Exploration of L-Shaped Wires in VTR and Implementation of node_is_wire() API

Ethan Steiner Rogers
Department of Electrical and Computer Engineering
Brigham Young University
Provo, Utah
ethanrogers529@gmail.com

***Abstract*—In the design of FPGA CAD tools, the routing resource graph provides necessary information regarding the layout of an FPGA. The position, length, and timing information for all of the wires on the device is included in the routing resource graph. VTR's RRGraph currently only supports the direct representation of wires that are strictly horizontal or vertical. To represent L-shaped or other complex wire types, a workaround must be used. This project analyzes the difficulty of modifying VTR to be more flexible and allow for complex wires to be represented directly. This project also implements an API function that allows VTR to determine if an RRGraph node is a wire without direct dependence upon the node type.**

***Index Terms*—VTR, Routing Resource Graph, L-shaped, Wires**

I. Introduction

Within the architecture of commercial FPGAs, there exist many different types of wires that provide connectivity to the device. These wires vary in size, shape, and timing parameters, with simple wires traversing only one direction and more complex wires traversing in both the X and Y direction. Several methods of representing these wires in FPGA CAD tools have been included in several open source CAD tools [1], [2].

One such CAD tool, Verilog-to-Routing [3] represents all wires as either `CHANX` wires that traverse in only the X direction or `CHANY` wires that traverse in only the Y direction. To represent wires that are L-shaped, VTR shorts together a `CHANX` and a `CHANY` wire with a non-configurable edge as can be seen in Figure 1.

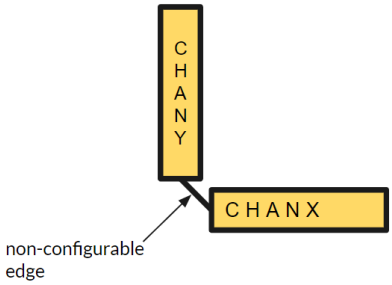

Fig. 1. L-shaped wire in VTR

Although this method is capable of representing any type of wire, there are some cons to using non-configurable edges in this manner. The first negative factor is that VTR's router must traverse every non-configurable edge in order to consider possible connections that can be made from a given wire. This additional step may reduce the performance of VTR's router. Another negative factor due to representing L-shaped wires in this manner is the resultant size of the routing resource graph. Representing wires in this manner requires two nodes and a non-configurable edge where a single more complex could be used instead.

This project aims to more fully understand the difficulty of modifying VTR's code base to support more complex wire types directly. Additionally, an API which removes some of VTR's dependence upon `CHANX` and `CHANY` node types is implemented.

II. Related Work

Before looking into implementation of complex wires in VTR, the presence of L-shaped wires in different architectures is explored. One previous work [4] explores several aspects of the Xilinx 7 Series FPGA architecture. In addition to other aspects of the architecture, the sources and destinations of different wires is determined. Figure 2 shows the structure of several different wires within the architecture.

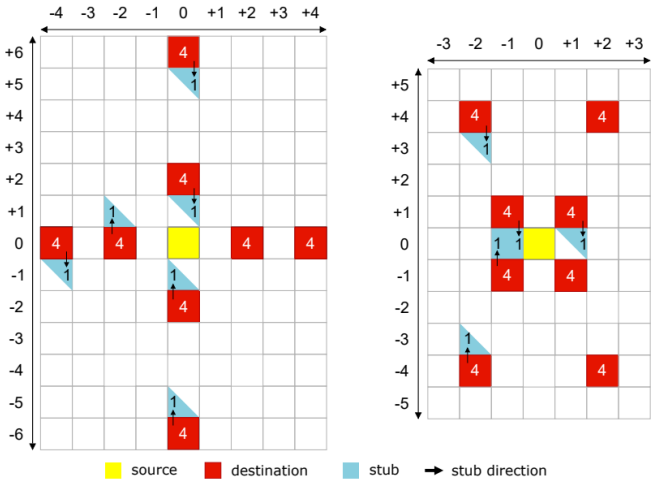

Fig. 2. Mid-range Connections

Of particular note to this project are the wires that traverse more than simply one cardinal direction. In the right side of Figure 2, for example, there are several instances of L-shaped wires. There are wires that traverse 1 unit in the X direction and 1 unit in the Y direction in addition to other wires that traverse 2 units in the X direction and 5 units in the Y direction.

III. Methodology

This project was split into two parts: (1) analyzing the RRGraphs in place within VTR and (2) determining the current reliance of VTR upon `CHANX` and `CHANY` nodes and implementing an API that begins the support of more complex wires within VTR.

A. Analysis of VTR's RRGraph

In order to determine the utility of supporting L-shaped wires within VTR, analysis was performed on an RRGraph that models Xilinx's XC7A50T device. This RRGraph was generated by the F4PGA [2] project and is among many different commercial-like architectures F4PGA is capable of modeling.

Figure 3 gives information regarding the number of shorted nodes within the `xc7a50t` RRGraph. The X axis gives the number of nodes that are shorted to the starting node. Of all the `CHANX / CHANY` nodes in this RRGraph, 24% contain shorted nodes. Of those that contain shorted nodes, 93% are a group of 2 nodes, 6.5% are a group of 3, and the largest number of shorted nodes in one group is 58. The primary reason this information is relevant is that nearly a quarter of the wires in this RRGraph contain shorts (or non-configurable edges). Were these to be replaced by L-shaped (or complex) wires, the RRGraph would require less edges and nodes and VTR's router may increase in performance.

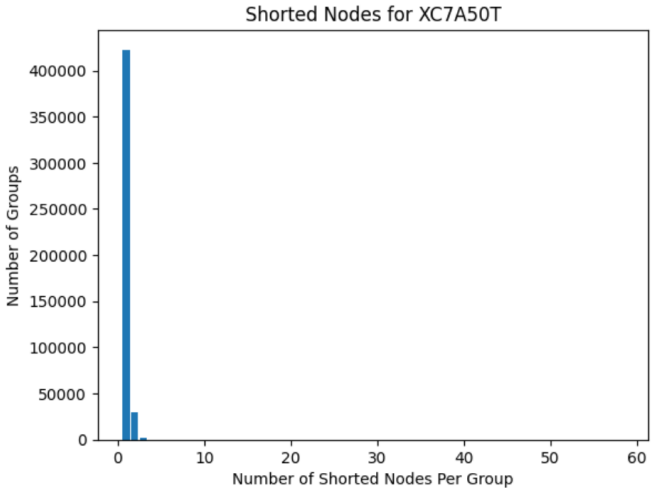

Fig. 3. Shorted Nodes

Although every configuration of shorted wires cannot be included in this paper, the following Figure 4 gives the layout of how 4 wires within VTR are shorted together.

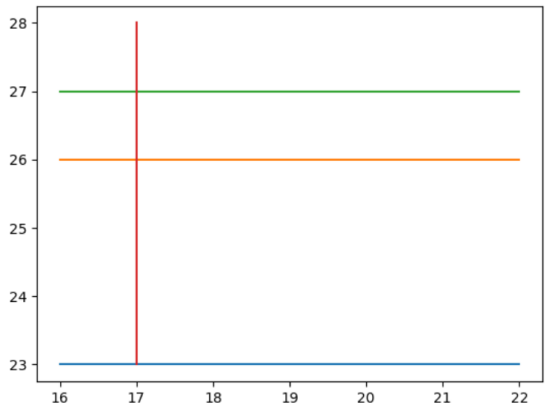

Fig. 4. An Example of Shorted Nodes

B. Implementation of node_is_wire() API

In order to determine the magnitude of VTR's reliance upon `CHANX and CHANY`, VTR's code base was parsed and every usage of these node types was analyzed for its purpose. The following list includes the reason `CHANX and CHANY` were used.

1) Set cost index accordingly
2) Check if node is a wire
3) Iterate over every tile of a wire
4) Determine how many wires are occupying each channel x/y location
5) Draw nodes and other elements in VTR's graphical interface
6) Determine placement cost along certain axis
7) Find max fanout of routing buffer
8) Determine routing usage and availability of wires
9) Check adjacent nodes
10) Lookup node
11) Build RRGraph
12) Record number of used resources in each x/y channel

In order to fully support L-shaped wires in VTR, each item in the list above would need to be able to function without direct reference to `CHANX or CHANY`. This is because L-shaped wires would not be of type `CHANX or CHANY`, but rather some other type such as `WIRE`. This section discusses the implementation of an API within VTR for item 2 in the list above.

Currently, VTR determines whether a node is a wire or not by checking if the node type is either `CHANX` or `CHANY`. Although this method works, it relies upon the presence of these 2 node types. Were an additional node type to be added that was also a wire, every instance where `CHANX` and `CHANY` were compared with would have to be augmented to include this additional node type. For example, instead of `type==CHANX || type==CHANY`, the following code would need to be used. `type==CHANX || type==CHANY || type==CHANZ`.

The implementation of `node_is_wire()` by this project removes the dependence upon `CHANX and CHANY` throughout VTR for determining if a node is a wire, and places that

dependence in one location. Thus, future changes to include other node types as wires would require changing only one function rather than VTR's entire code base. An example of one of the instances of calling `node_is_wire()` is given below.

```
- if (rr_graph.node_type(node) == CHANX ||
-     rr_graph.node_type(node) == CHANY) {
+ if (rr_graph.node_is_wire(node)) {
```

Fig. 5. node_is_wire() API

The implementation of this API throughout VTR replaced over 40 code snippets that were checking if a given node was a wire. A pull request of the changes has been made into the master branch of VTR, but has not yet been merged. This is the first of many more API functions that would need to be implemented for VTR to support L-shaped wires.

IV. Conclusion

This project has explored the possibility of supporting L-shaped and other complex wires within VTR. It has been found that L-shaped wires make up a sizable portion of the wires in Xilinx 7 Series devices. The abundance of these kinds of wires suggests that directly supporting more complex wires within VTR would be beneficial.

This project has also implemented the first of many API calls that would need to be included within VTR in order to support complex wires. This step provides a starting point should others wish to continue this work and complete the other APIs where VTR currently relies on referencing `CHANX or CHANY` node types. The addition of such support would add greater flexibility to VTR, reduce the RRGraph size, and potentially increase the performance of VTR's several FPGA CAD algorithms.

References

[1] V. Betz and J. Rose, "VPR: a New Packing, Placement and Routing Tool for FPGA Research," in Field-Programmable Logic and Applications, W. Luk, P. Y. K. Cheung, and M. Glesner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 213– 222.

[2] K. E. Murray, M. A. Elgammal, V. Betz, T. Ansell, K. Rothman, and A. Comodi, "SymbiFlow and VPR: An Open-Source Design Flow for Commercial and Novel FP- GAs," IEEE Micro, vol. 40, no. 4, pp. 49–57, July 2020.

[3] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "VTR 8: High-performance CAD and Customizable FPGA Architecture Modelling," ACM Trans. Reconfigurable Technol. Syst., vol. 13, no. 2, may 2020. [Online]. Available: https://dl.acm.org/doi/10.1145/3388617

[4] Morten B. Petersen, Stefan Nikolić, and Mirjana Stojilović. 2021. NetCracker: A Peek into the Routing Architecture of Xilinx 7-Series FPGAs. In The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '21). Association for Computing Machinery, New York, NY, USA, 11–22. DOI:https://doi.org/10.1145/3431920.3439285