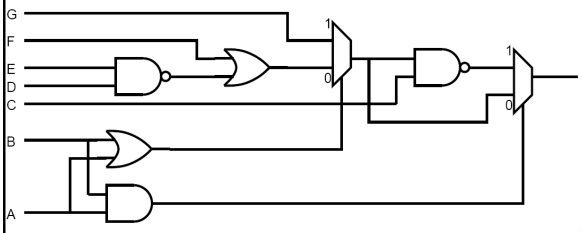
ESE535: Electronic Design Automation

Day 22: April 8, 2013
Static Timing Analysis
and Multi-Level Speedup

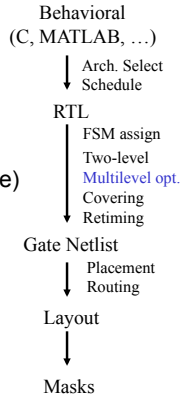
Delay of Preclass

- Delay?
- What transition causes?

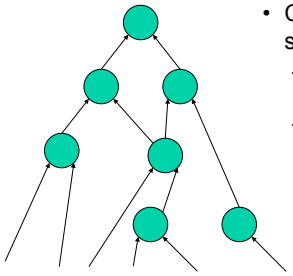


Today

- Topological Worst Case
  - not adequate (too conservative)
- Sensitization Conditions
- Timed Calculus
- Delay-justified paths
  - Timed-PODEM
- Speedup

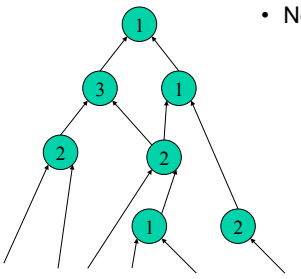


Topological Worst-Case Delay



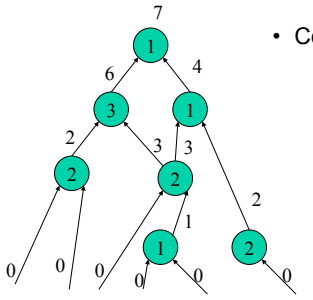
- Compute ASAP schedule
  - Take max of arrival times
  - Apply node Delay

Topological Worst-Case Delay



- Node Delays

Topological Worst-Case Delay



- Compute Delays

False Paths

- Once consider logic for nodes
  - There are logical constraints on data values
- There are paths that cannot logically occur
  - Call them ***false paths***

What can we do?

- Need to assess what paths are real
- Brute force
  - for every pair of inputs
  - compute delay in outputs from in1→in2 input transition
  - take worst case
- How many such delay traces?
  - 2^{2n} delay traces

Alternately

- Look at single vector and determine what controls delay of circuit
  - I.e. look at values on path and determine path *sensitized* to change with input

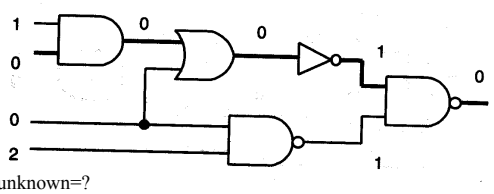
Controlling Inputs

- Controlling input to a gate:
  - input whose value will determine gate output
  - e.g.
    - 0 on a AND gate
    - 1 on a OR gate

Static Sensitization

- A path is statically sensitized
  - if all the side (non-path) inputs are non-controlling
  - I.e. this path value flips with the input

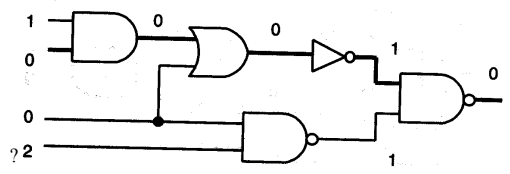
Statically Sensitized Path



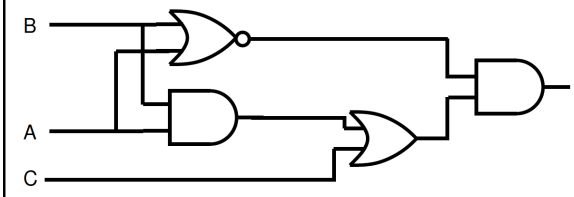
2=unknown=?

Sufficiency

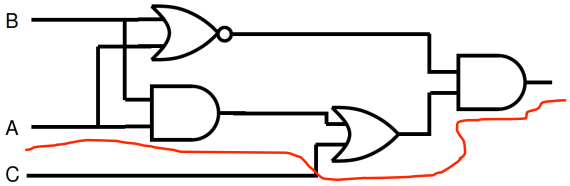
- Static Sensitization is **sufficient** for a path to be a **true** path in circuit



Static Sensitization not Necessary

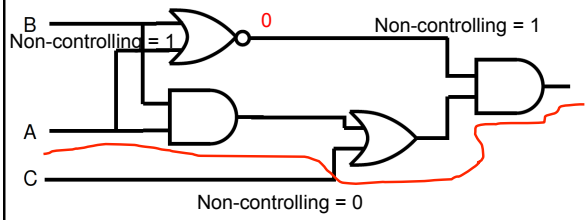


Static Sensitization not Necessary



- Possible path of length 3

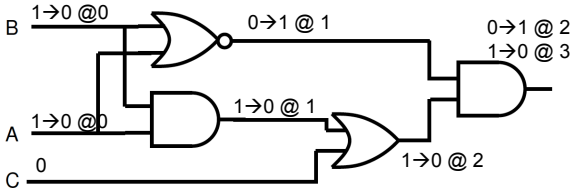
Static Sensitization not Necessary



- Paths of length 3 not statically sensitizable.

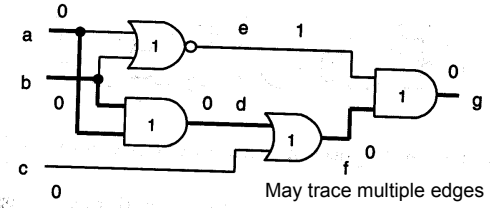
Static Sensitization not Necessary

- True Path of Delay 3 (simulate)



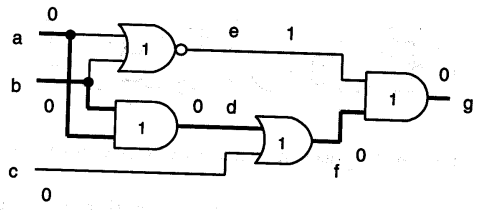
Static Co-sensitization

- Each output with a controlled value
  - has a controlling value as input on path
  - (and vice-versa for non-controlled)

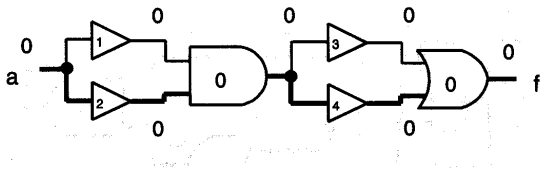


Necessary

- Static Co-sensitization is a **necessary** condition for a path to be true



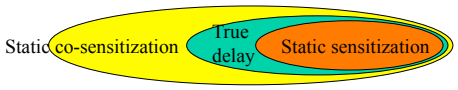
Cosensitization not Sufficient



Cosensitize path of length 6.
Real delay is 5.

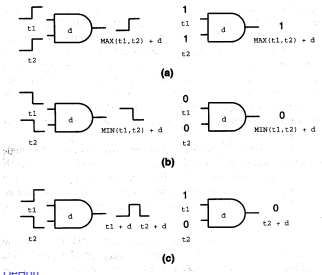
Combining

- Combine these ideas into a timed-calculus for computing delays for an input vector



Computing Delays

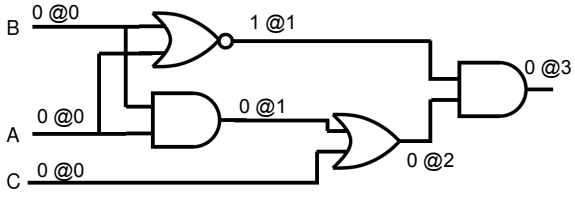
AND Timing Calculus



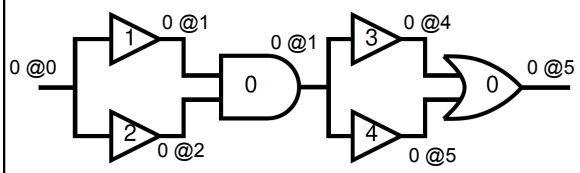
Rules

- If gate output is at a controlling value, pick the minimum input and add gate delay
- If gate output is at a non-controlling value, pick the maximum input and add gate delay

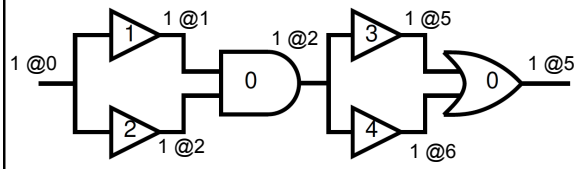
Example 1



Example 2



Example 2



Now...

- We know how to get the delay of a single input condition
- Could:
  - find critical path
  - search for an input vector to sensitize
  - if fail, find next path
  - …until find longest true path
- May be $O(2^n)$

Better Approach

- Ask if can justify a delay greater than T
- Search for satisfying vector
  - …or demonstration that none exists
- Binary search to find tightest delay

Delay Computation

- Modification of a testing routine
  - used to justify an output value for a circuit
  - similar to SAT
- PODEM (Path Oriented DEcision Making)
  - backtracking search to find a suitable input vector associated with some target output
  - branching search with implication pruning
    - Heuristic for smart variable ordering

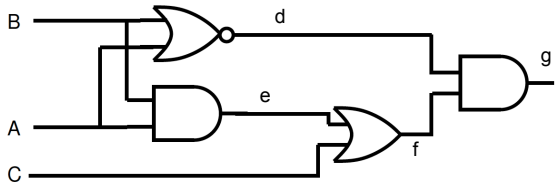
Search

- Takes two lists
  - outputs to set; inputs already set
- Propagate values and implications
- If all outputs satisfied → succeed
- Pick next PI to set and set value
  - Search (recursive call) with this value set
  - If inconsistent
    - If PI not implied
      - Invert value of PI
      - Search with this value set
      - If inconsistent → fail
      - Else succeed
    - Else fail
  - Else succeed

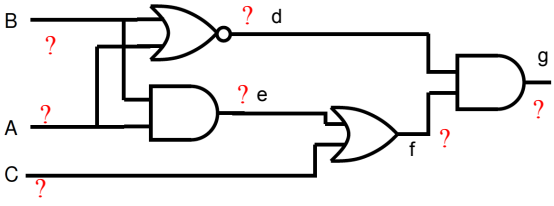
Picking next variable to set

- Follow back gates w/ unknown values
  - sometimes output dictate input must be
    - (AND needing 1 output; with one input already assigned 1)
    - Implication
  - sometimes have to guess what to follow
    - (OR with 1 output and no inputs set)
    - Uses heuristics to decide what to follow

Example

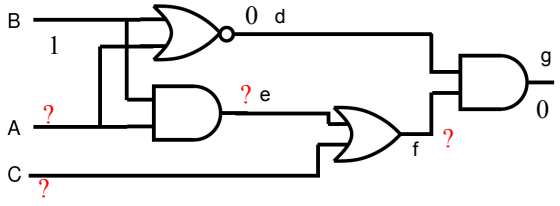


Example (goal g=1)



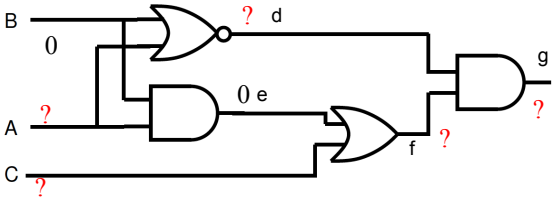
Example (goal g=1)

- Try B=1



Example (goal g=1)

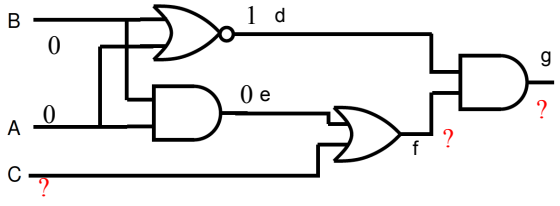
- Try B=0



Deduce any inputs?

Example (goal g=1)

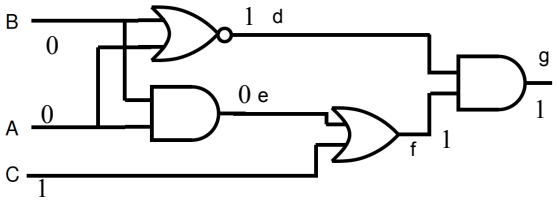
- Implied A=0



Deduce any inputs?

Example (goal g=1)

- Implied C=1



For Timed Justification

- Also want to compute delay
  - on incompletely specified values
- Compute bounds on timing
  - upper bound, lower bound
  - Again, use our timed calculus
    - expanded to unknowns

Delay Calculation

AND rules

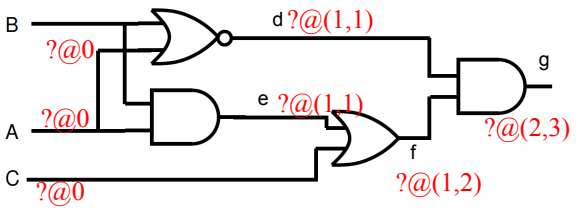
$i_1 \rightarrow$ / $i_2 \downarrow$	0	1	? 2
0	0 $MIN(l_1,l_2)+d$ $MIN(u_1,u_2)+d$	0 l_2+d u_2+d	0 $MIN(l_1,l_2)+d$ u_2+d
1	0 l_1+d u_1+d	1 $MAX(l_1,l_2)+d$ $MAX(u_1,u_2)+d$	2 l_1+d $MAX(u_1,u_2)+d$
2 ?	0 $MIN(l_1,l_2)+d$ u_1+d	2 l_2+d $MAX(u_1,u_2)+d$	2 $MIN(l_1,l_2)+d$ $MAX(u_1,u_2)+d$

Unknowns force us to represent upper/lower bound on delay.

Timed PODEM

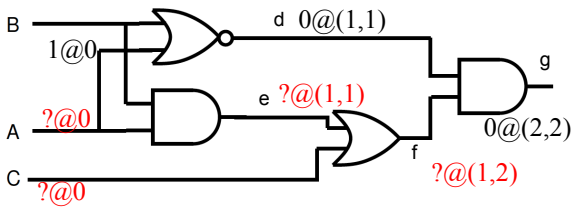
- **Input:** value to justify and delay T
- **Goal:** find input vector which produces value and exceeds delay T
- Algorithm
  - similar
  - implications check timing as well as logic

Example (goal g=1@3)



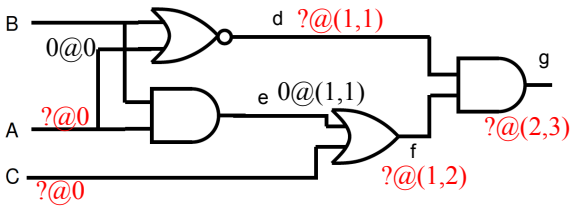
Example (goal g=1@3)

- Try B=1



Example (goal g=1@3)

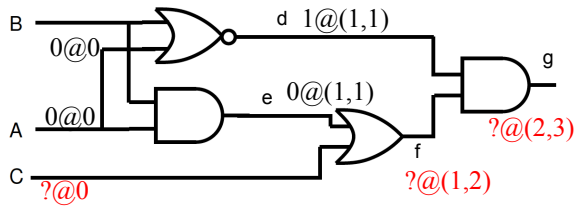
- Try B=0



Deduce any inputs?

Example

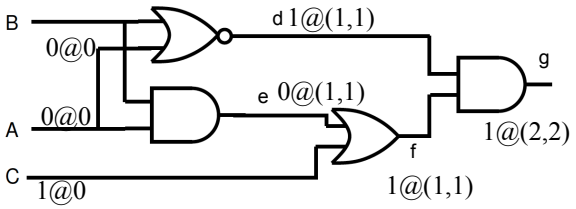
- Imply A=0



Deduce any inputs?

Example (goal g=1@3)

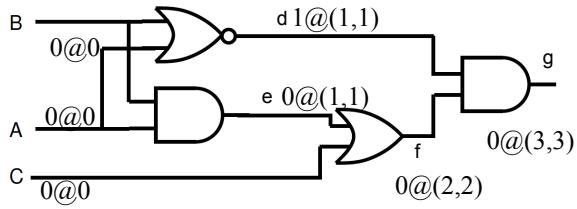
- Imply C=1



Failed to justify 1@3

Example (goal g=0@3)

- Try C=1



Search

- Less than 2^n
  - pruning due to implications
  - here saw a must be 0
    - no need to search 1xx subtree

Questions

- Any questions on static timing analysis?

Speed Up

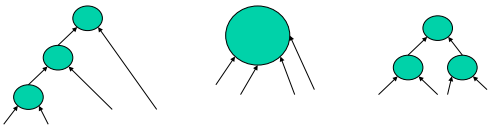
(sketch flavor)

Speed Up

- Start with area optimized network
- Know target arrival times
  - Know delay from static analysis
- Want to reduce delay of node

Basic Idea

- Improve speed by:
  - Collapsing node(s)
  - Refactoring collapsed subgraph to reduce height

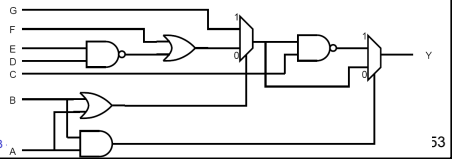


Speed Up

- While (delay decreasing, timing not met)
  - Compute delay (slack)
    - Static timing analysis
  - Generate network close to critical path
    - w/in some delay ε, to some distance d
  - Weight nodes in network
    - Less weight = more potential to improve, prefer to cut
  - Compute **mincut** of *nodes* on weighted network
  - For each node in cutset
    - Partial collapse
  - For each node in cutset
    - Timing redecompose

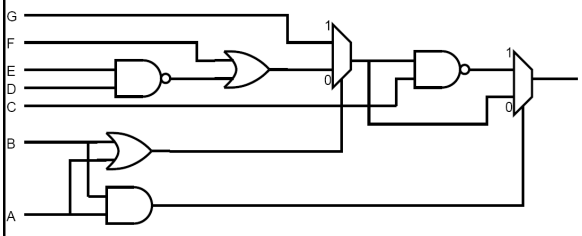
MinCut of Nodes

- Cut nodes not edges
  - Typically will need to transform to dual graph
    - All edges become nodes, nodes become edges
  - Then use maxflow/mincut

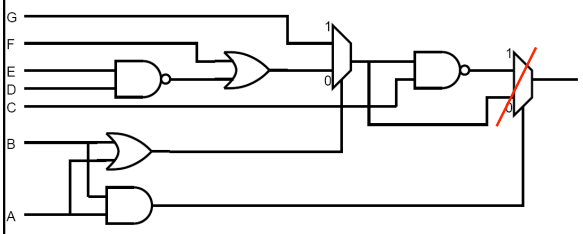


MinCut of Nodes

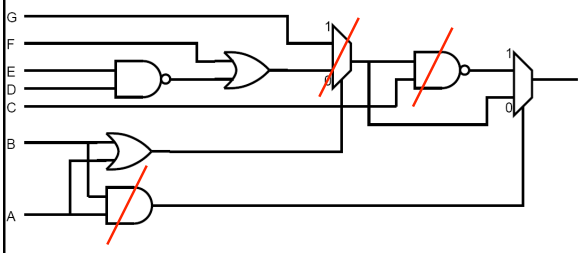
- What are possible cuts?



MinCut of Nodes



MinCut of Nodes

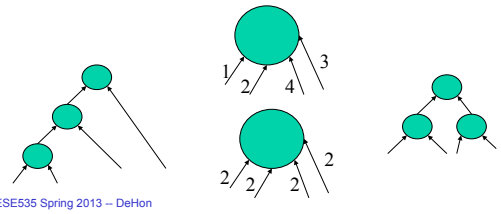


Weighted Cut

- $W=W_t+\alpha W_a$ → α tuning parameter
- Want to minimize area expansion (W_a)
  - Things in collapsed network may be duplicated
  - *E.g.* W_a=literals in duplicated logic
- Want to maximize likely benefit (W_t)
  - Prefer nodes with different input times to the "near critical path" network
    - Quantify: large difference in arrival times
  - Prefer nodes with critical path on longer paths

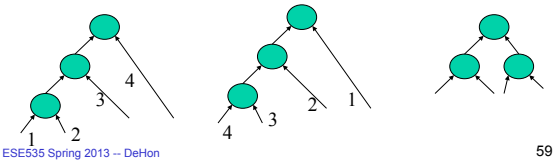
Weighing Benefit

- Want to maximize likely benefit (W_t)
  - Prefer nodes with different input times to the "near critical path" network



Weighing Benefit

- Want to maximize likely benefit (W_t)
  - Prefer nodes with critical path on longer paths



Timing Decomposition

- Extract area saving kernels that do not include critical inputs to node
  - f=abcd+abce+abef
  - Kernels={cd+ce+ef,e+d,c+f} (last time)
  - F=abe(c+f)+abcd, ab(cd+ce+ef), abc(e+d)+abef
  - What decomposition use (and how finish decompose)?
    - Critical input is f? e? d? {a,d}?
- When decompose (*e.g.* into nand2's) similarly balance with critical inputs closest to output

Delay Decompositions

- f last:
  - abc(e+d)+abef
  - f*((ab)e)+((ab)(c(e+d)))
- e last:
  - abe(c+f)+abcd
  - e*((*ab)(c+f))+((ab)(cd))

Delay Decompositions

- d last:
  - abe(c+f)+abcd
  - d*((ab)c)+((ab)(e(c+f)))
- {a,d} last:
  - abe(c+f)+abcd
  - a((be)(c+f)+d(bc))

Speed Up (review)

- While (delay decreasing, timing not met)
  - Compute delay (slack)
    - Static timing analysis
  - Generate network close to critical path
    - w/in some delay ε, to some distance d
  - Weight nodes in network
    - Less weight = more potential to improve, prefer to cut
  - Compute **mincut** of *nodes* on weighted network
  - For each node in cutset
    - Partial collapse
  - For each node in cutset
    - timing redecompose

Big Ideas

- Topological Worst-case delays are conservative
  - Once consider logical constraints
  - may have false paths
- Necessary and sufficient conditions on true paths
- Search for paths by delay
  - or demonstrate non existence
- Search with implications
- Iterative improvement

Admin

- Reading Wednesday on web