Evaluating FPGA Clusters under Wide Ranges of Design Parameters

Grace Zgheib and Paolo Ienne
Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences, 1015 Lausanne, Switzerland
{grace.zgheib, paolo.ienne}@epfl.ch

***Abstract*—The latest published studies with extensive explorations of look-up table and cluster sizes are now more than a decade old. However, CMOS technology as well as CAD and transistor modeling tools have improved so much since that it is reasonable to wonder whether the conclusions of such studies still hold. One of the major difficulties of conducting these studies, especially in academia, is producing credible delay and area models. In this paper, we take advantage of a recently developed architecture modeling tool to re-evaluate the effect of the various cluster parameters on the FPGA. We considerably extend the exploration space beyond that of the classic studies to include sparse crossbars and fracturable LUTs, and show some results that go against the current tenets of FPGA architecture.**

I. Introduction

"In all affairs, it is a healthy thing now and then to hang a question mark on the things you have long taken for granted." A quote by Bertrand Russell that, under the scope of "all affairs", can be be applied to any research field, including *Field Programmable Gate Arrays (FPGAs)*.

For over a decade, the general consensus on the optimal cluster and *Look-Up Table (LUT)* size for FPGAs has been influenced by the 2000 study done by Ahmed and Rose [1] and its extension in 2004 [2]. However, FPGA research has come a long way since then, especially at the technology and CAD tools levels. From one side, the technology node scaled down by more than a factor of 10, resulting in lower threshold voltage and producing smaller transistors that are faster and more power efficient. From the other side, academic research made numerous contributions to the different stages of the CAD flow, improving their optimization algorithms and the quality of the results obtained using the various tools. For instance, multiple new algorithms were introduced to the logic synthesis and technology mapping phases with the introduction, in 2005, of the synthesis and verification tool ABC [3]. Since then, ABC and its numerous internal algorithms evolved drastically to become one of the main providers of optimization algorithms for synthesis and mapping, in large open-source CAD tools like the VTR project [15]. The VTR project itself also witnessed a major evolution with its AA-pack algorithm [12] in 2011 and multiple releases up to the last version of VTR 7.0 [13] in 2014. As the FPGA research evolved, new benchmarks were also introduced to emulate real applications on which FPGAs might actually be used. The circuits became bigger and included memory and DSP blocks such as in the VTR [15] and Titan [14] benchmark suites, introduced in 2012 and 2013, respectively.

Even at the transistor level, FPGA architecture modeling tools emerged to automate the process and facilitate architectural explorations, the latest of which were COFFE [4] and FPRESSO [19]. COFFE runs SPICE simulations on clusters with pre-set, yet parameterizable, structure, and models the FPGA architecture within hours. In contrast, FPRESSO relies on libraries of pre-characterized cluster elements that it uses to model architectures, without any hard structural constraints, within minutes.

With this major evolution and dramatic improvement in the different aspects of FPGA research, and with the introduction of fast and automated FPGA modeling tools, it is time to hang some question marks on our "affairs" and re-evaluate our conventions by exploring the FPGA architectural space while exploiting the improvements achieved in this area. To this end, we start by re-evaluating the architectural conclusions of Ahmed and Rose's study [2], but using an experimental methodology that includes the most recent versions of the CAD tools and leverages the modeling capabilities of FPRESSO. We first limit our exploration to the almost-exact architectures explored in 2004 and use it to verify the validity of the original findings. And then, we gradually extend our exploration to a much larger search space, and focus on evaluating the effect of the crossbar density on the performance. We also show how fracturable LUTs can be modeled in automatic modeling tools like FPRESSO while assessing their architectural benefits.

The rest of the paper starts by first defining the general structure of the cluster and how it is modeled, and by introducing a new approach to automatically model fracturable LUTs within FPRESSO, in Section II. Then, Section III details our experimental methodology while elaborating on the benchmark selection process. Starting with a re-evaluation of the existing studies, we first limit the architectural exploration to the scope of these studies, in Section IV, before gradually increasing the search space. We then evaluate the effect of the crossbar density on the architectural performance in Section V. Widening further the exploration space, we compare the fracturable LUT architectures to the non-fracturable ones and analyze the effect of the CAD tools on the results, in Section VI, before concluding in Section VII.

II. Architecture Modeling

One of the main challenges in architectural explorations is to have correct delay and area modeling of the different architectures. Any minor modification to the cluster requires redesigning it at the transistor level, sizing the different transistors and running SPICE simulations to measure the

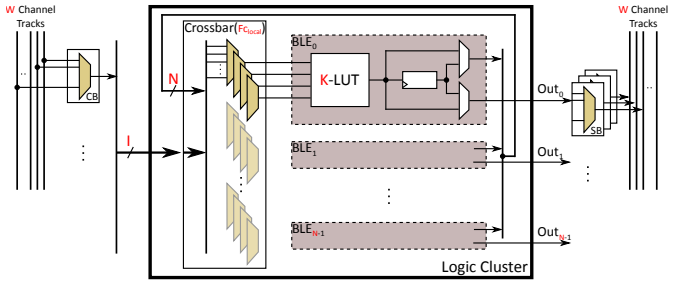


Fig. 1: The general structure of the cluster architecture used in our experiments and its design parameters (K, N, etc.).

pin-to-pin delays of every element of the cluster. In prior explorations [1], [2], this modeling had to be done manually, which imposed severe constraints on the feasible search space. Luckily, several tools have since been introduced to automate the process. In this section, we first define the cluster and its parameters and then we show how the architecture is modeled using FPRESSO. We focus on the challenges of modeling fracturable LUTs and explain how we extend FPRESSO to automatically support and model this feature.

A. Cluster Architecture and Parameters

State-of-the-art FPGAs have an island-style architecture that consists of a grid of clusters connected though vertical and horizontal routing channels [7]. Although the look-up table is used as the basic logic element in the clusters, the structuring of these LUTs with the additional logic blocks, such as multiplexers, crossbars, and registers, differs from one architecture to another.

In this paper, we base our exploration on an FPGA architecture similar to the Altera Stratix FPGAs while modifying its main characteristics. Figure 1 shows the general structure of the cluster along with the different parameters that can define a particular architecture. Each cluster consists of N *Basic Logic Elements (BLEs)* and has I inputs and N outputs. Each BLE has a K-input LUT, a register and two multiplexers to select between the registered and unregistered LUT output, before sending it either to the cluster output or as a local feedback. The I inputs, along with the N feedback signals, feed the input crossbar which then distributes them to the BLEs (and hence the LUTs). As such, the crossbar has $(I+N)$ inputs, $(N \times K)$ outputs and a density $F_{c_{local}}$ which indicates the fraction of the inputs connected to each output.

The inputs and outputs of the cluster are connected to the global routing through Connection Blocks (CBs) and Switch Blocks (SBs). The fraction of routing channels connected to each of the cluster's input/output is defined by the parameters $F_{c_{in}}$ and $F_{c_{out}}$, respectively.

B. General Modeling

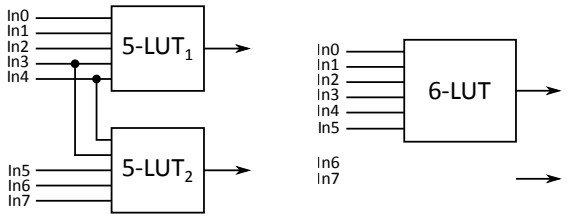
We use FPRESSO [19] to model the different architectures. It is an automatic modeling tool for FPGAs that takes as input a description of the cluster architecture, in a simplified XML format and returns the modeled architecture, fully annotated with the cluster area and the delay of every element.

Besides being the most recent FPGA modeling tool, FPRESSO is highly convenient for this kind of architectural explorations. First, it is a fully-automated tool that does not require any transistor-level expertise or effort from the user. The user just specifies the architectural parameters (e.g., K, N, I, $F_{c_{local}}$, etc.) then FPRESSO automatically models the cluster and returns the annotated architecture file as output. Furthermore, exploring a wide search space of hundreds if not thousands of architectures is a very time-consuming process, especially at the modeling phase. As a matter of fact, the previous explorations [1], [2] were limited to about 60 architectures due this constraint. However, FPRESSO models an architecture within minutes, providing a major runtime advantage without which modeling becomes the bottleneck of the exploration. The output file generated by FPRESSO follows the exact XML format requirements of the packer of the VPR tool, which makes it highly convenient and allows it to be easily integrated in the CAD flow.

We use the latest version of FPRESSO (version 2.0)[1] which improves the initial release by modeling global routing and inter-component wireload [18]. To model global routing, Ahmed and Rose [2] assume that the routing buffers scale proportionally with the length of the tile, while basing it on a certain assumed size for the small architecture of $K = N = 4$. To maintain consistency and properly compare with that study, we scale the routing buffers the same way; however, instead of assuming the size of the base architecture, we model it along with all the routing multiplexers and use the sizes reported by FPRESSO.

Since FPRESSO does not support fracturable LUTs, which is an important feature in the state-of-the-art FPGAs, we extend it so that it understands the notion of fracturability and model the fracturable LUTs, as explained in Section II-C.

[1] Available online at fpresso.epfl.ch



(a) Mode 1: Two 5-LUTs. (b) Mode 2: One 6-LUT.

Fig. 2: A fracturable LUT is seen by VPR as two mutually exclusive modes of operation. This example shows the two modes of a fracturable 6-LUT, given a BLE with 8 inputs: (a) mode 1 consists of two 5-LUTs with two shared inputs, while (b) mode 2 consists of a single 6-LUT.

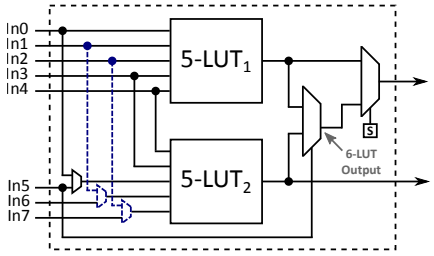


Fig. 3: The hardware implementation of the fracturable 6-LUT of figure 2. Multiplexers are added to switch between the two modes and to correctly satisfy the input/output constraints.

C. Modeling Fracturable LUTs

Fracturable LUTs are a fundamental feature in modern FPGAs, it is thus essential to evaluate their effect on the FPGA performance. However, modeling fracturable LUTs is not as straightforward as it may seem, for there is a major difference between how CAD tools perceive the notion of fracturability and how it is designed at the hardware level.

Looking at the VPR architecture files, a fracturable LUT is expressed as two different modes of operation, as if the packer can select between two independent BLEs, each with different LUTs inside. Figure 2 shows how a fracturable 6-LUT (in an 8-input BLE) is described as two modes in the VPR architecture file. The first mode, in which the LUT is fractured, the BLE contains two 5-LUTs with two shared inputs. The second mode consists of only one 6-LUT that uses the first six (out of eight) BLE inputs and the first BLE output.

VPR cares only about how the fracturable LUT can be used and not how it is actually implemented. However, in practice, the LUT must be designed with all the additional logic that makes it fracturable, in order to model it correctly and account for the delay and area overhead of this feature. Figure 3 shows the actual design of the fracturable 6-LUT of Figure 2, as we implement it in FPRESSO. The concept of fracturable LUTs is derived from the fact that a K-LUT can be built using two $(K-1)$-LUTs by assigning to these two LUTs the same inputs and adding a multiplexer to select between their outputs, depending on the value of the K^{th}-input. Thus, when implemented in FPRESSO, the two 5-LUTs must be able to share all of their inputs when operating as a 6-LUT while maintaining three independent inputs in the dual-LUT mode. Therefore, at least one multiplexer is added to select the correct K^{th} input, depending on the mode. The other input multiplexers are optional but can be used in the cases where the routing is not flexible enough to guarantee connecting the same signals to the inputs of the two LUTs when in the second mode (Figure 2b). Similarly, additional multiplexers are introduced to select the LUT outputs before connecting them to the registers of the BLE, depending on the mode of operation. Note that these registers are not shown in Figures 2 and 3 for simplicity.

Using these predefined rules, we integrate fracturable LUTs into FPRESSO so that the hardware implementation is automatically inferred from a simple description of the modes of operations and the input/output connections. We generalize the process: FPRESSO reads in the different modes, as described in a VPR architecture file, then transforms them into a detailed design by adding all the related multiplexers and connections, depending on the sizes of the LUTs, the shared inputs and the BLE's input/output specifications. This design is then sized and optimized along with the remaining of the cluster. After optimization, the delays/areas are measured and the architecture file is generated with the fracturable LUT expressed as modes again and annotated with the respective delays, depending on the logic used in each mode.

III. Experimental Methodology

To facilitate the exploration of a very wide search space, the entire experimental process is fully automated. In this study, we model and explore over 1,200 architectures, as opposed to the 60 architectures tested by Ahmed and Rose [2], and run more than 41,000 benchmark simulations. Such an extensive exploration could not have been possible if the experimental setup was not fully automated.

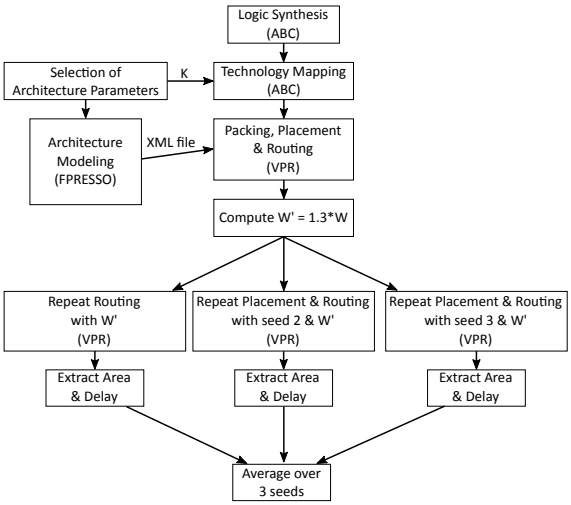


Fig. 4: The different steps of the experimental methodology.

A. General Flow

The experimental flow consists mainly of two phases: (1) architecture modeling and generation of architecture description files, and (2) running all selected benchmarks on each modeled architecture. The different steps of these phases are shown in Figure 4.

In the first phase, architectures are automatically modeled using FPRESSO [19] in a 65 nm UMC technology (typical corner). We only had to create a template of the desired architecture and varied the cluster parameters shown in Figure 1 before sending it to FPRESSO. The output is the architecture description file in the XML format required by the packer. Given that the prior work does not specify the routing density, we choose the default values of $F_{c_{in}}$ and $F_{c_{out}}$ (0.15 and 0.10 respectively) used in the architecture files provided with the VTR tool, with a Wilton switch box and a segment length of 4. The second phase handles the benchmark simulations on the modeled architectures. Each benchmark is first synthesized and technology mapped, knowing the size of the LUT (K) in the architecture, using the synthesis and verification tool ABC [3].

Using the architecture description file generated by FPRESSO, the benchmark is packed into the cluster, placed and routed, using VTR 7.0 [13], with unlimited routing constraints. Then, knowing the minimum channel width (W) required to route the benchmark, the channel width is increased by 30% and the routing step is repeated but now with a fixed channel width ($W' = 1.3 \times W$). Placement and routing is also repeated, for three different placement seeds, and the extracted delay and area results are averaged (over these seeds), to filter out the placement noise. In the different stages of the CAD flow, the tools are set to optimize for the critical path delay.

B. Benchmark Selection

There seems to be a growing consensus in the FPGA research that the MCNC benchmarks [17] are a rather outdated benchmark suite that does not represent realistic circuits on

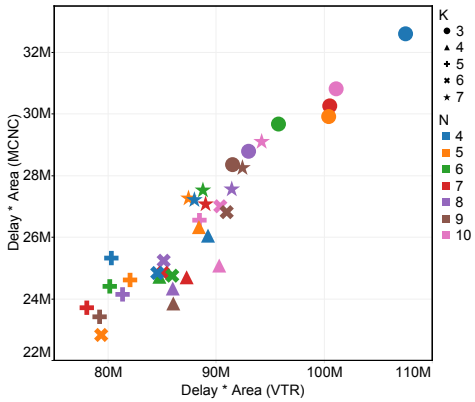


Fig. 5: Comparison of the delay-area product for the MCNC and VTR benchmark suites, over multiple K and N. The results of the two benchmark suites have a very similar behavior.

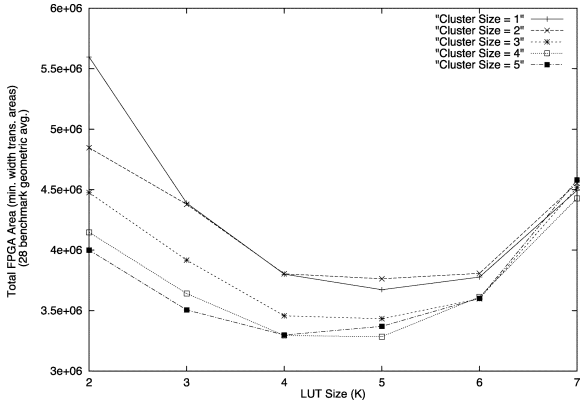
which the FPGA might be used. Even the big 20 MCNC circuits are often criticized for being small with some purely combinatorial designs and no heterogeneous circuits (memory and DSP blocks) [14]. This encouraged introducing new benchmark suites, such as the VTR benchmarks, in the VTR project [15], and the Titan benchmarks [14], as better alternatives to the MCNC benchmarks.

We set out to verify whether the MCNC circuits can be relied on in such architectural explorations. So, we design an experiment to compare the MCNC and VTR benchmarks over the same sets of architectures, modeled in FPRESSO so that the exact cluster delays and areas are used in both cases, even if the VTR benchmarks require additional RAM and multiplier blocks in the architecture. Since FPRESSO does not model RAM and DSP blocks, we use the delays and areas provided in the VTR 7.0 architectures, scaled to the correct technology node. We test the benchmarks of each suite on different architectures with a large range of K and N, and a fully populated crossbar.

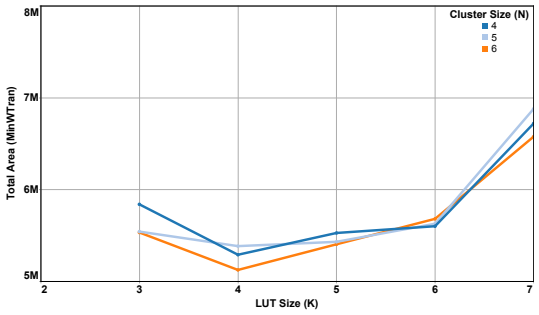
Figure 5 shows a comparison between the delay-area products obtained for each of the benchmark suites. Clearly, the scatter follows a linear trend indicating that the conclusions derived in such architectural explorations are valid using either of the two benchmark suites. There is no denying that the VTR benchmarks are bigger and have longer critical paths, resulting in about $3\times$ more delay-area product. However, this increase in size barely causes any deviation in the results since this ratio is maintained with very minor variations over the large set of architectures. This shows that the MCNC benchmarks can still be representative of the performance of an FPGA architecture and we would even claim that they are more advantageous in architectural explorations since they can lead to similar conclusions with a shorter experimental runtime. Hence, the MCNC benchmarks will be used in our experiments. We realize though that, if carry chains were to be added to the architecture, these conclusions might need to be re-evaluated since the VTR benchmarks might benefit from the hard adders and fast chains more than the MCNC benchmarks.

IV. Revisiting Existing Studies

We start first by re-evaluating the latest studies on the effect of the different cluster parameters onto the FPGA performance.



(a) Total area with respect to the LUT size K, for small cluster sizes, from the reference study [2].



(b) Total area with respect to the LUT size K, for small clusters of sizes (N) 4 to 6, measured using our methodology.

Fig. 6: Total area with respect to the LUT size K, for small cluster sizes, as measured in both the reference study of Ahmed and Rose [2] and our experiments.

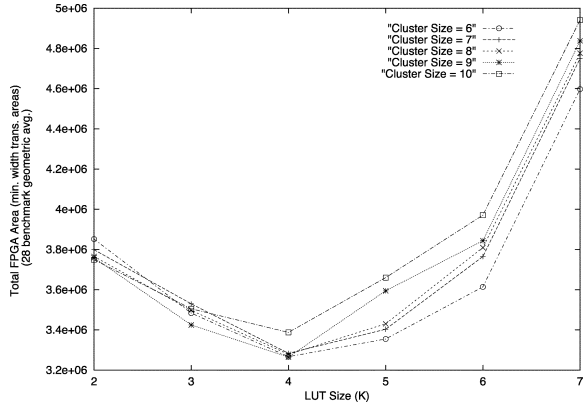
So, we limit our initial experiments to the search space of the these studies, with some minor differences. There are, in fact, several unspecified variables in the Ahmed and Rose papers, like the fraction of routing channels connected to each cluster input/output, $F_{c_{in}}$ and $F_{c_{out}}$, for example. As mentioned in Section III, we use the default $F_{c_{in}}$ and $F_{c_{out}}$ values given in the VTR architecture files, and this forces us to limit the cluster size to a minimum of 4, otherwise some routing channels will not be able to reach any cluster input. Furthermore, the new CAD tools introduce some limitations: for example, FPRESSO supports only up to 7-input LUTs while ABC does not map on only 2-LUTs.

Taking all these constraints into consideration, we select the architectures with the following cluster parameters:

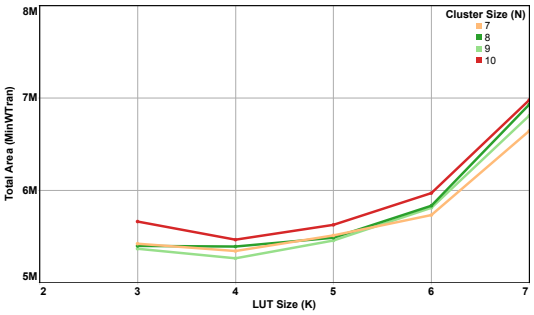
- $3 \leq K \leq 7$,
- $4 \leq N \leq 10$,
- $I = \frac{K}{2} \times (N+1)$, as used in the reference study [2], and
- $F_{c_{local}} = 1$ (i.e., a fully populated crossbar).

We test these architectures on the big 20 MCNC benchmarks and represent the results, reported using the geometric mean over all benchmarks, in the same format as in the original study.

Figures 6 and 7 show the total area, measured in minimum width transistors (MinWTran) as the LUT size varies from 3 to 7, for cluster sizes between 4 and 10. The results from Ahmed and Rose's study [2], for the same parameters, are also added for comparison. Figure 8 shows the total delay for the same

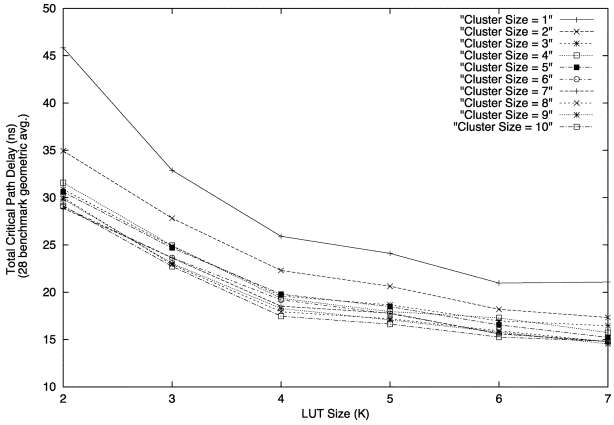


(a) Total area with respect to the LUT size K, for large cluster size, from the reference study [2].

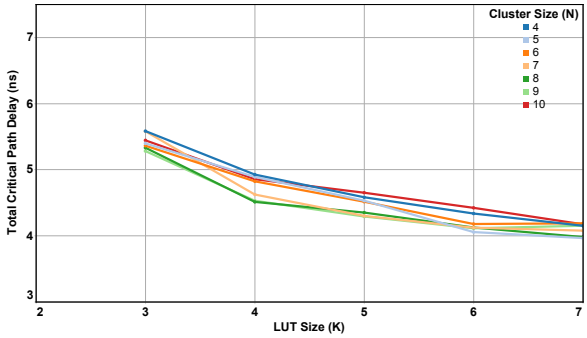


(b) Total area with respect to the LUT size K, for large clusters of sizes (N) 7 to 10, measured using our methodology.

Fig. 7: Total area of the MCNC benchmarks with respect to the LUT size K, for large cluster sizes, as measured in both the reference study of Ahmed and Rose [2] and our experiments.



(a) The total delay (in ns) as reported by the reference study [2].



(b) The total delay (in ns) measured using our experimental methodology.

Fig. 8: Total delay (in ns) with respect to the LUT size K, (a) as reported by the reference study [2] and (b) as measured from our experiments.

architectures and compares the reference results to ours. When compared with the prior work, one can clearly see the same behavior of the area and delay curves as K and N change. Certainly, the ranges of the area/delay values change due to the differences in technology; however, the same trends are generally preserved. Figure 9b shows the number of BLEs on the critical path for every combination of K and N and compares to the only curve available from the reference study in Figure 9a. The curves decrease at the same rate. We also show that, in general, the number of BLEs on the critical path decreases as well with the size of the cluster (N).

Having similar area and delay curves comes as a validation of the conclusions of the previous study on parameterizable clusters. These conclusions still hold despite all the changes in CAD and technology. According to our results, the architectures that lead to the best area have (K, N) values of (4, 6) and (4, 9), while the best delay was observed for (K, N) values of (7, 5), (7, 8), (7, 7), (6, 5) and (6, 7), which concurs with the findings of Ahmed and Rose [2].

A. Evaluation of Bigger Clusters

Setting the maximum cluster size to 10 can be a limitation to the search space, especially since bigger clusters may seem potentially promising: having more logic within the cluster can favor local feedbacks and reduce the use of global routing. Thus, we extend the experiments up to a cluster size of 15.

Figure 10 shows the change in area and delay as N increases, for all K. In general, larger values of N are not particularly advantageous, neither in area nor in delay. However, there are some particular cases for which some area and/or delay improvement is observed. Thus, increasing the cluster size is not particularly advantageous for all architectures but may present some improvement for particular parameters.

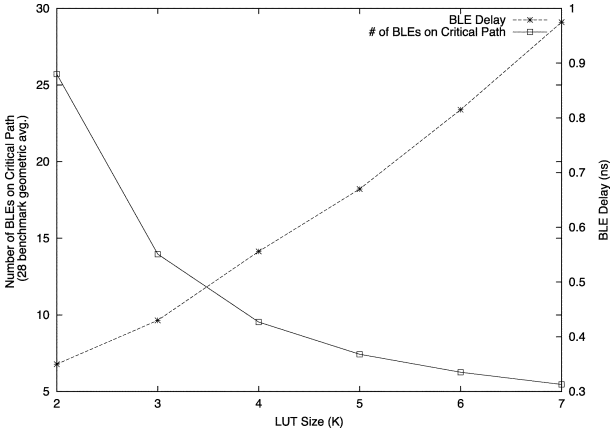
B. Evaluation of Input Assumptions

In the architectural explorations of Ahmed and Rose [2], the number of cluster inputs I was factored out as a parameter by computing it, for every architecture, as

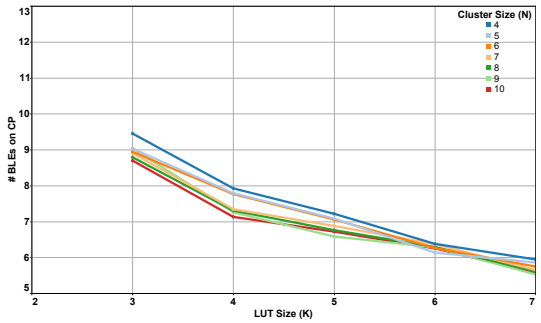
$$I = \frac{K}{2} \times (N + 1), \tag{1}$$

where K is the number of LUT inputs and N the size of the cluster. It was shown that with this I, a 98% cluster density can be achieved [2].

However, with the new packing algorithms of VTR, this relationship might not be valid anymore and Equation 1 might not achieve the same packing density. Furthermore, it was shown by Murray et al. [14] that a higher cluster density does not necessarily imply an improved FPGA performance. So, we decide to measure the effect of I on the performance of the FPGA and not only the density of its cluster. We run a set of experiments in which we vary the number of cluster inputs I by increasing the value computed using Equation 1 with



(a) The number of BLEs on the critical path as reported by the reference study [2].



(b) The number of BLEs on the critical path measured using our experimental methodology, for different cluster sizes (N).

Fig. 9: The number of BLEs on the critical path decreases as the LUT size (K) increases, both in the reference [2] and in our results. We show that it also decreases with as N increases.

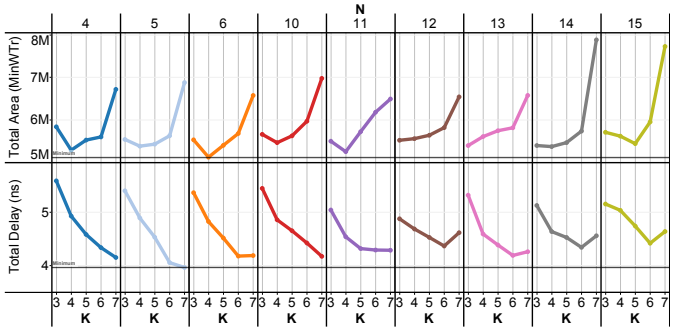


Fig. 10: The measured delay and area as the cluster size (N) varies, for different LUT sizes (K). In general, no clear trend emerges to favor large cluster sizes.

increments of 5. The objective of the experiment is to verify whether the cluster is being starved of inputs and if increasing I can improve its performance.

Figure 11 shows how the delay-area product changes with respect to the different steps of added inputs. In general, the delay-area product tends to increase with I. However, a closer look at the first part of the graph shows that adding 5 more inputs to the cluster actually improves the overall performance, for all cluster sizes. Adding more than 5 causes all the improvement to be lost. Thus, the number of inputs computed using Equation 1 is not sufficient and needs a boost by about 5 inputs.

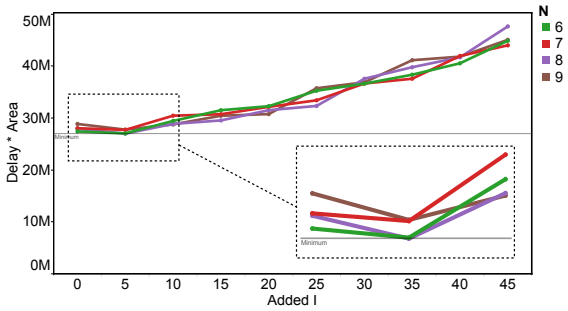


Fig. 11: The effect of the number of cluster inputs I on the delay-area product, for different N (averaged over K). The results are shown with respect to the number of added inputs ("Added I") to the original values computed using Equation 1.

V. Crossbars Density

The crossbars are used to distribute the cluster inputs (I) and the local feedback signals (N) to the LUTs of the cluster. Each crossbar consists basically of $N \times K$ multiplexers, as explained earlier, and the size of the multiplexer is determined by the density of the crossbar ($F_{c_{local}}$) along with I and N. Hence, as the clusters get bigger, the crossbars can become very expensive in terms of both area and delay. Sparse crossbars were introduced as a compromise between the flexibility of the routing and the cost of the full density [8], [9]. By depopulating a crossbar, each of its outputs can be connected to only a fraction ($F_{c_{local}}$) of its inputs. Therefore, using a sparse crossbar translates to smaller multiplexers that reduce the routing flexibility of the LUT inputs but, at the same time, reduce the area and critical path delay of the cluster.

Commercial FPGAs based on the Altera Stratix family use 50% sparse crossbars [10] in their clusters, while Microsemi has FPGAs with multi-level crossbars, some of which are highly depopulated, reaching a 25% density [5]. It was also observed through academic research that low-density crossbars are beneficial for some specific architectures (e.g., $K = N = 6$) [9]. However, with the new optimization algorithms in CAD tools and at smaller technology nodes (affecting the delay cost of long wires and large fanout nodes versus the delay of LUTs), the effect of the crossbars and of their sparsity on the FPGA performance might change dramatically. Would it change though, to the point that crossbars are no longer needed in the cluster? Or perhaps that denser crossbars are now needed? To study such effects and answer these questions generically, irrespective of the architecture, we evaluate the crossbar sparsity on a wide range of architectures by varying $F_{c_{local}}$ between 0.2 (i.e., each output can connect to 20% of the inputs) and 1 (i.e., a fully populated crossbar) for K between 3 and 7, and N between 4 and 10. FPRESSO distributes, by default, the inputs of the sparse crossbars uniformly among the created multiplexers. It is also designed in a way that, given a certain number of inputs per output, if the crossbar density cannot be achieved, this density is increased until all inputs can be connected. So, although a 20% crossbar is considered in these experiments, in several cases, the effective density is slightly higher than 20% (by a few percentages).

Figure 12 shows how the delay and area change with the density of the crossbar for the different K, averaged over N.

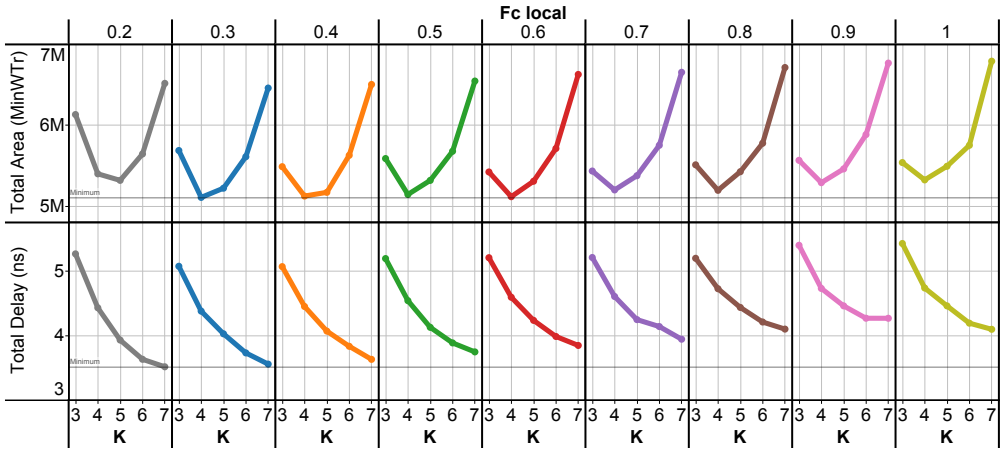


Fig. 12: The effect of sparse crossbars with different sparsity $F_{c_{local}}$ on the delay and area, for multiple LUT inputs (K), averaged over all cluster sizes. This figure shows that the best results are achieved for the crossbar with 30% density.

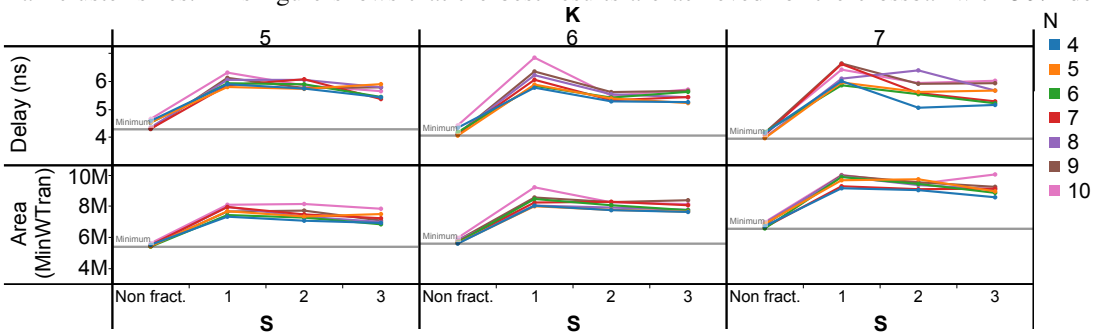


Fig. 13: Delay and area of the benchmarks tested on architectures with fracturable K-LUTs and up to 3 shared inputs S. The results for non-fracturable K-LUTs are also added for comparison.

The first observation is that, even after varying the density, our previous conclusions of Section IV on the optimal LUT sizes for area (4 and 5) and delay (6 and 7) still clearly hold for most $F_{c_{local}}$. However, more importantly, as $F_{c_{local}}$ decreases, area also decreases, until it reaches its optimum at 30% density, before increasing back at 20%. Interestingly enough, the best delay is also observed at around 20% to 30% density, making this an architectural sweet spot. Hence, the best compromise between the size of the crossbar and the routing flexibility is generally achieved, for most architectures, at the low density of around 30%. It benefits from the small-sized multiplexers to improve the delay and area contribution of the crossbar while maintaining enough flexibility not to introduce excessive routing overhead.

VI. Fracturable LUTs

Fracturable LUTs were commercially introduced in the Altera Stratix II [11] and Xilinx Virtex 5 [16] FPGAs. As discussed in Section II-C, a fracturable K-LUT can be split into two $(K-1)$-LUTs with S shared inputs.

To evaluate the effect of the fracturable LUTs on the FPGA performance, we run experiments for LUTs of sizes 5, 6, and 7, where each LUT can be fractured into two smaller ones, of sizes 4, 5, and 6, respectively, with shared inputs. The number of shared inputs S is also added as a parameter and varied between 1 and 3. The architectures are generated for clusters of sizes $N \leq 10$ and with full crossbars to filter out the effect of sparse crossbars on the results.

Figure 13 shows the delay and area of these fracturable K-LUT experiments. The graph shows the results over multiple cluster sizes, for the three choices of shared inputs. The same results for a non-fracturable 6-LUT are also added for comparison. The results vary, depending on the selected parameters but, in general, one can observe a clear trend irrespective of the LUT or cluster size. When compared to the results of the non-fracturable architecture, fracturable LUTs do not bring any improvement. A slight increase in area and delay was to be expected: the BLE has more inputs in the fracturable architecture, which results in bigger and slightly slower crossbars, and, with the extra logic added to support fracturability, the LUTs themselves are also slower. However, one would have thought that the additional flexibility of fracturable LUTs should have largely overcome this overhead—alas, this appears not to be the case.

To better understand the reasons behind this behavior, we evaluate the effect of the CAD tools on the results, by analyzing the outputs of the mapping, packing and routing stages as shows in Figure 14. Although this applies to any fracturable architecture, we take a cluster of size 10 with fracturable 5-LUTs ($K = 5$, $N = 10$). Any advantage the fractuable 5-LUT should bring is highly influenced by the number of 4-LUTs (or less) generated by the technology mapper and their shared inputs, since they can create opportunities to efficiently utilize the 5-LUT. However, only about 18% of the total LUTs have four inputs (and 34% have four or less inputs) which limits considerably these opportunities. Clearly, the default ABC's

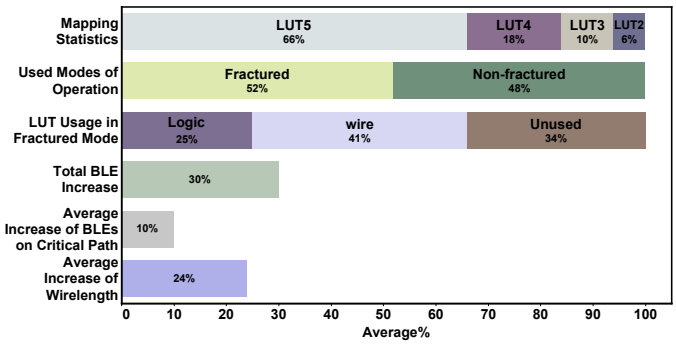


Fig. 14: Analysis of the mapping and packing results for a fracturable 5-LUT architecture with $N = 10$.

mapping algorithm is not aware that LUTs can be fractured and, as such, does not try to optimize its mapping accordingly. Furthermore, the packer is adding more damage by under-utilizing the available resources: almost half of the 5-LUTs are fractured into two 4-LUTs, and out of these 4-LUTs only 25% are used for logic, while 41% are used as wires and about 34% are completely unused. Using only 25% of the fractured LUTs is a major waste of resources and adds to this overhead, mainly since it increases the total number of BLEs by 30% (compared to the non-fracturable architecture), the number of BLEs on the critical path by about 10% and the average wirelength by 24%. All this added together gives an idea of the reasons behind the deterioration of the results for fracturable LUTs and is an indication that the existing academic CAD tools (specifically VTR) are not properly equipped or designed yet to support this feature. As for the commercial tools, Hutton et al. show that fracturable LUTs bring about 15% delay improvement for a 12% area reduction, on average, when combined with an LUT balancing phase integrated into Quartus' technology mapper [6].

VII. Conclusions

In this work, we evaluate the effect of the different cluster parameters on the performance of the FPGA, using a 65 nm technology and the latest academic CAD tools. After exploring over 1,200 distinct architectures and running more than 41,000 benchmarks through the flow, we show that the observations of Ahmed and Rose's classic study of 2004 are still correct today. However, we also show that some important FPGA architectural tenets appear not to always hold. We demonstrate, for instance, that MCNC benchmarks are still useful in architectural explorations and have, in this context, similar behavior as the more complex VTR benchmarks. We also refute, within the academic context, the common assumption that a half-populated crossbar is the best architectural compromise between crossbar complexity and routability; instead, we show that a 30% density crossbar systematically achieves better results for most architectures. We introduce fracturable LUTs into the modeling tool FPRESSO and show that, when using the academic CAD tools, the overhead of the fracturable LUTs overcomes any advantage they theoretically bring. We show through an analysis of the output of the different stages of the CAD flow that the current tools do not leverage properly the promised advantages of the fracturable LUTs. We plan, in the future, to extend our exploration to cover crossbars with different topologies, as well as clusters with hard adders and carry chains.

References

[1] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. In *Proceedings of the 2000 ACM/SIGDA 8th International Symposium on Field Programmable Gate Arrays*, pages 3–12, Monterey, Calif., Feb. 2000.

[2] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *IEEE Trans. Very Large Scale Integr. Syst.*, 12(3):288–298, Mar. 2004.

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

[4] C. Chiasson and V. Betz. COFFE: Fully-automated transistor sizing for FPGAs. In *Proceedings of the IEEE International Conference on Field Programmable Technology*, pages 34–41, 2013.

[5] J. Greene, S. Kaptanoglu, W. Feng, V. Hecht, J. Landry, F. Li, A. Krouglyanskiy, M. Morosan, and V. Pevzner. A 65nm flash-based FPGA fabric optimized for low cost and power. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 87–96, 2011.

[6] M. Hutton, J. Schleicher, D. Lewis, B. Pedersen, R. Yuan, S. Kaptanoglu, G. Baeckler, B. Ratchev, K. Padalia, M. Bourgeault, A. Lee, H. Kim, and R. Saini. Improving FPGA performance and area using an adaptive logic module. In *Proceedings of the 14th International Conference on Field Programmable Logic and Application*, pages 135–144, 2004.

[7] I. Kuon, R. Tessier, and J. Rose. *FPGA Architecture: Survey and Challenges*. Now, Delft, The Netherlands, 2008.

[8] 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 155–64, Monterey, Calif., Feb. 2000.

[9] G. Lemieux and D. Lewis. Using sparse crossbars within LUT clusters. In *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays*, pages 59–68, Monterey, Calif., 2001.

[10] D. Lewis, V. Betz, D. Jefferson, A. Lee, C. Lane, P. Leventis, S. Marquardt, C. McClintock, B. Pedersen, G. Powell, S. Reddy, C. Wysocki, R. Cliff, and J. Rose. The stratix routing and logic architecture. In *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays*, pages 12–20, Monterey, Calif., 2003.

[11] 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.

[12] 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.

[13] 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 (TRETS)*, 7(2):6:1–6:30, June 2014.

[14] 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*, pages 1–8, 2013.

[15] 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*, pages 77–86, 2012.

[16] Xilinx Inc. *Virtex-5 User Guide*. `http://www.xilinx.com/`.

[17] 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.

[18] G. Zgheib and P. Ienne. Automatic wire modeline to explore novel FPGA architectures. In *Proceedings of the IEEE International Conference on Field Programmable Technology*, pages 181–184, Xi'an, China, 2016.

[19] G. Zgheib, M. Lortkipanidze, M. Owaida, D. Novo, and P. Ienne. FPRESSO: Enabling express transistor-level exploration of FPGA architectures. In *Proceedings of the 24th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 80–89, Monterey, Calif., Feb. 2016.