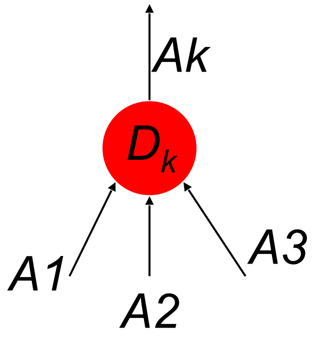
Logic Synthesis

Timing Analysis

Courtesy RK Brayton (UCB) and A Kuehlmann (Cadence)

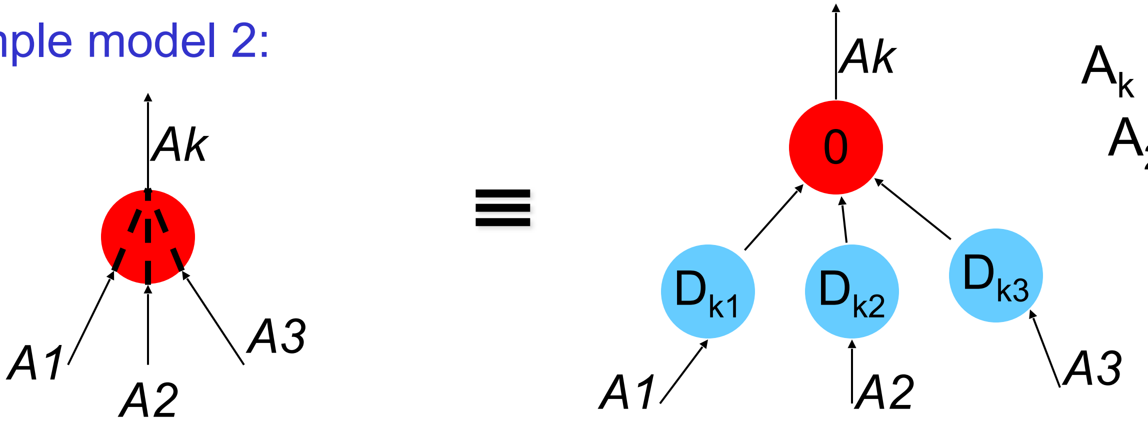
Timing Analysis - Delay Models

- Simple model 1:



A_k = arrival time = max(A_1,A_2,A_3) + D_k
D_k is the delay at node k, parameterized according to function f_k and fanout node k

- Simple model 2:



$A_k = \max\{A_1+D_{k1}, A_2+D_{k2}, A_3+D_{k3}\}$

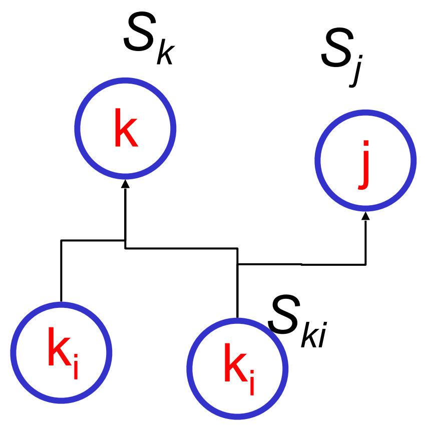
- Can also have different times for rise time and fall time

Static delay analysis

```
// level of PI nodes initialized to 0,
// the others are set to -1.
// Invoke LEVEL from PO
Algorithm LEVEL(k) { // levelize nodes
  if( k.level != -1)
    return(k.level)
  else
    k.level = 1+max{LEVEL($k_i$)|$k_i \in$ fanin(k)}
  return(k.level)
}

// Compute arrival times:
// Given arrival times on PI's
Algorithm ARRIVAL() {
  for L = 0 to MAXLEVEL
    for {k|k.level = L}
      $A_k = \mathbf{MAX}\{A_{ki}\} + D_k$
}
```

Required Times



Required times:

given required times on primary outputs

- Traverse in reverse topological order (i.e. from primary outputs to primary inputs)
- if $(k_i\,,\,k\,)$ is an edge between k_i and k, $R_{k_i\,,k} = R_k - D_k$

  (this is the edge required time)
- Hence, the required time of output of node k is

  $R_k = \min\,(\,R_{k,k_j} \mid k_j \in \text{fanout}(k)\,)$

Propagating Slacks

Slacks: slack at the output node k is $S_k = R_k - A_k$

Since $R_{ki,k} = R_k - D_k$

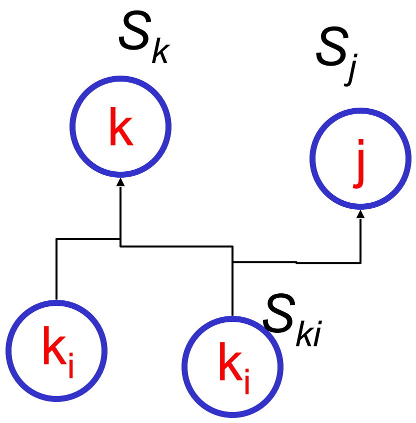
$$S_{ki,k} = R_{ki,k} - A_{ki}$$

$$S_{ki,k} + A_{ki} = R_k - D_k = S_k + A_k - D_k$$

Since $A_k = \max\{A_{kj}\} + D_k$

$$S_{ki,k} = S_k + \max\{A_{kj}\} - A_{ki} \qquad k_j, k_i \in \text{fanin}(k)$$

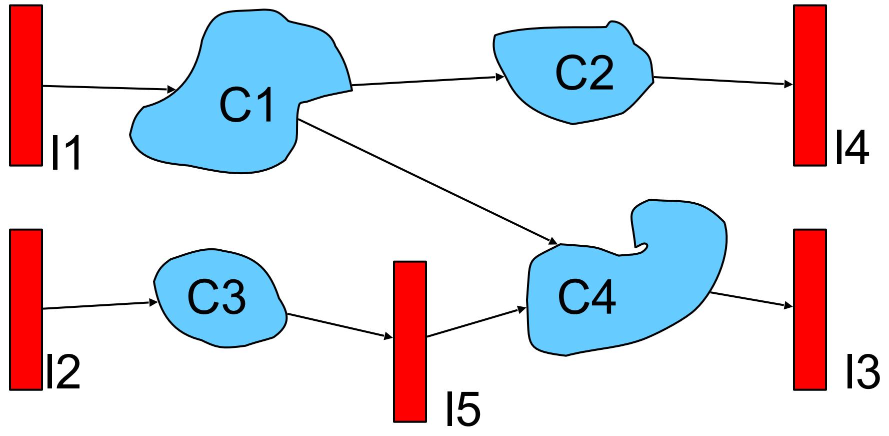
$$S_{ki} = \min\{S_{ki,j}\} \qquad j \in \text{fanout}(k_i)$$



Notes:

- Each edge is the graph has a slack and a required time
- Negative slack is bad.

Sequential networks



- Arrival times known at I_1 and I_2
- Required times known at I_3, I_4, and I_5
- Delay analysis gives arrival and required times (hence slacks) for C_1, C_2, C_3, C_4

Static critical paths

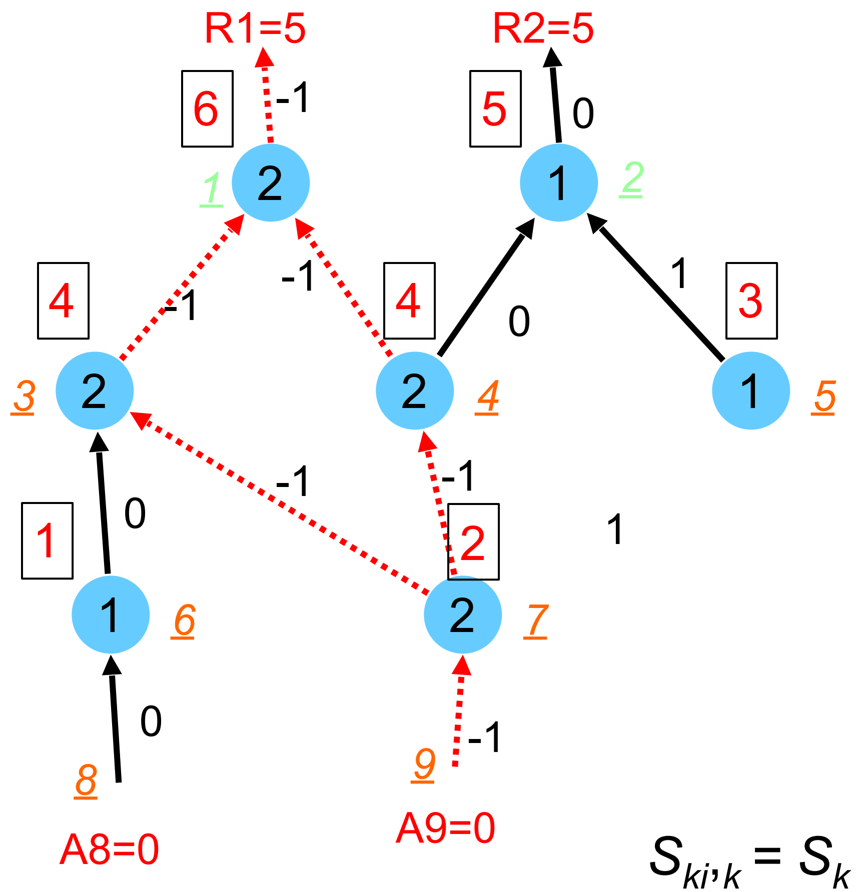
Min-Max problem: minimize max$\{-S_i, 0\}$

A static critical path of a Boolean network is a path $P = \{i_1, i_2, \ldots, i_p\}$ where $S_{i_k, i_{k+1}} < 0$

Note: if a node *k* is on a static critical path, then at least one of the fanin edges of *k* is critical. Hence, all critical paths reach from an input to an output.

Note: There may be several critical paths

Example: Static critical paths



A1=6	R1=5
A2=5	R2=5

S_1=-1	R3=3
S_2=0	R7=1
$S_{3,1}$=-1	R9=-1
$S_{4,1}$ = -1	
$S_{4,2}$ = 0	
$S_{5,2}$ = 1	
$S_{6,3}$ = 0	
$S_{7,3}$ = -1	
$S_{7,4}$ = -1	
$S_{7,5}$ = 1	
$S_{8,6}$ = 0	
$S_{9,7}$ = -1	

.... critical path edges

$$S_{ki,k} = S_k + \max\{A_{kj}\} - A_{ki}, \; k_j, k_i \in \text{fanin}(k)$$
$$S_k = \min\{S_{k,kj}\}, \; k_j \in \text{fanout}(k)$$

Timing analysis problems

We want to determine the true critical paths of a circuit in order to:

- determine the minimum cycle time that the circuit will function
- identify critical paths from performance optimization - don't want to try to optimize the wrong (non-critical) paths

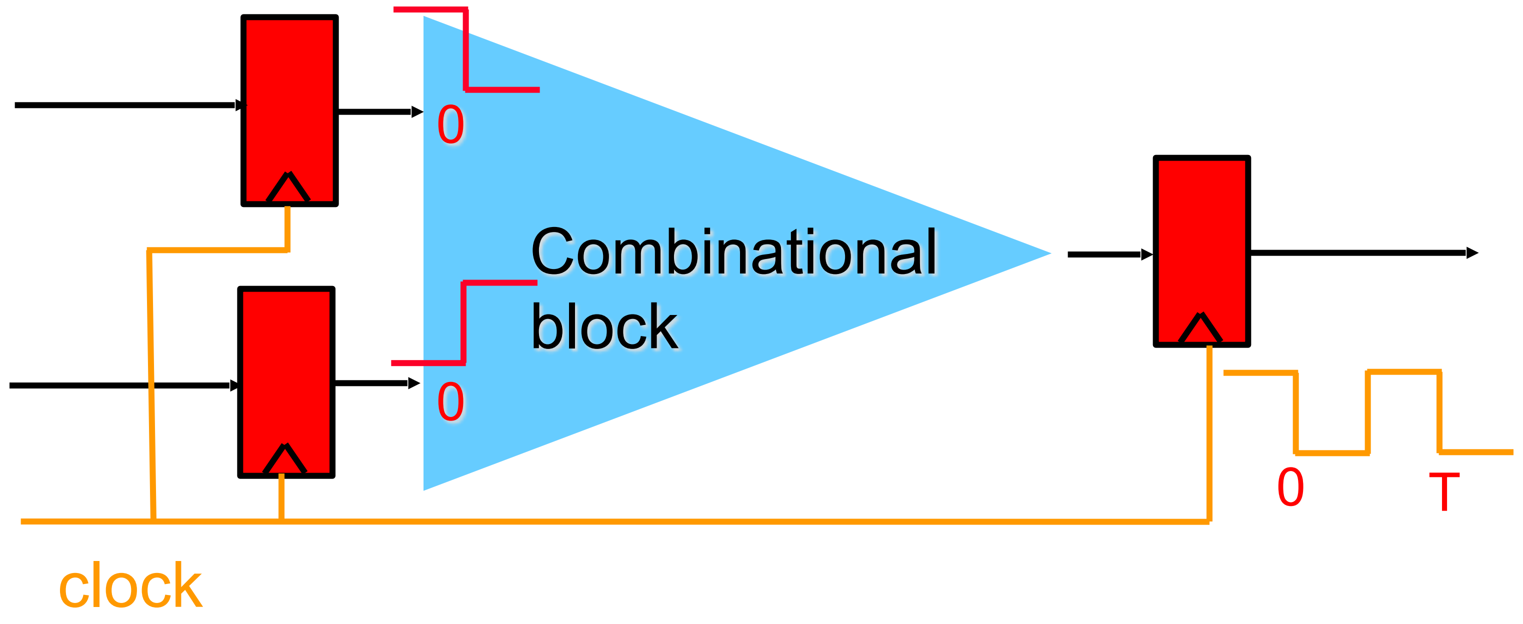
Implications:

- Don't want false paths (produced by static delay analysis)
- Delay model is worst case model. Need to ensure correctness for case where ith gate delay ≤ $\mathbf{D_i^M}$

Functional Timing Analysis

What is Timing Analysis?

Estimate when the output of a given circuit gets stable



Why Timing Analysis?

Timing verification
- Verifies whether a design meets a given timing constraint
  - Example: cycle-time constraint

Timing optimization
- Needs to identify critical portion of a design for further optimization
  - Critical path identification

In both applications, the more accurate, the better

Timing Analysis - Basics

Naïve approach - Simulate all input vectors with SPICE

- Accurate, but too expensive

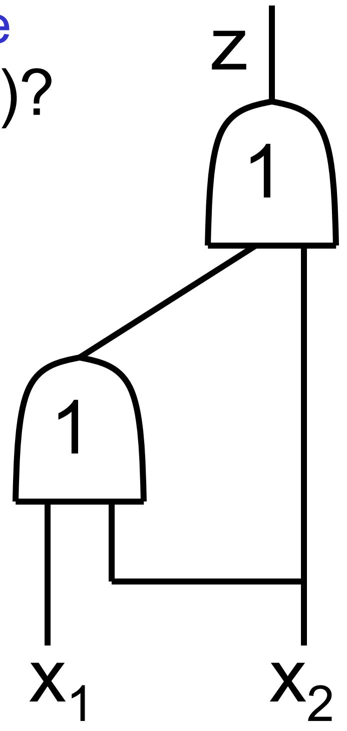
Gate-level timing analysis

Focus of this lecture

- Less accurate than SPICE due to the level of abstraction, but **much** more efficient
- Scenario:
  - Gate/wire delays are pre-characterized (accuracy loss)
  - Perform timing analysis of a gate-level circuit assuming the gate/wire delays

Gate-level Timing Analysis

False path aware arr(z)?



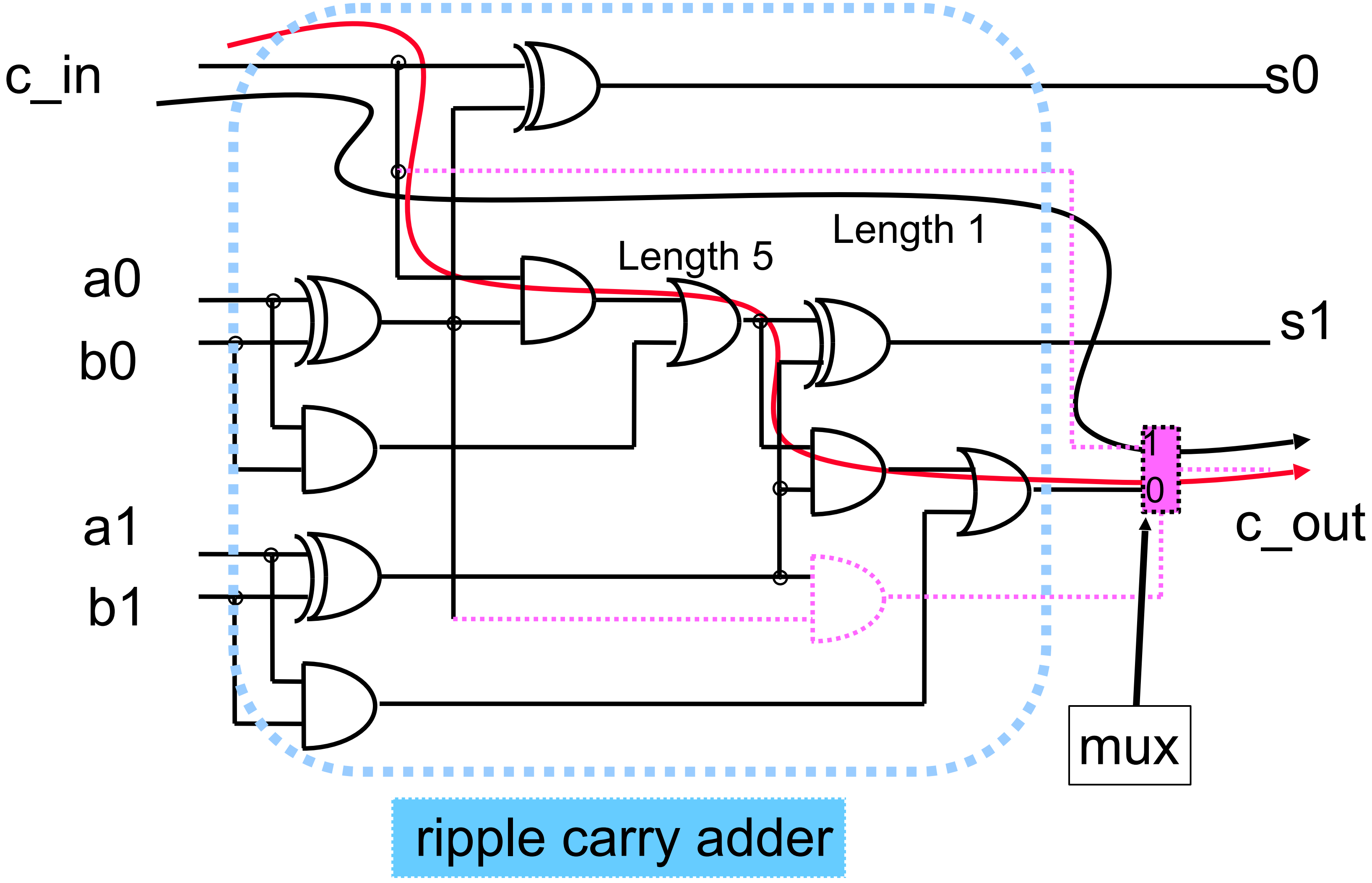
$arr(x_1)=0$ $arr(x_2)=0$

A naive approach is topological analysis
- Easy longest-path problem
- Linear in the size of a network

Not all paths can propagate signal events
- False paths
- If all longest paths are false, topological analysis gives delay overestimate

Functional timing analysis = false-path-aware timing analysis
- Compute false-path-aware arrival time

Example: 2-bit Carry-skip Adder



False Path Analysis - Basics

Is a path responsible for delay?

- If the answer is no, can ignore the path for delay computation

Check the falsity of long paths until we find the longest true path

- How can we determine whether a path is false?

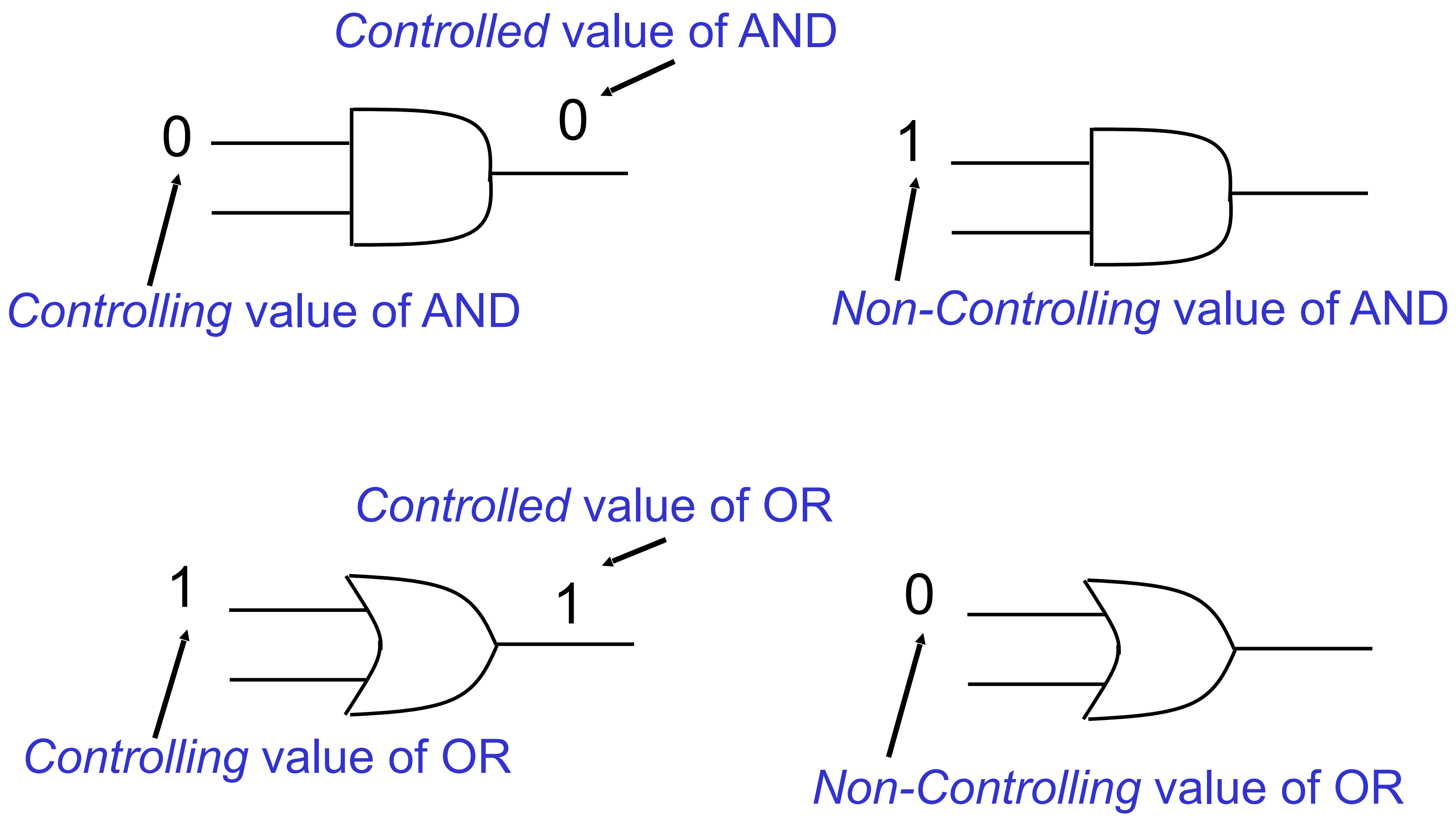
Delay underestimation is unacceptable

- Can lead to overlooking a timing violation

Delay overestimation is not desirable, but acceptable

- Topological analysis can give overestimate, but never give underestimate

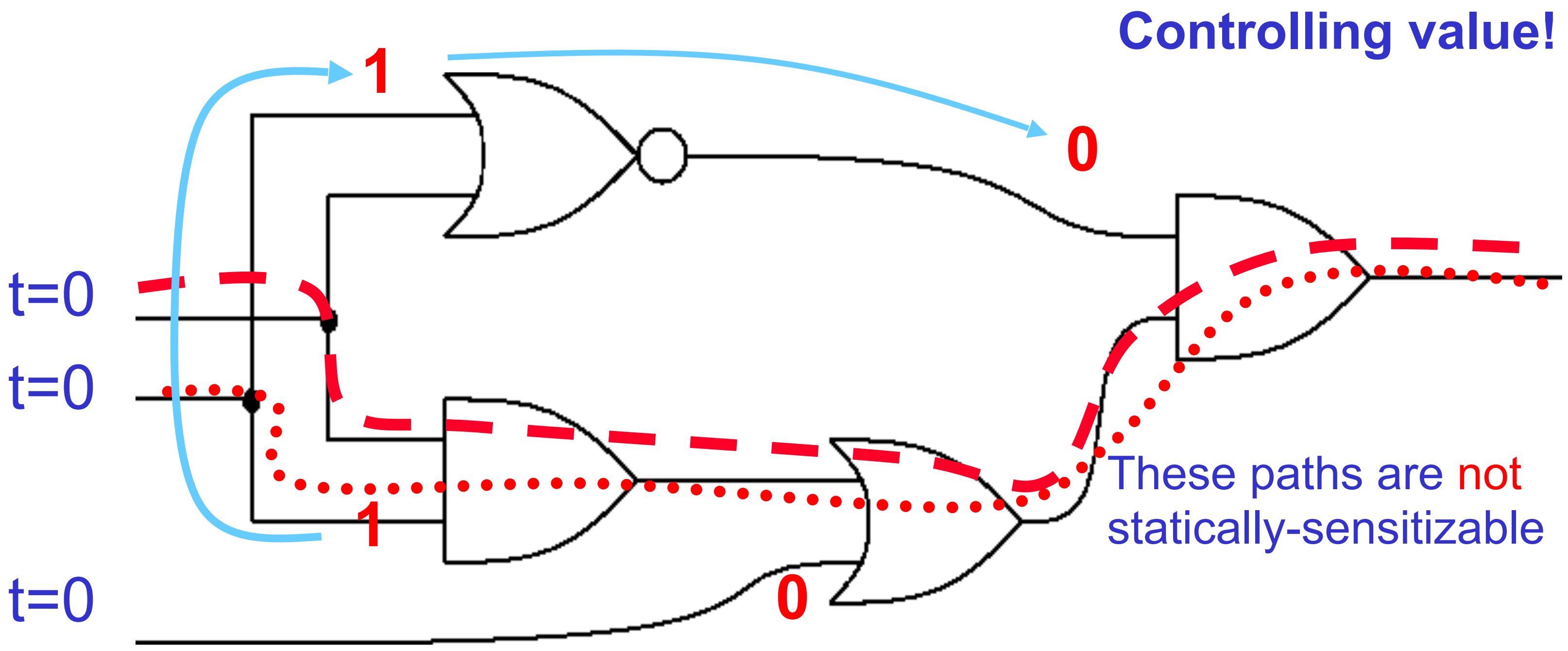
Controlling/Non-Controlling Values


Controlled value of AND
0
0
Controlling value of AND
1
Non-Controlling value of AND
Controlled value of OR
1
1
Controlling value of OR
0
Non-Controlling value of OR

Static Sensitization

A path is *statically-sensitizable* if there exists an input vector such that all the side inputs to the path are set to non-controlling values

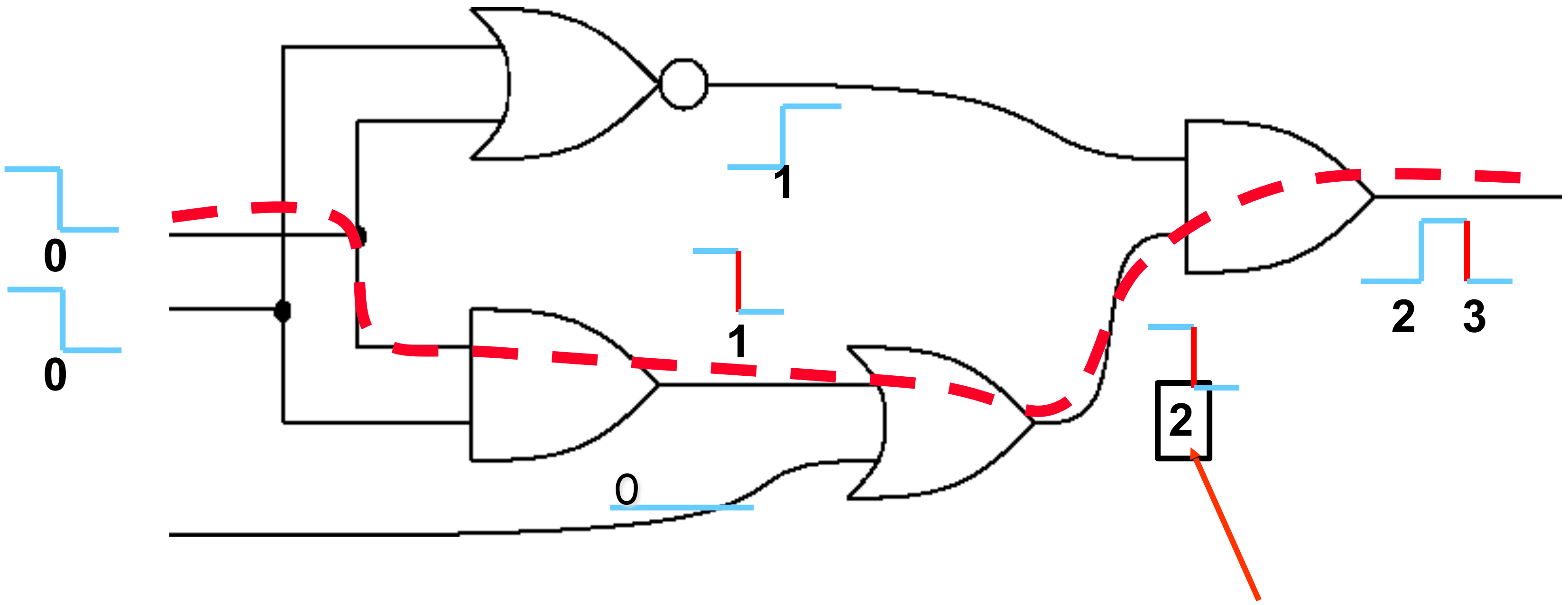
- This is independent of gate delays



The longest true path is of length 2?

Static Sensitization

- The (dashed) path is responsible for delay!
- Delay underestimation by static sensitization (delay = 2 when true delay = 3)
  - incorrect condition



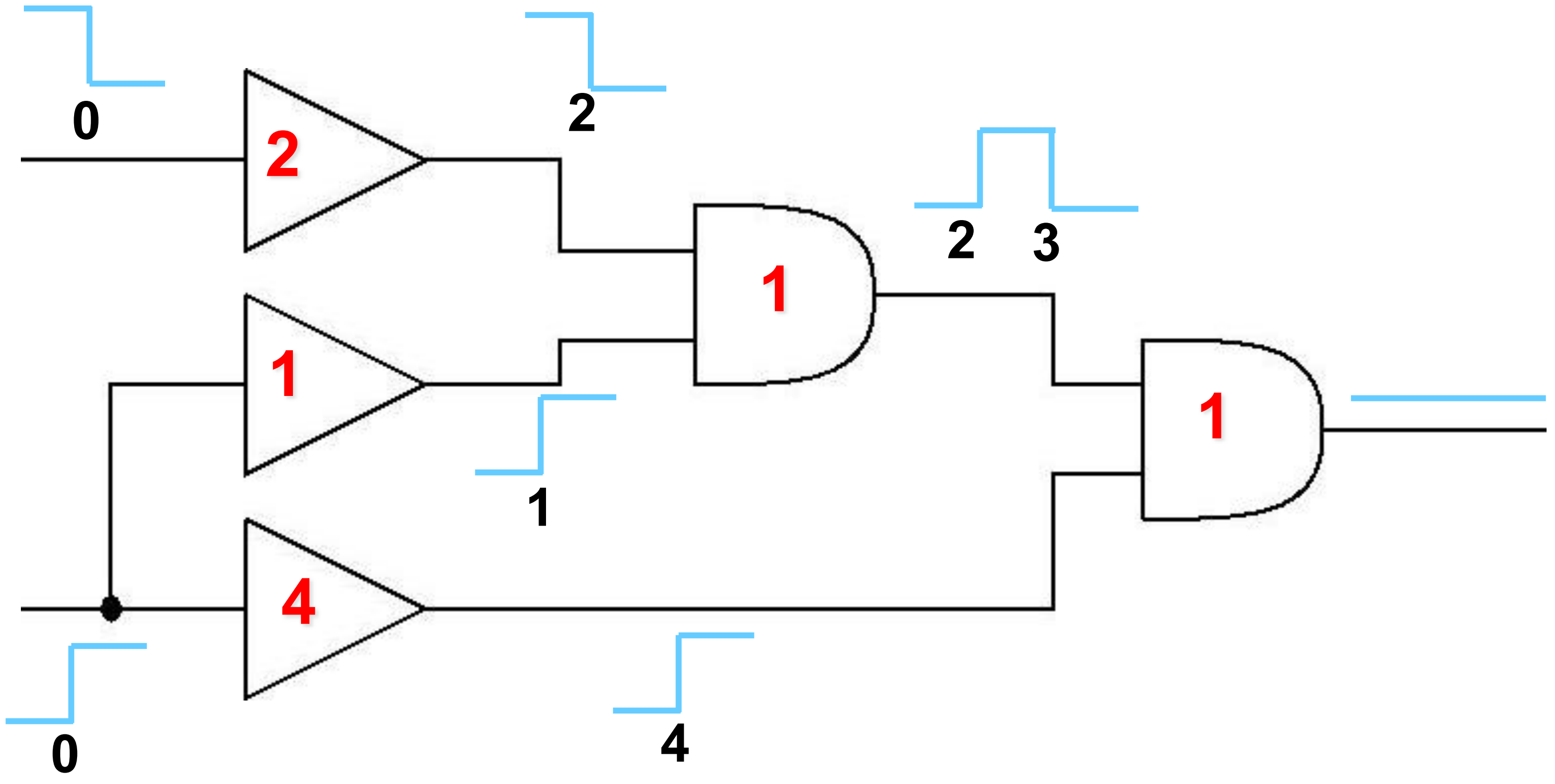
What is Wrong with Static Sensitization?

The idea of forcing non-controlling values to side inputs is okay, but timing was ignored

- The same signal can have a controlling value at one time and a non-controlling value at another time.

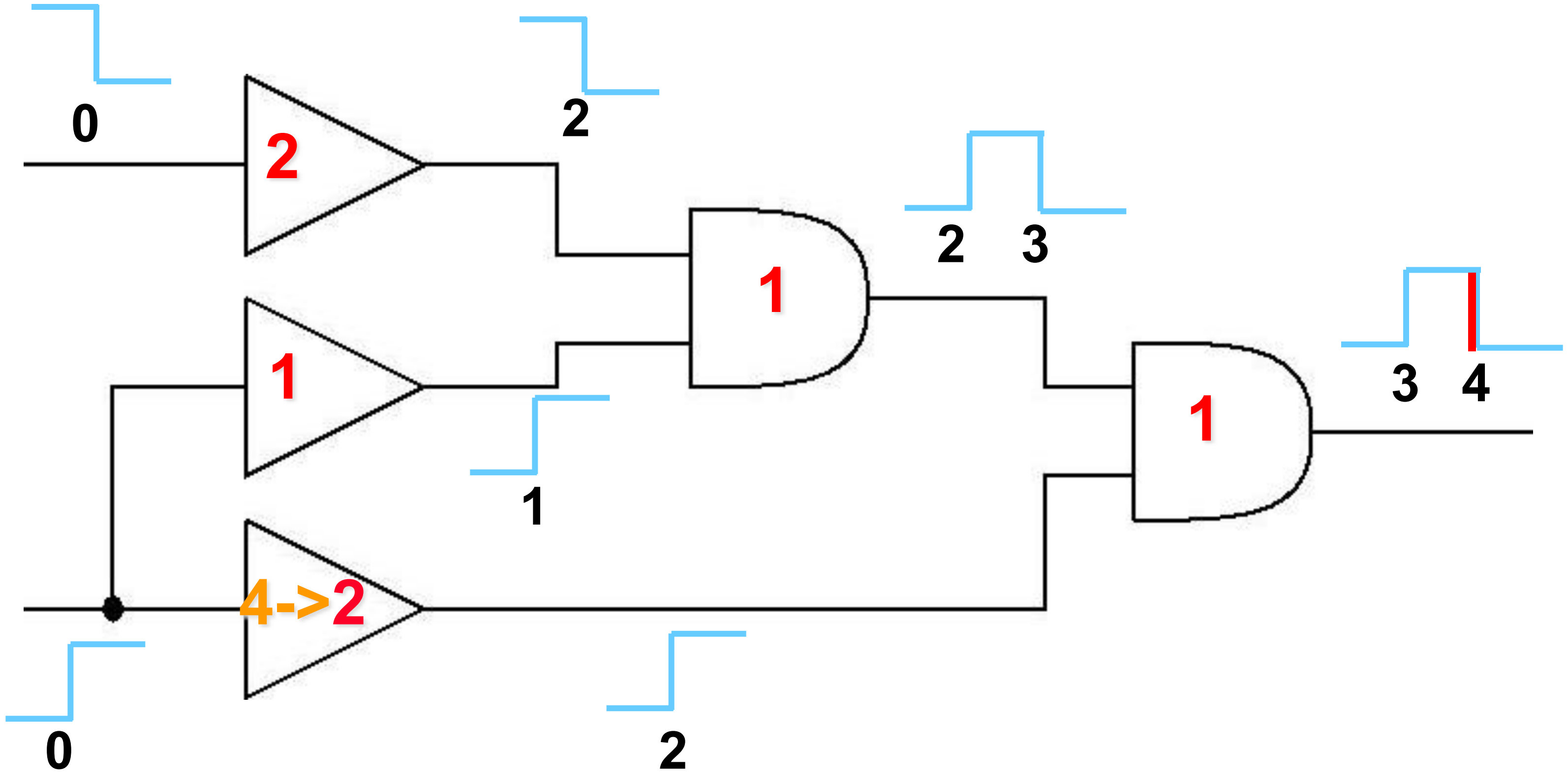
How about timing simulation as a correct method?

Timing Simulation



Implies that delay = 0 for these inputs
BUT!

Timing Simulation



Implies that delay = 4 with the same set of inputs.

What is Wrong with Timing Simulation?

If gate delays are reduced, delay estimates can increase

Not acceptable since

- Gate delays are just upper-bounds, actual delay is in [0,d]
  - Delay uncertainty due to manufacturing
- We are implicitly analyzing a family of circuits where gate delays are within the upper-bounds

Monotone Speedup Property

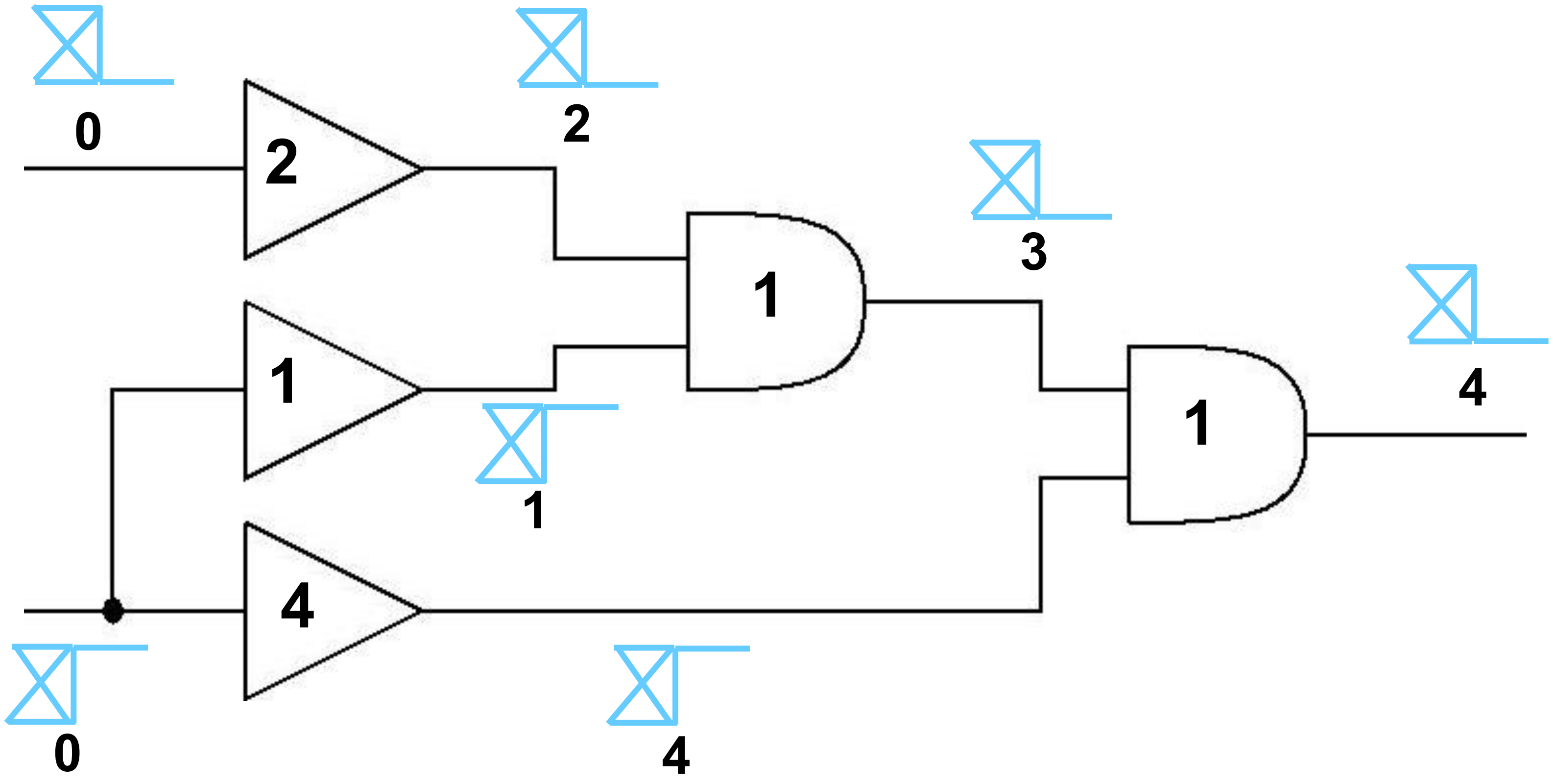
Definition: For any circuit C, if

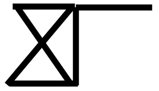
- C’ is obtained from C by reducing some gate delays, and
- delay_estimate(C’) ≤ delay_estimate(C),

then delay_estimate has *Monotone Speedup*

Timing simulation does not have this property

Timing Simulation Revisited




means that the rising signal occurs anywhere between t = -infinity and t = 4.

X-valued simulation

Timing Simulation Revisited

Timed 3-valued (0,1,X) simulation

- called X-valued simulation

Monotone speedup property is satisfied.

Underlying model of

- *floating mode condition* [Chen, Du]
  - Applies to "simple gate" networks only
- *viability* [McGeer, Brayton]
  - Applies to general Boolean networks

False Path Analysis Algorithms

Checking the falsity of every path explicitly is too expensive - exponential # of paths

State-of-the-art approach:

1. Start: set $L = L_{top} - !$ = topological longest path delay - !

   $L_{old} = 0$

2. Binary search:

```
If (Delay(L)) (*)
        $! L = |L-L_{old}|/2,\ L_{old} = L,\ L = L + ! L$
    Else, $! L = |L-L_{old}|/2,\ L_{old} = L,\ L = L - ! L$
If ($L > L_{top}$ or $! L <$ threshold), $L = L_{old}$ , done
```

(*) Delay(L) = 1 if there an input vector under which an output gets stable only at time *t* where $L \leq t$?

Can be reduced to

- a SAT problem [McGeer, Saldanha, Brayton, ASV] or
- a timed-ATPG [Devadas, Keutzer, Malik]

SAT-based False Path Analysis

Decision problem:

Is there an input vector under which the output gets stable only after t = T ?

Idea:

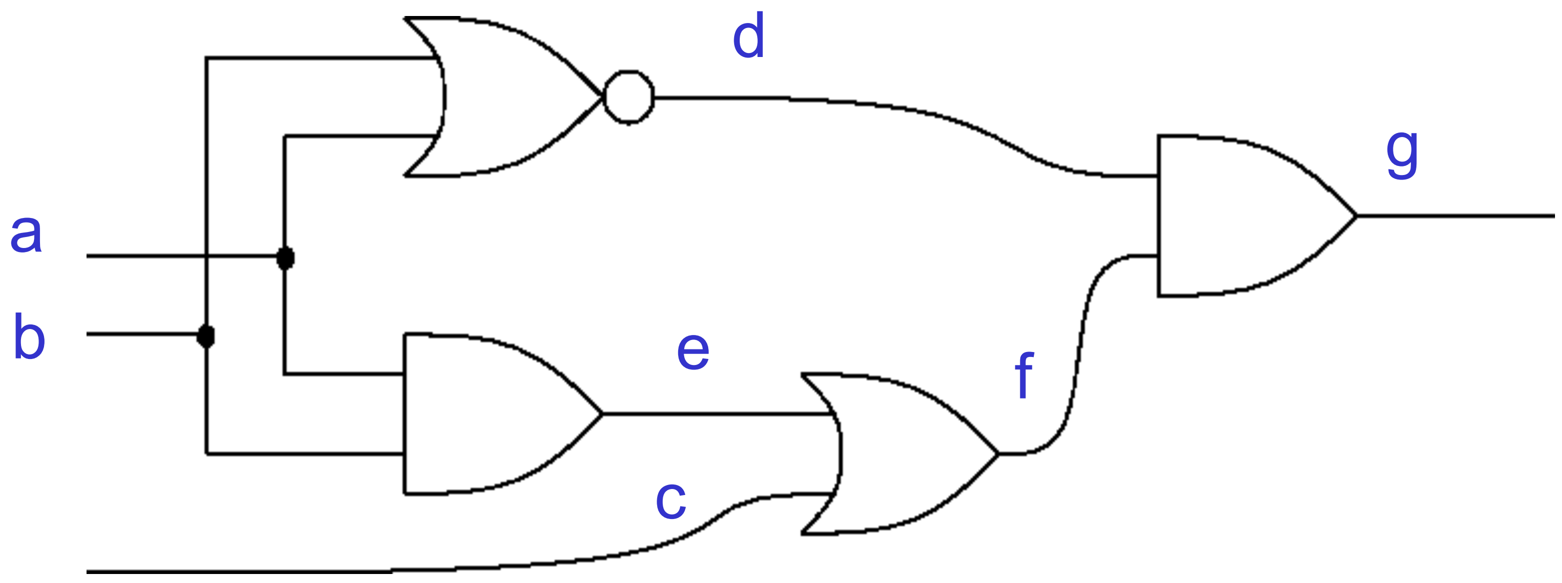
1. characterize the set of all input vectors S(T) that make the output stable no later than t = T
2. check if S(T) contains S = all possible input vectors

   This check is solved as a SAT problem:

   Is S \ S(T) empty? - set difference + emptiness check

   - Let F and F(T) be the characteristic functions of S and S(T)
   - Is F !F(T) satisfiable?

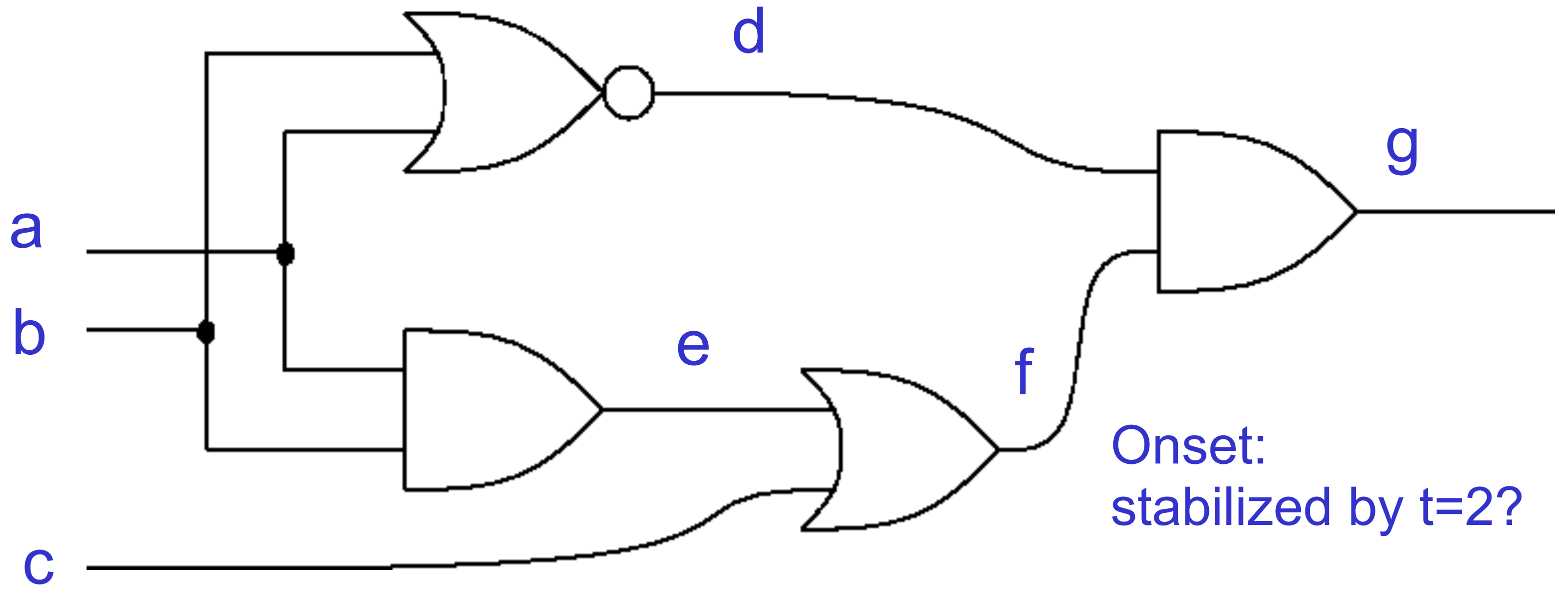
Example



Assume all the PIs arrive at t = 0, all gate delays = 1
Is the output stable time t > 2?

Example

g(1,t=2) : the set of input vectors under which g gets stable to value = 1 no later than t =2

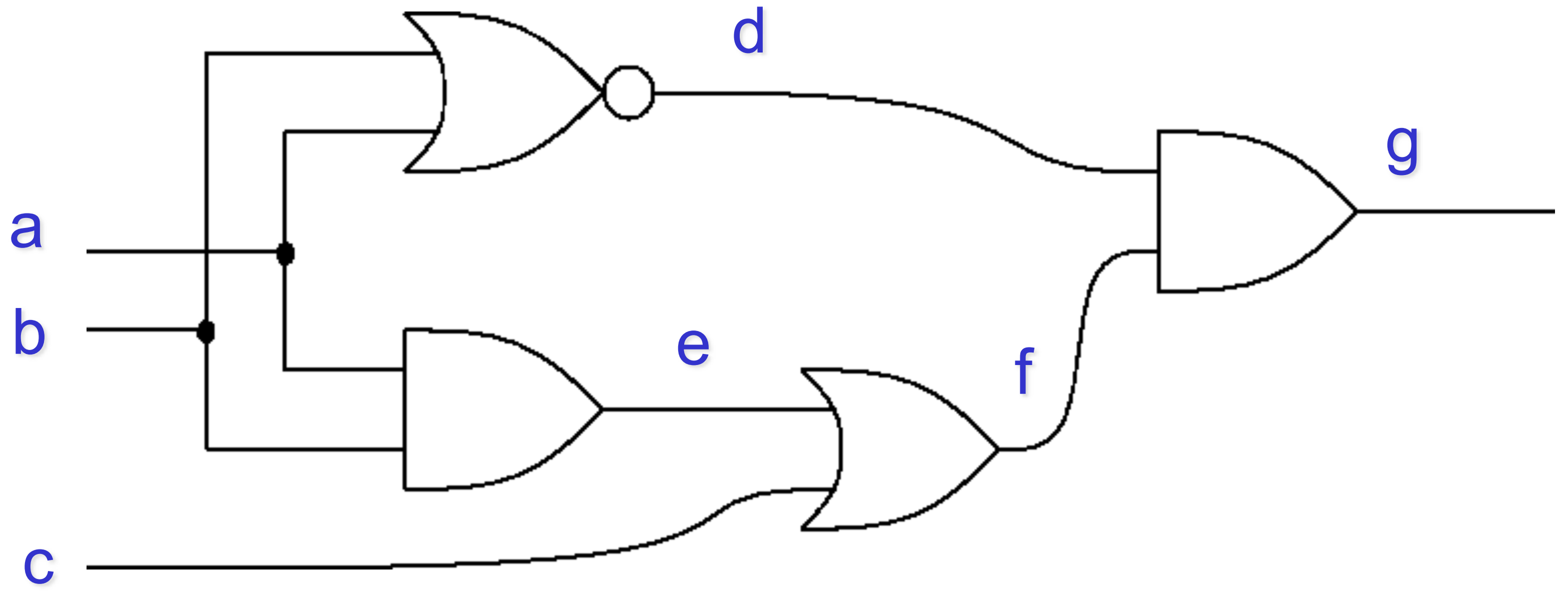


Onset:
stabilized by t=2?

$$g(1,t=2) = d(1,t=1) \cap f(1,t=1)$$
$$= (a(0,t=0) \cap b(0,t=0)) \cap (c(1,t=0) \cup e(1,t=0))$$
$$= !a!b(c \cup \emptyset) = !a!bc = S_1(t=2)$$
$$g(1,t=\infty) = \text{onset} = !a!bc = g(1,t=2) = S_1$$

Example

g(0,t=2) : the set of input vectors under which
g gets stable to value = 0 no later than t=2



$g(0,t=2) = d(0,t=1) \cup f(0,t=1)$

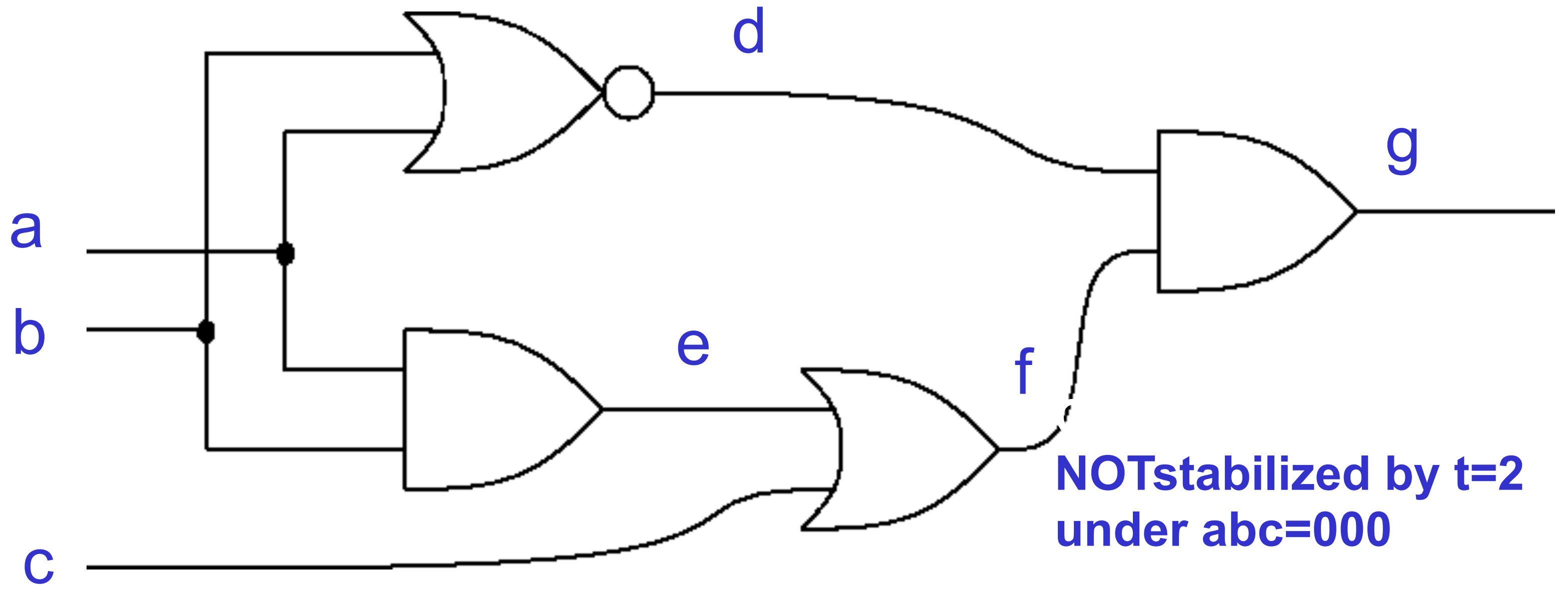
$= (a(1,t=0) \cup b(1,t=0)) \cup (c(0,t=0) \cap e(0,t=0))$

$= (a+b) + (!c \cap \varnothing) = a+b = S_0(t=2)$

$g(0,t=\infty) = \text{offset} = a+b+!c = S_0$

Example

g(0,t=2) : the set of input vectors under which g gets stable to 0 no later than t=2



NOTstabilized by t=2 under abc=000

g(0,t=2) = a+b

g(0,t=∞) = offset = a+b+!c

g(0,t=∞) \ g(0,t=2) = (a+b+!c) !(a+b) = !a !b !c = satisfiable

Summary

False-path-aware arrival time analysis is well-understood

- Practical algorithms exist
  - Can handle industrial circuits easily

Remaining problems

- Incremental analysis (make it so that a small change in the circuit does not make the analysis start all over)
- Integration with logic optimization
- DSM issues such as cross-talk-aware false path analysis

Timed ATPG

Yet another way to solve the same decision problem

A generalization of regular ATPG:

- regular ATPG
  - find an input vector that differentiates a fault-free circuit and a faulty circuit in terms of functionality
- Timed ATPG
  - find an input vector that exhibits a given timed behavior
  - Timed extension of PODEM