University of Pennsylvania
Department of Electrical and Systems Engineering
Electronic Design Automation

ESE535, Spring 2008 — Assignment #4 — Monday, March 3

Due: Monday, March 24th, beginning of class.

Resources You are free to use any books, articles, notes, or papers as references. Provide citations in your writeup as appropriate.

Collaboration Please work independently on this assignment. You may discuss general algorithmic strategies for the coding exercise (problem 2) and help each other with the compiler, build environment, and debugging, but each student should develop his or her own solution. If you do discuss strategy or getting debugging help, please acknowledge it in your writeup.

Writeup Writeup should be in an electronically readable format (HTML or PDF preferred—I do not want to decipher handwriting or hand-drawn figures). State any assumptions you need to make.

Problems

1. Considering the delay model in `cplace` described in the previous assignment:
   (a) What is the 2D delay cost function you are optimizing?
   (b) Why is this cost function a bad driver for incremental optimization?
   (c) List at least 4 cost functions that might drive an incremental optimizer to reduce circuit delay. Discuss the pros and cons of each cost function.
   (d) List at least 4 different techniques for approaching placement. Considering quality of results, algorithmic runtime, and implementation complexity, list pros and cons for each technique.
   (e) What lower bounds can you place on the achievable delay for an optimally placed netlist?

2. Develop a placement routine for `cplace` which minimizes circuit delay. For this assignment your goal is to do better than the default sequential placement and you will not be asked to optimize the IOs/cluster (*i.e.*, for K-input LUTs, we will allow $K \times N_{luts/cluster}$ inputs per cluster). In the next assignment we will add the constraint that you deal with restricted inputs **and** expect a more aggressive optimization aimed at beating `vpr4.3` placements. Turnin:
   (a) Your code (tar file ready to unpack and make like the one provided for you on assignment 3)
   (b) Writeup for your placer, including:
      i. Summary of your results on the provided benchmark set. (table with columns: benchmark, sequential place delay, your delay, % improvement, you placement runtime in seconds; a final row should summarize the geometric mean of your improvements across the benchmark set. You may have multiple absolute delay/% improvement/runtime columns if you have results under different options worth reporting.
      ii. Explanation of your placement routine
      iii. Explanation of any tuning, benchmarking, and profiling you performed to arrive at your current solution. Where appropriate show quantitative graphs or tables.
      iv. Documentation for running your version of `cplace` including description of all new options you have added to `cplace`.
   (c) List six techniques or variations which you might be able to use to improve upon your results.

Benchmarks

Pickup the benchmark directory in `assign4_bench.tar` from `~ese535/spring2008/assign4_bench.tar` on `eniac`. Unpack it as a subdirectory of your `cplace` directory by running `tar -xvf assign4_bench.tar` in your `cplace` directory. The `makefile` provided here will run the benchmark set for the sequential place option. You will need to modify the options given to `cplace` in the `makefile` so that it calls your placement routine properly.

Timing

The `makefile` uses `/usr/bin/time` to compute the complete runtime of an invocation of `cplace`. It is possible to do finer-grained timing inside the C code, but I expect the runtime of your placement routine to dominate all the time for other things (*e.g.* reading the netlist, packing the flip flops, checking the legality, computing the final circuit delay).

Note that the remote login `eniac` machines have a limit on how long a single job can run (20 minutes). If you need to run your optimizer for longer time, you will need to run it on some other machine (*e.g.* visit lab and run on a machine directly, your own desktop or laptop).

Placement Interface Routines

`mesh.c`/`mesh.h` contain the code for dealing with physical placement. The main two routines you may need are:

- `place_block` – take an unplaced block and place it into a particular mesh location
- `move_block` – take a placed block and move it to a different mesh location

These two routines take care of all the lower-level details of linking/unlinking blocks into slots, so you probably do not want to directly access the `mesh_position` data structure. These routines do check:

- If the position is legal for the block type—producing a fatal error with an error message if the position is not legal.

They do not check:

- If you have exceeded the number of allowed LUTs (or IOs) per mesh position.
- If you have exceeded the limit on the inputs to a logic mesh position.

In some approaches, you may allow these limits to be violated during optimization, using your cost function to ultimately drive them to be corrected in the course of the optimization. The `count_placement_violations` routine will check the entire array for placement violations and report them. Until this returns zero, the placement is not legal.

The provided placement routines (`seqplace.c`, `oneplace.c`) give examples of using `place_block` and generally working with the mesh. Option 3 in `cplace` performs a swap using `simple_move_test` to demonstrate the use of `move_block`.

The mesh structure is currently only accessible from within `mesh.c`. As part of your cost calculations, it may be necessary for you to add code to `mesh.c`.

Delay Recording

To ease the collection of data, you might want to print the critical path delay to a file so you can write scripts to automatically collect critical path delays and runtimes.