

Homework # 2

Due: October 12

FPGA Placement and Routing: Understanding the
Tradeoffs

1 Introduction

Perhaps the most challenging part of implementing a new FPGA architecture is developing an appropriate set of CAD tools to map a design consisting of logic gates to a physical device implementation. While many companies have succeeded in designing FPGA devices that have an abundance of logic gates coupled with ultra-fast routing resources, only those that have developed decent CAD software support have been able to survive. In the last problem set we evaluated a set of metrics that indicated how we should apportion VLSI area in an FPGA device. In this assignment we evaluate CAD algorithm tradeoffs of algorithm run time versus performance.

We've seen that four main steps are needed to translate a circuit from a generic logic representation consisting of simple 2-input gates to a successful implementation in a physical device: technology-independent optimization, technology mapping, placement, and routing. In the following exercises we primarily focus on the latter two issues through the application of two standard algorithms, simulated annealing-based placement and maze routing, and evaluate various tradeoffs of run time versus placement and routing quality. The CAD algorithm implementations evaluated here are similar to those found in software tools for island-style devices from Xilinx Corporation and Altera Corporation.

2 Getting Started

Completion of this assignment will require the use of VPR, one of the tools used in problem set 1. If you have not already done so, I suggest reviewing the VPR User's Guide, *manual430.pdf*, which describes many VPR features in detail. Needed modifications to the source files used in the first problem set are located in the course web site in file *ps2.tar.gz*. These files should be used in place of the files originally distributed in *ps1.tar.gz* and in the original VPR distribution. Be sure to delete all VPR object files and recompile VPR from scratch once the new files have been moved into place. While you are only required to do experiments with one design **s298**, I recommend exploring results for other netlists. VPR is a powerful tool that includes a number of features not exercised in this assignment such as a timing-driven router. Please see me if you are interested in learning more about how to use some of these features.

3 FPGA Placement

In this first set of experiments you will get a chance to learn more about simulated annealing, an iterative FPGA placement technique we have talked about in class. Before starting the experiments you may wish to look at sections related to FPGA placement in [2] and review [5] to elaborate on basic FPGA placement issues and standard techniques for addressing them.

While the basic goal of simulated annealing is to locate a low-cost placement based on a pre-specified cost, a number of different parameters can be used to control the algorithm's operation. For this first set of exercises you will vary several of these and evaluate their effects on run time and resulting placement quality.

Ex 1: Simulated Annealing Parameters

In several short paragraphs, summarize Section 3 of [1] especially focussing on the discussion of annealing parameters. Specifically, how is the start temperature for simulated annealing determined? How is the temperature update scheme implemented for VPR? Note that these parameters are also discussed in [5].

For most formulations, annealing parameters include β (the inner number parameter), α (the temperature update parameter), and *StartT* (the start temperature). For the following experiments, β can be varied by changing the INNER_NUM variable of the makefile, *StartT* can be set by changing the makefile INIT_T variable and uncommenting the appropriate line under the .p makefile entry, and α can be set by using the *-alpha_t* option in the .p makefile entry. Unless otherwise specified $\beta = 1$, INIT_T is commented out in the makefile, and α is not set by the user (e.g. not set in the makefile). Collect run time and wire length information from the following placement exercises and plot all points on a graph similar to Figure 4-3 of [5].

Ex 2: Inner Number Variation

Keeping other parameters unchanged, determine placement for **s298** for $N = 1$, $I = 4$, and $\beta = 0.1, 0.5, 1$, and 5 keeping track of wire lengths and placement run times. Wire lengths can be determined by scaling the final placement cost values by 100. Run time can be estimated by using the **time** Unix command and observing on-screen time statistics after each placement finishes. For example:

```
prompt%> time make s298.p

<< run info >>

20.79u 1.45s 0:18.46 120.4%
```

indicates that the program took 20.79 seconds to complete. Be sure to save the placement found with $\beta = 1$ for use with subsequent routing experiments.

Ex 3: Start Temperature Variation

Reset the inner number back to 1 and then rerun placement with $StartT = 1, 10, 50, 100$ noting run times and resulting final wire lengths. Use the `-init.t` option in the makefile to set appropriate $StartT$ values.

Ex 4: Fixed Temperature Update Schedule

Reset the start temperature back to the default and set α to a fixed value of 0.5 and 0.8 using the `alpha.t` switch. Perform placement and note run times and resulting final wire lengths.

Plot all results on a single graph. Summarize your findings in a short paragraph.

Ex 5: Bounding Box Scaling

In class, we discussed that net bounding box sizes are sometimes adjusted during placement and routing to a value which is larger than the Manhattan distance between the most-distant net points. Why is this adjustment performed and how are the adjustments determined? (Hint: note reference [3])

4 A* Routing

For the next set of exercises, you will have the opportunity to learn more about routing for FPGAs by varying a set of parameters that control router run time. In class, we learned that *maze routing* can be thought of as an A* search that evaluates possible routing paths from a net source to net destinations. Typically, this search is performed in such a way as to minimize a prespecified cost function at intermediate points in routing paths. In VPR, the per-net intermediate cost along a path is:

$$Cost = Cost_{prev} + C_0 + \alpha(\Delta D) \quad (1)$$

where $Cost_{prev}$ is the cost of previous track segments, C_0 is the cost of the current track under evaluation, and ΔD is the Manhattan distance from the current track to the destination [3]. α is a tunable parameter which indicates which part of the cost function should be weighted most heavily. A large α value indicates that closeness to the destination is the overriding factor in selecting a new track to add to an existing route while an α value near 0 indicates that track congestion is the more important factor. More complete descriptions of this cost function and maze routing in general can be found in Section 2.2.1 of [3] and Section 2.2 of [4].

Ex 6: Varying the A* parameter, α

Perform routing of design **s298** using α values (`astar_fac` in the makefile) of 0, 1.1, 1.5, 2, 5, and 10 for a device track count of 10 (use `-route_chan_width option`) and using the placement you found above from setting β to 1. Plot a graph showing the change in run time versus α at this fixed channel width. Rerun the above experiments without the `-route_chan_width` option to determine the minimum track counts that can be achieved

for each case. In a table indicate the minimum routable track count for each α . Briefly explain these findings. (Note: some of these routing experiments can take several minutes)

Routing algorithms can be very sensitive to the order in which nets are routed. Frequently, each net is routed in an initial routing *iteration*, and then ripped-up and re-routed in subsequent iterations to eliminate routing congestion. The ordering of nets in an iteration can vary between longest-net-first, shortest-net-first, and random ordering depending upon user parameters.

Ex 7: Net Ordering

Rerun routing for **s298** with $\alpha = 1.2$ for different net orderings. The default currently is largest-first. Modify routine *load_net_order* in *vpr/route.breadth.first.c* to order nets randomly and smallest-first and perform routing with each. How do the three minimum track counts (including largest-first) compare for design **s298**? Comment on associated run times.

By now you should be quite familiar with the planar switchbox from class and the first problem set. Several class sessions ago I introduced you (through use of a two-page handout) to the Wilton switchbox, a new switchbox that has the same transistor count as the planar switchbox but with better routability.

Ex 8: Wilton Switchbox

Briefly explain why the Wilton switchbox makes maze routing easier. Modify file *vpr/4lut_sanitized.arch* to use the Wilton switchbox instead of the planar (also called subset) switchbox for routing. Rerun routing for $\alpha = 1.2$ with nets ordered largest-first in an attempt to determine the minimum routable track count. How does your result compare to that found for the planar switchbox? Can you think of any negatives regarding the use of the Wilton switchbox?

Ex 9: One-step versus Two-step Routing

In class, we discussed the difference between one-step and two-step routing. Briefly describe the differences between the two approaches and the benefits and drawbacks of each.

References

- [1] V. Betz and J. Rose. VPR: A New Packing, Placement, and Routing Tool for FPGA Research. In *Proceedings, Field Programmable Logic, Seventh International Workshop*, Oxford, UK, Sept. 1997.
- [2] S. Hauck and A. Agarwal. Software Technologies for Reconfigurable Systems. *submitted to IEEE Computer*, 1997.
- [3] J. Swartz, V. Betz, and J. Rose. A Fast Routability-Driven Router for FPGAs. In *6th International Workshop on Field-Programmable Gate Arrays*, Monterey, Ca, Feb. 1998.

- [4] R. Tessier. Negotiated A* Routing for FPGAs. In *Proceedings: Fifth Canadian Workshop on Field-Programmable Devices*, Montreal, Quebec, June 1998.
- [5] R. Tessier. *Fast Place and Route Approaches for FPGAs*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1999. Chapter 4.