

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

ESE535, Spring 2011

Assignment #1

Monday, January 24

Due: Monday, January 31, beginning of class.

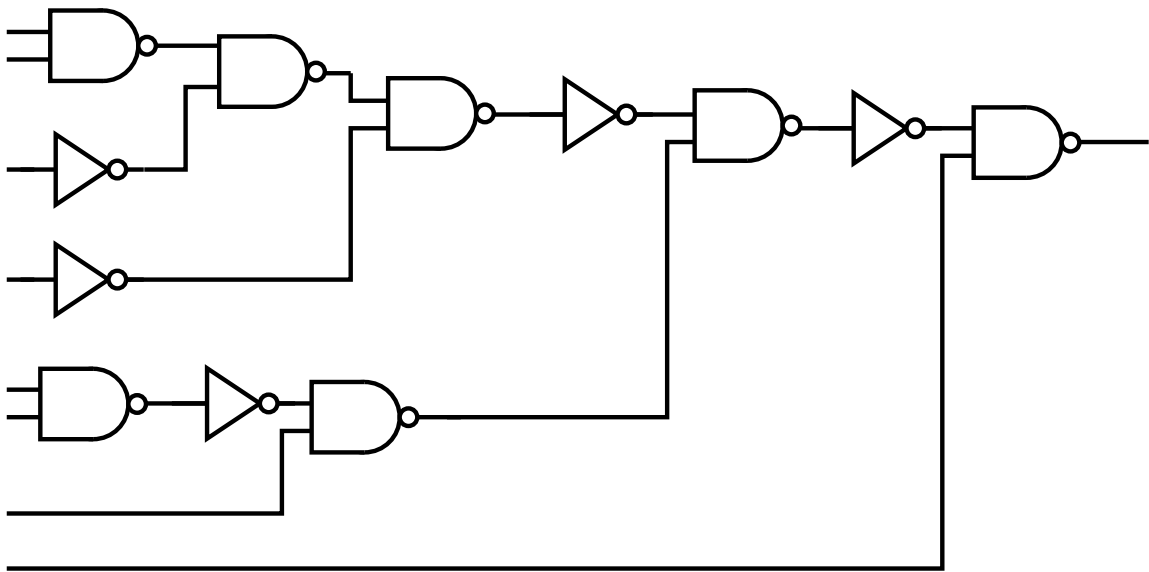
Resources You are free to use any books, articles, notes, or papers as references. Provide citations in your writeup as appropriate.







Collaboration Please work independently for the written problems 1–3. You may help each other get tools working for problem 4, but make sure you will be able to use the tools on your own for future assignments.

Writeup Turn-in assignments on blackboard (PDF preferred). See details on course web page. No handwriting or hand-drawn figures. State any assumptions you need to make.

Problems

1. [3pt] 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. [3pt] 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).

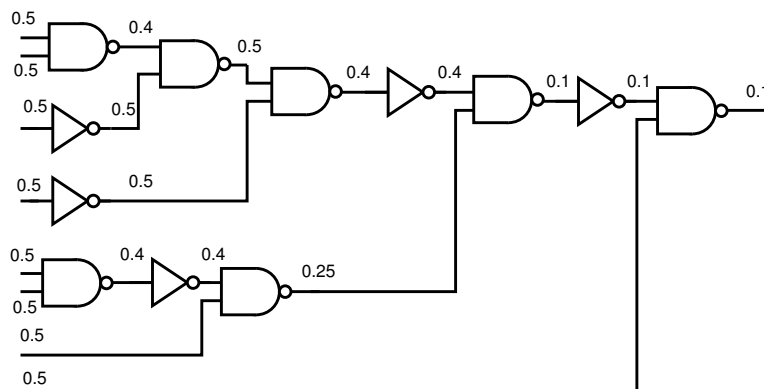
For reference, we state the area and delay cost functions as:

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

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

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)



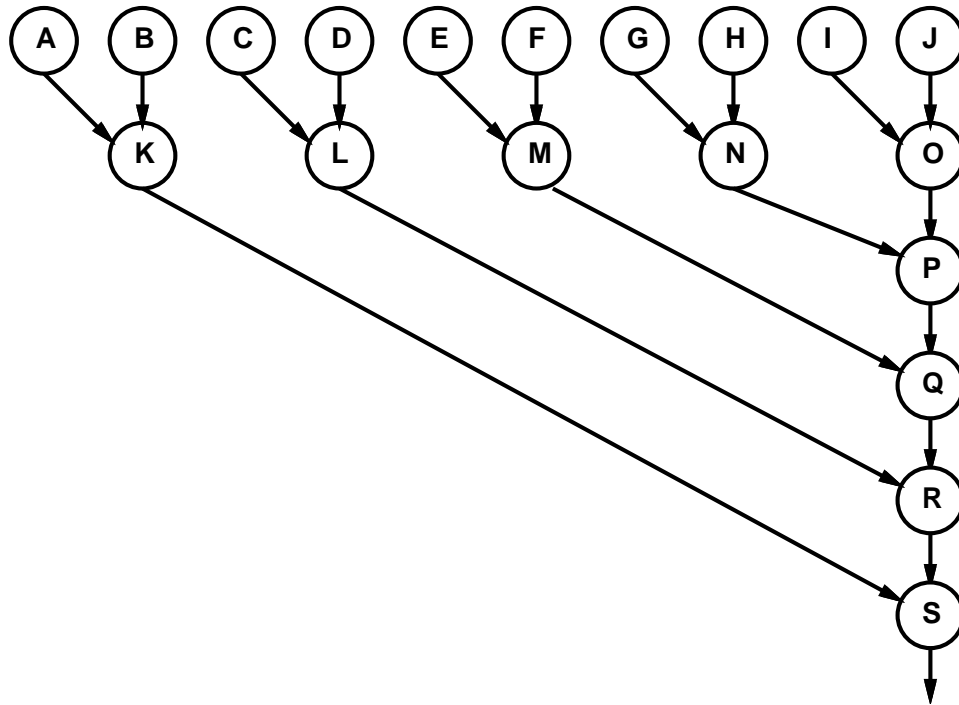


Figure 1: Sample Graph to Schedule

3. [3pt] Consider the graph in Figure 1. Assuming this is scheduled onto a VLIW datapath with 5 functional units, each of which takes one cycle to perform an operation:
- What is the latency bound? (critical path bound in class)
 - What is the compute bound? (resource bound in class)
 - Show the schedule that results from using the algorithm shown in Figure 2 to schedule the graph ($F = 5$, $T_{op} = 1$). (*N.b.* This algorithm is not necessarily good; in fact we have seen better in class. This is a setup for your comparison in the following part.)
 - Identify a better schedule for the graph (perhaps using an algorithm from class). Compare the makespan of the schedules.

```

Input: Directed Dataflow Graph  $G = (V, E)$ ,
       Number of functional units  $F$ ,
       Cycles per Operation  $T_{op}$ 

//Unscheduled is a set that holds all unscheduled nodes.
Unscheduled=Empty Set
// ReadyQueue holds nodes that are ready to execute.
// Data extracted from Queue is in strict First-In-First-Out (FIFO) order.
ReadyQueue=Empty Queue
// Cycle is an integer indicating the cycle we are scheduling
Cycle=0
// Mark all nodes as unscheduled.
for each  $v_i \in V$ 
    Unscheduled.Add( $v_i$ )
    // Initialize the ReadyQueue with nodes which depend on nothing.
    if  $v_i$  has no predecessors in  $G$ 
        ReadyQueue.Add( $v_i$ )
// Assign nodes to functional units and time slots.
While (not(Empty(Unscheduled)))
    fu=0; // fu is a counter to keep track of assigned functional units
    Running=Empty Set // nodes scheduled at Cycle
    // Start as many nodes running as possible.
    while ((fu <  $F$ ) and (not(Empty(ReadyQueue))))
        tmpNode=ReadyQueue.removeOldest()
        Assign tmpNode to Functional Unit fu at cycle Cycle
        Running.add(tmpNode)
    // Advance Cycle to completion time of these jobs.
    Cycle=Cycle+ $T_{op}$ 
    // Mark nodes as schedule.
    for each node  $n_i \in$  Running
        Unscheduled.Remove( $n_i$ )
    // Put all nodes enabled by the nodes which just completed into the ReadyQueue
    for each node  $n_i \in$  Running
        for each successor  $s_i$  to node  $n_i$ 
            if no predecessors to  $s_i \in$  Unscheduled
                ReadyQueue.Add( $s_i$ )

```

Figure 2: Simple First-Come-First-Served Scheduling Algorithm

4. [1pt] Download the assignment 1 version of the scheduling framework, build it, and run the benchmarks given. This task is entirely to make sure you have your tools gathered for the programming assignments that will start with assignment 2—so, you work out any tool issues this week and are ready to focus on programming next week.
- (a) Create a new directory for this build: `mkdir assign1`
 - (b) Copy or download `assign1.tar` from `/home1/e/e535/spring2011`
 - (c) In your new, use tar to unpack: `tar -xvf assign1.tar`
 - (d) Use `make` to build: `make`
 - (e) Change to the test subdirectory and use `make` to run tests: `cd test; make`
 - (f) A successful run will produce `.place` and `.sched` files for each of the three benchmarks.
 - (g) For each of the three benchmarks in the test directory, report: array geometry for mapped design and final makespan (largest timestep used). You can find the array geometry in the `.place` file (second line) and the final timestep at the end of the `.sched` file.
 - (h) In your writeup, summarize the develop environment you built this in and what you needed to do to get it to work.
 - (i) If you collaborated with others to get this working, acknowledge in your writeup.

It should be possible to perform these operations exactly on eniac linux cluster machines. On your personal laptop or other machines, you may need to change tools (*e.g.* C Compiler using the `CC` variable in the makefile) or paths. You are welcome to use different development environments (*e.g.* eclipse, VisualStudios...), but figuring out how to make it work in that framework is up to you. I was able to build this on my Mac; in previous years, I've used Cygwin on Windows.