.4% (3.0%). This shows that the direct required time calculation method is superior to the previous ones. The direct cut selection method leads for most of the circuits to an area change of less than $\pm 5\%$ compared to the priority cut list method. The on average slightly worse results for the VTR circuits are outperformed by the additional area recovery round, which gets feasible as the runtime is reduced. Note that, the effect of the TM is for the VTR circuits smaller than for the MCNC ones, as their implementation relies on memory and multiplication cells.

The critical path of the MCNC (VTR) circuits changes on average by +3.1%, 2.5% and $\sim 0\%$ (+0.2%, -1.1%, $\sim 0\%$) for the A1pS. A1dS, A2dS mapping compared to A1p. The delay changes for the different mapping options is for most of the circuit implementations less than 5 %. However, for some circuits, like *pdc*, see figure 6a, it changes over 10 %. We ensured, that the timing levels of the mapped netlists are equal for the different mapping options. Therefore these delay changes must be rooted in the place and route phase. We propose, that these delay differences are mainly caused by different packings of the AICs: (i) The packing highly depends on the distribution of the netlist wires, which are not explicitly controlled by the TM; (ii) the local routing delay is much smaller than the global routing delay; which is not considered in the proposed model; (iii) this is coherent with the high sensitivity of the delay to the inital VPR (placement) seed. Similarly, for LUT-based FPGAs have in comparison to the A2pS mapping an average area of +35.2% (-0.5%) and delay of 56.7 % (29.2%) for the MCNC (VTR) benchmark circuits.

VI. Summary and Conclusion

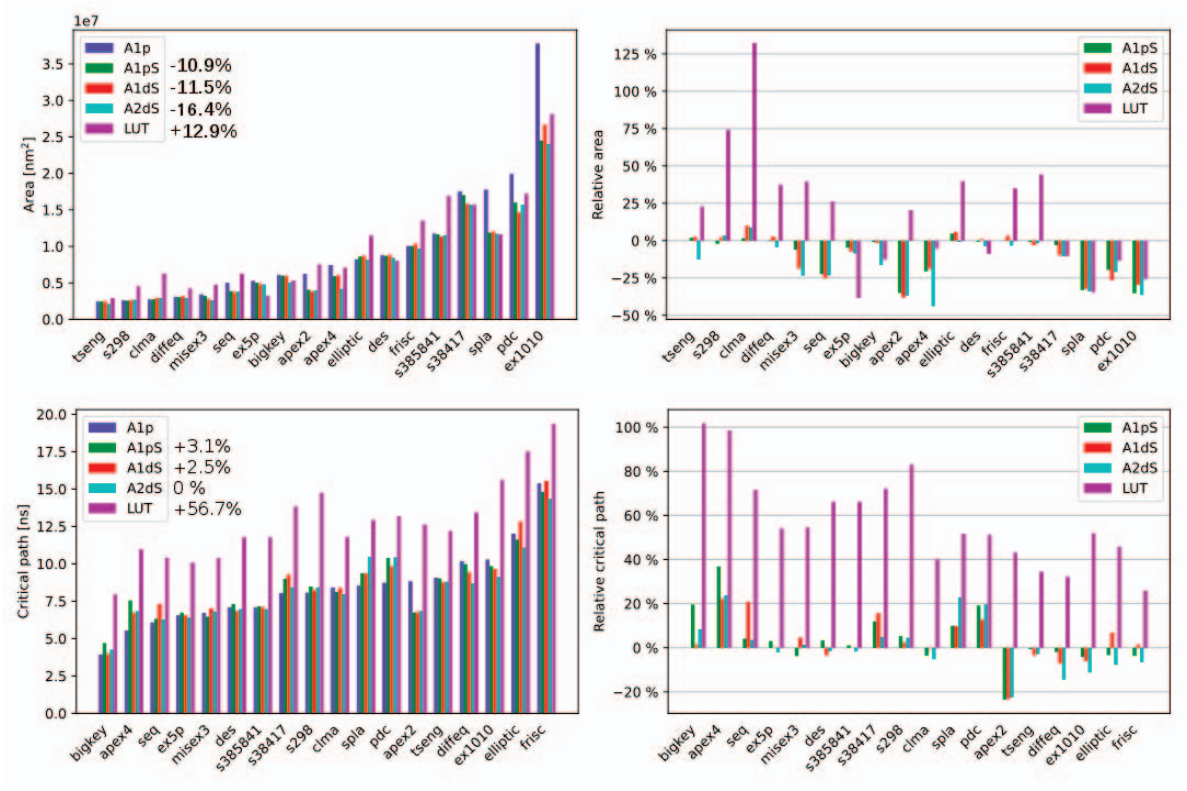
We demonstrated and proofed how to set an sensible required time for each node for an AIC technology mapper. Additionally, we showed that for AICs a priority cut list approach can be replaced by a much faster direct cut selection method, which made a second area recovery round feasible. On average we improved the area by 16.2 % (23 %) and with a delay change of 12 % (12 %) for the MCNC (VTR) benchmark. The next step is to adapt the nearly 10 year old architecture files to a more reliable comparison between LUTs and AICs. A future work is to investigate the effect of the architecture and placement on the delay further, to develop a enhanced placement aware TM to reduce the delay variation. The source code of the proposed TM can be downloaded at: *https://github.com/chacheline/abc_aic_mapper/tree/aic*

Acknowledgments

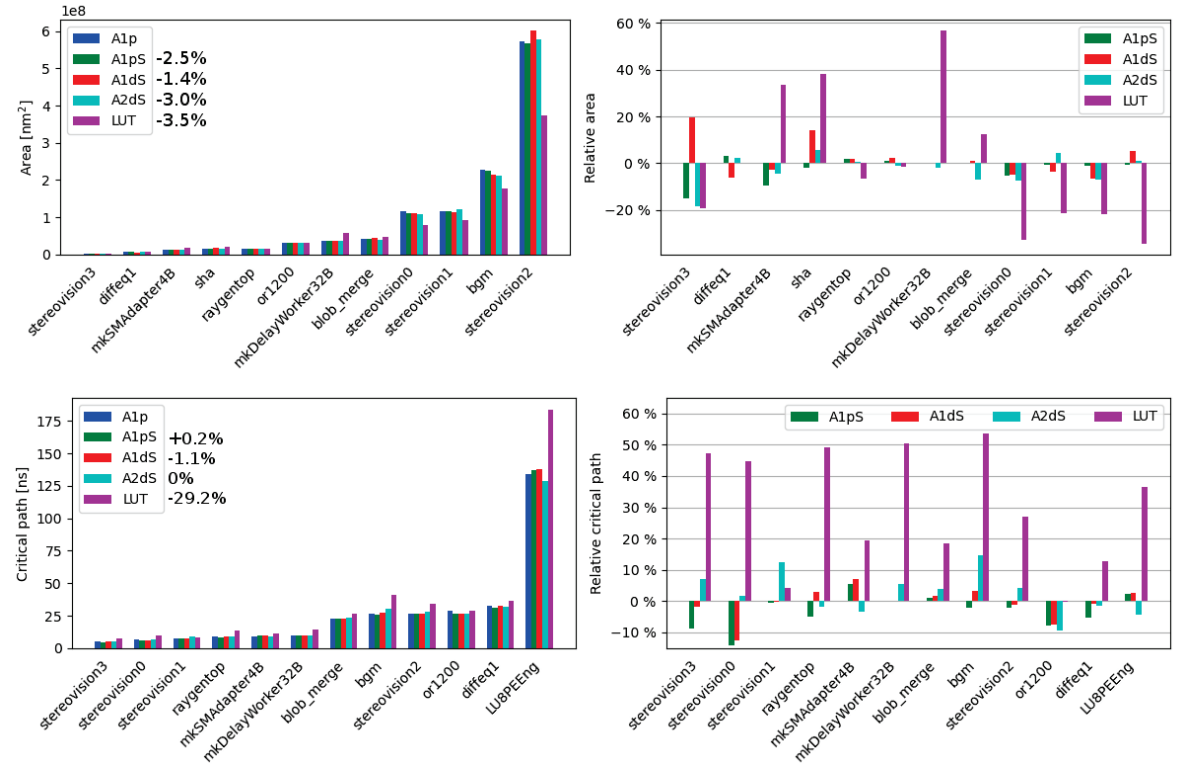
This research was supported in part by the German Research Foundation (DFG), project SecuReFET (Project Number: 439891087).

References

[1] Hadi Parandeh-Afshar et al. “Rethinking FPGAs: Elude the Flexibility Excess of LUTs with and-Inverter Cones”. In: *ISFPGA*. 2012.
[2] Zhihong Huang et al. “NAND-NOR: A Compact, Fast, and Delay Balanced FPGA Logic Element”. In: *ISFPGA*. FPGA ’17. 2017.
[3] Stephen M. Williams and Mingjie Lin. “Architecture and Circuit Design of an All-Spintronic FPGA”. In: *ISFPGA*. 2018.
[4] S. Rai et al. “Designing Efficient Circuits Based on Runtime-Reconfigurable Field-Effect Transistors”. In: *TVLSI* (2019).
[5] S. Rai et al. “A Survey of FPGA Logic Cell Designs in the Light of Emerging Technologies”. In: *IEEE Access* (2021).
[6] Wenyi Feng, Jonathan Greene, and Alan Mishchenko. “Improving FPGA performance with a S44 LUT structure”. In: *ISFPGA*. 2018.
[7] Satrajit Chatterjee et al. “Reducing structural bias in technology mapping”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.12 (2006).
[8] Robert Brayton and Alan Mishchenko. “ABC: An Academic Industrial-Strength Verification Tool.” In: vol. 6174. 2010.
[9] V. Manohararajah, S.D. Brown, and Z.G. Vranesic. “Heuristics for Area Minimization in LUT-Based FPGA Technology Mapping”. In: *IEEE TCAD* 25.11 (2006).
[10] Zhenghong Jiang et al. “A technology mapper for depth-constrained FPGA logic cells”. In: *FPL*. 2015.
[11] Alan Mishchenko et al. “Combinational and sequential mapping with priority cuts”. In: *ICCAD*. 2007.
[12] Kevin E. Murray et al. “VTR 8: High-Performance CAD and Customizable FPGA Architecture Modelling”. In: *ACM TRETS* 13.2 (May 2020).
[13] Grace Zgheib et al. “Revisiting And-Inverter Cones”. In: *ISFPGA*. 2014.



(a) MCNC



(b) VTR

Fig. 6: Absolute and relative change of area and delay of different circuits. The numbers indicate the average (geomean) relative area and delay changes.