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

ESE535, Spring 2008 Assignment #3 Monday, February 18

Due: Monday, March 3rd, 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 exercises (problem 1) 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.

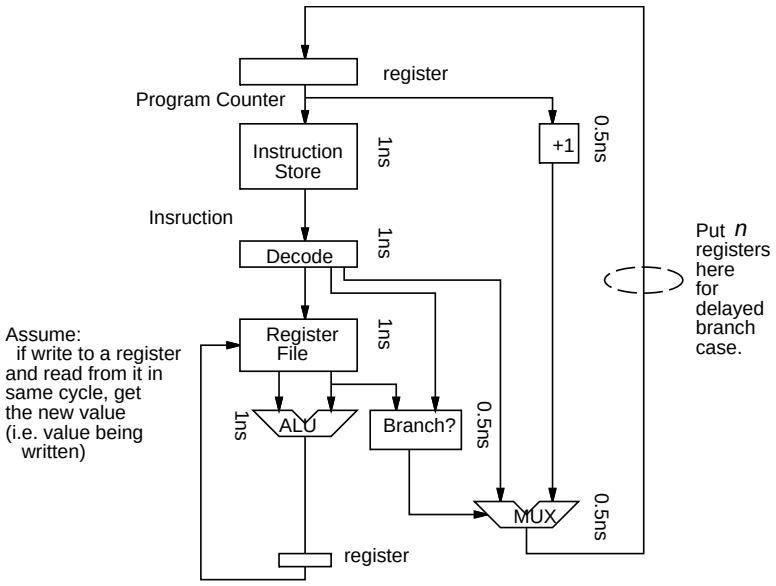
Writeup Writeup should be in an electronically readable format (HTML or PDF preferred—I do not want to decipher handwriting or hand-drawn figures). State any assumptions you need to make.

Problems

1. Add support to the provided CPLACE infrastructure (described at end) to compute the delay of a netlist and report the slack at each input/output/gate node (simply compute the topological delay—you do not need to worry about the function of each logic gate, and hence, do not need to eliminate possible false paths). Run the code and report the delay for each netlist in the provided benchmark set. Turnin:

   - Your code (no binary files, but in an archive like the provided support so it can be unpacked and built; this means the make file should be updated to build the application with your additions)
   - Description of your algorithm and the code the supports it
   - Table summarizing the delays for all netlists in the provided benchmark set.

2. Consider the following, simple microcontroller datapath:



(a) What is the delay of the datapath as shown?

Consider changing the branch instruction to a delayed branch instruction that transfers control n cycles after the branch is issued. This is equivalent to putting n registers on the path as shown.

(b) Using the retiming procedure from class:

i. How does the clock cycle change with n?

ii. Above what value of n does further increase of n result in no further decrease in cycle time? Why? (Show the G-1/C graph and use that in your explanation.)

iii. For this maximum useful value of n, show the retimed circuit which achieves the minimum cycle time.

(c) Starting from the retimed circuit with the maximum useful branch delay, n. C-slow the circuit so that the cycle time of the C-slowed circuit is equal to the delay of the ALU.

i. What is the value of C necessary to achieve this?

ii. Show the circuit after C-slowing and retiming the circuit. Color each of the C different register sets differently similar to the example in the Day 8 lecture.

[Note: This is a deliberately simple datapath. While processors did go through a period where they used delayed branches, that was ultimately considered a poor idea for technology-abstract ISA scalability. Modern processors use many more techniques to deal with this cyclic dependence, but some do run interleaved streams using a similar idea to C-slowing. For more details see CIS501.]

3. Consider 1D placement (*e.g.* of a datapath or a standard-cell row). You have a netlist graph $G = (V, E)$. You want to assign each node ($v_i \in V$) to a unique location $j \in [0..(|V| - 1)]$.

   (a) What is the ideal cost function you wish to minimize in order to:

      i. Minimize energy (assume the energy of each v_i is proportional to the (sum of length of the wire to the most distant nodes to the left and right of v_i) and (the activity factor of v_i)).

      ii. Minimize delay of the netlist (assume each v_i can be evaluated in time T_v, and the time it takes to route a signal from location i to location j is $|j - i| \times T_{int}$).

      iii. Minimize area of the netlist (assume all cells are the same width and height is linear in channel width)

   Cost function provided for each case should be for the full assignment.

   (b) Considering the algorithms recently discussed in class (KLFM partitioning, Hall's Spectral 1D placement, maxflow-based partitioning), for each of the three targets above:

   - Select a base approach.
   - Describe how you would adapt/modify/use the selected approach to optimize for the target.
   - Identify how the selected approach trends the design toward minimizing the ideal cost function.
   - Describe or show scenarios where the optimization does not perfectly track the ideal cost function (or argue why the approach does optimize the ideal cost function).

Initial `cplace` Project Details

For the project this term, we will cluster and place a netlist to minimize delay. Concretely, we will consider placement for an island-style FPGA. Interconnect for the island-style FPGA is a mesh. Each mesh-location (island or CLB) can hold a number (*e.g.* 4) of individual Lookup-Tables. Similarly, each periphery position around the central logic mesh can hold a number of inputs or outputs. The delay of each LUT is a constant (T_{block_delay}). The delay for each interconnect link depends on the locations of the source and sink LUT. If the source and sink are in the same block, the delay is short (T_{intra_block}). If the source and sink are in different blocks, the delay is the sum of a fixed delay for leaving the block ($T_{inter\_block\_initial}$) and a delay proportional to the Manhattan distance between the source and sink ($(|src_x - dst_x| + |src_y - dst_y|) \times T_{hop}$). Since your task is only to place the IOs and LUTs, you cannot change the total LUT delay on each path. Your placement will define the interconnect delays. In a typical FPGA, the majority of the delay is in the interconnect.

An academic package that solves this problem is t-vpack/vpr from the University of Toronto [2, 1, 3]. This package solves the problem in two separate phases: (1) clustering (`t-vpack`) and (2) placement (`vpr`). We will look at solving clustering and placement simultaneously. We will use code from the t-vpack/vpr distribution as a basis for our work (reading the initial netlist, representing the netlist in C, writing out the final cluster and placement). Using this code base, we avoid having to rewrite these I/O and representation routines, allowing us to focus on the optimization. This also allow us to run out final clusters and placements through VPR (which also performs routing) for a head-to-head comparison at the end.

Please look at parts of the VPR manual (available in `~ese535/spring2008/manual_430.pdf`) for descriptions of the architecture and placement coordinate system. Particularly Figure 2 shows what the basic module of a LUT and FF looks like. Figure 10 shows the coordinate system. The manual also defines the netlist format. Since we provide code to read and write this formats, you do not have to implement it, but you will likely find it useful for debugging to be able to look at these files and make sense of them.

In future weeks you will select an approach and implement code to place the LUTs in an attempt to minimize this delay. As a warmup exercise this week, you will write a routine to compute the delay of a given placement and the slack of each node.

We are providing an infrastructure in C which we call `cplace`. In addition to providing the base t-vpack/vpr netlist capabilities, we are also providing a representation for the mesh and islands and some initial (dumb) placements. The initial placements give us something to work with this week while we're developing the delay calculator. They also give you an easy target against which to show improvement as you first start developing your own placement optimization in following weeks.

Pickup the code in `cplace.tar` from `~ese535/spring2008/cplace.tar` on `eniac`. Unpack it with `tar -xvf cplace.tar`. Run `make` to build. This should produce an executable `cplace` which you can run. `cplace` with no arguments will produce a usage message. The `makefile` in the `test` subdirectory runs `cplace` on three test cases and provides an example of how to use it. Please use the architecture and delay parameters in the `Makefile` for producing your results for this assignment. At the moment there are only three example

input files, one without flip flops and two with flip flops. We will update `cplace.tar` to include additional test cases in the next few days.

For this assignment, we provide you a stub for the delay calculation in the file `compute_delay.c`. Currently the routines does nothing. You should complete the routines, including the routine `print_slack` which prints the slack of the nodes. **Caveat:** blocks of type `LATCH` or `LUT_AND_LATCH` have flip flops which will both start and end paths. As such, you may need to exercise some care to deal separately with the delays of these nodes based on whether you are looking at them as inputs to the flip-flop or outputs from the flip-flop. Particularly, if you think about the slack of these flop nodes, they effectively have two slacks to deal with—one for the path which starts at the flop and one for the path which ends at the flop.

A quick overview of code in `cplace`:

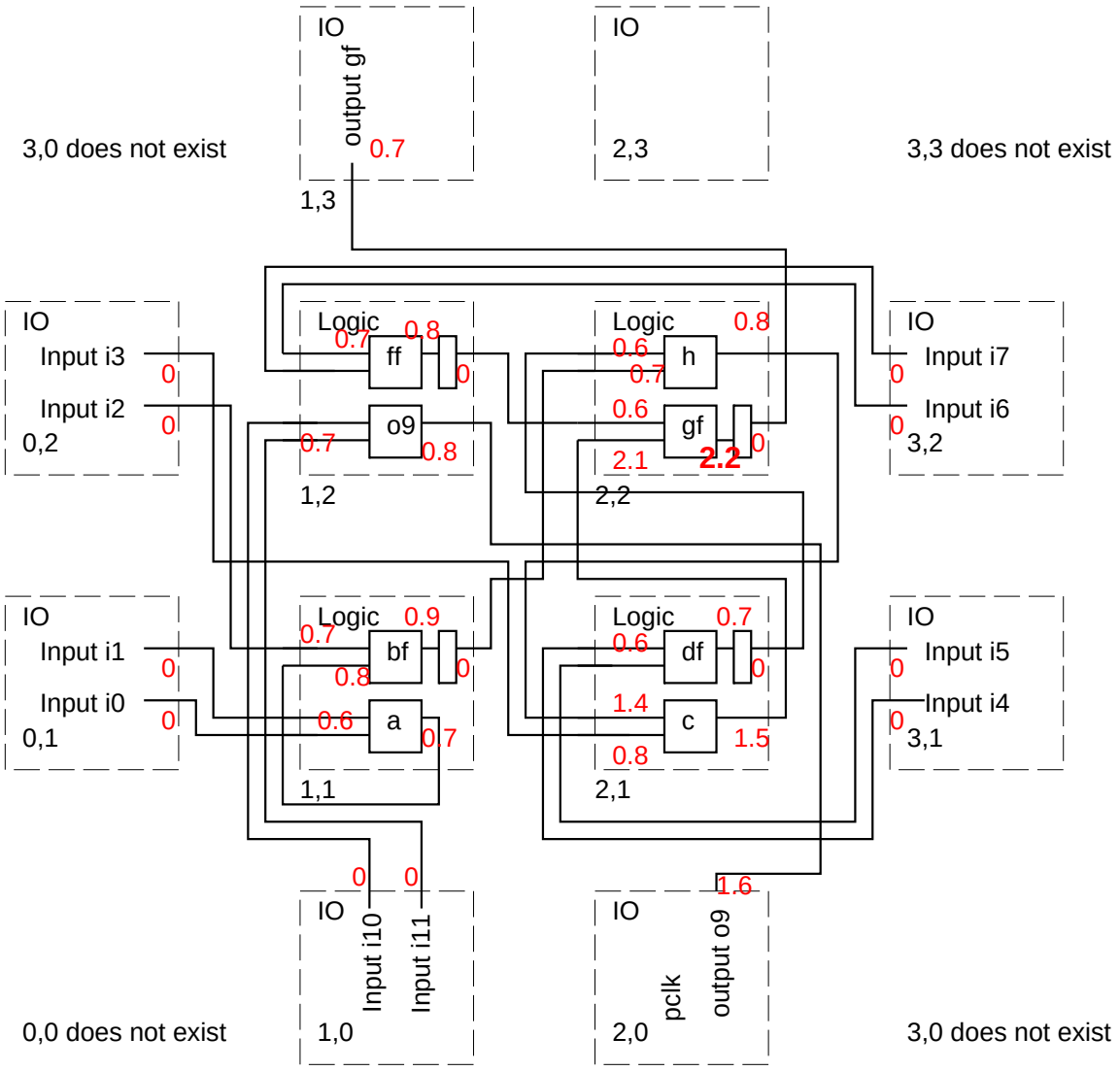
- `cplace.c` — contains the `main` function which drives the overall optimizer; it also contains the command-line option parsing. You will most likely **not** need to modify it this week, but you will need to in subsequent weeks.
- `globals.h` — defines global data structures: notably the `block` and `net` datastructure which represent the netlist.
- `cplace.h` — defines the type structure for `block` and `net`.
- `mesh.c`, `mesh.h` — routines for working with the physical mesh, including placing and moving blocks, legality checks, and printing out the placement and clustering. The high-level place/unplace/move operations may be the only ones you need to use.
- `output_clustering.c` — prints out the cluster. You should not need to touch.
- `read_blif.c` — prints out the cluster. You should not need to touch.
- `ff_pack.c` — packs LUTs and FFs. You should not need to touch.
- `heapsort.c` — a sort implementation. You should not need to touch. You may find it useful to use this.
- `util.c` — various utilities. You should not need to touch. You may or may not want to use some of these utilities.
- `onespot_place.c`, `sequential_place.c` — these are our **dumb** placement routines. They may be useful examples for you on how to work with the netlist and mesh once you start performing optimizations in subsequent weeks. `onespot_place.c` places all the LUT blocks in a single island and all the IO in a single IO location; this placement does not obey the limit on the number of items per island. `sequential_place.c` places blocks and IOs sequentially. It is careful not to exceed the logic capacity of each island, however, it does not check the input capacity.

We'll say more about `cplace.c` and `mesh.c` in the writeup for the next assignment as you need to work with them.

Caveat: The code not borrowed from t-vpack/vpr (mesh code, cplace, dumb placement routines) was newly written for this assignment. While we have tried to test it, like any recently developed code it may contain bugs. Let us know if you have any problems. Similarly, we may need to provide updated source as we fix bugs or add additional functionality.

[2/18/08] Note that the `num_nets` on the `s_block` includes the clock. So, when you are dealing with a `LATCH` or `LUT_AND_LATCH`, the actual data inputs are [1..`num_nets`-2].

Following is the placement generated by `sequential_place` for the sample design `example` annotated with delays assuming ($T_{inter\_block\_initial}$=0.5, T_{block_delay}=0.1, T_{intra_block}=0.1, T_{hop}=0.1)



References

[1] Vaughn Betz. VPR and T-VPack: Versatile Packing, Placement and Routing for FPGAs. `http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html`, March 27 1999. Version 4.30.

[2] Vaughn Betz and Jonathan Rose. VPR: A new packing, placement, and routing tool for FPGA research. In Wayne Luk, Peter Y. K. Cheung, and Manfred Glesner, editors, *Proceedings of the International Conference on Field-Programmable Logic and Applications*, number 1304 in LNCS, pages 213–222. Springer, August 1997.

[3] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts, 02061 USA, 1999.