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

ESE535, Spring 2013 — Assignment #3 and #4 — Monday, February 11

Due: Assign 3: Wednesday, February 20, beginning of class.
Due: Assign 4: Wednesday, February 27, 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 You may **not** discuss algorithmic and testing approaches. You may give tutorial assistance on using OS, compiler, and debugging tools. All code development should be done independently. You may **not** share code or show each other code solutions. All writeups must be the work of the individual.

Writeup Turn-in assignments on blackboard. See details on course web page. No handwriting or hand-drawn figures. See details below on what you need to turn in and the format.

Project Goal for this phase Locally assign logical LUTs to partially defective physical LUTs to maximize yield.

Opportunity An FPGA cluster in a modern, island-style FPGA contains multiple physical LUTs (See Figure 1) [1]. We can freely reassign logical LUTs to physical LUTs within the cluster.

Assignment 3 Task Assign logical LUTs to partially defective physical LUTs locally within a cluster to avoid defects. For this, you will use clusters created in a fairly dumb manner.

Assignment 4 Task Generate LUT clusters to maximize the probability that all LUTs within the cluster can be mapped to a suitable partially defective physical LUT in the cluster. Note that a legal cluster will obey two constraints:

- The number of LUTs assigned to the cluster will not exceed a specified cluster size.
- The total number of distinct inputs to the cluster will not exceed a specified number of cluster inputs.

You should try to minimize the number of clusters produced. The default packer (sequential_place) is not very high quality, so you should certainly use fewer cluster than it does. We had to give it an extra row and column (`-pside 1`) for it to fit. You should try to fit into the chip without the extra column (`-pside 0`).

You may want to read [2]. Your cost function from Assignment 2 is likely to be useful for both pieces.

We expect Assignment 3 to be somewhat lightweight (less than a week of work), and Assignment 4 to be heavier. So, it would be good to start thinking about Assignment 4 before the Assignment 3 due date.

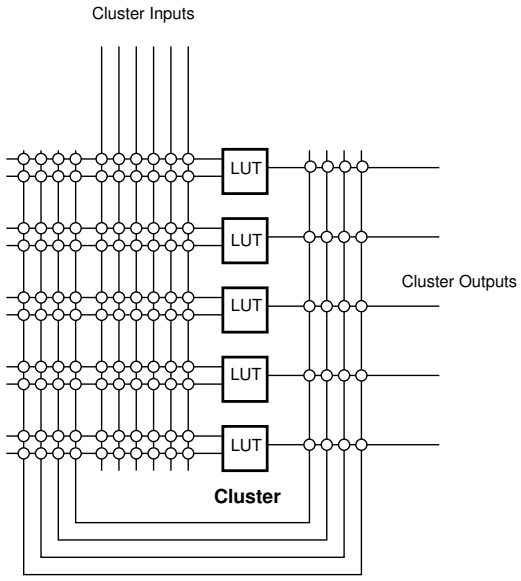

Figure 1: FPGA Cluster with 6 inputs containing five 2-LUTs

Assignment 4 Warmup Questions:

1. If you were packing into clusters that could contain four 4-LUTs and had 8 functions, 4 of which were xor4's and 4 of which were and4's, what would be the least defect tolerant and most defect tolerant ways to create 2 clusters of 4 LUTs?

2. Assume you are packing into clusters with N physical LUTs. A legal, but not very dense, packing would place one logical LUT into each cluster.
   (a) How many more clusters will this need than a dense packing of LUTs into clusters?
   (b) What would the impact of this mapping be on defect tolerance?

3. Based on the example above, what should your goal be when packing LUTs into clusters to maximize defect tolerance?

4. How should this goal be reflected in a cost function? How can your Assignment 2 cost function contribute?

Code Base: We extend the same basic code base we started with on Assignment 2.

Pickup the code in `assign34.tar` from `~ese535/spring2013/assign34.tar` on `eniac`. Unpack it with `tar -xvf assign34.tar`. Run `make` to build. This should produce an executable `dpack` which you can run. The `makefile` in the `test` subdirectory runs `dpack` on the various cases needed for this assignment and provides an example of how to use it. Please use the architecture and target parameters in the `makefile` for producing your results for this assignment.

Representations: For this lab we add a representation for defects in particular chips. You can see these in the test/chips subdirectory. Each line starts with the x and y position of a cluster in an array of LUT clusters and is followed by the defects for each of the LUTs in the cluster. Our primary tests will be on clusters of size N=4, so a sample line looks like:

```
22 1 0 1024 0 0
```

This says the cluster at position (x=22,y=1) has 3 non-defective LUTs, and one LUT (the one in the second slot, which will be indexed 1) with a single defect in mux 10. `mesh.c` has been extended to read in these defect files and store the defects in the mesh representation.

You may want to consult the VPR manual (available in `~ese535/spring2013/manual_430.pdf`) for descriptions of the mesh 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.

We are **not** really doing placement for this assignment (we'll get to that in Assignment 5). So, the task being performed by the default `sequential_place` and your Assignment 4 packer at this point is just the packing of LUTs into a cluster. We use the mesh structure here as a convenience to collect the cluster then later associate a set of defects with the cluster. For now, the only significance of the (x,y) positions is that different positions represent different clusters.

A quick overview of code:

- `dpack.c` — contains the new `main` function that drives the generation of clusters. You will use this driver for both Assignments 3 and 4. There are now two different case statements operating on approach inputs. The first is a matching approach (for assignment 3) and the second is a packing approach (for assignment 4). These are selected by the `-approach` (for packing) and `-match` (for matching) options. The makefile in the test directory is setup to provide the options. The cryptic file name `.mXpY` (.m0p0, .m1p0, .m2p0, .m2p1) correspond to these options.
- `test_chip.c` – reads in test chips, actually calls your matching functions, and collects and writes results.
- `sequential_place.c` – the dummy packing routine. It is greedy and inefficient both in runtime and function. It does **not** look at the LUT functions when making its decision—so, you should certainly be able to do better by looking at the LUT functions. It also isn't very clever about dealing with IOs to create clusters. Nonetheless, it illustrates how you can use place_block to put something into a cluster.

- `mesh.c` – contains the mesh representation and functions for operating on the mesh. You definitely want to understand the mesh_position structure (the mesh is an array of these) that holds the slots and defects. There is also checking code here that is called by `dpack` to validate if your packing is legal. That will be an important sanity check for you in Assignment 4.

You need to complete code in:

- `match.c` – your_match – this is where your Assignment 3 answer goes. Your code should return a permutation for assigning logical LUTs to physical LUTs (with associated defects) in the cluster.
- `your_pack.c` – your_pack – this is where your Assignment 4 answer goes. In this case you will be assigning the logical LUTs to clusters by putting them in some (x,y) mesh position.

You will also need your `transform.c` and `cost.c` from Assignment 2.

Note that the packing routine (your_pack, Assignment 4) is called **before** any chip defect maps are loaded. The packer should try to create LUT clusters that are good for **any** chip. The matching (your_match, Assignment 3) is called for a specific cluster on a specific chip with its defects and logic, so it should identify the best way to match logic with partially defective LUTs in the cluster for the specific defects.

The .mXpY files contain the summary of defect mapping attempt. The lines report the file used for the defect map and the number of clusters that were incompatible with the defects. A count of 0 indicates a successful yield, anything larger than 0 indicates failure (a -1 is reported if it cannot read the file). The final line gives the yield; this is the number you should report in the comparison tables in your writeup. With the 50 chip benchmark set provided, it will be a number between 0 and 50, with larger numbers meaning more successfully mapped (yielded) chips.

Caveat: The code not borrowed from t-vpack/vpr (`test_chip`, `mesh`, `sequential_place`) was newly written or heavily revised 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.

Assignment 3 turnin: You will need to upload two files. We have created separate assignments on blackboard so that you only need to submit a single file to each assignment

1. **assign3-writeup**: a single PDF with
   - Pseudocode for your LUT assignment algorithm
   - Explanation of your LUT assignment algorithm
   - Table of results reporting the number of chips successfully mapped (bottom line, yield number from .mXpY file) for each benchmark three cases: (1) sequential_place with no LUT input permutations or inversions and no LUT mapping (m0p0), (2) sequential_place with your LUT input permutations and inversions and no LUT

mapping (m1p0), (3) sequential_place with your LUT input permutations and inversions and your LUT mapping (m2p0).

2. **assign3-code**: a single tar file with your code (no binary files, but in an archive like the provided support so it can be unpacked and built)
   - run `make clean` in both the code and test directories
   - use `make assign34.tar` to create the tar file
   - test that you can unpack your `assign34.tar` and build and run tests from there before you upload to blackboard; we will build your code and test it.

Assignment 4 turnin: You will need to upload two files. We have created separate assignments on blackboard so that you only need to submit a single file to each assignment

- **assign4-writeup**: a single PDF with
  1. Answers to the assignment 4 warmup questions
  2. Pseudocode for your clustering algorithm
  3. Explanation of your clustering algorithm
  4. Extend the table described for assignment 3 to include: (4) number of chips successfully mapped for dpack clusters with your LUT input permutations and inversions and your LUT mapping (m2p1), (5) the number of clusters produced by dpack, (6) number of clusters produced by sequential_place.
- **assign4-code**: a single tar file with your code (no binary files, but in an archive like the provided support so it can be unpacked and built – same as above)

References

[1] V. Betz and J. Rose, "How much logic should go in an FPGA logic block?" *IEEE Design and Test of Computers*, vol. 15, no. 1, pp. 10–15, 1998.

[2] A. Marquardt, V. Betz, and J. Rose, "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, February 1999.