Automatic Wire Modeling to Explore Novel FPGA Architectures

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*—Correct modeling of the FPGA architecture is a fundamental step in the design and testing of new architectures. Such modeling must include the logic components of the FPGA as well as the wires connecting these components. In recent years, researchers have been investigating tools to perform automatic modeling of FPGAs with either a fixed structure (e.g., COFFE) or even unconstrained architecture designs (e.g., FPRESSO). And, although these tools highlight the importance of intercomponent wire modeling, FPRESSO does not model wire loads due to the difficulty in determining the length of the wires without a fixed cluster structure, while COFFE approximates the wirelength by assuming a simplified topology and a predefined placement of components. In this paper, we present an automatic and generic method to not only model the intercomponent wires but also to minimize the total wirelength, by floorplanning the cluster and finding the best placement of its components. Our algorithms were integrated into FPRESSO to improve its modeling by including wire loads in its FPGA architecture modeling.**

I. Introduction

With the introduction of FPGAs into cloud and mobile computing, bridging the gap between FPGAs and ASICs has become more urgent, motivated by the demand for faster and more energy efficient FPGAs. This triggered the search for novel architectures that can satisfy the new requirements.

The FPGA software flow is sufficiently generic to support a wide range of FPGAs, enabling the exploration of unconventional architectures, especially with the *Verilog To Routing (VTR)* project [8]. So, the attention turned to the problem of providing these CAD tools accurate delay and area measurements of the FPGA's logic and routing elements, which is essential for any architectural exploration. Several tools emerged recently, in the attempt of solving this problem, focusing on modeling the FPGA architecture to estimate the area and delays as accurately as possible.

COFFE [2], an automatic transistor sizing tool, models FPGA architectures with a predefined structure but allows the user to customize it by varying several parameters such as the number of *Look-Up Tables (LUTs)*, the size of the LUTs, etc. Depending on the configured architecture, COFFE automatically sizes the different components and runs SPICE simulations to measure the delays of the relevant paths. Although practical, the limitations in the COFFE supported architectures reduce the search space of the architectural explorations. So another tool was more recently introduced, FPRESSO [12], to overcome the limitations of COFFE and cover a wider range of architectures. FPRESSO is a fast transistor modeling tool that uses existing standard cell tools to model FPGAs. It builds, offline, a library of all the components that can be used to compose an FPGA, filled with all the delay and area details needed. Then, when the user provides a description of the architecture, FPRESSO models and optimizes it within minutes using a standard cell synthesizer and the library of components. With this approach, the architectures that FPRESSO can model are limited only to the types of components available in the library, which enables the exploration of novel and unconventional FPGAs.

978-1-5090-5602-6/16/$31.00

However, despite the major effort put into correctly modeling FPGAs, any modeling remains too far from correct unless coupled with proper wire modeling. It was observed, in COFFE, that the critical path delay almost doubles after modeling the wire loads. In FPRESSO, however, wire loads were modeled within each component (between the different transistors) but no intercomponent wire model was considered. The authors observed that the delay of the feedback path, which consists of a long wire, was about 20% more optimistic. That is due to the large capacitance on that wire, that was not accounted for.

So, modeling the wire loads is an essential step towards a correct modeling of FPGA architectures. But, as stated in FPRESSO, without a predefined structure of the FPGA architecture, the tools lack any indication on the length of the wires. In this work, we present a method to automatically determine the length of the interconnects within the FPGA cluster, generically, irrespective of the architecture or components used within the FPGA. We were able to integrate our approach into FPRESSO to improve the correctness of its modeling and of future explorations of potentially novel architectures.

Perhaps the most related research to our work is GILES [7], a tool that generates transistor-level schematics and presents a place and route algorithm to obtain a compact layout of the architecture. However, our goals are fundamentally different since we do not plan on generating layouts from our design, but we target a fast modeling and estimation of the delay/area of an FPGA architecture, while taking into account the parasitics of the wires within that architecture.

II. The wire modeling problem

To model the wires and measure their length, the dimensions of the different cluster components must be known as well as their placement within the cluster. And, to minimize the wire load, all the components connected through the same net must be placed as closely as possible to reduce the length of that net. So, the architecture can be seen as a network of components that must be placed within the cluster, as densely as possible. And by that, the wire modeling problem can be transformed into a floorplanning problem with the optimization objective of minimizing the wirelength.

In this work, we start by converting the cluster architecture, with all its components and connections into a B*-Tree

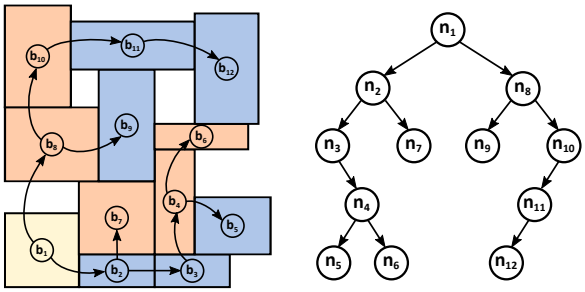


Fig. 1: An *admissible* floorplan, compact on the lower left corner, and its equivalent B*-Tree.

representation. Then a simulated annealing-based algorithm is implemented to floorplan the tree and optimize the placement of each module while minimizing the overall wirelength. Once the floorplan is optimized, the best solution is chosen as the final placement of all the components and the length of each wire is computed to derive its respective load and resistance.

In the next sections, we explain the different steps of this approach starting with the B*-Tree representation and its characteristics in Section III. Then, we present in Section IV the simulated annealing-based floorplanning algorithm used to optimize the B*-Tree representation of the cluster. Then, in Section V, we discuss the integration of our wire modeling approach into FPRESSO and its results in Section VI along with a comparison with COFFE.

III. B*-TREE REPRESENTATION

A *B*-Tree* is a representation in the form of a binary tree that was introduced to overcome the limitations of existing floorplanning representations [1], [11]. A B*-Tree depicts the floorplan through a set of predefined geometric relationships. Each node (n_i) of the tree corresponds to a placed module (b_i) where the root of the tree is the module placed at the lowest left corner of the floorplan. The placement of each node is relative to the placement of its parent. If node n_j is the left child of node n_i, then b_j is placed on the right-hand side of b_i, directly adjacent to it. In this case, the x-coordinate of b_j is determined by $x_j = x_i + w_i$ where x_i is the x-coordinate of b_i and w_i its width. If n_j is the right child of node n_i, then b_j is placed immediately above b_i so that $x_j = x_i$. Figure 1 shows a B*-Tree and its equivalent floorplan, generated following the geometric relationships of the B*-Tree nodes.

To convert a B*-Tree to a floorplan, the tree is traversed in a way similar to the *Depth-First Search* by placing the root first at the lower left corner and then recursively tracing and placing its left subtree first then the right one. Similarly, an existing placement can be transformed into a B*-Tree if it is *admissible*, meaning that it is compact on the lower left corner such that no module can be further moved left or down. The floorplan of Figure 1 is an example of an admissible placement. The B*-Tree is derived by starting from the module in the lower left corner (b_1), setting it as root to the tree and then recursively traversing the modules on its right, building the left subtree of the B*-Tree before visiting the modules above it, building the right subtree.

Although the x-coordinates of the B*-Tree nodes are easily derived, the y-coordinates must be computed during its conversion to a floorplan. When placing a module b_i at x_i, it must not overlap with the existing modules, so its y-coordinate is determined by the maximum height of the current floorplan between x_i and $x_i + w_i$. For that purpose, a *horizontal contour* of the existing floorplan is saved, keeping track of the maximum height of the modules already placed [4], [1]. Figure 2a shows the horizontal contour before and after placing module b_{11}.

IV. FLOORPLANNING ALGORITHM

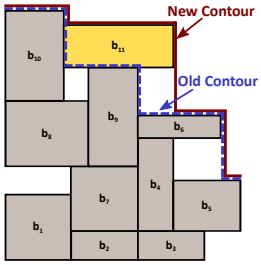
Simulated annealing [6] is commonly used to optimize the placement of floorplans in VLSI design [9], [10]. The adopted algorithm starts by taking a randomly-generated solution in the form of a B*-Tree. Each node of the tree is equivalent to a module in the floorplan or a component in the cluster architecture. Having the node at a particular location in the tree is like specifying its (x, y) coordinates in the floorplan, as explained in Section III.

Starting with an initially high temperature T, the tree is perturbed M times, where each time the cost of the new tree is computed. If the new solution is better than the existing one, it is selected for the next iteration. However, if the new tree has a higher cost, it is accepted with a probability *p_uphill*, defined by $\text{p\_uphill} = \mathrm{e}^{\frac{-\Delta\text{cost}}{T}}$ where Δcost is the difference in cost between the new solution and the current one, and T is the current temperature. Since, initially, T is much larger than Δcost, the probability of accepting a worse solution is very high, allowing for the search to escape local minima. The temperature is gradually decreased, every M iterations, using $T = \lambda^n T$, where n is the number of times the temperature is decreased and λ is a variable used to control the speed with which the temperature drops. The entire process is then repeated and, as the algorithm advances, worse solutions are less likely to be selected (p_uphill gets smaller), leading to an eventual freezing of the system around the best seen solution.

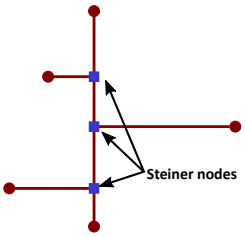
Perturbation: The tree is perturbed in two steps: First, a node is selected at random and removed from the tree while its sub-branches are re-arranged, randomly and recursively. Then, the removed node is reinserted in the tree as the child of another randomly-selected node.

Cost function: The total wirelength is used as a cost function to guide the algorithm in the search for a solution that reduces the wireload effect. To minimize and measure the wirelength, we use the *Rectilinear Steiner Minimal Tree (RSMT)* [5] which determines the tree that connects a set of nodes using the minimum wirelength, measured in Manhattan distance. To build this tree, new *Steiner nodes* can be added as shown in Figure 2b. To solve the RSMT problem, we use a fast look-up table based approach called *FLUTE* [3] which can determine the net's tree and its length very fast and accurately for any number of nodes. For each net of the floorplan, the RSMT and length is computed using FLUTE and then, the sum of the length of all nets is used as a cost functions for the simulated annealing algorithm.

Initial temperature: To compute the initial temperature T_0, a random B*-Tree is perturbed and, if the new tree has a worse cost than the previous one, the difference in cost, known as uphill cost, $\Delta\text{cost} = \text{cost(new)} - \text{cost(old)}$ is saved. This process is repeated over multiple randomly generate B*-Trees perturbed several times and the average uphill cost Δavg is computed. The initial temperature should be set much higher than the average uphill cost, so it is computed using $T_0 = \frac{\Delta avg}{\alpha}$, where α is a tuning parameter of the algorithm.



(a) Horizontal contour



(b) RSMT of 5 nodes

Fig. 2: (a) The horizontal contour before and after adding module b_{11} to the floorplan. (b) The Rectilinear Steiner Minimal Tree connecting 5 nodes by adding 3 Steiner nodes.

V. Integration in the FPRESSO modeling tool

We integrated our wire modeling technique into the FPGA modeling tool FPRESSO [12], which takes a description of the architecture, written in XML format, and identifies its different components and connections. Having all the components/connections identified, we convert them into a network of modules ready to be placed, where each module is associated with a node ready to be added to the B*-Tree.

The width and height of each module is required before running the algorithm, to be able to correctly place the modules and abide by the geometric relationships of the B*-Tree. But since the cluster has not been modeled yet, we don't know at this phase the area of its components. Thus, we use the average area of each component over its different sizings. This data is easily derived, since the different components with all their characteristics and sizes are stored in the library of FPRESSO. It is assumed, for simplicity, that each component has a square shape (aspect ratio of 1). Since the location of the pins of the modules is not known, we assume that a wire starts/ends at the centroid of each module. Once the cluster is floorplanned and the placement optimized, the length of the nets is computed along with their respective capacitance and resistance. Then the FPRESSO design is back annotated with this information before optimization.

Once optimized, the actual size used for each component is determined and, by that, its correct dimensions are known, so the floorplan can be corrected accordingly. As such, the cluster is floorplanned again but using the correct area derived after optimization. As in the previous iteration, the best solution is determined by the algorithm and the parasitics are again back annotated for another round of optimization in FPRESSO, before returning the final results to the user. Floorplanning the architecture again while using accurate area measures helps minimize the noise added by the modeling assumptions.

VI. Experiments

Once the simulated annealing algorithm is well tuned, by varying the different parameters of the heuristic (such as λ and M), the efficiency of the floorplanner is evaluated by testing it on a set of FPGA architectures.

A. Comparison within FPRESSO

To test our approach and evaluate the effect of wire modeling on the results of FPRESSO, we model a set of architectures in (1) FPRESSO without wire modeling and (2) FPRESSO running with our wire modeling approach. Figure 3 shows the general structure of the tested architectures.

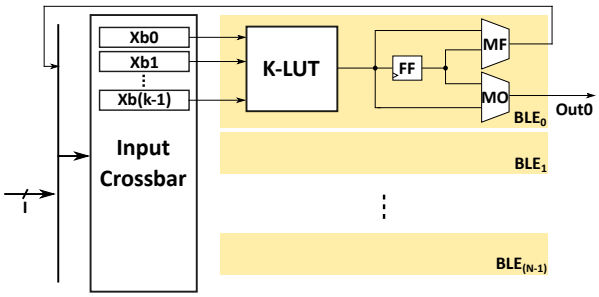


Fig. 3: A standard FPGA cluster with N BLEs, K-input LUTs, I cluster inputs and a fully populated input crossbar.

Figure 4 shows the relative delay and area of the feedback path, for different architectures, measured using our version of FPRESSO (with wire modeling), with respect to the original FPRESSO. The feedback path starts from the output of the flip-flop to its input, going though the feedback multiplexer, input crossbar and LUT. The results show that adding the wire load to the modeling process increases the delay by about 60% to 85%, depending on the size of the architecture and the optimization, while maintaining almost the same area. This increase in delay is mainly due to the wire delays and loads that are not accounted for in the original version of FPRESSO. The authors of FPRESSO discuss in their paper [12] the lack of inter-component wire modeling as a source of error and assume that the 20% difference in delay, when compared with COFFE, is due to the long wires that they are not accounting for. However, we can clearly see here, that the wire delays add generally more than 20% to the overall delay. We show in Section VI-B that COFFE is actually under-estimating the length of the wires which results in delay difference between our approach and COFFE's.

Moreover, we observe a similar behavior in results between the feedback path and direct IO path (from the inputs to the outputs of the cluster), despite a decrease in relative delay by about 25%, on average, for the IO path with respect to the feedback path. The feedback path is usually assumed to be a long wire connecting the output multiplexer to the input crossbar; however, our algorithm does not only measure the length of the wires but also floorplans the FPGA cluster in order to minimize the total wirelength. This means that the floorplanner tries to place connected components as closely as possible to reduce the delay of that connection. As such, the feedback multiplexer could be placed near the multiplexers of the input crossbar, reducing the feedback delay. Figure 5a shows the floorplan of a simplified cluster with one BLE using our floorplanning algorithm. It clearly shows how the feedback multiplexer is placed close to (1) the input crossbar multiplexers and (2) the LUT and flip-flop, reducing the feedback delay.

B. Modeling comparison with COFFE

To evaluate the correctness of our wire modeling, we compare our approach with the modeling done in COFFE, by measuring the net capacitances generated by each, for specific architectures. To have a correct comparison, we need both approaches to use the same assumptions while measuring the wirelength. COFFE has a set of predefined assumptions on the length of the wires, depending on the considered components. For example, the length of the wire connecting the LUT to the feedback multiplexer is equal to the width of the flip-flop,

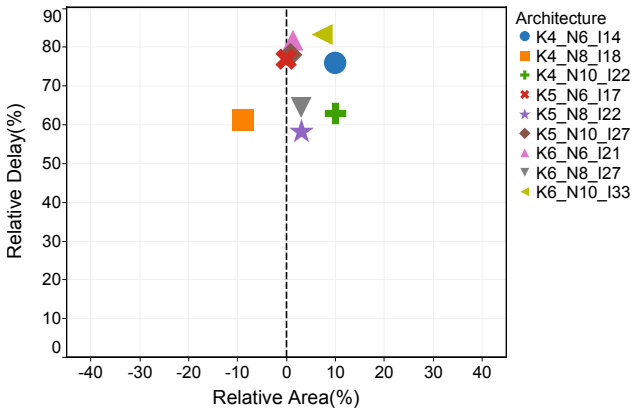


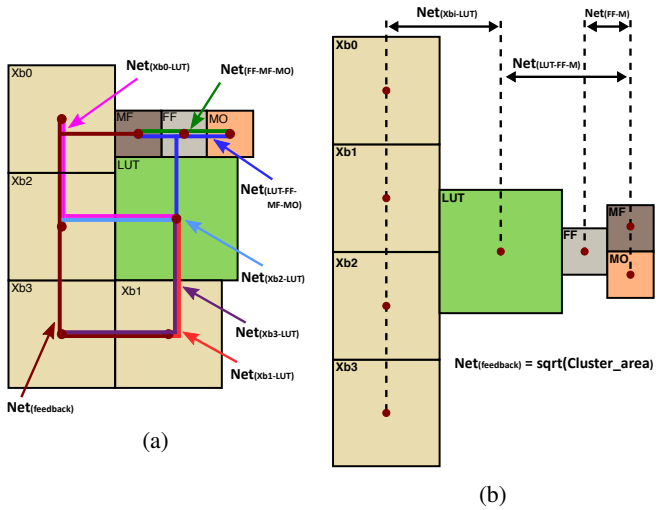
Fig. 4: Relative delay and area of the feedback path of an FPGA architecture, modeled in FPRESSO with our wire modeling approach, with respect to the original FPRESSO.

while the wire connecting an input crossbar multiplexer to the LUT input is equal to one fourth the width of the multiplexer. These assumptions cannot be applied to our floorplan since, for example, the flip-flop might not be placed between the LUT and feedback multiplexer. So, since we cannot use COFFE's assumptions, we modified COFFE to assume that the wires start and end at the centroids of the components, the way we do, and measure the wirelength accordingly. In general, our total intercomponent wirelength is longer than COFFE's by a factor of 3x to 5x, depending on the size of the architecture.

There are two main reasons behind this difference: First, COFFE assumes a fixed floorplan of the cluster irrespective of the size of the components. Since COFFE optimizes only a single representative path of the architecture, its floorplan consists of a simple linear ordering of the components as generally described in architecture files and as shown in Figure 5b for a single BLE. However, the placement generated by our floorplanner changes according to the description of the architecture and the sizes of the components. For example, floorplanning the same components of Figure 5b results in the placement shown in Figure 5a, where the modules are rearranged to minimize the total wirelength. The second reason behind this difference is the way the length of the nets is computed. We compute the wirelength using the rectilinear Steiner minimal tree and Figure 5a shows the nets used to connect the different components of the tree (not showing the input/output wires to/from the cluster). However, COFFE ignores the height of the nets, as if all the components were aligned vertically and measures the length along the x-axis (horizontally) only, as shown in Figure 5b. This results in a significant difference in wirelength, especially for architectures with bigger clusters. Therefore, it is clear that COFFE uses a fairly simplistic floorplanning to compute the wire loads and, by that, under-estimates its effect on the critical path delay.

VII. Conclusion

In this paper, we introduce a generic wire modeling method for FPGA architectures. We use a simulated annealing-based technique to floorplan the different components of the FPGA cluster while minimizing the total wirelength, and then measure the length, capacitance, and resistance of all the interconnecting wires. We successfully integrate this approach into the most recent FPGA modeling tool, FPRESSO, and show that wire loads have a substantial effect on the delays of the FPGA



(a) (b)

Fig. 5: Difference in the floorplanning and the wirelength measurement, for the architecture of Figure 3 but with only 1 BLE and a 4-LUT, using (a) our approach and (b) COFFE.

architectures. Although we do not ignore that many other factors influence the floorplanning of FPGA components (e.g., SRAM placement), we believe that this paper introduces a more realistic methodology to assess automatically and quickly the parasitics between components. This is essential for a fast and correct evaluation of novel FPGAs, in a vast search space of possible architectures.

References

[1] Y. Chang, Y. Chang, G. Wu, and S. Wu. B*-Trees: a new representation for non-slicing floorplans. In *Proceedings of the 37th ACM/IEEE Design Automation Conference*, pages 458–463, 2000.

[2] 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.

[3] C. Chu and Y. C. Wong. FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(1):70–83, Jan 2008.

[4] P.-N. Guo, C.-K. Cheng, and T. Yoshimura. An O-tree Representation of Non-slicing Floorplan and its Applications. In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, pages 268–273, 1999.

[5] F. Hwang, D. Richards, and P. Winter. *The Steiner Tree Problem*. Annals of Discrete Mathematics. Elsevier Science, 1992.

[6] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. *Science*, 220(4598):671–680, 1983.

[7] K. Padalia, R. Fung, M. Bourgeault, A. Egier, and J. Rose. Automatic transistor and physical design of FPGA tiles from an architectural specification. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 164–172, 2003.

[8] 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.

[9] C. Sechen and A. Sangiovanni-Vincentelli. The timberwolf placement and routing package. *IEEE Journal for Solid State Circuits*, SC-20:510–522, 1986.

[10] C. Sechen and A. Sangiovanni-Vincentelli. Timberwolf3.2: A new standard cell placement and global routing package. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pages 432–439, 1986.

[11] G. Wu, Y. Chang, and Y. Chang. Rectilinear block placement using B*-Trees. *ACM Trans. Design Autom. Electr. Syst.*, 8(2):188–202, 2003.

[12] 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.