Problem Formulation and Algorithm Application in Computer-aided Design

Prof. Kurt Keutzer

EECS

University of California

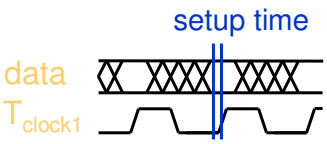
Berkeley, CA

Critical Path Delay – Setup Time

Delay is a function of

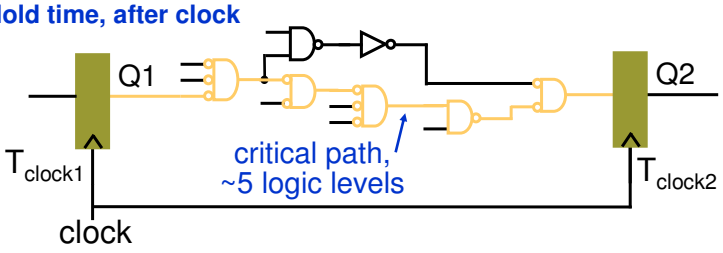
Total gate, wire delays

- **measure of delay between registers**
  - **logic levels**



Data stable during

- **Setup time, before clock**
- **Hold time, after clock**



Critical Path Delay – Hold Time

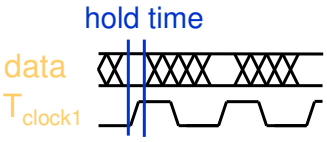
Delay is a function of

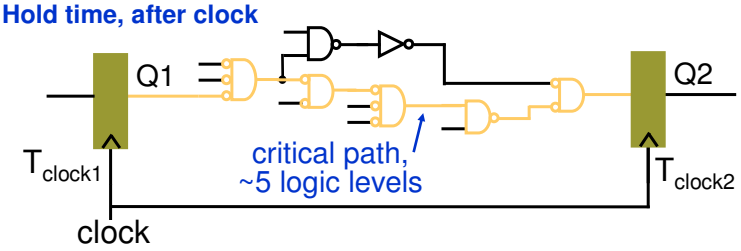
Total gate, wire delays

- measure of delay between registers
  - logic levels

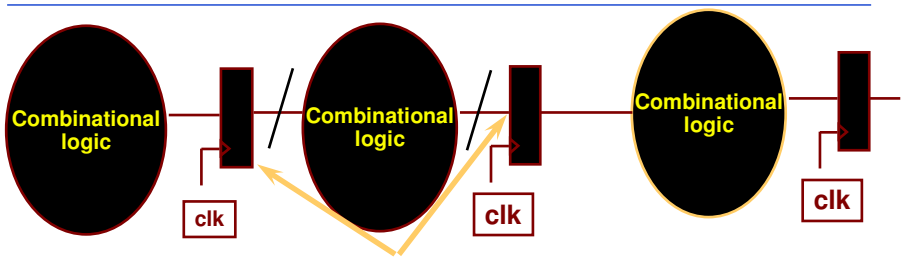
Data stable during

- Setup time, before clock
- Hold time, after clock





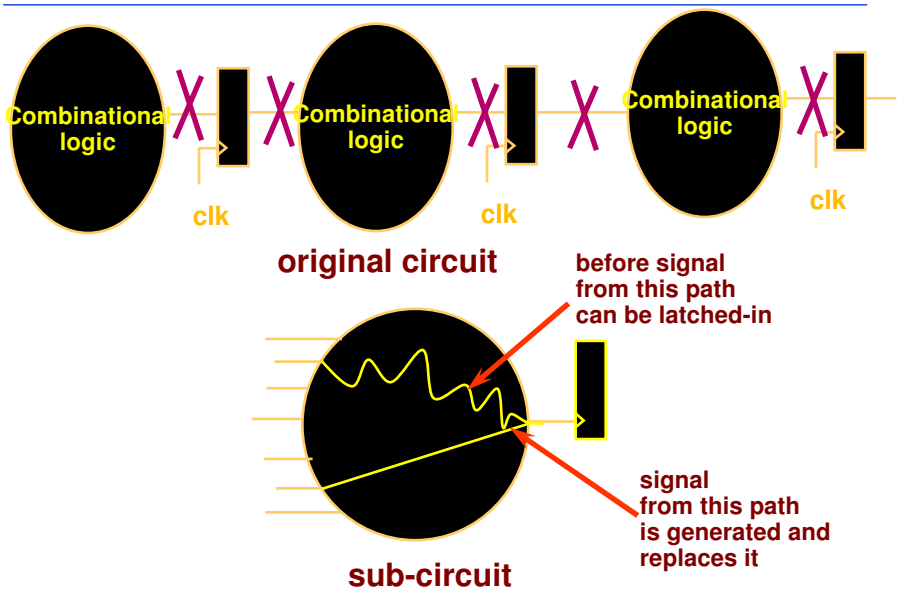
Problem 1: Hold-time check



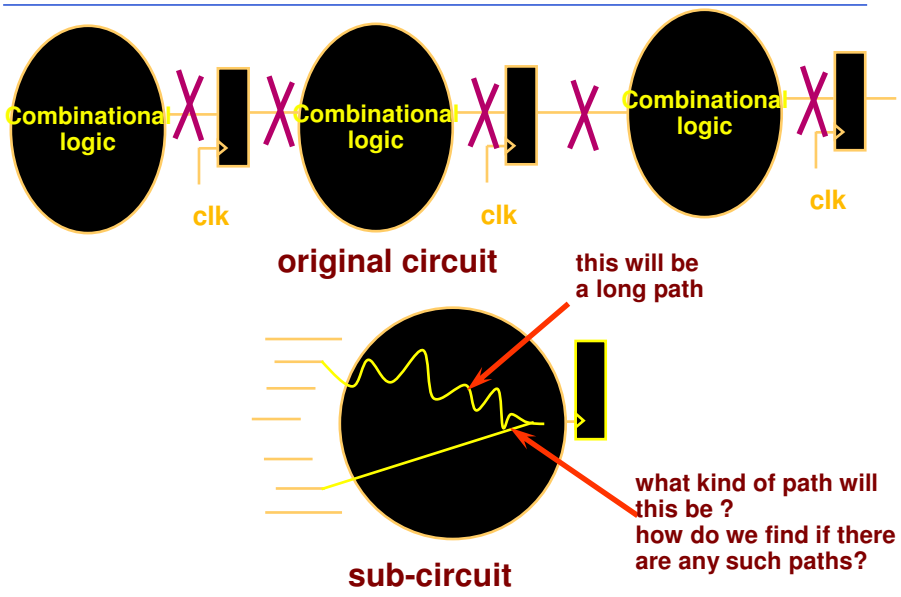
• in addition to checks for *set-up time violations* there need to be checks for *hold-time violations*

• the *hold-time* of a circuit is the amount of time that a signal needs to be held steady so that it can be ``latched into'' the register

How can we check for this problem?

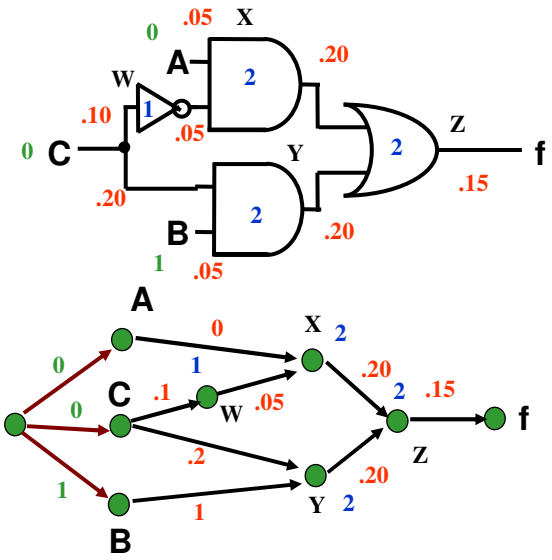


Hold-time problem



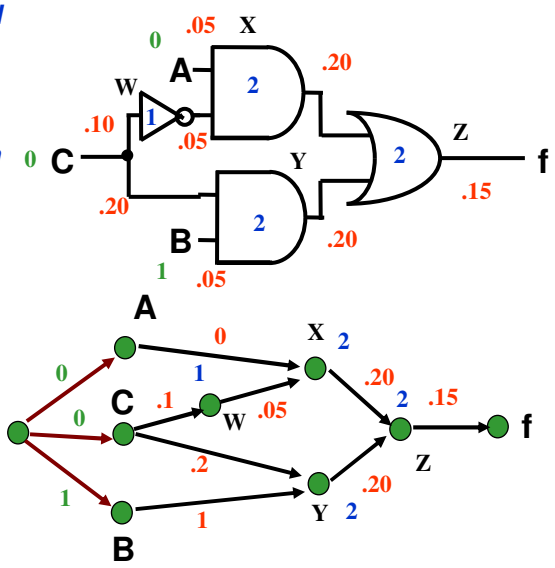
Problem formulation - 1

- Use a labeled *directed* graph
- *G = <V,E>*
- *Vertices* represent gates, primary inputs and primary outputs
- *Edges* represent wires
- *Labels* represent delays
- Now what do we do with this?

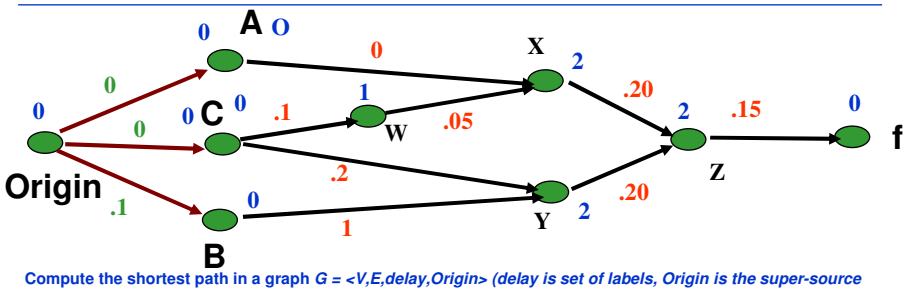


Problem formulation - 2

- Use a labeled *directed* graph
- *G = <V,E>*
- Find the *shortest path*



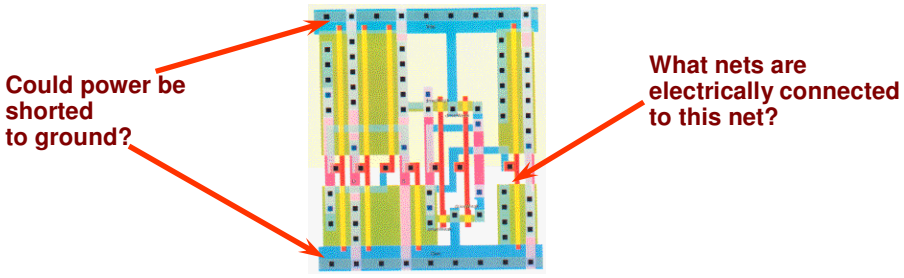
Shortest Path Algorithm



Compute the shortest path in a graph *G = <V,E,delay,Origin> (delay is set of labels, Origin is the super-source of the DAG)*

```
Forward-prop(W){
    for each vertex v in W
        for each edge <v,w> from v
            Final-delay(w) = min(Final-delay(w), delay(v) + delay(w) + delay(<v,w>))
            if all incoming edges of W have been traversed
                add w to W
}
shortest_path(G)
    Forward_prop(Origin)
}
```

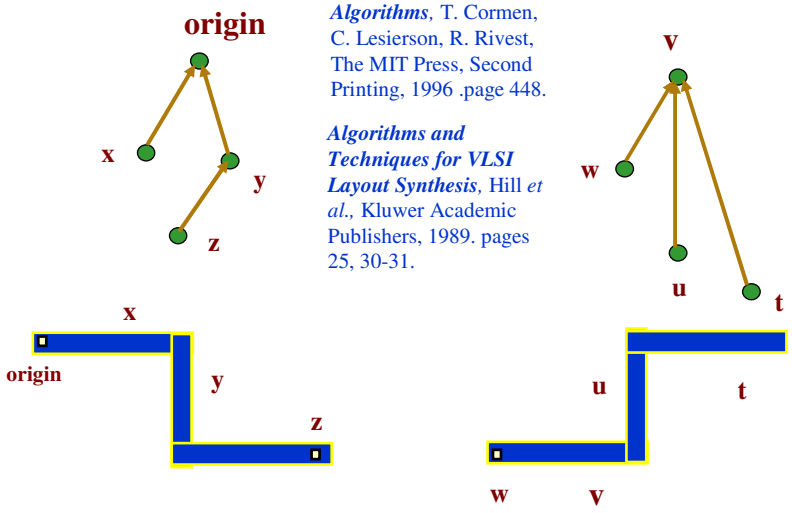

Problem 2: Electrical Connectivity Checks



1) Extract physical geometry
2) Identify electrical connectivity polygon by polygon - build database for future queries
3) Query database

What data-structure would you use in step 2 to make step 3 efficient?

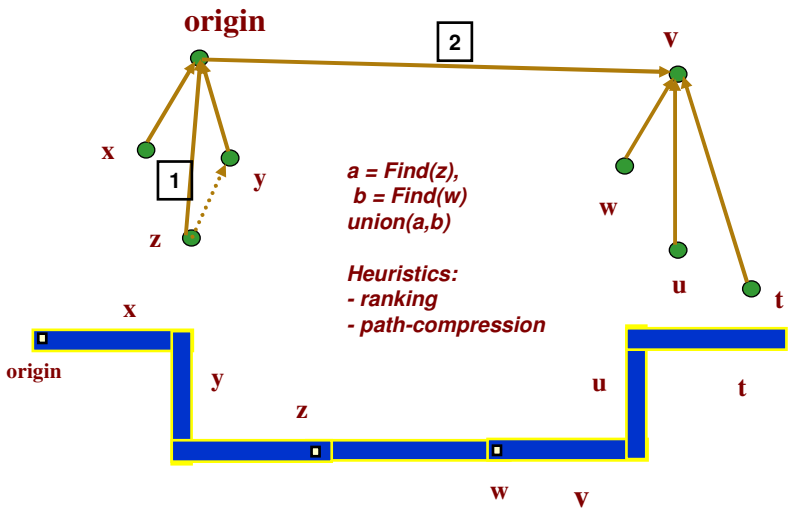
Applying the union-find algorithm - 1



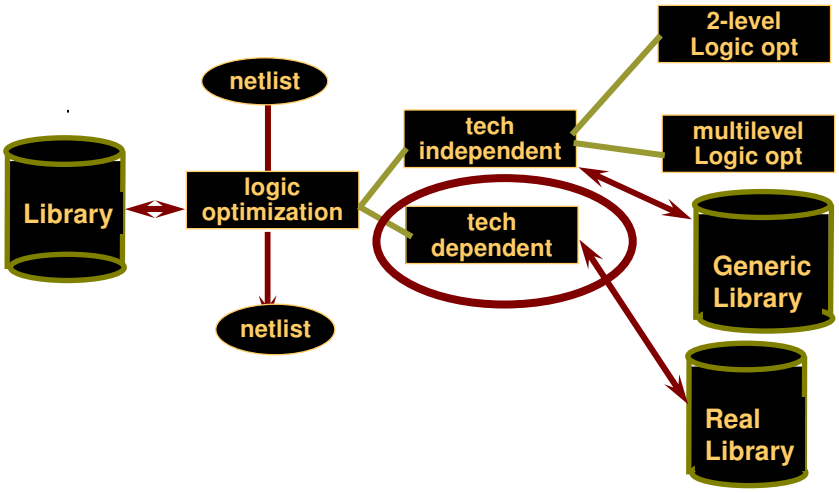
Introduction to Algorithms, T. Cormen, C. Lesierson, R. Rivest, The MIT Press, Second Printing, 1996 .page 448.

Algorithms and Techniques for VLSI Layout Synthesis, Hill *et al.*, Kluwer Academic Publishers, 1989. pages 25, 30-31.

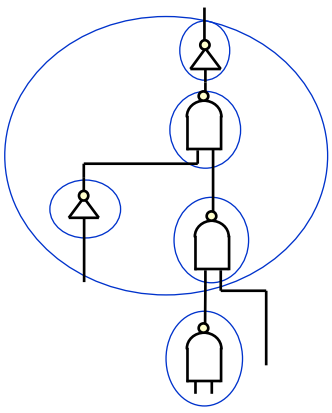
Applying the union-find algorithm - 2



Problem 3: Mapping for Delay



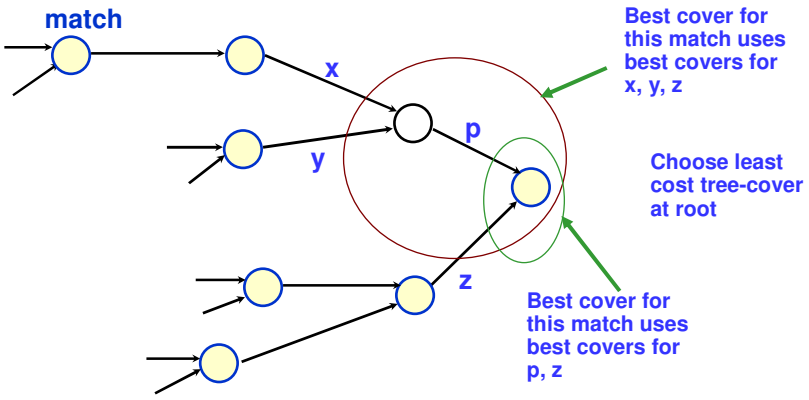
Can we use dynamic programming?



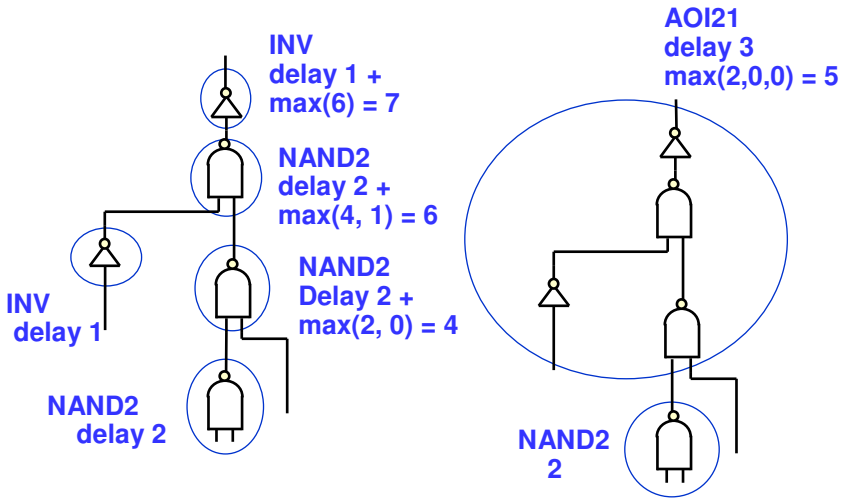
Can we use a dynamic programming formulation to find a minimum *delay* cover of the candidate tree?

Does Dynamic Programming Still Work?

Principle of optimality: Optimal cover for a tree consists of a best match at the root of the tree plus the optimal cover for the sub-trees starting at each input of the match



Dynamic Programming for Min Delay



What else do we need to consider?

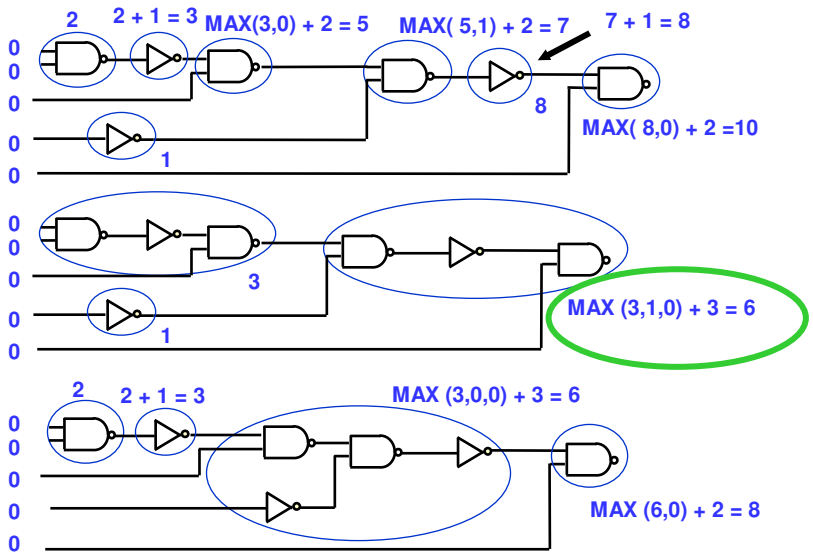
We need to time the cover based on proper arrival times

- Arrival times will only be known when the arrival times of prior (topologically) trees in the DAG are known
- Map from inputs to outputs

Mapping of the tree may produce too much slack on off-critical paths – we'll discuss this later in the lecture

Selection of a cell in the network depends on the load it is facing!

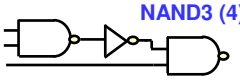
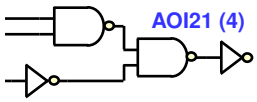
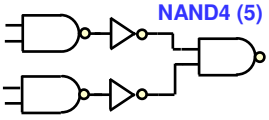
Three covers for delay - min load



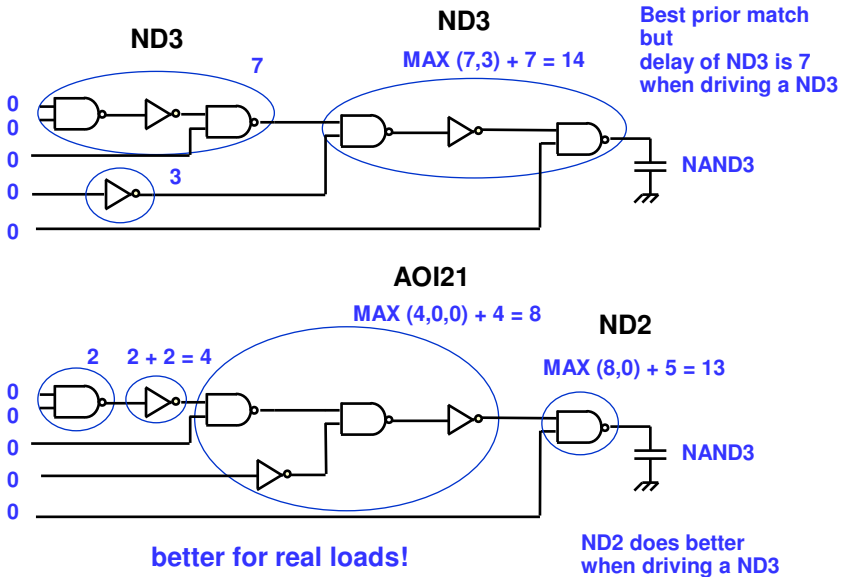
What about higher load?

Library and Delay Information

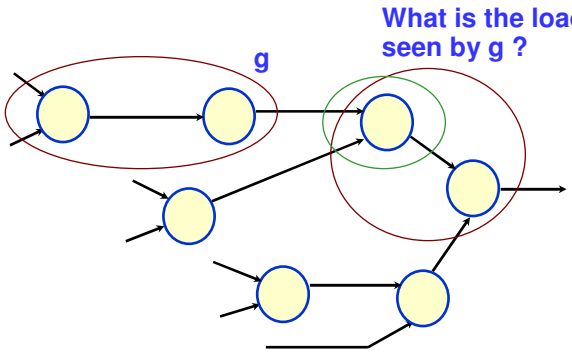
Load-Dependent Delay When Driving

Area	CELL	Delay of Cell Driving		
 INV (1)	INV (1)	NAND2 (2) AOI21	NAND3 (3) NAND4	
 NAND2 (3)	ND2 (2)	NAND2 (4) AOI21	NAND3 (5) NAND4	
NAND3 (4) 	ND3 (3)	NAND2 (5) AOI21	NAND3 (7) NAND4	
AOI21 (4) 	AOI21 (3)	NAND2 (4) AOI21	NAND3 (7) NAND4	
NAND4 (5) 	ND4 (5)	NAND2 (9) AOI21	NAND3 (12) NAND4	

Variable Load



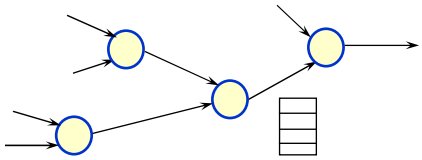
Incorporating load-dependent delays



Optimum match depends on forward (unmapped) part of the tree

How can we handle this in the dynamic programming framework?

Variable Load Delay Optimization



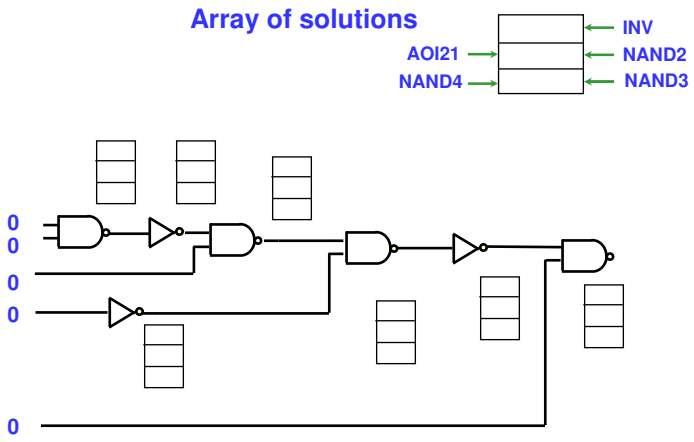
Create bin for each load value that we may face

Array of solutions at each node, one per load value

Compute arrival time for each match for each load value

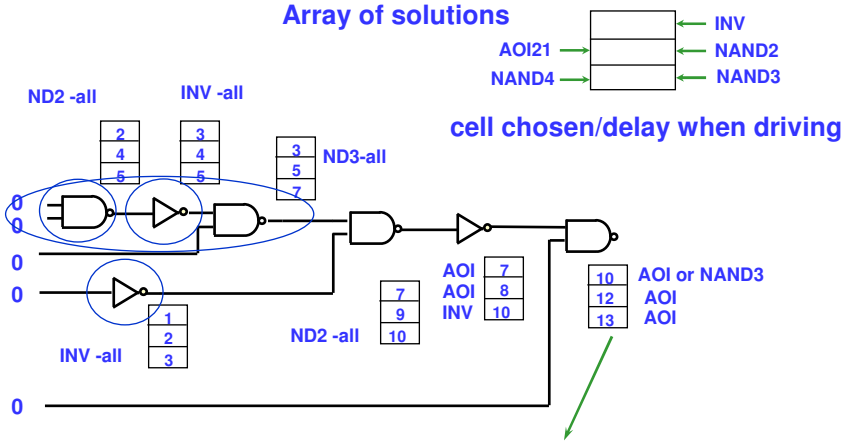
When evaluating a match, use the optimal solution at the input node which is appropriate for the load presented by this match

Variable Load Covering



Variable Load Covering Result

(all solutions NAND2 sees INV)

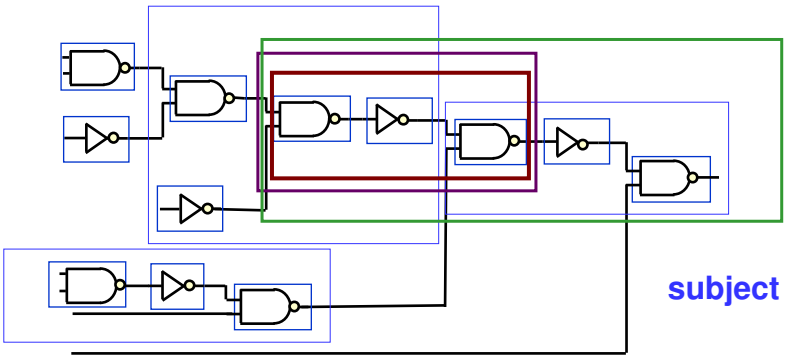


If driving NAND3 will get AOI21 solution with arrival time 13
If driving AOI21 or ND2 will get AOI21 solution
If driving INV choose either AOI21 or NAND3 solution

Summary of load-dependent mapping

- Load-dependent delay shows how dynamic programming paradigm can be extended
- What's the computation time of this approach?
- What's wrong or incomplete with this picture?
- When do we know wiring capacitance?
- What can we do to address it?

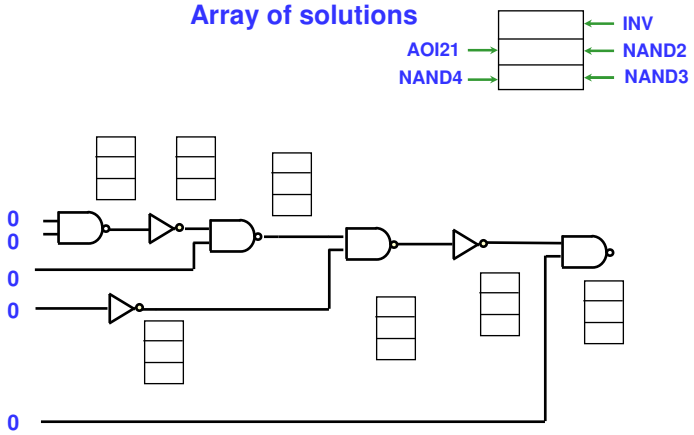
Problem 4 – Techmap for low dynamic power



subject tree

Given input load capacitance for each cell in the library and switching frequencies on the net – find a cover that minimizes dynamic (1/2 CV **2f) power.

Hint - Variable Load Covering



Destination capacitances will be especially important here. Use the variable loading approach that was used for performance-driven mapping.

Problem 5: Layer Assignment

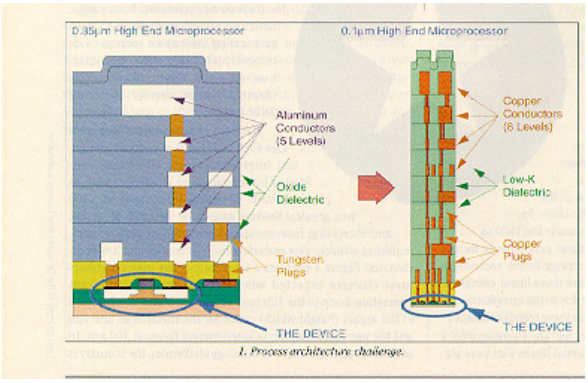


Photo courtesy:
Jan M. Rabaey
Anantha Chandrakasan
Borivoje Nikolic

Given:

- **Cell placement**
- **100K nets**
- **8 layers**
- **Assign each net to a (primary) layer**

Intel 65nm

- **M1 105nm,**
- **M2 105;**
- **M3 110;**
- **M4 140;**
- **M5 165;**
- **M6 240;**
- **M7, 360;**
- **M8 540)**

Routing Objectives

Minimize path delay

- **Timing driven → maintain timing constraints**
- **Minimize wire length**
- **Balance congestion**

Minimize noise

- **Noise driven → minimize cross-coupled capacitance**

Minimize clock skew

- **Balance clock trees**
- **Keep buses together**

When to do layer assignment?

Advantages/disadvantages for global routing

Advantages/disadvantages for detail routing

Global routing

- Identify routing resources to be used
- [Identify layers (and tracks) to be used]
- Assign particular nets to these resources
- Also used in floorplanning and placement

- Within global routing
  - Advantages/disadvantages to doing layer assignment before routing region assignment
  - Advantages/disadvantages to doing layer assignment after routing region assignment

Linear time algorithm

Before routing region assignment?

- Simply take Euclidean distance of each net
- Bin nets according to distance – longer nets higher layers
- Improvement?
- Use Rectilinear Steiner Trees

After routing region assignment?

- Simply take length of RST of each net
- Bin nets according to distance – longer nets higher layers

The Class NP

NP is a class of decision problems for which

- a given proposed solution (called certificate) for a given input
- can be checked quickly (in polynomial time) to see if it really is a solution.

A non-deterministic algorithm

- The non-deterministic "guessing" phase.
  - Some completely arbitrary string s, "proposed solution"
  - each time the algorithm is run the string may differ
- The deterministic "verifying" phase.
  - a deterministic algorithm takes the input of the problem and the proposed solution s, and
  - return value true or false
- The output step.
  - If the verifying phase returned true, the algorithm outputs yes. Otherwise, there is no output.

The Class NP-Complete

A problem Q is *NP-complete*

- it is NP-hard.
- if it is in NP and

A problem Q is *NP-hard*

- if *every* problem in NP is reducible to Q.

A problem P is *polynomially reducible* to a problem Q if

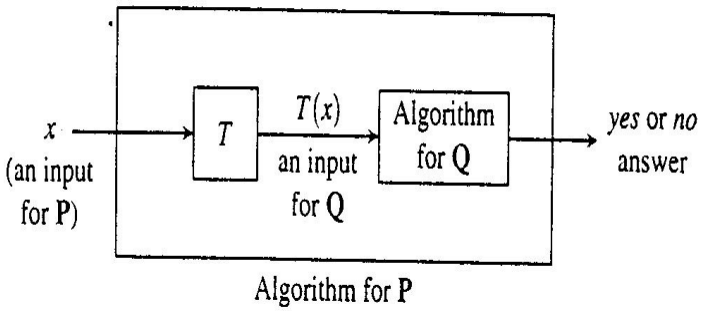
- there exists a polynomial reduction function T such that
  - For every string x,
  - if x is a yes input for P, then T(x) is a yes input for Q
  - if x is a no input for P, then T(x) is a no input for Q.
  - T can be computed in polynomially bounded time.

Polynomial Reductions

Problem P is polynomially reducible to Q

- P ≤p Q
- Transforming inputs of P
  - to inputs of Q

Reducibility relation is transitive.



Problem 5:

Given Boolean satisfiabilty problem is NP-hard

Prove Circuit-satisfiability problem is NP-Complete

Circuit-satisfiablity problem

- we say that a one-output Boolean combinational circuit is satisfiable
  - if it has a satisfying assignment,
  - a truth assignment (a set of Boolean input values) that
  - causes the output of the circuit to be 1

Problem: Show that Circuit-satisfiability problem

- belongs to the class NP
- is NP-hard, i.e.
  - Show any instance of satisfiabilty problem can, in poly time, be turned into an instance of circuit sat

Problem 5: Hints

Problem: Show that Circuit-satisfiability problem

- **belongs to the class NP**
  - **Hint: If you guessed a solution how long would it take to verify it**
- **is NP-hard, i.e.**
  - **Hint: Show any instance of the Boolean satisfiabilty problem can, in poly time, be turned into an instance of circuit sat**