

EECS 219B

Spring 2003

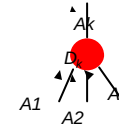
Timing Analysis

Andreas Kuehlmann

1

Timing Analysis - Delay Models

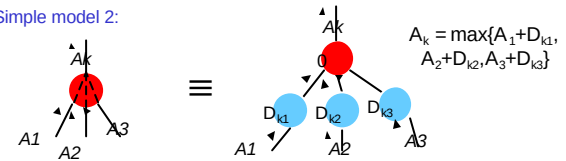
- Simple model 1:



$$A_k = \text{arrival time} = \max(A_1, A_2, A_3) + D_k$$

D_k is the delay at node k , parameterized according to function f_k and fanout node k

- Simple model 2:



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

2

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 ∈ fanin(k)}
    return(k.level)
}

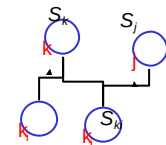
// Compute arrival times:
// Given arrival times on PI's
Algorithm ARRIVAL() {
  for L = 0 to MAXLEVEL
    for {k|k.level = L}
      A_k = MAX{A_{k_i}} + D_k
}
```

3

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

4

Propagating Slacks

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

Since $R_{k \rightarrow k} = R_k - D_k$

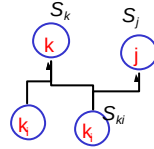
$$S_{k \rightarrow k} = R_{k \rightarrow k} - A_{k \rightarrow k}$$

$$S_{k \rightarrow k} + A_{k \rightarrow k} = R_k - D_k = S_k + A_k - D_k$$

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

$$S_{k \rightarrow k} = S_k + \max \{A_{k_j}\} - A_{k \rightarrow k} \quad k_j, k_j \in \text{fanin}(k)$$

$$S_{k \rightarrow j} = \min \{S_{k \rightarrow j}\} \quad j \in \text{fanout}(k)$$

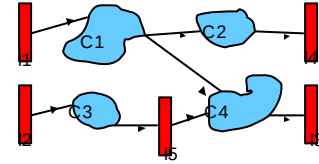


Notes:

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

5

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

6

Static critical paths

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

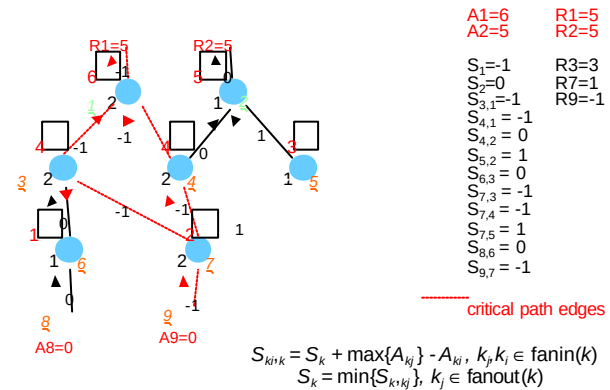
A **static critical path** of a Boolean network is a path $P = \{i_1, i_2, \dots, i_p\}$ where $S_{k_i} + 1_{k \rightarrow k+1} < 0$

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

7

Example: Static critical paths



8

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:

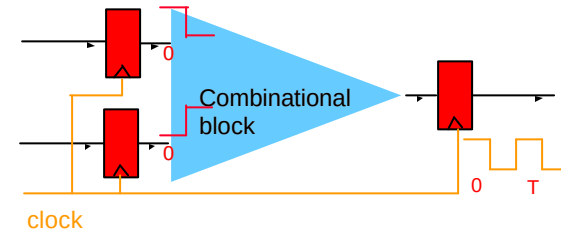
- Don't want false paths (produced by static delay analysis)
- Delay model is **worst** case model. Need to ensure correctness for case where i^{th} gate delay $\leq D_i^M$

9

Functional Timing Analysis

What is Timing Analysis?

Estimate when the output of a given circuit gets stable



10

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

11

Timing Analysis - Basics

Naïve approach - Simulate all input vectors with SPICE

- Accurate, but too expensive

Gate-level timing analysis

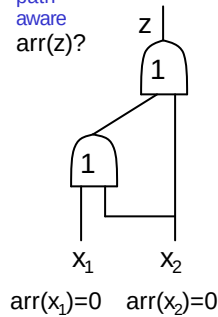
Focus of this lecture

- 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**

12

Gate-level Timing Analysis

False
path
aware
arr(z)?



A naive approach is topological analysis

- Easy longest-path problem
- Linear in the size of a network

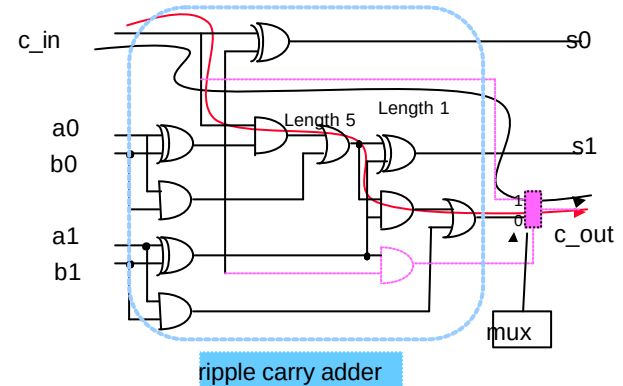
Not all paths can propagate signal events

- 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**

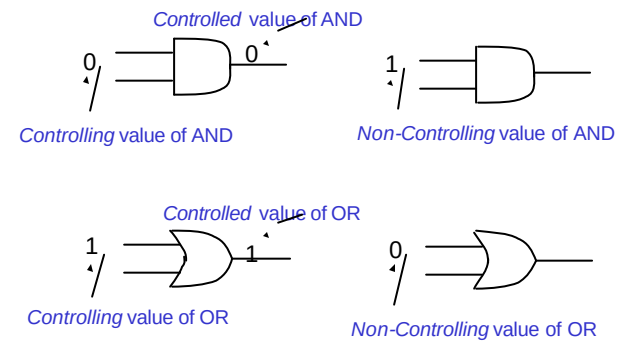
- Can lead to overlooking a timing **violation**

Delay overestimation is not desirable, but **acceptable**

- **Topological** analysis can give overestimate, but never give underestimate

15

Controlling/Non-Controlling Values

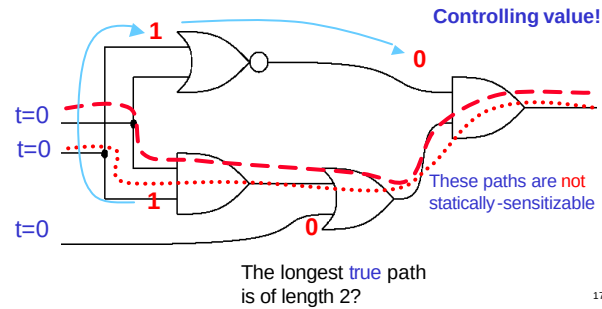


16

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

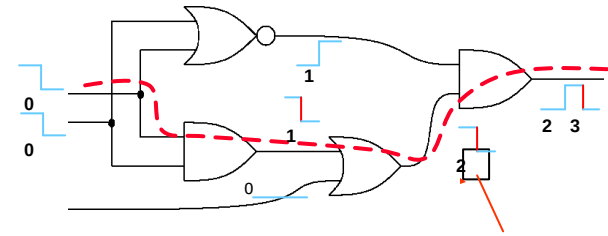
- This is *independent* of gate delays



17

Static Sensitization

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



18

What is Wrong with Static Sensitization?

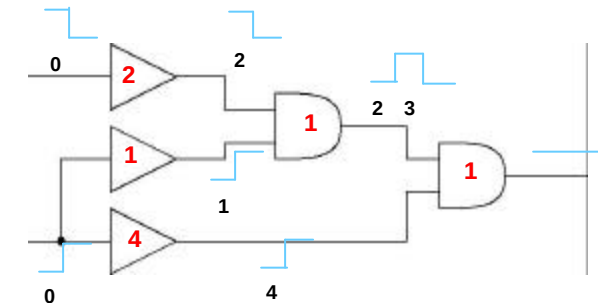
The idea of forcing non-controlling values to side inputs is okay, *but* timing was ignored

- 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?

19

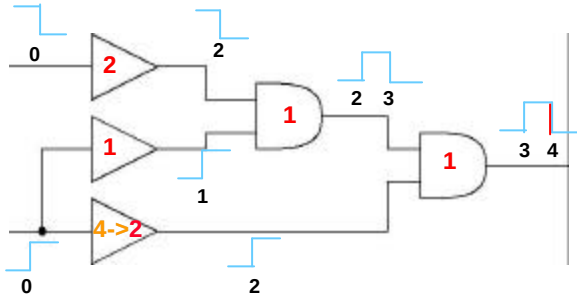
Timing Simulation



Implies that delay = 0 for these inputs BUT!

20

Timing Simulation



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

21

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

22

Monotone Speedup Property

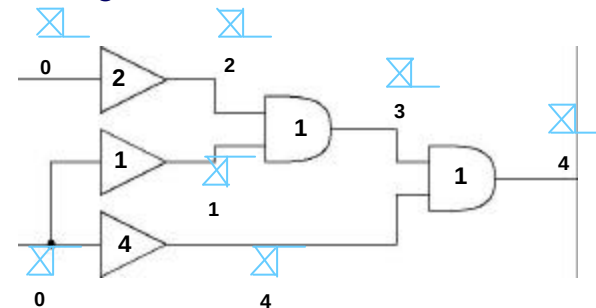
Definition: For any circuit C , if

- C' is obtained from C by reducing some gate delays, and
 - $\text{delay_estimate}(C') \not\leq \text{delay_estimate}(C)$,
- then delay_estimate has **Monotone Speedup**

Timing simulation does **not** have this property

23

Timing Simulation Revisited



X means that the rising signal occurs anywhere between $t = -\infty$ and $t = 4$.

X-valued simulation

24

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 ! $L < \text{threshold}$), $L = L_{old}$, 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

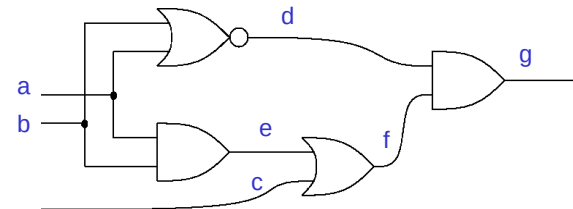
This check is solved as a SAT problem:

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

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

27

Example

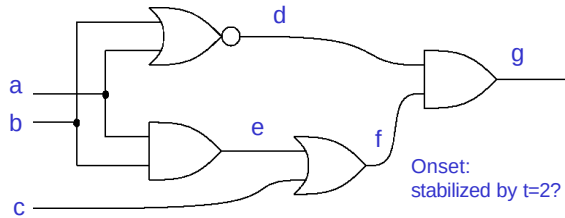


Assume all the PIs arrive at $t = 0$, all gate delays = 1
Is the output stable time $t > 2$?

28

Example

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

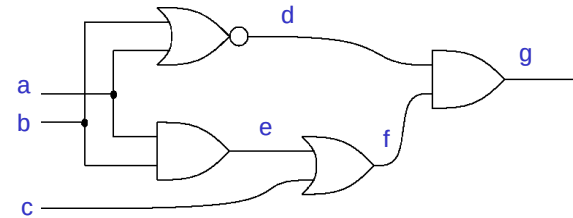


$$\begin{aligned} g(1, t=2) &= d(1, t=1) \cap f(1, t=1) \\ &= (a(0, t=0) \cap b(0, t=0)) \cap (c(1, t=0) \cup e(1, t=0)) \\ &= !a!b(c \cup \emptyset) = !a!bc = S_1(t=2) \\ g(1, t=\infty) &= \text{onset} = !a!bc = g(1, t=2) = S_1 \end{aligned}$$

29

Example

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

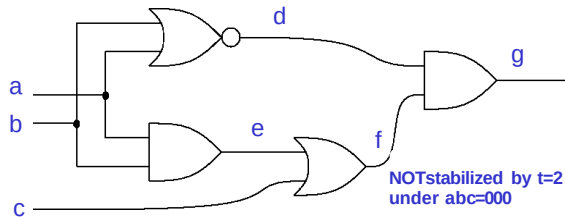


$$\begin{aligned} g(0, t=2) &= d(0, t=1) \cup f(0, t=1) \\ &= (a(1, t=0) \cup b(1, t=0)) \cup (c(0, t=0) \cap e(0, t=0)) \\ &= (a+b) + (!c \cap \emptyset) = a+b = S_0(t=2) \\ g(0, t=\infty) &= \text{offset} = a+b+!c = S_0 \end{aligned}$$

30

Example

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



$$\begin{aligned} g(0, t=2) &= a+b \\ g(0, t=\infty) &= \text{offset} = a+b+!c \\ g(0, t=\infty) \setminus g(0, t=2) &= (a+b+!c) \setminus (a+b) = !a!b!c = \text{satisfiable} \end{aligned}$$

31

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

32

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