Revisiting And-Inverter Cones

Grace Zgheib[†] Liqun Yang[‡] Zhihong Huang[‡] David Novo[†]
Hadi Parandeh-Afshar[†] Haigang Yang[‡] Paolo Ienne[†]

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

[‡]Chinese Academy of Sciences
Institute of Electronics, Beijing, China
yangliqun10@mail.ucas.ac.cn, {huangzhihong, yanghg}@mail.ie.ac.cn

ABSTRACT

And-Invert Cones (AICs) *have been suggested as an alternative to the ubiquitous* Look-Up Tables (LUTs) *used in commercial FPGAs. The original article suggesting the new architecture made some untested assumptions on the circuitry needed to implement AIC architectures and did not develop completely the toolset necessary to assess comprehensively the idea. In this paper, we pick up the architecture that some of us proposed in the original AIC paper and try to implement it as thoroughly as we can afford. We build all components for the logic cluster at transistor level in a 40 nm technology as well as a LUT-based architecture inspired by Altera's Stratix IV. We first determine that the characteristics of our LUT-based architecture are reasonably similar to those of the commercial counterpart. Then, we compare the AIC architecture to the baseline on a number of benchmarks, and we find a few difficulties that had been overlooked before. We thus explore other design possibilities around the original design point and show their detailed impact. Finally, we discuss how the very structure of current logic clusters seems not perfectly appropriate for getting the best out of AICs and conclude that, even though they are not confirmed as an immediate blessing today, AICs still offer rich research opportunities.*

Categories and Subject Descriptors

B.6.1 [**Logic Design**]: Design Styles—*Logic arrays, Combinatorial Logic*; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Gate arrays*

Keywords

FPGA Logic Block, And-Inverter Cones, Transistor Design

FPGA'14, February 26–28, 2014, Monterey, CA, USA.

http://dx.doi.org/10.1145/2554688.2554791.

1. INTRODUCTION

Field-Programmable Gate Arrays (FPGAs) are an important class of semiconductors with a tangible and growing market share. The key of their future growth is in the capability of the vendors to improve their effectiveness (e.g., the ability to run user designs at higher speeds) and their efficiency (e.g., run the same user design at a considerable lower silicon real-estate cost). Vendors and researchers alike actively work on the architecture of FPGAs, and seldom two generations of devices by the same vendors have shared the very same architecture. Yet, some important architectural elements have now been constant for many years, and among them the use, at least in all widely available commercial devices, of *Look-Up Tables (LUTs)* as the basic logic element.

A couple of years ago, some researchers argued that the costly versatility of LUTs goes wasted in most cases when the logic synthesis process uses representations other than the classic algebraic one [4]. Inspired by some modern synthesizers representing circuits as networks of AND-invert operations [8], Parandeh-Afshar et al. suggested replacing LUTs with a new logic element in the form of a binary tree of AND gates with conditional inversion. This new logic element is called *And-Inverter Cone (AIC)* [10] and its configuration bits would decide the function of a specific AIC by selecting whether the output of each AND gate is inverted or not. The authors argued that the similarity of the logic element to the representation used for synthesis would guarantee an efficient mapping and somehow compensate for the lack of versatility of AICs compared to LUTs.

The original paper on AICs claimed that hybrid FPGAs made of a blend of AICs and LUTs could reduce delay up to approximately 30% and area by some 15%. These results were obtained on a set of relatively small benchmarks, all of them combinational. Perhaps more importantly, the modeling of the new logic cluster was somewhat rough and no real global routing results were presented, the overall routing contribution being estimated through statistical parameters extrapolated from LUT-only experiments. Some aspects of the architecture have been refined in successive papers without really addressing the fundamental modeling questions [12, 11]. We think that a more comprehensive analysis of AICs is worth it, before they can be considered as serious alternative to LUTs.

This paper attempts a more thorough analysis of AIC architectures close to the suggested original design [10]. To this end, in Section 2 we develop at the transistor-level precise models of both a LUT logic cluster (primarily to benchmark our design ability and to establish a fair reference) and an AIC logic cluster. We compare the resulting designs with the original expectations of the authors and notice a few divergences, leading to some exploration of the design space. Section 3 explains how we use the data collected in the previous section. We run a tangibly larger set of benchmarks, not limited to combinational ones and including some large designs. Since results are not necessarily what we would expect, we use them to drive some changes in the architecture, and analyze which aspects of AICs remain promising and which ones somewhat disappointing; all this is covered in Section 4. Finally, we draw our conclusion on the perspectives for AICs in Section 5.

2. LUT- AND AIC-BASED CLUSTERS

In this section we build the components that will make up our architectures: we start by presenting our design methodology and we then use it to produce the building blocks for the LUT-based reference logic cluster. We then pause in Section 2.3 to compare our results with common models used in the literature, and the results are a mixed blessing; we discuss why we think that the differences do not compromise our final results. In the following subsections, we attack the components of the AIC-based logic cluster and in Section 2.7 we discuss how our findings relate to the original AIC paper [10]. We conclude this part by exploring the design space, at transistor-level, of critical components.

2.1 Full-Custom Design Methodology

Although the original AIC paper contained some analysis of the critical path [10], no comprehensive design experiments at the transistor level have been performed. This is one of the main contributions of this paper.

All the components needed for our designs and described in the upcoming sections have been implemented in a commercial 40 nm CMOS technology. The circuits have been developed full-custom at the transistor level in the Cadence Design Platform Virtuoso. Speed characterization has been performed by simulating the circuits with Cadence simulation tool Spectre using the transistor models provided by the foundry. We report timing data in the typical corner (typical process, room temperature, nominal voltage). Unless specified differently in the following sections, transistor sizes are optimized for speed. Approximately, this means that we usually increase the size of critical transistors until we observe only marginal gains in speed. Although not strongly formalized, we apply this strategy consistently on all our functional blocks. We mainly use normal V_t transistors and low V_t transistors. The sizing of all transistors is performed manually.

Of course, the effort needed prevented us from layouting the circuits we developed. The use of front-end data has two impacts: on the measurements of timing and of area. Concerning timing, we used a number of other designs for another FPGA architecture we have designed (with complete layout and back-annotation): our experience suggests that, on average, paths degrade by a factor of 1.6× after layout, and thus we have corrected our front-end data by applying the same factor. Concerning area, for consistence

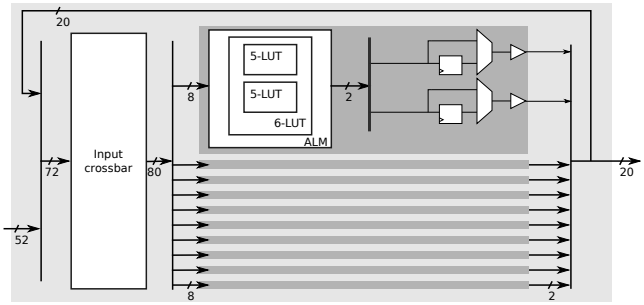

Figure 1: Reference LUT-based logic cluster. It is composed of ten fracturable 6-LUTs and all outputs can be registered; an input crossbar creates some local routing network.

with the numbers reported by VPR, we have applied the same classic methodology [2]. Specifically, we measure area using *minimum-width transistor areas* (or T_{mw}) as the unit, assume as usual that the die size is determined by the active area, and estimate the area (in minimum-width transistor areas) occupied in the actual layout by each transistor T as

$$\text{Area}(T) = 0.5 + \frac{\text{Drive Strength}(T)}{2 \times \text{Drive Strength}(T_{Minimum-Width})}. \quad (1)$$

As explained in Section 3.2, we use our measured values to build appropriate architecture files for VPR. It should be noted, though, that we restricted ourselves to the design of all of the logic clusters that we use and describe in detail in the next sections; on the other hand, we rely on existing architectural descriptions, provided with the *Verilog-to-Routing (VTR)* toolflow [13], for what concerns the characterization of the global routing. Indeed, these descriptions, widely used in the research community, exist and target commercial technology nodes similar to ours.

2.2 Reference LUT-Based Logic Cluster

For our reference LUT-based cluster we chose to loosely mimic the Altera Stratix-IV architecture. The logic cluster of this architecture, shown in Figure 1, was implemented drawing inspiration from the details provided in the Altera datasheets [1] and other descriptions in the literature. Each cluster, or *Logic Array Block (LAB)*, is composed of ten *Adaptive Logic Modules (ALMs)* having eight inputs each. The ALM is the basic logic block of the Stratix-IV architecture and can be configured as either one 6-input LUT, or *6-LUT*, or two 5-LUTs with some input sharing. The output(s) of the LUT(s) are used either as combinational outputs or sequential outputs, using two registers.

In addition to the LUTs and registers, the ALM architecture of the Stratix-IV includes two dedicated full adders and a carry chain used to implement arithmetic functions. These adders and carry chain, along with all related multiplexers, were not included in the implemented reference cluster to provide a more accurate area to delay comparison between the AIC-based and the LUT-based architectures, since carry chains are not used in our experiments.

Each LUT-based logic cluster has 52 inputs from the global interconnect and 20 outputs from the LUTs. The input crossbar receives the 52 external inputs and the 20 outputs from the LUTs, and has 80 outputs to feed the ten ALMs. It is implemented as a half-populated crossbar with

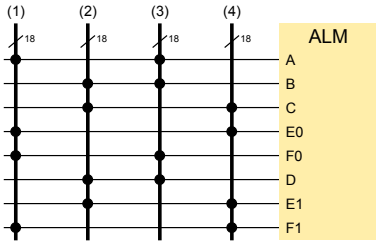

Figure 2: Topology of the LAB input crossbar. The half-populated crossbar has 72 inputs and 80 outputs, connecting the LAB input to the ALM inputs using a specific pattern.

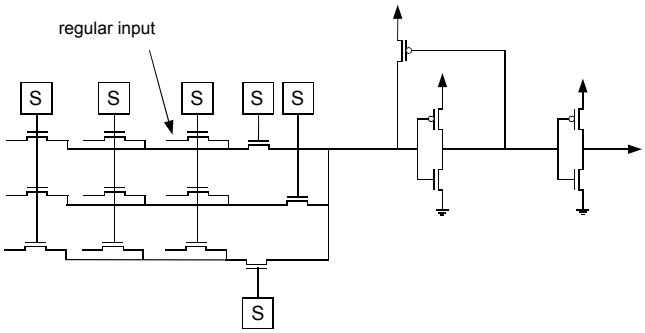

Figure 3: Two-level multiplexer [6]. All our crossbars are implemented with this type of two-level structure (without any fast inputs). Their topology differs only in the size of the $h \times v$ first-level array. The LUT-based logic cluster crossbar has a 6×6 structure.

the topology shown in Figure 2 [9]. It may be noted here that the crossbar in the original AICs paper was fully populated [10].

The eighty 36-input multiplexers of the crossbar are implemented as a two-level structure, similar to the ones used in Stratix-II [6], as indicated in Figure 3. Using the optimization criteria mentioned in Section 2.1, the transistors of the first and second layers result in transistors around 600 nm wide. The transistors in the multiplexers of the LUTs are usually somewhat larger than 600 nm except the last level which are significantly larger. These transistor sizes are not fundamentally different from those which could be obtained from recently published automated transistor sizing techniques [3].

2.3 How Good Is Our Reference?

The natural question at this point is whether our model of LUT-based cluster has any resemblance, quantitatively, to the commercial architecture that inspired it. We will report all numbers on the LUT-based cluster in Section 2.7, but anticipate some key figures in Table 1. We took as a reference numbers extracted from the standard VTR architecture files [13] which are mainly based on simulation data and approximations. Firstly, we notice that some critical values match remarkably well, such as the 6-LUT delay or the total cluster area. Besides these, there are a number of parameters which are different but within what appear to us reasonable margins (for instance, if one considers the sum of setup time and clock to output of the flip-flops the difference is around 50% which is significant but, we believe, not

		Our	[13]
6-LUT		298 ps	261 ps
FF	T_setup	192 ps	66 ps
	T_clock_to_Q	96 ps	124 ps
Mux	comb. inputs	224 ps	25 ps
	seq. inputs	163 ps	45 ps
Input crossbar		192 ps	95 ps
Cluster area		53,600 T_{mw}	53,894 T_{mw}

Table 1: Reference LUT-based cluster vs. Stratix-IV. We compare here some key parameters of our reference design with some available model of Stratix-IV [13].

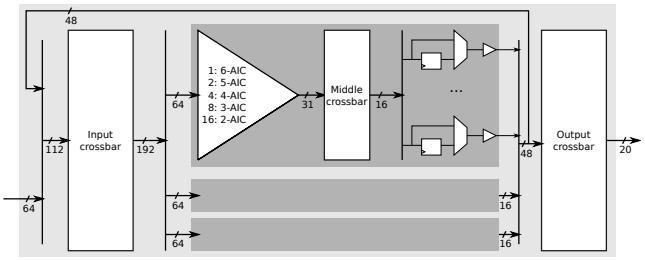

Figure 4: AIC-based cluster with three 6-AICs. The cluster architecture is unmodified from the original AIC paper [10].

truly worrying). Finally, there are some parameters which are unjustifiably different: the output multiplexer delays are very much slower in our design. We believe that our model contains a buffer to drive the highly loaded cluster outputs whereas the competing model does not: 25 ps are certainly sufficient only for a simple multiplexer in a 40 nm node. We believe that such differences do not jeopardize the quality of the results of Section 4, though. Firstly, the key computational element is reasonably well matched in our model, and thus does not suggest any suspicious low-quality implementation of LUTs that would bias the results in favor of AICs. Secondly, all worst discrepancies correspond to components which either exist practically identical in the AIC cluster or which are redesigned for it following the very same methodology and circuit structure. In other words, if we are making a mistake, the error is independent of the cluster type and is systematic across the simulated architectures: it should thus bring only a minimal bias to our conclusions.

2.4 AIC-Based Cluster

The AIC-based logic cluster proposed in Parandeh-Afshar et al. [10] has three 6-level AICs (or *6-AIC*) with 64 inputs and 31 outputs. Figure 4 shows a block diagram of the AIC-based logic cluster. A small crossbar, referred to as the *middle crossbar*, is placed at the end of each 6-AIC to select only 16 out of the 31 outputs. These 16 outputs can either be used as combinational outputs or sequential outputs using dedicated registers, exactly as in the LUT-based cluster. In order to maintain the same number of outputs per cluster as in the reefrence architecture, an additional crossbar, referred to as *output crossbar*, selects 20 cluster outputs out of the total 3×16 signals, registered or non-registered, coming from the middle crossbar or the registers.

The cluster has 64 inputs—the original paper assumed that the reference Stratix-III logic cluster also has 64

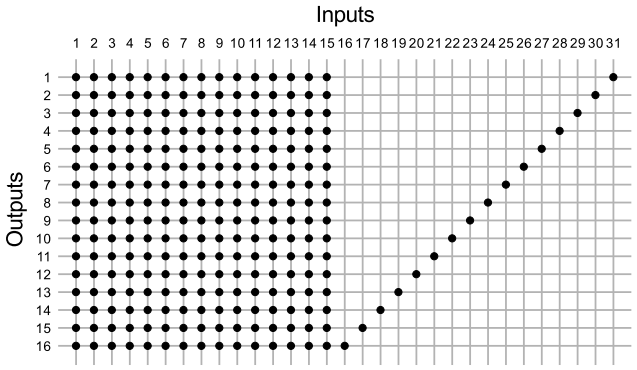

Figure 5: AIC-based logic cluster middle crossbar. The connectivity of the middle crossbar is that of a full-capacity minimal crossbar, but the implementation is as in Figure 3.

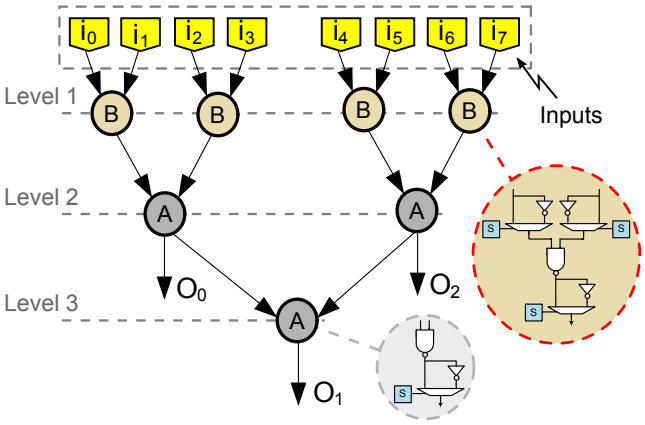

Figure 6: AIC logic element. The enhanced AIC architecture with programmable inversion at the inputs of the first level nodes, reproduced from the paper which introduced the improvement [12].

inputs—coming from global routing and 48 feedback connections coming from the AICs selected outputs. A third crossbar, the *input crossbar*, connects the 64 cluster inputs and the 48 feedbacks—112 in total—to the 192 AICs inputs. (A small detail is that, due to the symmetry of the AND gates at the input of the 6-AIC, only 111 inputs are implemented, but this has hardly any impact on area and delay.)

2.5 Topology of the Crossbars

The three crossbars of the AIC-based logic cluster are also all implemented as two-level multiplexers (see Figure 3). The basic implementation of the input crossbar is fully-populated with a 14×8 first level array. Our transistor-size optimization results in very similar transistors as the input crossbar of the LUT-based cluster: transistors are also about 600 nm wide. We considered two additional input crossbars: a 75%-populated sparse crossbar and a half-populated crossbar built with the same transistors sizes. The 75% and 50% crossbars are referred to using the *75p* and *50p* notations, respectively, in the experimental results. Note that Parandeh-Afshar et al. used a 75%-populated crossbar [10].

The middle and output crossbars have the connectivity of full-capacity minimal crossbars. Full-capacity minimal crossbars, or simply *minimal crossbars*, have fewer switches than full crossbars but maintain the full-capacity property. This property guarantees that any set of inputs that does not exceed the number of outputs can be connected to all outputs; however, the order of the outputs cannot be fully controlled [5]. Figure 5 shows the connectivity of the middle crossbar, with 31 inputs and 16 outputs. It is implemented with multiplexers with $in - out + 1 = 31 - 16 + 1 = 16$ inputs, in a 4×4 array of large transistors: 1 μm in width. The output crossbar has 48 inputs and 20 outputs. It is implemented with 29-input two-level multiplexers whose input array is 6×5; the transistors are also 600 nm wide as in the input crossbar.

2.6 AIC Logic Element

The And-Inverter Cone was introduced as a full binary tree of AND gates with programmable output inversion [10]. Then an enhanced AIC architecture was proposed as a shadow logic, placed behind LUTs [12]. Irrespective of its architectural placement, this enhanced design adds to the initial AIC architecture programmable inversions at the inputs of the first level nodes, as shown in Figure 6. Naturally, allowing the inputs to be inverted internally uses more compact AICs with less depth and helps reducing the number of clusters used per circuit. We designed the logic element using the enhanced AIC architecture. The optional inversion is implemented literally with an inverter and a multiplexer. Transistor sizes vary between 240 nm and 1 μm in width.

2.7 The Good, the Bad and the Ugly

Table 2 summarizes the two clusters designed and the break out in the different elements. There is some good, as expected: the AICs are not very significantly slower than the LUTs and yet are much smaller. They contribute for a quite small fraction to the cluster size (less than 10%), suggesting that even if they were not sufficiently well utilized, more could be added, allowing more logic to be packed within the same cluster, or that their structure could be made more complex at a very limited cost.

There is some bad, or at least surprise, in that the delay of the crossbars is only moderately variable with their size, which is a significant penalty in the case of the middle crossbar. Essentially, this reflects the fact that the topology of the crossbars is the same and, to a good extent, optimizing transistor sizes only helps compensating the different parasitic capacitances.

But, finally, there is something really ugly: the input crossbar, which was anticipated by the original authors to be the dominant component of the cluster area-wise, is in fact humongous; not only it takes 80% of the cluster, but it is itself almost twice the size of the complete LUT cluster itself. Of course, this is also due to the fact that the crossbar is full, but this is not the whole story: even with a 75% populated crossbar (80% of the area, approximately) or half-populated (60% of the area), we are far from having two clusters of similar size as we originally expected, and the AIC-based cluster remains at best 60% larger than the LUT-based cluster.

2.8 Exploring Transistor Sizing

From Table 2, the area of the AIC-based cluster is about 2.3× that of the LUT-based cluster. This area difference

		AICs		LUTs	
		Area (T_{mw})	Delay (ps)	Area (T_{mw})	Delay (ps)
Crossbars	Input crossbar	102,393.6 (80.85%)	203.2	12,330 (22.4%)	192
	Middle crossbar	7,310.4 (5.77%)	118.4	-	-
	Output crossbar	3,849 (3.04%)	166.4	-	-
Logic	5-LUT	-	-	24,550	302.08
	6-LUT	-	-	(44.6%)	298.4
	Adder	-	-	1,440 (2.6%)	-
	6-AIC	9,870.21 (7.79%) [3 AICs]	449.6	-	-
	5-AIC		388.8	-	-
	4-AIC		318.4	-	-
	3-AIC		252.8	-	-
	2-AIC		184	-	-
FF Mux Buff	Tclk2Q	3,223.2 (2.55%)	171.2 17.6 80	16,720 (30.4%)	96 193 (avg)
	Total	126,646.41		55,040	

Table 2: Delay and area comparison between the AIC and LUT Clusters.

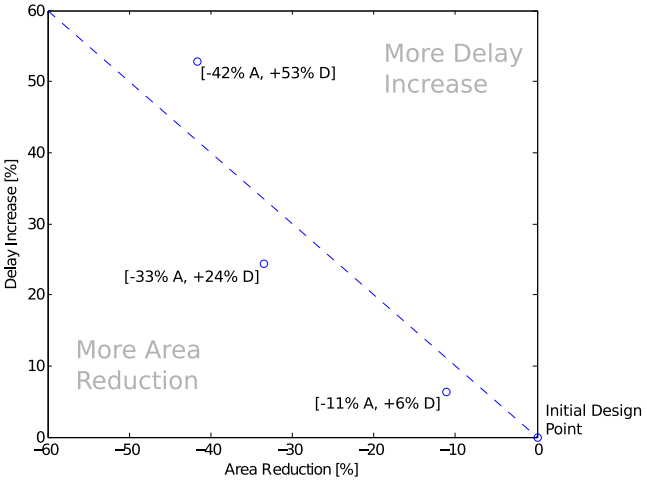

Figure 7: Transistor sizing delay vs. area trade-off. Different transistor sizes in the AIC cluster's input crossbar result in sensibly different and potentially interesting Pareto solutions.

is mainly due to the AIC input crossbar—accounting for more than 80% of the cluster area; thus, we decided to go back to the transistor-level circuits and see if other design points would make sense by trading off delay for area. Figure 7 shows three other optimization points with their relative area and delay values. Clearly, there exists other design points of potential interest: the second smallest (68,140 T_{mw}, 252.8 ps) saves a significant area (33% area reduction) for an additional delay (24% delay increase, roughly 50 ps) which is fairly small compared to the total combinatorial cluster delay (of the order of the nanosecond). Accordingly, we decided to use also this design point in our experiments and will be referring to it throughout this paper as *68k*.

3. EXPERIMENTAL SETUP

Now that we have all circuit elements we need, we can proceed to assemble them and compare the two architectures on benchmarks. In this section we explain how we run the experiments of Section 4. In particular, we explain first how we performed technology mapping and then we explain the various choices we have made to create architecture description files for VPR, in order to perform packing, placement, and routing.

3.1 AIC Technology Mapper

We use the AIC technology mapper described in Parandeh-Afshar et al. [10] and include a final area recovery pass [12]. The mapping includes basically three phases. Initially, the input AIG circuit description is forward traversed—from inputs to outputs—generating for each AIG node all the possible cuts, both for an AIC and a LUT mapping. Then, the circuit is backward traversed selecting for the visible nodes the cuts that are optimal for delay, and amongst those, the one leading to a smaller area. Note that after having chosen a cut for a node, this cut will cover all the AIG nodes between its inputs and output. The inputs become visible nodes and the process is repeated until the whole circuit is covered. However, this covering has prioritized delay over area in all the paths of the circuit, including the non critical ones. Accordingly, we implement a final area recovery phase to choose area optimal cuts for the paths that are not critical. The area optimal cuts still need to satisfy the constraint that the relaxed path cannot include more logic levels than the critical path obtained in the previous phase.

3.2 VPR Architecture Files

After technology mapping, the *AAPack* [7] tool of VPR is used to pack the mapped logic blocks into clusters of the predefined FPGA architectures. Thus, each FPGA architecture was modeled using the XML format. The basic logic blocks are defined along with their respective configuration modes

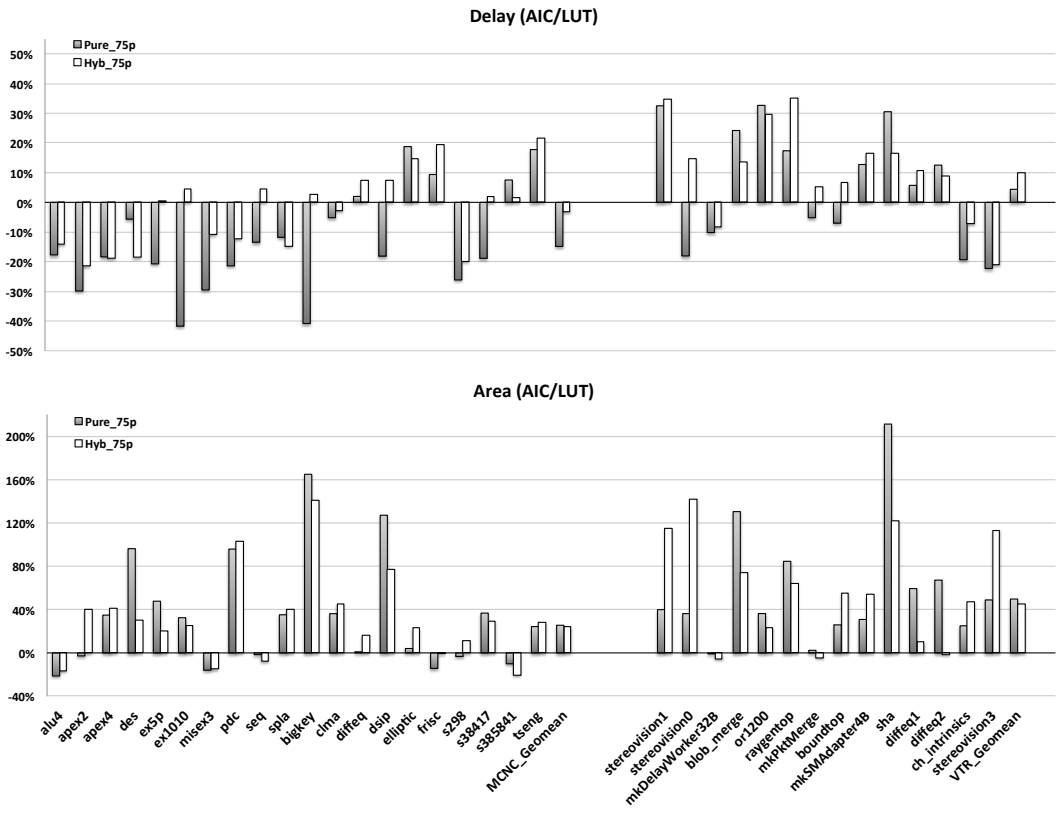

Figure 8: Speedup and area reduction of reference architectures. Reproducing the results of both reference architectures [10] shows that the pure AIC architecture has a more direct effect on the speedup of the MCNC benchmarks than the hybrid architecture, with respect to the LUT architecture. This speed up comes at the expense of around 25% area increase. However, the performance of both these architectures deteriorates on the VTR benchmarks.

and delay metrics. All logic delays and local routing delays were taken from the transistor level implementation, as reported in Table 2. Starting with the LUT-based architecture, used as reference, the Stratix-IV cluster is designed in which LUTs are configured as two 5-LUTs or one 6-LUT and the crossbar is half populated as described in Section 2.2.

The basic AIC architecture is modeled as described in Section 2.4, where three 6-AICs are used as basic logic blocks. The fracturable nature of the AIC is explicitly defined in the XML file by allowing multiple configurations of the same cone. The hybrid architecture file is created by combining the two cluster definitions of the LUT and AIC architectures, respectively into one XML file. The LUT clusters are placed at every third column of the FPGA grid. The different areas of the LUT and AIC clusters are accounted for by specifying the relative size of the AIC cluster with respect to the LUT, and the length of the global routing is adapted accordingly.

For the first comparison between AICs and LUTs [10], the MCNC benchmark set [14] was used to run experiments. More precisely, and due to some tool flow limitations, the benchmarks of the original paper were limited to the biggest 10 combinatorial circuits of the MCNC set. However, in this work, we were able to overcome these limitations and test the performance of the AICs on sequential circuits. Furthermore, we also run our experiments on the more recent set of benchmarks provided with the VTR tool flow. These VTR benchmarks are bigger and use multipliers and memory blocks. For this purpose, we updated our architecture files to include the multipliers and RAM blocks that we took from the sample architectures of VTR. Some of the biggest benchmarks take too long to pack and were omitted from the experiments.

To compute the area of each packed and routed circuit, we are relying on the VPR built-in estimation of the routing area, while the logic area is computed by counting the number of blocks, for each block type, and multiplying it by its respective area.

All architectural files can be downloaded here:

`http://lap.epfl.ch/downloads/fpga14`

4. EXPLORING ARCHITECTURES

We finally report our results through VPR over all benchmarks. We start by running experiments for the basic AIC architecture as proposed by the original authors and we repeat some of the design explorations they originally suggested. Then, analyzing our use of the resources, we explore a few neighboring design points which do not change radically the structure of the AIC cluster but try to minimize the problems of the straightforward architecture. We conclude this section with some analysis of what we think works well with AICs and what needs further research or dramatic improvement.

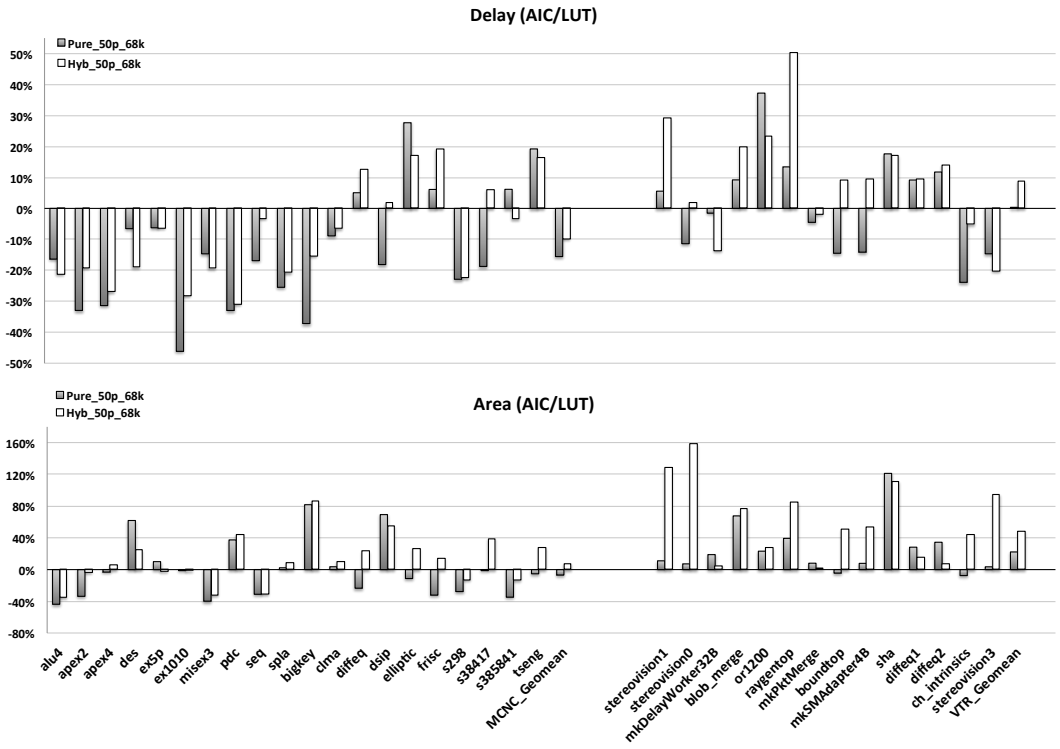

Figure 9: Effects of the input crossbar's architectural modifications on delay and area. Decreasing the population of the input crossbar to 50% and choosing an optimization point that reduces the area of the crossbar, improves the performance of both pure AIC and hybrid architecture. The best speedup and area results are achieved using the pure AIC architecture when considering only the MCNC benchmarks.

4.1 Plain-Vanilla AIC Architecture

In the first experiment, the benchmarks are mapped, packed, placed and routed onto three different architectures: (1) the reference architecture, which is made of LUT-based logic clusters such as the one depicted in Figure 1; (2) the pure AIC architecture as originally described, which is made of AIC-based logic clusters such as the one depicted in Figure 4; and (3) the hybrid LUT-AIC architecture, which combines LUT and AIC logic clusters in a ratio of 1 to 3, respectively. Throughout the experimental setup and results sections, we will use *pure* to identify AIC-only architectures and *hybrid* (or *hyb* for short) to refer to mixed LUT-AIC architectures.

Figure 8 shows in the top graph the relative delay of the pure and hybrid architectures with respect to the reference LUT architecture. The results are separated per benchmark suite. The MCNC benchmarks, the combinatorial subset which was used in the original AIC paper as well as the sequential subset, show consistent speedups when executed on AICs, especially in the pure architecture. The results show an average speedup of 15% for the pure architecture and a 3% for the hybrid one. However, the subset of VTR benchmarks, which are bigger and make use of memories and DSP blocks, performs generally worse in the AIC architectures than in the reference LUT architecture. In particular, the pure and hybrid architectures achieve a 4% and 10% slow down, respectively.

Moreover, Figure 8 shows in the bottom graph the relative areas (logic and routing areas) corresponding to the same architectures and benchmarks with respect to the LUT reference architecture. The pure AIC architecture requires on average 25% and 50% more area for the MCNC and VPR benchmarks, respectively. The hybrid architecture requires slightly less area but the overhead is still quite significant. Some benchmarks are particularly big in AICs increasing the area by more than a factor of 3.

In summary, the original AIC architectures only show interesting speedups for a limited set of benchmarks. The main reasons are the rather big crossbars, which increase logic cluster area beyond the reference logic cluster, and the inclusion of bigger size and synchronous benchmarks. In the following, we explore a few AIC architectural variations to try and unleash the AICs' theoretical potential.

4.2 Playing with the Input Crossbar Sparsity

Realizing from Table 2 that the input crossbar contributes to about 80% of the total cluster area, we considered changing the input crossbar in two ways: (1) by reducing its population and making it more sparse, and (2) by changing its optimization so that we gain in area even if it comes at the expense of delay.

Depopulating the input crossbar by reducing the number of used switches, we first explored a 75% sparse crossbar and then a 50% sparse crossbar. Having less switches constrains the packer, making it harder to cluster logic blocks. However, the experiments revealed that such a constraint is still acceptable and promises good results. Moving from a full crossbar to a 50% sparse crossbar reduces the overall cluster

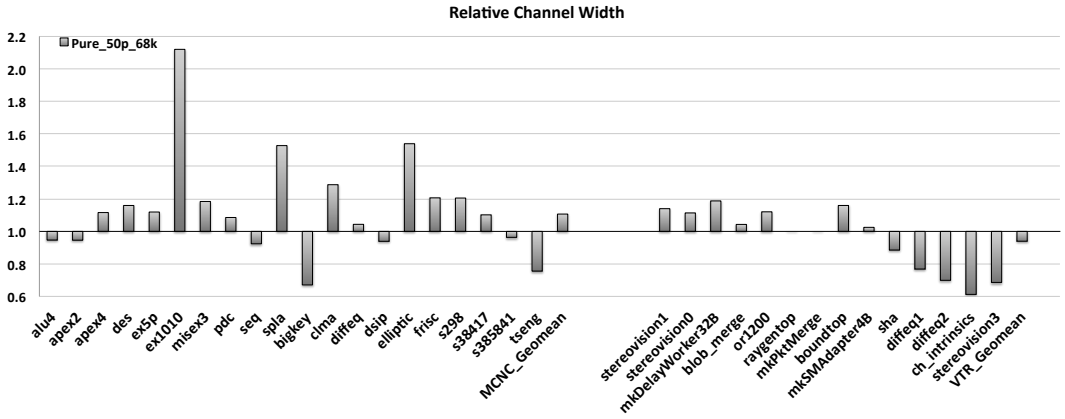

Figure 10: Relative channel width of the pure AIC architecture with an optimized half-populated crossbar, with respect to the reference LUT architecture.

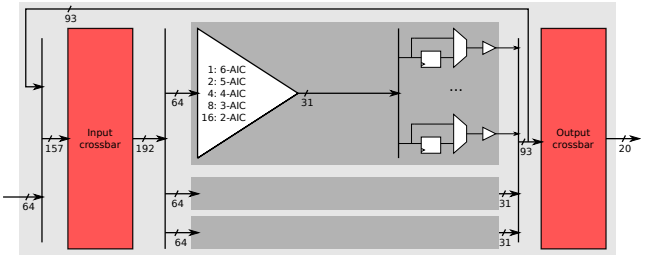

Figure 11: AIC-based cluster without middle crossbar.

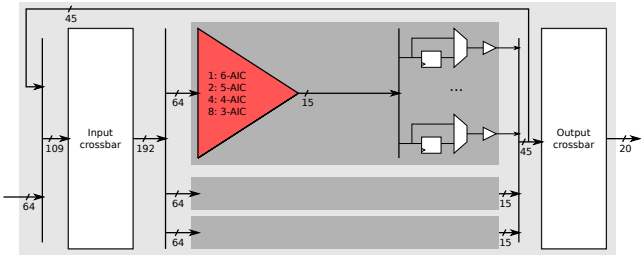

Figure 12: AIC-based cluster without middle crossbar and without outputs for 2-AICs.

area by about 10%. In order to further reduce the cluster area, we explore different optimizations of the input crossbar as mentioned in Section 2.8. From Figure 7, we choose the 68k optimization point where the crossbar size decreases by 33% at the expense of a delay increase of 24% (referred to as 68k).

The results of Figure 9 confirm our intuition since the experiments performed using a 50% sparse input crossbar with the chosen design point reveal to be the most promising: the delay decreases by 16% for the pure AIC architecture and 10% for the hybrid architecture with respect to the reference LUT architecture. For the pure AIC architecture, not only the delay is reduced but also the overall area is reduced by around 7% for the MCNC benchmarks. This particular architecture appears to be the most rewarding in terms of delay and area, especially for the MCNC set.

To gain a sense of the stress the AIC architectures might add on global routing, Figure 10 shows the relative channel width of the pure AIC architecture with 50% sparse crossbar (and 68k optimization point) over the reference LUT architecture. The demand for routing channels depends on the benchmark used, and although on average the channel widths used for AICs and LUTs are comparable, some oddities in the results can be highlighted. For instance, the MCNC benchmarks, which are known to be small, tend to require more channels on AICs than the VTR benchmarks which are rather large and complex. We believe that the nature of the benchmarks directly impact the efficiency with which it can map and pack on AIC clusters.

4.3 Removing the Middle Crossbar

Table 2 suggests that the middle crossbar adds to the delay of the logic path around 166 ps in order to reduce the number of AIC outputs. In this experiment, we remove the middle crossbar trying to gain more in delay at the expense of almost doubling the number of flip-flops and output multiplexers, as shown in Figure 11. However, the experimental results of Figure 13 show that removing the middle crossbar (with a 75% sparse input crossbar designed using the 68k optimization point that trades delay for area) does almost as well as if the middle crossbar were kept and the input crossbar sparsity were reduced to 50%, on average (the results of Section 4.2). Nevertheless, removing the middle crossbar has a bigger impact on the area. It is true that the number of flip-flops and multiplexers doubled; but that is not the main source of area overhead. The feedback connections to the input crossbar as well as the direct connections to the output crossbar doubles, which has a direct impact on the area of the crossbars, the area of the cluster and by that the overall circuit area.

4.4 Limiting the AIC Element Outputs

Limiting the AIC outputs is one possible remedy to the extra area overhead observed after removing the middle crossbar. In this experiment we remove the outputs that exist at the second level of the AICs, which results in a reduction from 31 to 15 in the number of AIC outputs. Figure 12 depicts the corresponding logic cluster architecture. Note that the input and output crossbars are slightly smaller than the

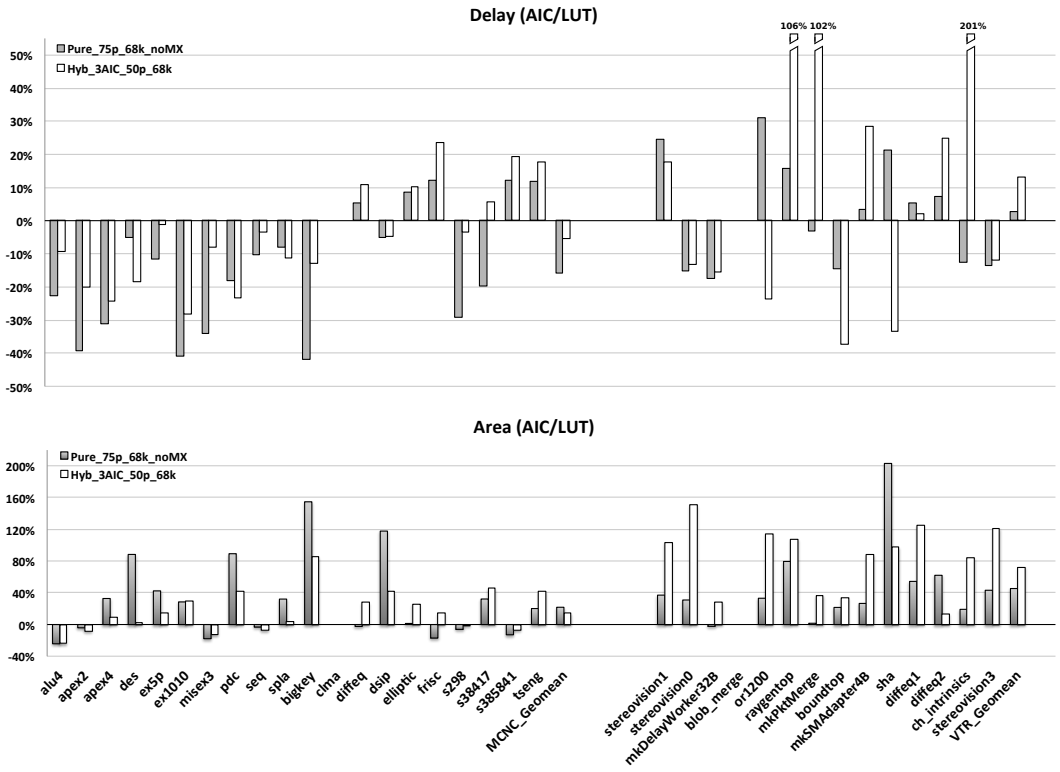

Figure 13: Minimal effects resulting from removing the middle crossbar. To compensate for the middle crossbar either bigger input/output crossbars are used (*Pure_75p_68k_noMX*, Figure 11) or the AIC's second-level outputs are not used (*Hyb_3AIC_50p_68k*, Figure 12).

ones used in the original AIC architecture of Figure 4. Such an architectural change required some modifications in the mapper in order to prevent the use of 2-AICs and restrict the AIC outputs to levels 3 to 6.

Figure 13 shows the relative delay and area of this new architecture (*Hyb_3AIC_50p_68k* in the figure) with respect to the reference LUT architecture. In general, removing the 2-AIC outputs seems detrimental for both area and speed. Often, the logic that would ideally map onto a 2-AIC is now forced to go onto a LUT. This results in some substantial impact on the global routing due to the forced change of clusters. LUT and AIC clusters are located in different columns, thus switching between LUTs and AICs over a small subfunction compromises the locality of the design and introduces major routing overhead.

4.5 What Is Promising and What Is Not

We have seen that despite the theoretical potential of AICs, realizing a logic cluster architecture that enables competitive implementations is far from trivial. We believe that the original work on AICs underestimated the effects of the crossbars, especially in area. We have shown that the circuit area/delay numbers are heavily influenced by the different AIC cluster architectures but also by the area/delay tradeoffs played at the component level, as illustrated in Figure 7.

In order to provide further insights on why the AIC performance varies so much depending on the benchmark, Figure 14 details the critical path of four selected benchmarks when running on five different architectures. The *bigkey* benchmark shows delay advantages in all its AIC executions being an example of a small benchmark with a very short critical path that can fit in about 3 logic blocks. At the opposite end of the spectrum, the *sha* benchmark includes a long critical delay path and shows an increase in delay in all its AIC executions. The reference LUT execution manages to reduce significantly the global routing delay by using very short hops. We suspect that the area increase of the AIC implementations reduces the locality and forces longer global wires, also in the critical path. The *tseng* benchmark is another example that systematically performs poorly in AIC architectures, suffering from the same problem of *sha* despite a much shorter critical path. Finally, the *boundtop* benchmark shows speedups for the pure AIC architecture while it slows down in the hybrid one. For this benchmark the penalty of moving from LUT to AIC clusters and vice versa—each type of cluster is placed in different columns, not always contiguous—overcomes the benefits in logic delay.

Further analysis reveals that the AIC packing is extremely focused on achieving the maximum density, thus apparently resulting in poor locality and seldom using any local feedback within the same cluster. Thus, the AICs are mainly using global routing, leading to a longer critical path.

5. CONCLUSIONS

We have revisited the idea of replacing LUTs in FPGAs with AICs, tried to build at transistor level the complete logic necessary and have adapted VPR to place and route

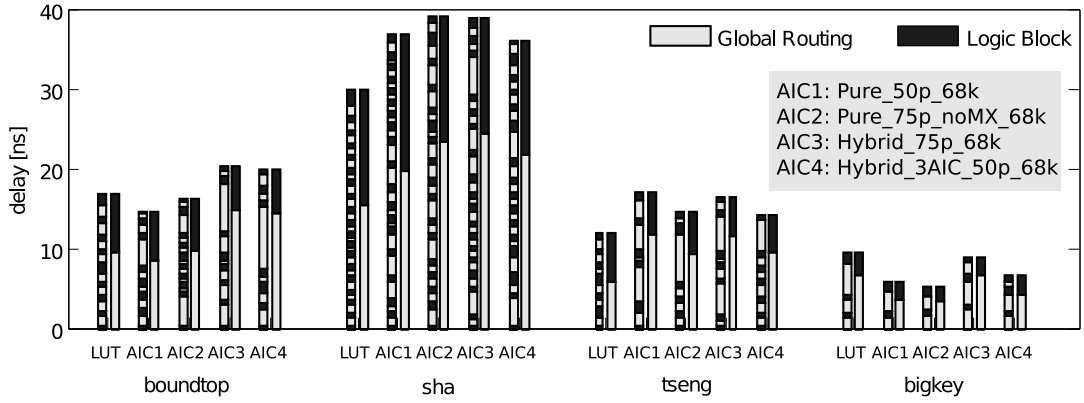

Figure 14: Detailed critical path. Distribution between logical delay and global routing delay of four selected benchmarks when running on five different architectures. Each pair of bars shows the same path twice: on the left with the succession of logic and routing delays as they occur on the path and on the right accumulating the two components.

benchmarks. The results are a mixed blessing: some aspects of the AIC-based architecture are significantly worse than expected, most notably the size of the input crossbar and of the cluster. Although this cannot but impact the overall results, it is surprising that even with very fat logic clusters we could find some design points which improve delay reasonably while not deteriorating the overall silicon real-estate cost. Although average results are never truly excellent, results on individual benchmarks display a remarkable heterogeneity, with several benchmarks achieving delay improvements of around 30-40% at almost no area cost. We conclude that AICs still promise some potentially significant improvements, although achieving them does not appear as immediate as we hoped at first. Looking at the resulting implementations, we foresee opportunities to better utilize AICs by tailoring the logic synthesis and technology mapper to AICs, and using a packer that can trade off wire length for a lower packing density [9].

6. REFERENCES

[1] Altera Corporation. *Stratix IV Device Handbook, vols. 1 and 2*. `http://www.altera.com/literature/`.

[2] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for deep-submicron FPGAs*. Kluwer Academic, Boston, Mass., 1999.

[3] C. Chiasson and V. Betz. COFFE: Fully-Automated Transistor Sizing for FPGAs. In *Proceedings of the IEEE International Conference on Field-Programmable Technology*, 2013.

[4] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, New York, 1994.

[5] G. Lemieux, P. Leventis, and D. Lewis. Generating highly-routable sparse crossbars for PLDs. In *Proceedings of the 8th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 155–164, 2000.

[6] D. Lewis et al. The Stratix II logic and routing architecture. In *Proceedings of the 13th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 14–20, Monterey, Calif., Feb. 2005.

[7] 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.

[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] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz. Titan: Enabling large and complex benchmarks in academic CAD. In *Proceedings of the 23rd International Conference on Field-Programmable Logic and Applications*, Porto, Portugal, Sept. 2013.

[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 AICs: Reaping the benefits of And-Inverter Cones with minimal architectural impact. In *Proceedings of the 21st ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, page 279, Monterey, Calif., Feb. 2013. Poster.

[12] 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*, Porto, Portugal, Sept. 2013.

[13] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson. The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing. In *Proceedings of the 20th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2012.

[14] S. Yang. Logic synthesis and optimization benchmarks user guide, version 3.0. Technical report, Microelectronics Center of North Carolina, Research Triangle Park, N.C., Jan. 1991.