

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

ESE535, Spring 2015

Assignment #3–6

Wednesday, January 28

Due: Assign 3: Thursday, February 5, 10PM

Due: Assign 4: Thursday, February 12, 10PM

Due: Assign 5: Thursday, February 19, 10PM

Due: Assign 6: Thursday, February 26, 10PM

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.

We expect everyone to abide by Penn's Code of Academic Integrity. http://www.upenn.edu/academicintegrity/ai_codeofacademicintegrity.html If there is any uncertainty, please ask.

Writeup Turn-in assignments on canvas. 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 Use partitioning to assign blocks (LUTs, Inputs, Outputs) to PEs to minimize energy costs. We will particularly use the cost functions calculated in assignment 2 to metric cost. Ultimately, this means by the end of assignment 6, your code will have performed full recursive bisection of the input design attempting to minimize both the total growth schedule and the maximum, per-level growth schedule.

This task is broken into 4 milestones to help pace your work.

Suggested milestone goals:

Assign 3 single bipartition only concerned about total nets cut

Assign 4 single bipartition also minimizing max level cut

Assign 5 recursive bipartition based on total nets cut per tree

Assign 6 recursive bipartition based on all costs, total nets cut and max level cut

The actual turnin requirements will be slightly less aggressive than this to give you flexibility in when you do it and to allow you more time to get initial code fully debugged and running.

Assign 3 data structures and algorithm plan for total net cut optimization, including plan for recursion; you are not required to have running code or results for this first week.

Assign 4 plan for how will accommodate level max cut optimization; running code and results for single bipartition targetting only total nets cut

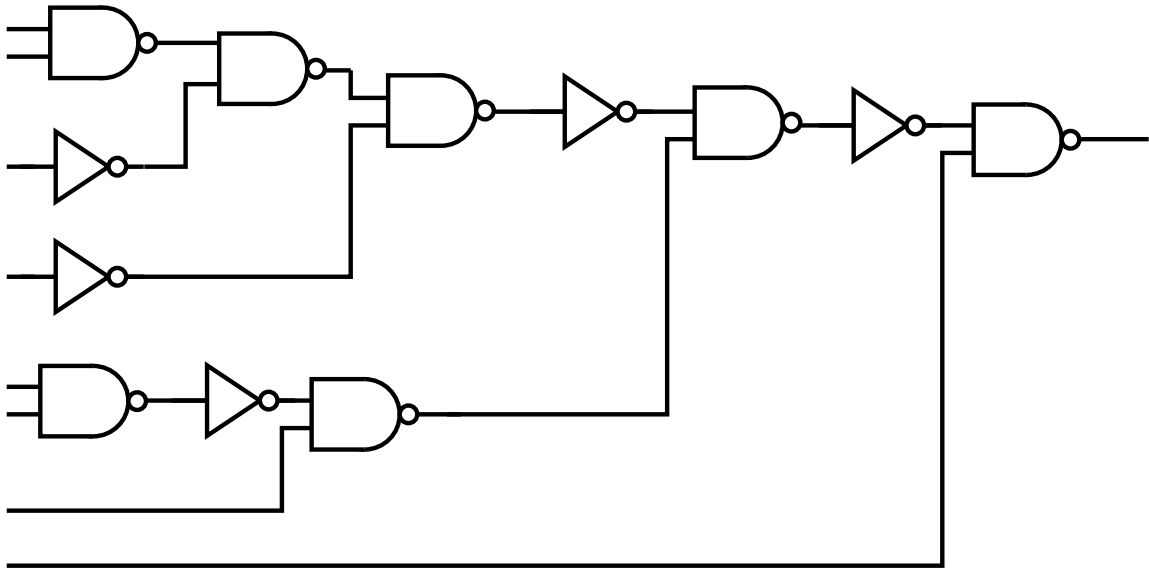
Assign 5 running code, results, and writeup for recursive bipartition based on total nets cut per tree.






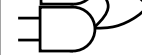
Assign 6 full solution with all results.

Points for these assignment will be weighted toward the later assignments, with assignment 6 weighted more than assignment 3. You should incorporate feedback from earlier writeups in your later ones.

There will be a few non-code questions to go with each assignment. The Assignment 3 questions are here, 4-6 will come out weekly as we get to them.

1. Use the dynamic-programming covering algorithm from lecture and reading to cover the tree from lecture (repeated below) for **minimum delay**. Use the delays given below.
 - (a) Show the resulting cover (circle the collection of gates in the tree for each library gate in the final cover and annotate the circle with the assigned library gate).
 - (b) Report the delay and area for this cover.
 - (c) Compare to the delay and area for the area-minimizing cover (from class).



Gate						
Delay	1	2	3	4	4	4
Area	2	3	4	5	4	5

2. How do we change the cost function for the dynamic-programming covering algorithm to minimize energy?

Here, we will assume the dominant energy is CV^2 switching energy, and the dominant capacitance is assignable as gate input capacitance. Consequently, we assign each gate input a number to represent its capacitance. Each net, net_i , is further assigned a switching probability, $P_{switch}(net_i)$, between 0 and 1, representing the fraction of cycles on which the node toggles. The total energy of a mapped circuit is thus:

$$E_{circuit} \propto \sum_{\text{all gate inputs}} (P_{switch}(\text{input}) \times C_{input})$$

Note that we hold V constant for the mapping, so while we need to know V to compute the absolute energy, it simply becomes a proportionality constant and we can omit it in our cost model for minimization. For covering, assume the input netlist is already annotated with the switching probability of each net, net_i . ($P_{switch}(\text{input}) = P_{switch}(net_i)$ for the net_i which is the input to the gate). Particularly, gate inputs in the input netlist that are hidden inside a mapped gate during covering do **not** contribute to the energy for the mapped circuit.

- (a) State cost function for computing the energy of a tree cover (analogous with the ones in class for computing the area or delay of a tree cover).

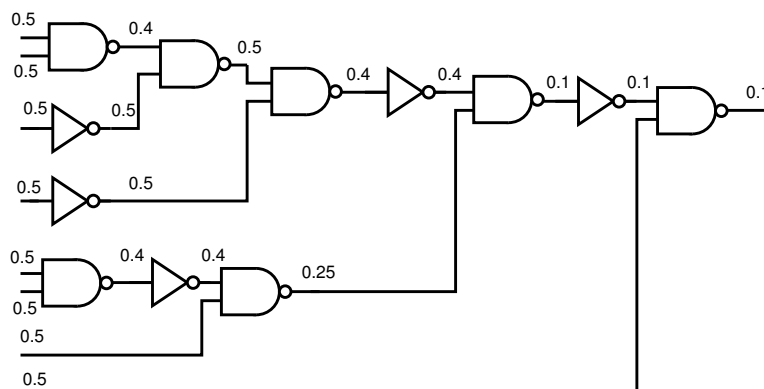
For reference, we state the area and delay cost functions as:

$$A(\text{tree}) = A(\text{gate}) + \sum_{\text{input trees } i} (A(\text{tree}_i)) \quad (1)$$

$$D(\text{tree}) = D(\text{gate}) + \max_{\text{input trees } i} (D(\text{tree}_i)) \quad (2)$$

Use $C_{in}(\text{gate}, j)$ to identify the capacitance on the j -th input to a gate.

- (b) Assuming all gate inputs have unit capacitance, C_1 , and the nets toggle with the probabilities shown, identify the energy minimizing cover for the example tree circuit below (same as previous problem).
- (c) Compare the area and delay of this cover to the area-minimizing and delay-minimizing covers (area, delay as shown on previous page with Problem 1)



Code notes and instructions:

- Code base is only slightly changed from assign2. Pickup updates in `~ese535/spring2015/assign3.tar` on eniac.
- The benchmark set has expanded from assign2. We've added more and larger benchmarks.
- We have added an empty `part_place.c` where you will place your partitioning code. The routine `part_place` is called from `main` with arguments to distinguish the various goals and steps in this assignment.
- We have revised `tree.c` to create slots and domains for switches. Previously, these were only allocated within PEs. The idea is that you can use these at the switch level to keep track of the nodes assigned to each subtree during partitioning.
 - added `insert_block_switch` to place a block at a switch. This allows you to place blocks into the switch slots.
- [\[1/30/14\]](#) Added code to `part_place` that puts all the blocks that need to be partitioned at the root switch to illustrate use of `insert_block_switch` and the general setup of using switch slots to keep track of blocks in a subtree; also added dummy code to do a dumb partition and put blocks on left and right switches from root to illustrate and to setup for calculating the top cut.
- Note that you can associate information for blocks (or nets) using an array like level (`asap.h,c`). For example, you might use an array index by block to keep track of whether the block is assigned to the left partition or right partition (or is outside of the current subtree) during partitioning. You might use another array to keep track of the current gain (KLFM, Day4 lecture) on a block.
- [\[1/29/14\]](#) Fix bugs in counting in `domain` .
- [\[1/30/14\]](#) Clocks are no longer placed and routed. This means changes to `main`, `check_route`, `global_route`, `dummy_tree_place` and `tree`.
- [\[1/30/14\]](#) Added `calculate_and_write_top_cut` to report out single cut for assignment 4.
- For recursive partitions, the cut size (number of parents) out of the left and right subtrees may be different. It may be useful to track that directly as part of your partitioning cost. **TODO:** provide an example.

Assignment 3 turnin: a single PDF with

- Summary of the data structures you will use for partitioning
- Pseudocode for your partitioning algorithm, including the plan for performing the partitioning recursively down to the PE leaves
- Explanation of your partitioning algorithm and data structures
- Answers to the two covering problems.

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

1. **assign4-writeup:** a single PDF with

- Answers to assignment 4 problems (problems to be provided separately).
- Explanation of your final partitioning algorithm and data structures; this should be stand-alone from assignment 3. It's likely your algorithm and data structures have changed somewhat from assignment 3.
- Description of how you will modify your partitioning algorithm to minimize the maximum level cut, include a description of the new or refined data structures required.
- Table of results for the provided benchmark set reporting the single-partition cut size when using (a) the existing dummy partition and (b) your bipartitioner.

2. **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)

- run `make clean` in both the code and test directories
- use `make assign3456.tar` to create the tar file
- test that you can unpack your `assign3456.tar` and build and run tests from there before you upload to blackboard; we will build your code and test it.

Assignment 5 turnin: You will need to upload two files. We have created separate assignments on canvas so that you only need to submit a single file to each assignment

1. **assign5-writeup:** a single PDF with
 - Answers to assignment 5 problems (problems to be provided separately).
 - Explanation of your final partitioning algorithm and data structures; this should be stand-alone from previous assignments. It's likely your algorithm and data structures have changed somewhat from previous assignments. If there are no changes or need for revision, it can be a copy of what you turned in before.
 - Table of results for the provided benchmark set reporting the results of the assignment 2 cost functions (interconnect cost, context factor) when using (a) the existing dummy partition and (b) your recursive bipartitioner.
2. **assign5-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 assign3456.tar` to create the tar file
 - test that you can unpack your `assign3456.tar` and build and run tests from there before you upload to blackboard; we will build your code and test it.

Assignment 6 turnin: You will need to upload two files. We have created separate assignments on canvas so that you only need to submit a single file to each assignment

1. **assign6-writeup:** a single PDF with
 - Answers to assignment 6 problems (problems to be provided separately).
 - Explanation of your final partitioning algorithm and data structures to handle full recursive partitioning and both total cut and maximum level cut; this should be stand-alone from previous assignments. It's likely your algorithm and data structures have changed somewhat from previous assignments. If there are no changes or need for revision, it can be a copy of what you turned in before, but should address any feedback you received on previous writeups.
 - Table of results for the provided benchmark set reporting the results of the assignment 2 cost functions (interconnect cost, context factor) when using (a) the existing dummy partition (b) your recursive bipartitioner targetting only the total wires cut at each partition, (c) your level-max-aware recursive bipartitioner.
2. **assign6-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 assign3456.tar` to create the tar file
 - test that you can unpack your `assign3456.tar` and build and run tests from there before you upload to blackboard; we will build your code and test it.