FPRESSO: Enabling Express Transistor-Level Exploration of FPGA Architectures

Grace Zgheib
grace.zgheib@epfl.ch

Manana Lortkipanidze
manana.lortkipanidze@epfl.ch

Muhsen Owaida
mohsen.ewaida@epfl.ch

David Novo
david.novobruna@epfl.ch

Paolo Ienne
paolo.ienne@epfl.ch

Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences, 1015 Lausanne, Switzerland

ABSTRACT

In theory, tools like VTR—a retargetable toolchain mapping circuits onto easily-described hypothetical FPGA architectures—could play a key role in the development of wildly innovative FPGA architectures. In practice, however, the experiments that one can conduct with these tools are severely limited by the ability of FPGA architects to produce reliable delay and area models—these depend on transistor-level design techniques which require a different set of skills. In this paper, we introduce a novel approach, which we call FPRESSO, to model the delay and area of a wide range of largely different FPGA architectures quickly and with reasonable accuracy. We take inspiration from the way a standard-cell flow performs large scale transistor-size optimization and apply the same concepts to FPGAs, only at a coarser granularity. Skilled users prepare for FPRESSO locally optimized libraries of basic components with a variety of driving strengths. Then, ordinary users specify arbitrary FPGA architectures as interconnects of basic components. This is globally optimized within minutes through an ordinary logic synthesis tool which chooses the most fitting version of each cell and adds buffers wherever appropriate. The resulting delay and area characteristics can be automatically used for VTR. Our results show that FPRESSO provides models that are on average within some 10-20% of those by a state-of-the-art FPGA optimization tool and is orders of magnitude faster. Although the modelling error may appear relatively high, we show that it seldom results in misranking a set of architectures, thus indicating a reasonable modelling faithfulness.

1. EXPLORING FPGA ARCHITECTURES

Better FPGA architectures are of great value. Efforts towards reducing the existing efficiency gap between current FPGAs and dedicated circuits have the potential to bring the many benefits of reconfigurable computing to new domains, such as faster FPGAs for cloud computing or more energy efficient FPGAs for mobile computing.

A software flow able to synthesize circuits onto a wide range of different FPGA architectures is clearly an essential component to enable proper architectural exploration. Fortunately, the *Verilog To Routing (VTR)* project [16] already provides such a retargetable flow supporting a wide variety of easily-described hypothetical FPGA architectures. For example, precursors of VTR have successfully been used in industrially plausible FPGA architectures to explore optimal logic block configurations [2] or to show area and delay tradeoffs [11]. Furthermore, VTR has also been used to explore radically new architectures such as those based on And-Inverter Cones [15] or to assess the feasibility of adding area-efficient hard-logic cells of limited flexibility to reduce the dependence on *Look-Up Tables (LUTs)* [1]. The caveat, however, is that VTR architecture files need reasonable accurate area and delay models if the results are to be meaningful.

Creating a good model of a hypothetical FPGA architecture is difficult because it is hard to predict the effect of transistor-level optimizations on the circuit. We assume that in an FPGA architecture there is not much logic restructuring involved and the transistor-level circuits are generally well known (at least modulo a handful of alternate implementations of some components). Still, two elements change dramatically the area and delay characteristics of an architecture: appropriate transistor sizing and correct signal buffering. Both these elements critically depend on the architecture of the FPGA being explored, such as how many LUTs are connected to a given crossbar or how many crossbar inputs are connected to the output stage of the flip-flop assembly. Implementing the required transistor-level optimizations to obtain a reliable estimate of area and delay is a significant challenge due to the sizes of the circuits at hand.

2. MODELLING CHALLENGES

One possible, slightly wild, idea, would be the use of a semicustom design flow, based on standard cells, to design the hypothetical FPGA from a register-transfer level description. The idea would be to use the results of the semicustom flow as conservative estimates of what good de-

FPGA'16, February 21-23, 2016, Monterey, CA, USA
© 2016 ACM. ISBN 978-1-4503-3856-1/16/02...$15.00
DOI: http://dx.doi.org/10.1145/2847263.2847280

signers could conceive at transistor-level. Actually, the idea is not necessarily that wild since designers have discovered over the years that for many complex components (e.g., fast arithmetic components) semicustom approaches are today even superior to hand-crafted circuits [6]—and thus represent perfect estimates of what is achievable. Unfortunately, for FPGAs, Kim et al. [10] have recently shown that FPGA architectures designed with standard cells still incur severe area and delay overheads when compared to commercial full-custom FPGAs. Furthermore, these overheads largely vary across FPGA components rendering the models hardly faithful and thus unusable to drive realistic FPGA architecture exploration.

The other possibility is the one attempted by Chiasson and Betz with COFFE [4]: reduce the complexity of transistor size optimization in an FPGA by exploiting the structure of the circuit and by building a tool implementing ad hoc but efficient optimization strategies. The result is a reference and parameterizable architecture built at transistor level and a set of scripts implementing programmatically the required optimization for a set of parameters, using appropriate SPICE simulations for measurement. Although this works quite well for the given standard parametric architecture supported by COFFE, the optimization process is quite slow (in the order of hours). More importantly, there is no support for other quite different architectures researchers might want to play with: the optimization strategy is built into the scripts which constitute COFFE and, although in principle adaptable, porting it to wildly different architectures might essentially mean rewriting the tool from scratch, albeit with an excellent starting point. This does not seem in line with the level of generality built in VTR and which we think the research community requires.

In this paper, we address this modelling problem by proposing a novel approach that facilitates proper and quick modelling of FPGA components for users who are not transistor-level circuit designers. Our tool, which we call Fpresso, is able to model with an acceptable accuracy the delay and area of a wide range of largely different FPGA architectures without requiring the users to understand the issues of transistor sizing. For most users, all it is required is a topological description of the cluster of an FPGA and the automatic results are VTR architecture files annotated with area and timing estimations. We will demonstrate that Fpresso estimations, when a reference from COFFE is available, are quite close to those produced by a direct transistor-level optimization. We will explain the principles of our optimization strategy in the next section; we defer to the end of it an overview of the rest of the paper.

3. OPTIMIZING LARGE CIRCUITS

Our approach to making sound optimizations of complete FPGA architectures is somehow modelled on the divide and conquer approach used in semicustom design: Firstly, transistor-level designers construct highly-optimized libraries of standard building blocks (the standard cells). Libraries do not only limit the functionality of the cells to a set of basic classes, but contain several replicas of the same cell spanning a wide variety of transistor sizes. Secondly, once a library is available in a given technology, they characterize the cells, measuring in detail their area and delay characteristics. Finally, logic synthesizers, besides logically restructuring the target design and implementing it with the available standard cells, choose the most appropriate functionally equivalent cells and add required buffers to meet detailed high-level delay or area constraints. From the optimization point of view, the key of the semicustom design process is in locally optimizing transistors within cells to display a variety of potentially useful timing behaviours, and then in optimizing the overall circuit at a much higher abstraction than individual transistors and SPICE simulations.

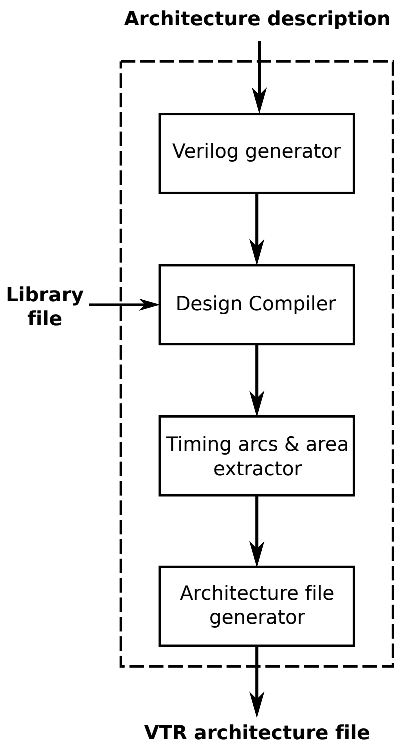


Figure 1: Tool flow for inexperienced users: Taking only an arbitrary description of the architecture (i.e., how LUTs, crossbars, etc. are interconnected) and a library of precharacterized components, Fpresso models the timing behaviour of the architecture and returns a VTR-compatible architecture file complete with all required area values and timing arcs.

Fpresso does exactly the same, conceptually, but avoids the unacceptable modelling errors of using standard cells by changing the granularity of the process. In Fpresso, expert users construct at transistor-level all the usually required components for FPGAs, such as LUTs, crossbars, multiplexers, flip-flop assemblies, etc. In the first step, *Cell Optimization*, an automated procedure generates a variety of implementations in terms of transistor sizes (e.g., drive strengths) and optimizes all versions of the components. In the second step, *Cell Characterization*, SPICE simulations extract the delay characteristics of the cell library. Both steps require a level of expertise that not every user has, but once this is done and as long as no new functional blocks are introduced, a completely automated *Architecture Optimization* step takes from inexperienced users a topological description of the hypothetical FPGA (essentially, a circuit typically composed of a few hundreds of known cells) and optimizes all drive strengths as well as adds buffers to generate a reliable model of the achievable area and delay. It is

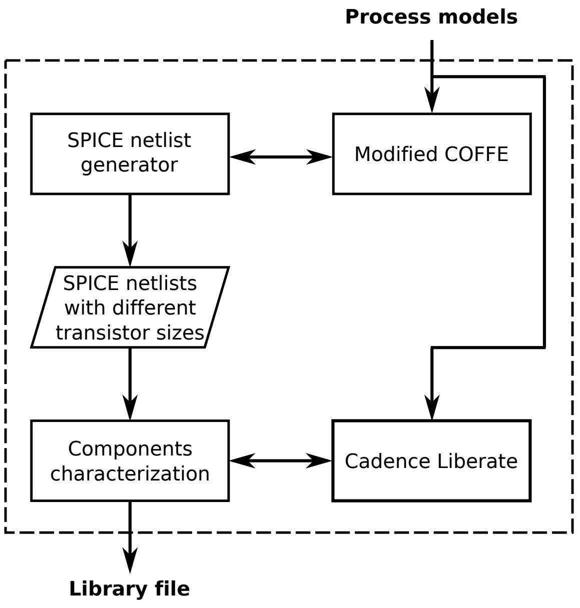


Figure 2: Library generation diagram.

important to realize that this last step, shown in Figure 1, demands no more transistor-level knowledge than specifying the cluster topology for VTR and yet is not limited to particular, predefined architectures.

So much this optimization procedure resembles the classic standard-cell design flow, that we strived to use wherever possible widely available off-the-shelf tools. Yet, the fact that our granularity is quite different (LUTs instead of NAND gates, so to speak) creates challenges in every step of this conceptually simple flow. In Sections 4 to 6 we describe in detail these challenges for the three steps mentioned above. In Section 7, we describe how we automated critical steps so that inexperienced users can get ready-made VTR models within minutes for arbitrary architectures. Section 8 compares our modelling results with the only immediately comparable tool, COFFE [4] while Section 10 elaborates on the value of the results. Section 11 discusses related work.

4. CELL OPTIMIZATION

The classes of coarse cells we need is pretty much implicitly determined by the architectures that we need to model: in typical cases, we may want to have LUTs, crossbars (that is, multiplexers with very large number of inputs), flip-flops, and muxes. The challenges of building our library will come from two sides: (1) how to automatically create a variety of versions of each cell with different driving strengths and (2) how to characterize each of these cells to enable a proper optimization. The former challenge is the topic of this section while the latter will be discussed in the next section.

4.1 Cell Transistors Sizing Flow

In a standard cell based ASIC design, a rich library with a wide range of drive strengths is essential to approach full-custom design efficiency [5]. Similarly, for our library of macro component we provide a wide range of different sizes, sweeping from very small components optimized for small loads, up to significantly large ones dimensioned to drive heavy loads.

For simple components such as standard cells, the sizing problem is not a terribly complex optimization problem as most cells are composed of just a handful of transistors. However, for complex components such as LUTs with tens of transistors to size, it becomes quite challenging, even for an expert circuit designer, to decide the optimal transistor sizes and automation of the process is essential. For this optimization, we decided to leverage COFFE [4] which, as mentioned in Section 2, is a tool that optimizes transistor sizes of a parametrized standard FPGA architecture. It uses SPICE simulations to iteratively search a range of transistor sizes to find the optimal combination for a given optimization criteria. COFFE happens to be quite suitable for our purpose since it already contains all the components needed for our cells; we simply modify it in order to isolate and size every component separately.

In the original COFFE flow, a component is sized while considering its context in the cluster, i.e., other components driving its input and connected to its output. We isolate the component from its surroundings and size it for a certain load capacitance C_L. To generate multiple drive strengths per component, we vary the load capacitance over a representative range of values. We use an exponential distribution obeying the following formula:

$$C_L(n) = C_{\text{Inv}} \cdot 2^{\frac{n}{2}}, \tag{1}$$

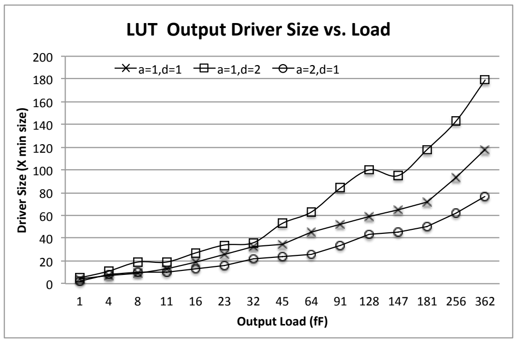
where C_{Inv} corresponds to the input capacitance for a minimum width inverter and $n = 0, 1, 2, ...N$, with the maximum capacitance defined by N. Accordingly, when substituting $n = 0$ in Equation 1 we obtain a minimum load capacitance equal to the input capacitance of the minimum width inverter in a given process technology. From there, we increment n to a maximum value N for which we obtain adequately large load capacitances. An adequately large load capacitance means that the components so optimized will never be used by the synthesis tool. With this, we would hope to provide a representative set of drive strengths for every component—but there is a catch as we will see in Section 4.2.

COFFE relies on a heuristic and not on the exhaustive exploration of all transistor sizing combinations—which would not be feasible given the very large search space and the need for time-consuming SPICE simulations. Instead, COFFE exploits the symmetries in the transistors netlist of each cell to reduce the number of transistors to be sized. For example, all the paths from one SRAM cell to the output in an LUT are identical, hence, only the transistors of a single path are sized. To further speed up the sizing process, the sizable set of transistors is split into several groups of a controlled number of transistors (5 or 6 transistors). This way, the number of transistor sizing combinations to be explored is further reduced. To compensate for any side effect of this clustering of the transistors, COFFE iteratively sizes the transistor groups until no further improvement is obtained.

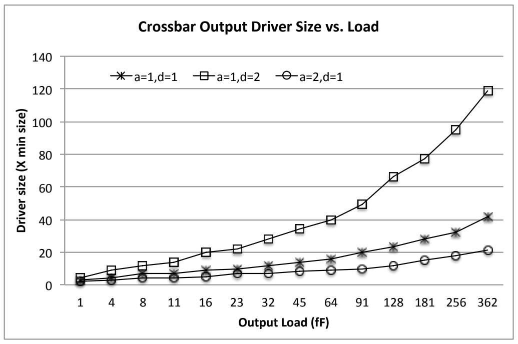
To rank the transistor sizing combinations, COFFE uses the following weighted area-delay product as a cost function:

$$Cost = area^a \cdot delay^d, \tag{2}$$

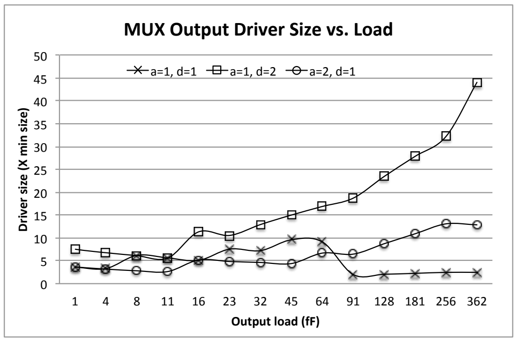
where a and d are user-defined parameters that prioritize area over delay, or vice-versa. The circuit delay for a transistor sizing combination is estimated by a SPICE simulation. The area, however, is calculated by component-dependent formulas that are programmatically encoded in the tool. At



(a) 5-LUT output driver size.



(b) 2-level 25:1 multiplexer output driver size.



(c) 2:1 multiplexer output driver size.

Figure 3: Analysis of the output driver size for different cells, optimization criteria and output loads.

the end of each sizing iteration, the transistor sizing combination that achieves a minimum cost is selected.

4.2 Generating Variety of Components

We understand that a rich library should include a large variety of driving strengths per component. This variety should be evident from the component output driver, whose size must vary significantly from a minimal size ($\approx 1\times$) to significantly larger sizes ($> 100\times$). This is due to the fact that the output driver is mainly responsible of restoring the output signal and delivering enough drive strength for the component load.

Experimenting with different weight values (a and b) in the optimization cost function (2), we found out that sticking to a certain combination of weight values (e.g., $a = 1, d = 1$) would not produce the type of variety we want in our library. On the other hand, we found that generating a large set of transistor sizings with different optimization weights coupled with a later pruning creates the desired variety.

Figure 3 shows the size of the output driver of several components, for different loads. A balanced optimization (weight values $a = 1, d = 1$) does not generate enough variety, and in some cases it stops increasing the size of the output driver way too early. This happens when the area increases by a factor that is larger than the delay reduction. Accordingly, an optimization that favours delay but that does not completely ignore area (e.g., weight values $a = 1, d = 2$) will generate larger drive strengths. At the same time, an optimization that gives more weight to area (e.g., weight values $a = 2, d = 1$) will not generate large drive strengths but components that may still be useful to optimize area in non critical paths of the FPGA architecture.

5. CELL CHARACTERIZATION

Cell characterization is a very well understood process: conceptually, tens or hundred of SPICE simulations are run within some simple testbenches with varying driving slopes or load capacitances (and process corner, temperature, and voltage, but this is outside the scope of our goals). EDA tools that perform the task in a completely automated way exist and, of course, they interface very well with any other part of semicustom toolchains. For our purpose, we have decided to use Cadence Virtuoso Liberate: an advanced library characterization tool that creates electrical views, such as timing and power, in standard formats like the Synopsys Liberty format (.lib). To characterize a cell, Liberate takes as input the SPICE netlist along with the foundry device models.

Our components are indeed qualitatively similar to standard cells (and in some cases are practically identical, such as flip-flops or 2-to-1 multiplexers). Unfortunately, many of the critical components, such as LUTs and large multi-level muxes for crossbars, are much bigger (in terms of both transistor count and number of inputs) than the biggest typical standard cells. This implies that, for understandable scalability issues, a tool like Liberate naturally fails to characterize some necessary components. We will discuss in this section how we circumvented the problems.

5.1 LUT Characterization

Liberate is capable of characterizing small size LUTs, such as LUTs with 2 or 3 inputs. However, as the number of inputs increases, the number of SRAMs grows exponentially, making the LUT characterization a major challenge for Liberate. Having a complex functionality and a high number of variables to track, Liberate fails to perform the required task.

In order to simplify the problem, we reduce the LUT complexity for the characterization tool by exploiting the symmetries in the LUT transistor design. As seen in Figure 4, a signal can travel from an input (e.g., in_A) to the output through what seems to be different paths. In practice, however, these paths include transistors of identical sizes because each transistor belonging to a level of the binary tree has the same size. The only actual difference is whether the input inverter is used or not. Thus, it is sufficient to pick a limited number of *representative paths* during the characterization to cover all the different input-to-output timing arcs, which can done by fixing the configurations of the SRAMs

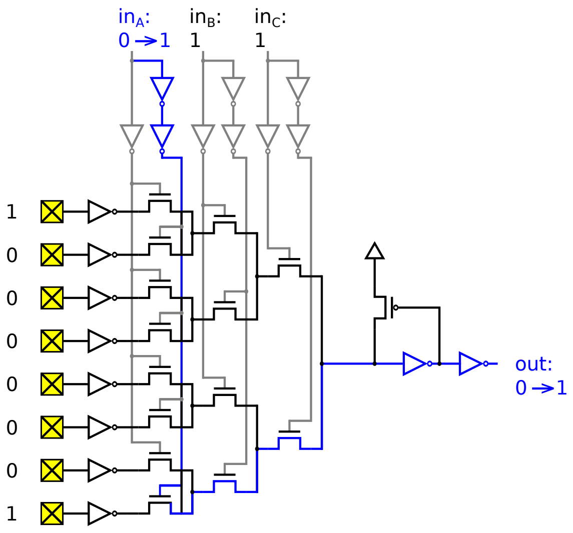


Figure 4: Transistor design of a 3-input LUT. Representative paths that ensure all signal transitions between the inputs and the output are selected to simplify the characterization of an LUT.

(i.e., setting them to ‘1’ or ‘0’). These representative paths must be selected in a way that guaranties the characterization of both inverted and non-inverted input-to-output paths. Furthermore, these paths must enable all possible signal transitions (rise-fall, rise-rise, fall-fall and fall-rise) between the inputs and the output.For that purpose, we set the configuration bits S_0 and S_7 in the example of Figure 4 to ‘1’ and the remaining ones to ‘0’. By configuring the 3-LUT in this way, all the input-to-output timing paths, along with their respective rise and fall delays, will exist in the generated library. To configure the LUT, the SRAM cells are connected directly to *VDD* and *GND*, reducing the number of variables visible to Liberate.

Once the LUT is characterized for the representative paths, the library is then corrected with the respective Boolean function (logic synthesis tools need the exact function to operate properly) and the SRAMs are added back as input variables. It is important to bring back the complete LUT interface into the library so that it can be directly associated with its circuit description when read into the later stages of FPRESSO.

5.2 MUX Characterization

FPGA crossbars are generally designed as 2-level multiplexers as shown in Figure 5. Each multiplexer is usually connected to all or to a fraction of the crossbar inputs, depending on the desired sparsity. In the Stratix-IV FPGA for instance, the crossbar has 72 inputs and is half populated [12, 13], which means that each 2-level multiplexer has 36 inputs and 12 SRAMs.

So, similar to LUTs, these multiplexers can have a large number of inputs making it impossible for Liberate to characterize. However, to ensure the crossbar functionality, only two SRAMs are set during one configuration (one SRAM in the first level and one in the second) to connect a single input to the output. This property is used to simplify the characterization of the 2-level multiplexers: For each input, the mux is configured to enable the path from that particular input to the output by connecting the two related SRAMs

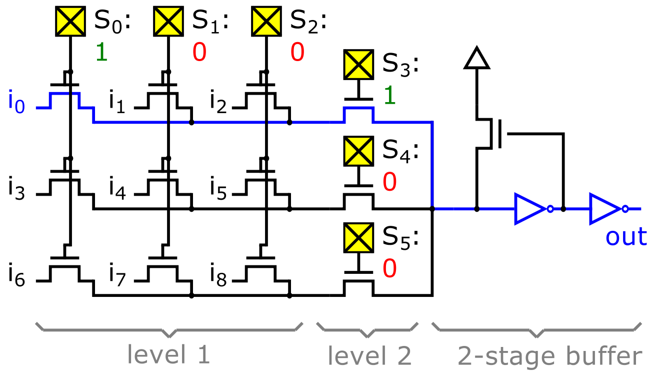


Figure 5: Transistor design and configuration of a 2-level multiplexer.

to *VDD* and the remaining ones to *GND*. Figure 5 shows the general structure of a 9:1 mux, where, to connect input i_0 to the output, S_0 and S_3 are set to ‘1’ while the remaining configuration bits are set to ‘0’. Once an input-to-output path is enabled, the remaining inputs are no longer relevant and can be forced to a logic value (‘1’ in our case). The mux input is then characterized and the process is repeated for all inputs, which generates multiple library files. Thus, in a final step, all the generated libraries are merged back into one, the Boolean function is corrected and the SRAMS are reintroduced as input variables.

6. ARCHITECTURE OPTIMIZATION

The tasks described in the two previous sections are needed to prepare the library of cells used to construct the architecture. As with standard cells, their development needs to be performed off-line by fairly experienced designers: in case of porting the library from one technology node to another, the process is almost automatic, but the expertise of a transistor-level designer is required if one is to add new components to the library (for instance, non-LUT logic

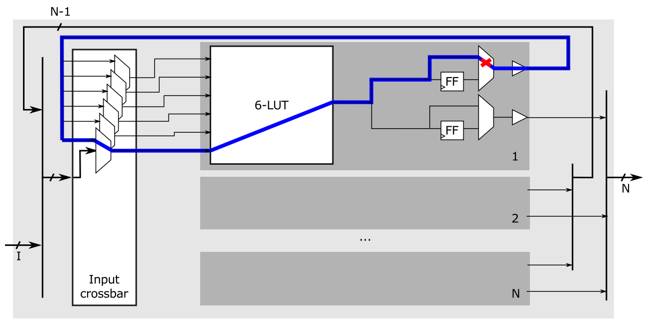


Figure 6: Timing loops within the logic cluster.

blocks). Once the library is available, architects can compose circuits using the defined functional blocks (we will discuss in detail how in Section 7): they define functionally the architecture of the FPGA and completely ignore electric and timing issues. To obtain an area or timing model of the architecture, it is necessary to take the netlist and optimize it for some specific constraints (such as minimizing the delay along some path) in two senses: (i) every functional block can be replaced with another of equal functionality but different characteristics and (ii) buffers can be added wherever it makes sense to. Although this is only a little part of what a logic synthesizer for a semicustom flow does, it does it remarkably well and it is readily available. We thus decided to use *Synopsys Design Compiler* for the global architecture phase—that is, for the optimization that needs to be run for each and every specific architecture a researcher is interested in exploring.

Since Design Compiler is not used within its typical flow as a logic synthesizer but mainly as a driving-strength optimization tool, the FPGA design must be annotated with proper attributes (such as the *set_size_only* attribute for the FPGA cells) indicating that they must only be sized. And, although SRAM cells exist in the FPGA design, Design Compiler has no notion of the reconfigurability of the FPGA and identifies feedbacks as timing loops. For instance, Figure 6 shows one of these timing loops that starts at the feedback multiplexer and continues through the crossbar, the LUT and then back into the feedback multiplexer. This combinatorial loop does not occur in configured FPGAs, but Design Compiler cannot identify this from the design. It can break the timing arcs itself, but for consistency, we decided to specifically tell Design Compiler that the loop does not exist, by breaking the timing path on the feedback multiplexer. Even SRAMs can be seen as a loop of two inverters which is then interpreted by Design Compiler as a timing loop. However, since there is no interest in sizing the SRAMs, Design Compiler must be instructed to ignore them (by assigning the *set_dont_touch* attribute to those inverters) so that it does not try to optimize this part of the circuit.

7. AUTOMATING FPRESSO

Having the main optimization part handled by Design Compiler, the major task of FPRESSO is taken care of. We are just left with the task of abstracting the complexity of the flow from the user and automating the different parts of FPRESSO as shown in Figure 1. This mainly consists of preparing a circuit of the FPGA design for Design Compiler and extracting the static timing analysis data from its timing reports to produce the VTR architecture file. We discuss in this section how we automate these parts of the tool flow.

7.1 Architecture Generation

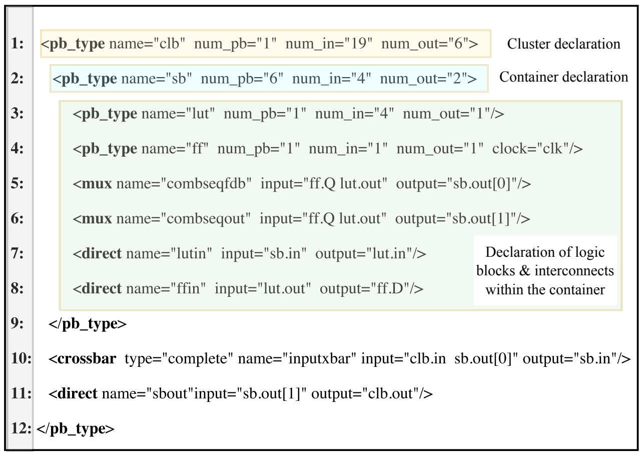
To model any FPGA, FPRESSO requires, as input, a description of the architecture, along with the library of cells. The user can either provide a complete VTR architecture file (XML format) or a simplified version of this architecture file. In the simplified version, it is sufficient to specify the different components of the FPGA, their interface (i.e. inputs, outputs and number of instances) and the way they are connected.

Figure 7 gives an example of the user architecture description using our simplified XML format. The example describes the architecture of Figure 8 using 4-input LUTs, 19 cluster inputs, and 6 LUTs per cluster. The description of the architecture is quite simple. Any hierarchical structure can be easily described as a hierarchy of XML elements, similar to the VTR architecture file. The different XML tags allow the user to declare any structural or functional component (e.g., cluster or LUT). The tool flow automatically identifies the hierarchy between the different components and determines which are functional cells and which are container components used only to maintain the hierarchical structure of the architecture. Interconnection blocks such as crossbars can be described simply by their input and output connections as well as they density (i.e. complete, sparse with a user-specified density, etc.) and FPRESSO will automatically translate it into 2-level multiplexer with the appropriate input connectivities. The positions of the switches of the 2-level multiplexer can be specified by the user using elaborate XML specifications. Some default topologies exist also in FPRESSO and can be directly used.

Generally, any FPGA cluster topology that can be expressed in XML format can be modelled by FPRESSO. However, a typical XML description (like the ones used in the VTR flow) can only provide a generic view of the architecture without having to detail accurate architectural specifications. For example, modes of operation can be easily specified using XML (e.g., a 6-LUT can also be configured as two 5-LUTs); however, FPRESSO needs to understand how these modes are translated into hardware components (e.g., the two 5-LUTs are connected through a multiplexer to form the 6-LUT). This kind of information needs to be added by the user to the XML file, providing FPRESSO with a detailed view of the circuit behind the architecture. This gives the user full flexibility and allows for the exploration of a wide range of FPGA architectures. This flexibility is ensured throughout the different stages of the tool flow. FPRESSO takes the architecture description file and translates it into a complete Verilog circuit where all components are instantiated and connected hierarchically the way they are expressed in the input description file.

7.2 Model Extraction

Knowing the different logic blocks and their interconnects, FPRESSO automatically generates the scripts needed to read the FPGA's Verilog description into Design Compiler, constrain it, optimize it, and report the timing arcs of each element. Once the Verilog description of the FPGA design is read into Design Compiler and assigned the right attributes, it is constrained from every cluster input to every cluster



```
<pb_type name="clb"  num_pb="1"  num_in="19"  num_out="6">
  <pb_type name="sb"  num_pb="6"  num_in="4"  num_out="2">
    <pb_type name="lut"  num_pb="1"  num_in="4"  num_out="1"/>
    <pb_type name="ff"  num_pb="1"  num_in="1"  num_out="1"  clock="clk"/>
    <mux name="combseqfdb"  input="ff.Q lut.out"  output="sb.out[0]"/>
    <mux name="combseqout"  input="ff.Q lut.out"  output="sb.out[1]"/>
    <direct name="lutin"  input="sb.in"  output="lut.in"/>
    <direct name="ffin"  input="lut.out"  output="ff.D"/>
  </pb_type>
  <crossbar  type="complete" name="inputxbar" input="clb.in  sb.out[0]" output="sb.in"/>
  <direct name="sbout"input="sb.out[1]" output="clb.out"/>
</pb_type>
```

Cluster declaration
Container declaration
Declaration of logic blocks & interconnects within the container

Figure 7: Example of the user's architecture description file for the architecture of Figure 8, using 4-inputs LUTs, 19 cluster inputs, and 6 logic blocks per cluster.

output. However, local feedbacks are a crucial aspect of the FPGA architecture, hence, additional constraints are set on the feedback paths, between the outputs and the inputs of the registers.

During the architecture optimization, FPRESSO targets delay, by default. It starts first by constraining the architecture for a maximum delay of zero, which obviously will not be met. However, at the end of this initial iteration, Design Compiler returns the minimum achievable delay, which is then increased by 10% and set as delay target in a second iteration of the optimization. This time, the timing constraint will be met and the provided extra slack is used by the tool to sensibly reduce the associated area.

Knowing the various cells that compose the FPGA logic cluster and their interconnections, FPRESSO can request the delays of all existing timing arcs from Design Compiler. All the timing reports are then parsed and the cluster cells are back annotated with the delay of every input-to-output timing arc. The overall area of the cluster is also extracted. Having the architecture description, the timing arcs, and the area estimation of the FPGA cluster design, FPRESSO returns a fully annotated architecture file in XML format, compatible with the VTR flow.

8. EXPERIMENTAL SETUP

FPRESSO can read in an architecture description to understand the FPGA cluster design it is modelling, and can thus be used on any architecture topology, as explained in the previous sections. However, in order to benchmark its performance, we restrict our experimental setup to the architectures supported by COFFE, the state-of-the-art modelling tool for FPGA components.

COFFE's architectural exploration is limited to a given FPGA topology customizable through some parameters. Figure 8 shows the cluster architecture supported by COFFE and used in our experiments, along with its three main parameters K, N, and I which represent the number of LUT inputs, total number of LUTs (or BLEs) in a cluster, and number of cluster inputs, respectively. Accordingly, we generate different FPGA clusters by varying K, N, and I and optimize the corresponding components using

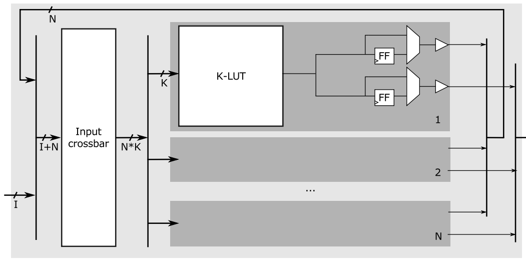


Figure 8: The FPGA architecture used in our experiments, with parameters K, N, and I.

the two tool flows: FPRESSO and COFFE. To limit variance in the comparison, all local register feedbacks are disabled in COFFE (and not included in FPRESSO), the crossbar is fully populated and the LUTs are not fracturable. A 65nm UMC technology is used in a typical corner in both flows.

Although *Switch Blocks (SBs)* and *Connection Blocks (CBs)* should be modellable in FPRESSO—they simply are 2-level multiplexers which already exist in the library, we decided to primarily concentrate in modelling the logic cluster components. Thus, the SBs and CBs are not included in the architecture seen by FPRESSO in these experiments. Instead, we properly constrain and size the multiplexers at the outputs of the cluster by specifying a particular load, which is a typical practice in semicustom design flows. This load is computed by taking the channel width assumed in COFFE, the fraction of the channels to which each output is connected (known as f_{Cout}) and the typical load observed for a 2-level multiplexer of that size. The load is then approximated and added before optimizing the cluster in Design Compiler.

FPRESSO is designed to optimize the circuit for delay, by default. So, for a fair comparison, we optimize for delay as well in COFFE by doubling the delay-to-area ratio of the cost function parameters (setting $d = 2$ and $a = 1$).

9. EXPERIMENTAL RESULTS

Multiple cluster architectures are generated (by varying K, N, and I) and optimized in both tools. To represent the delay, we differentiate between two delay paths: (i) the feedback path and (ii) the direct IO path. Using Figure 8 as reference, the feedback path starts at the output of the flip-flop and goes through the feedback multiplexer, the crossbar and the LUT, back to the input of the flip-flop. The direct IO path starts from the cluster inputs and passes through the crossbar, the LUT and the output multiplexer, into the cluster output. Figures 9a and 9b show the relative delay and area measured in FPRESSO with respect to COFFE for the two paths, respectively.

We observe that the differences between COFFE and FPRESSO is around 10% in terms of area and less than 35% in terms of delay—typically, in the 25% range for the feedback path and 10% for the IO path. In general, these results are quite encouraging, since these differences account for both modelling errors due to our library-based approach and differences in the optimization procedure—Design Compiler optimizes for a cost function that is certainly different than the one used in COFFE. However, it is obvious that the optimization of the IO path is closer to the reference than that of the feedback path. One possible explanation could

Table 1: Runtime improvement provided by FPRESSO, when compared to COFFE, for a range of architectures.

Architecture	Speed-up
K5_N6_I19	409
K5_N6_I30	413
K5_N6_I43	347
K6_N6_I19	411
K6_N6_I30	492
K6_N6_I43	278
K4_N6_I19	394
K4_N6_I30	305
K4_N6_I43	308
K5_N8_I41	208
K5_N8_I28	332
K6_N8_I41	267
K6_N8_I28	229
K5_N10_I39	239
K5_N10_I26	193

be the lack of adequate wire load modelling in FPRESSO: given that FPRESSO is not limited to a specific FPGA cluster topology, identifying which wires are longer than others (e.g., long feedback wires) can be a challenge and might require additional physical information from the user. Nevertheless, other factors could have also contributed to these differences, such as the advanced sizing capabilities of Design Compiler or the selective insertion of buffers on the high-fanout feedback path. We intend to further investigate the reasons behind these results and plan to include some wire load modelling capabilities in future versions of FPRESSO.

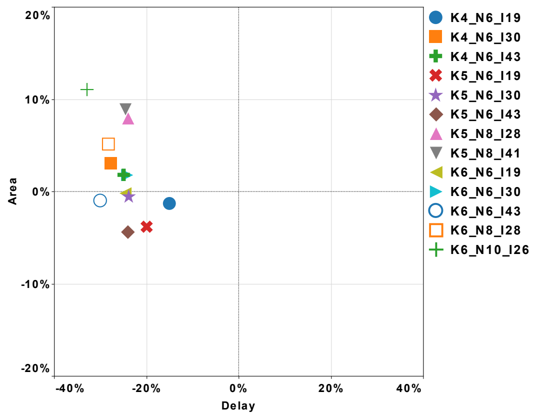
In a second experiment, we select a single architecture (i.e., $K = 5$, $N = 6$, and $I = 30$) and change the optimization cost functions to vary their relative priority in area vs. delay. Figure 10 shows the resulting Area-Delay Pareto front of the reported solutions. We observe that COFFE is able to cover a wider range of area-delay tradeoffs whereas FPRESSO concentrates in the arguably most interesting regions of that space. We also observe that, within the ranges spanned by our tool, FPRESSO consistently Pareto dominates COFFE. We are not able to conclude whether this is due to a superior optimization of our tool or to the fact that FPRESSO does not yet model intercomponent wire load.

In architecture, usually the relative ranking of some critical delay for various architectures is more important than the absolute delays themselves. To see how faithful FPRESSO is, we ordered many architectures in function of growing delay in COFFE and plotted the delay reported by FPRESSO for the same architectures (see Figure 11). Ideally, the curve should be monotonic and we see that only in four cases it is not. It is worth noticing the scale and the fact that these inversions are relatively modest, in the worst case in the order of 3% and significantly smaller in the others.

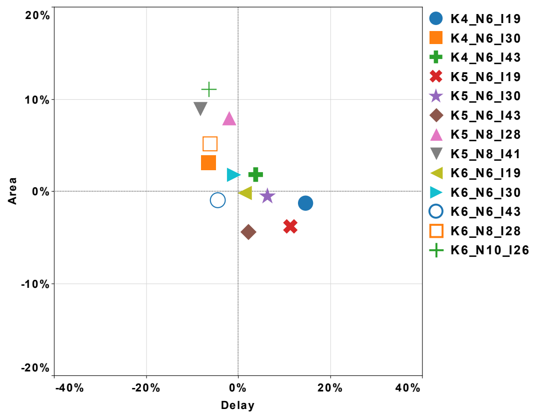
Furthermore, COFFE is on average 300 times slower than FPRESSO once all the components are in the library, as seen in Table 1. This is a game-changing difference that can enable far more comprehensive architectural explorations.

10. MODELLING OR DESIGNING

It would be tempting to believe that our tool *designs* optimized transistor-level architectures, instead of simply *mod-*



(a) Feedback path: from flip-flop to flip-flop.



(b) Direct IO path: from cluster inputs to cluster outputs.

Figure 9: Delay and area of the main paths of an FPGA, modelled in FPRESSO with respect to COFFE, for multiple architectural parameters. Figure 9a shows the results for the feedback path, which goes from the feedback multiplexer through the crossbar and the LUT. Figure 9b shows the same results but for the path going from the cluster inputs through the crossbar, LUT, and output multiplexer.

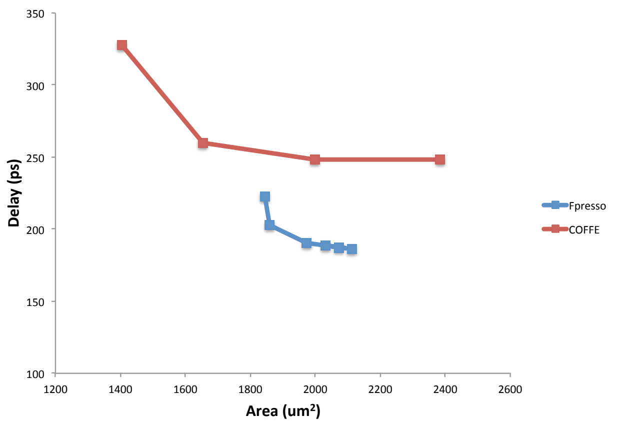


Figure 10: The delay and area Pareto fronts of multiple optimizations performed by FPRESSO and COFFE, for a single architecture ($K = 5$, $N = 6$ and $I = 30$).

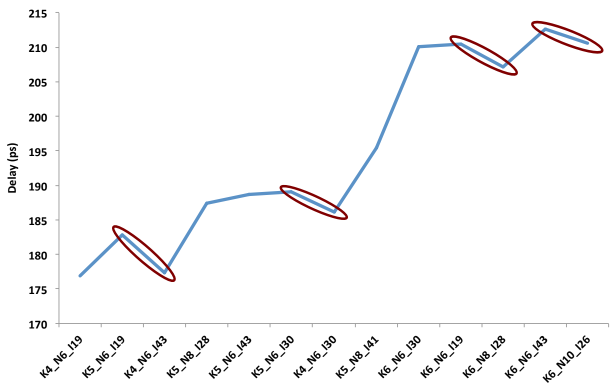


Figure 11: Critical delays reported by FPRESSO for various architectures ordered for growing critical delay as reported by COFFE. Ideally, the graph should be monotonic whereas it is not in a few cases, most pretty marginal. Note that the vertical axis does not start from zero.

elling them. Although this is a tempting claim, we do not think it is a granted one. The reason is that standard cells and a classic semicustom design flow have a number of built-in electrical safeguards to guarantee functionality under any constraint; our flow, purposely, does not. For instance, standard cells never expose pass transistors to the external pins of the cell and this is one of the reasons why standard cell designs cannot match in many practical cases perfectly crafted manual designs. In our case, although on the outputs we always have buffers, we omit input buffers to mimic the way a hand-crafted transistor-level circuit would be built. We have studied some of the circuits resulting from our flow and, within the range of fairly conventional architectures reported here, we have not observed any electrical error; yet, our methodology is such that we do not guarantee functionality for every possible conceivable circuit. At best, we can affirm that our flow helps fast and sound modelling (our prime goal) and actively suggests architectural solutions in the buffering structure which designers may want to study in case they want to produce a production transistor-level implementation. One should note that FPRESSO benefits from the advanced buffer optimization strategies of Design Compiler which largely exceed the resizing capabilities of COFFE, for instance: Design Compiler can not only build multistage optimal buffers when required, but can also add buffers after fanout points when load is divided unevenly across different circuit branches. The frequent Pareto dominance of FPRESSO results over COFFE may be partly due to this effect, but detailed analysis of the results has not been conclusive as yet.

11. STATE OF THE ART

The modelling problem addressed in this paper is very related to the transistor sizing problem of custom circuits. The latter is a well-studied optimization problem that targets the improvement of circuit performance by adequately increasing the size of its transistors. Fishburn et al. [7] show that modelling transistors as linear resistances and capacitances, and calculating the delay of the resulting RC circuits allows the transistor sizing problem to be formulated as a convex optimization problem, which guarantees that any local minimum is the global minimum. Accordingly, several algorithms guaranteeing the optimal solution of such formulation have subsequently been proposed [3, 17]. However, all this prior work relies on linear device models that are known to be quite inaccurate, especially in the latest CMOS technology nodes [9].

For the concrete case of FPGAs, Kuon and Rose [11] propose a two-phased approach consisting of an exploratory phase that uses linear device models followed by a SPICE-base fine-tuning phase that adjusts the transistor sizes to account for the inaccuracies of linear models. In contrast, Chiasson and Betz propose COFFE [4], a transistor sizing tool for FPGAs that completely relies on SPICE. FPRESSO differs from both in the fact that it only uses SPICE—a very detailed and slow simulation—in the characterization phase of the components and not in the optimization of every single FPGA architecture. As a result, our tool is capable of modelling new complete architectures significantly faster, provided that there is component reuse, which is typically the case in FPGA architectures (e.g., the same 6-input LUT component can be included in many different architectures). Furthermore, our tool provides full flexibility allowing the user to interconnect the components as desired, as opposed to the reference tools where local interconnects are programmatically encoded. Hence, FPRESSO provides a more versatile solution that enables the exploration of largely different FPGA architectures without requiring any transistor sizing expertise from the users.

12. CONCLUSIONS

Retargetable toolchains are one of the keystones of architectural research, but are somehow more tricky to use than one thinks. In computer architecture, everyone has access to a variety of fairly accurate architectural simulators, customizable in many aspects including, naturally, every aspect of the memory hierarchy. A few years back, someone noticed how difficult it was for most researchers to predict the effect of some architectural changes in the memory hierarchy on its area and latency: understanding most of the implications requires a deep knowledge of the transistor-level implementation options of leading-edge memories, which is clearly outside of the classic skill set of an architect. CACTI [18] was born out of that need: an easy-to-use and sound model for caches and other memory hierarchy elements. Over a couple of decades, six major revisions [14], and continuous new extensions [8], CACTI has helped literally hundreds of research groups in their scientific quest.

We think, as perhaps others in the area, that FPGA architectural research suffers today from the same syndrome that afflicted in the early nineties the computer architectural community: namely, the difficulty of combining in the same researcher or even research group acute architectural intuition and leading-edge transistor-level design skills. Infinitely more modestly compared to the CACTI endeavour, we have shown a new path towards quick and efficient modelling of almost arbitrary FPGA architectures: we can generate within minutes optimized VTR models with reasonably faithful delay and area characteristics. Although we are not there yet, we strive to develop a web interface to our tool that will allow researchers to submit architectural descriptions online and obtain almost immediately reasonable VTR models enabling sound and consistent architectural explorations.

13. REFERENCES

[1] I. Ahmadpour, B. Khaleghi, and H. Asadi. An efficient reconfigurable architecture by characterizing most frequent logic functions. In *Proceedings of the 25th International Conference on Field-Programmable Logic and Applications*, 2015.

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

[3] C.-P. Chen, C. C. Chu, and D. Wong. Fast and exact simultaneous gate and wire sizing by lagrangian relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(7):1014–1025, 1999.

[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] D. Chinnery and K. Keutzer. *Closing the Gap Between ASIC & Custom: Tools and Techniques for High-Performance ASIC Design*. Springer US, New York, NY., 2002.

[6] H. Eriksson, P. Larsson-Edefors, T. Henriksson, and C. Svensson. Full-custom vs. standard-cell design flow: an adder case study. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 507–510, 2003.

[7] J. P. Fishburn and A. E. Dunlop. TILOS: A posynomial programming approach to transistor sizing. In *The Best of ICCAD*, pages 295–302. Springer, 2003.

[8] N. P. Jouppi, A. B. Kahng, N. Muralimanohar, and V. Srinivas. CACTI-IO: CACTI with OFF-chip power-area-timing models. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, VLSI-23(7):1254–67, July 2015.

[9] K. Kasamsetty, M. Ketkar, and S. S. Sapatnekar. A new class of convex functions for delay modeling and its application to the transistor sizing problem [CMOS gates]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(7):779–788, 2000.

[10] J. H. Kim and J. H. Anderson. Synthesizable FPGA fabrics targetable by the Verilog-to-Routing (VTR) CAD flow. In *Proceedings of the 25th International Conference on Field-Programmable Logic and Applications*, 2015.

[11] I. Kuon and J. Rose. Exploring area and delay tradeoffs in FPGAs with architecture and automated transistor design. *IEEE Transactions on Very Large Scale Integration Systems*, 19(1):71–84, 2011.

[12] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff, and J. Rose. 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.

[13] D. Lewis, E. Ahmed, D. Cashman, T. Vanderhoek, C. Lane, A. Lee, and P. Pan. Architectural enhancements in Stratix-III$^{\mathrm{TM}}$and Stratix-IV$^{\mathrm{TM}}$. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '09, pages 33–42, New York, NY, USA, 2009. ACM.

[14] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0: A tool to model large caches. Technical Report HPL-2009-85, Hewlett-Packard Development Company, Palo Alto, Calif., Apr. 2009.

[15] 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.

[16] 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.

[17] V. Sundararajan, S. S. Sapatnekar, and K. K. Parhi. Fast and exact transistor sizing based on iterative relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(5):568–581, 2002.

[18] S. J. Wilton and N. P. Jouppi. An enhanced access and cycle time model for on-chip caches. Technical Report WRL-93-5, Digital Equipment Corporation, Palo Alto, Calif., July 1994.