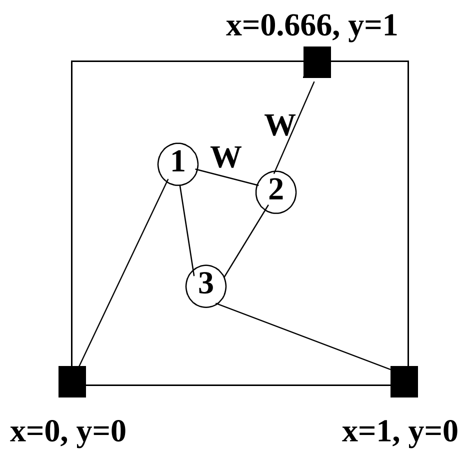
CMU Fall 2001 18-760 VLSI CAD

[125 pts] HW 5. Out: Tue Nov 27, Due: Thu. Dec 11 ‘01, in class. (V1)

1. Quadratic Placement [25 pts]

Consider this simple netlist with fixed pins, which has 3 placeable objects. All the 2-point wires have Cij=1 *except* the two wires labeled “W” in the figure:



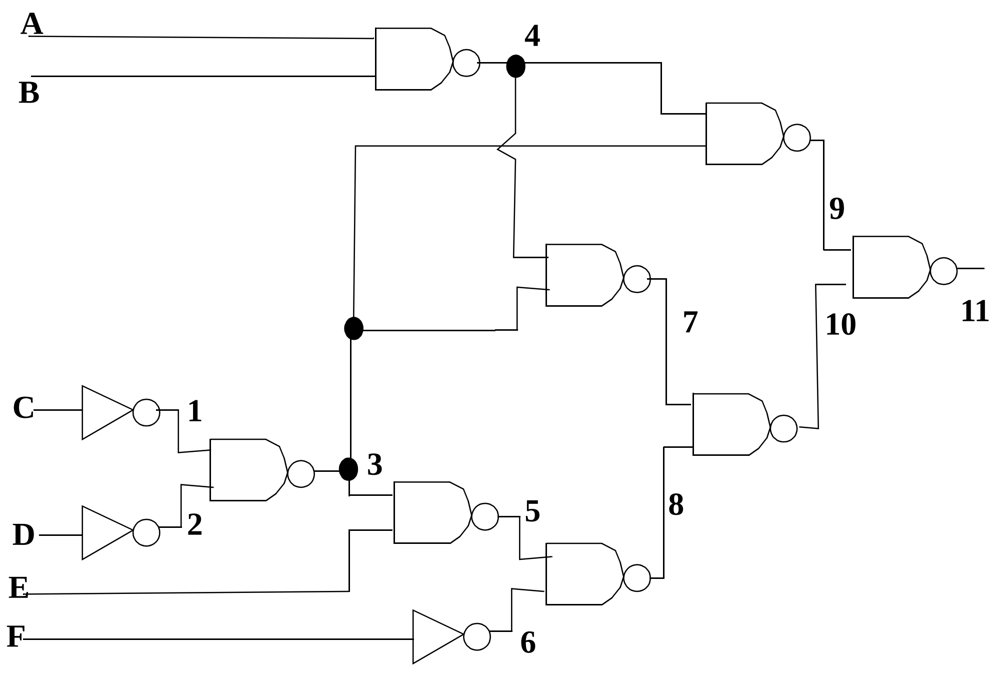
Do this:

- Assume **W=1** so all wires have unit weight, and show how to formulate and solve the quadratic placement problem as in the class notes. Show the [Cij] matrix, the [A] matrix, the two b vectors (one for solving the x problem, one for the y problem). Solve the two resulting 3x3 matrix problems (however you want to do it) to get a placement. Plot the placement as in class (you can do it by hand, or you can be fancy and use a program; either is OK).
- Now, assume **W=10**, and repeat the above placement exercise, again showing all the matrices, vectors, and final placement.

2. Static Timing Analysis: Computation [25 pts]

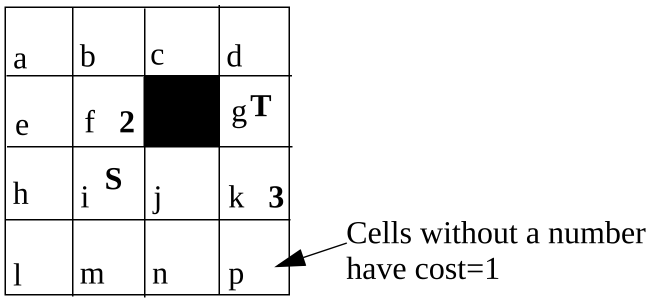
Consider the simple circuit below. Do the following:

- Assume each **gate** has delay equal to the number of its inputs, i.e., the inverters have delay=**1**, the NANDs have delay=**2**. Ignore the wires for delay. Draw the full delay graph with one source and one sink node.
- What the fastest cycle time we could use if this was the logic we had between latch stages in our design? Show how you computed this.
- Using this cycle time number, show the Arrival Time (**AT**(node)) and Required Arrival Time (**RAT**(node)) times for each node in your graph. Use the algorithms from the class notes: you can do the topological sorts by eye, then use these to generate the AT and RAT numbers. Don't worry about "early mode" and "late mode" delays; just use the single delay per gate (delay = the number of gate inputs). Given the AT and RAT numbers, show the slacks at each node too.
- Is the **longest** path statically sensitizable? Explain why or why not.



3. Basic Maze-Routing Algorithms [25 pts]

Consider the simple 2-point maze-routing problem shown below. All the unblocked cells are white, and for convenience are labeled with a single letter in the lower left corner so we can refer to them later. Assume cells have unit cost unless they are labeled with a number in the lower right corner (e.g., cells f, j). Black cells are blockages.



Answer these questions:

- Suppose we use the plain vanilla maze routing scheme (no predictor function) to route from the source S cell to the target T cell in this grid. Assume that when a cell is removed from the wavefront (expanded) and used to reach its neighbors, you look at (i.e., reach) the neighbor cells in this compass order: North, East, South, West. Assume also that reached cells added to the wavefront with the *same* cost are stored in a queue: first in is first out. Redraw the grid twice, and in one drawing label the cells in the order they are *reached* (added to the wavefront); in the other drawing label the order in which cells are *expanded*. Also show the final path and tell its cost.

  To be clear here: you start with just cell **i** on the wavefront with cost=1. You expand **i**, and you reach first **f**, and then **j**, and then **m**, then **h**, in order. **f** has cost=3, all others have cost=2. These cells go on the wavefront in order **f, j, m, h**. Given the queue structure of the wavefront, the next cell to expand is **j**, since it's the first cell of cost=2 on the wavefront. Continue from here...

- Suppose instead that we use a depth-first predictor scheme? Assume the same Rubin scheme as discussed in class. Assume the order for reaching neighbors is the same order as above. Recall: each cell now has a pathcost, and a distance-to-target (estimated) cost. Answer the same questions for the last part for this new search scheme.

  Again, to be clear. Cell **i** is the first one on the wavefront with pathcost=1 and estimated distance-to-target = 3. So, cell **i** is on the wavefront with total cost = 4. You reach the neighbor cells in this order: **f, j, m, h**. **f** has pathcost =3 and distance-to-target =2, so **f** goes on the wavefront at total cost = 5. **j** has pathcost=2, and distance-to-target = 2, so **j** goes on the wavefront at total cost = 4. Similarly, **m** and **h** end up on the wavefront each at a total cost of 2+4=6. Continue from here...

4. Elmore Delay for a Parameterized Clock Tree [25 pts]

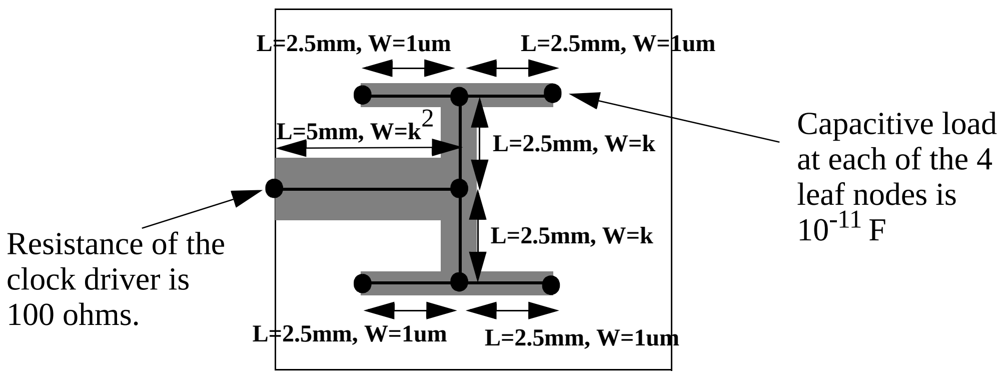
Consider the symmetric clock tree layout shown below. It has 7 segments with fixed length, with 3 of these having variable width. It has 4 leaf nodes. Now assume that the resistance and capacitance can be calculated as:

- $R = r \cdot L/W$, where $r = 50 \times 10^{-3}$
- $C = c \cdot L \cdot W$, where $c = 0.2 \times 10^{-15}$
- Assume that L, W must be in units of **microns** for the above formulas to work right. Note that some lengths in the picture are in **millimeters**. Remember to convert!

Suppose we want to make the delay as **small** as possible here. One simple way is a *mono-tone* widening scheme: the bottom-most wires are all width=1um, and each level up the clock tree has wires **k** times wider than the level below. There are 7 segments in this clock, but only one designable variable now: **k**. Do this:

- What is the root to leaf delay of this clock if we just set **k**=1?
- How fast can we make this clock run, ie, what is the minimum Elmore delay τ that we can achieve if we can set **k** to *any* positive number? (**k** does not have to be an integer).
- How fast can we make this clock if we limit the **maximize width** of the widest wire to 5 um?
- Suppose we also have a **power** limit. To first order, the power in the clock is CV^2f, where C is the total capacitance of the entire clock (segments + leaf nodes), V is the voltage of the clock signal (let's call it 1V to make life easy), and f is the frequency of operation (Hz) of the clock. Let's assume we can approximate f as $1/\tau$. How **big** can we make **f** if we have a total limit of **2 milliWatts** dissipated in the clock, i.e., $CV^2f \le 0.002$?

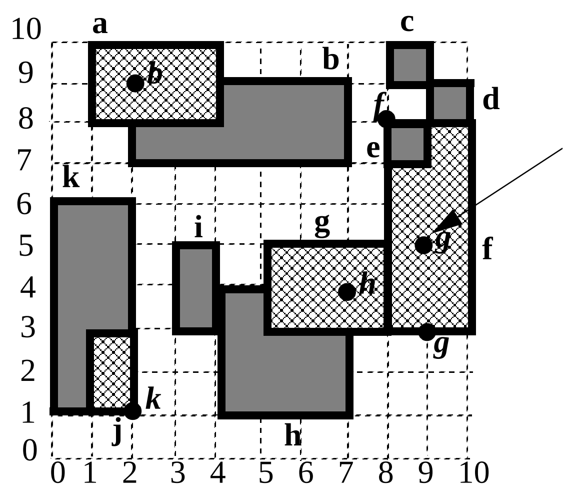
(**Hint**: Turn each segment into a pi model as a function of **k**, write delay as a function of **k**, and total capacitance as a function of **k**. You can solve nonlinear equations graphically by plotting for various values of the variable)



L=2.5mm, W=1um
L=2.5mm, W=1um
L=5mm, W=k²
L=2.5mm, W=k
L=2.5mm, W=k
L=2.5mm, W=1um
L=2.5mm, W=1um
Capacitive load at each of the 4 leaf nodes is 10⁻¹¹ F
Resistance of the clock driver is 100 ohms.

5. Geometric Data Structures [25 pts]

Consider this simple layout, which has 11 rectangles (labelled a-k) on 2 layers: the dark grey layer, and the lighter (it's actually striped) layer:



Dots with letters show the corners of rectangles that are hidden under another rectangle in this picture, just to avoid any ambiguity

Do this:

- Draw the **quadtree** that represents this layout. You don't need to try to draw the actual rectangle in each node of the quadtree, just show the quadtree's overall structure (quadrants, cut lines, hierarchy) and what rectangles (by letter) live in each part of the tree. Explain any assumptions you need to make.
- Draw the **maximally horizontal corner stitched tileplane** that represents this layout. Assume we want **one single tileplane**, so that each tile will be a unique combination of layers. So, we will have space tiles, dark-grey tiles, striped tiles, and dark-grey-AND-striped tiles. Label things clearly so we can tell what each tile is. Draw the grid coordinates as above so we can see where the edges of each tile fall. Remember the canonical form for the tileplane: solid tiles are maximally wide and then maximally high; solid tiles cannot have "same-color" tiles at their extreme left/right ends. In this case, it's OK to have another solid tile with a different "color", ie, a different set of solid layers inside a horizontally neighboring tile. Explain any assumptions you need to make.