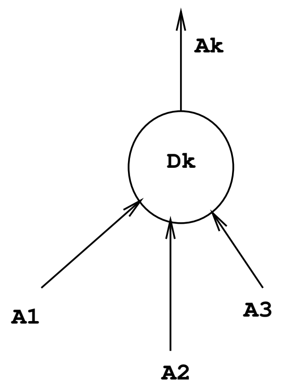
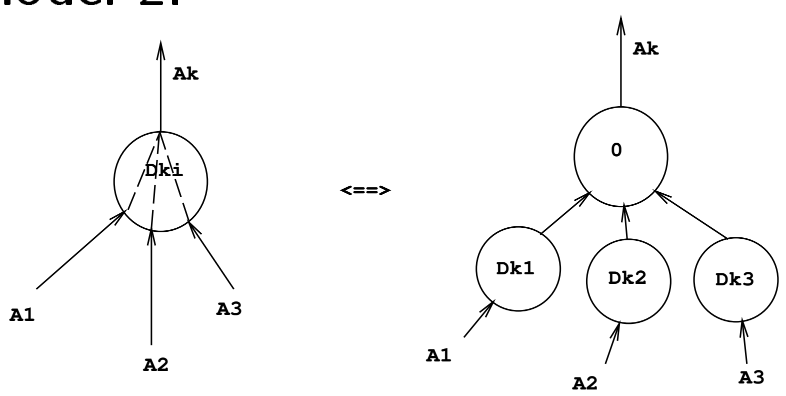
Timing Analysis - Delay Analysis Models

Simple model 1:



A_k = arrival time
$A_k = \max\{A_1, A_2, A_3\} + D_k$
D_k is the delay at node k, parametrized according to function f_k and fanout of node k

Simple model 2:



$$A_k = \max\{A_1 + D_{k_1}, A_2 + D_{k_2}, A_3 + D_{k_3}\}$$

Can also have different times for rise time delay and fall time delay.

Static delay analysis

Levelize Nodes:

/* Level of PI nodes initialized to 0,
rest to -1. Invoke LEVEL from PO */
LEVEL(k)
If k.level $\neq -1$
    Return k.level
Else
    k.level $\leftarrow 1 +$ max$\{$LEVEL$(k_i) | k_i \in$ fanin$(k)\}$
Return k.level

Compute arrival times:

/* Given arrival times on PI's */
For $1 \leq L \leq$ MAXLEVEL
    For $\{k | k$.level $= L\}$
        $A_k \leftarrow$ max$\{A_{k_i}\} + D_k$

Required times and Slacks

Required Times: Given required times on primary outputs

Traverse in reverse topological order (i.e from primary outputs to primary inputs)

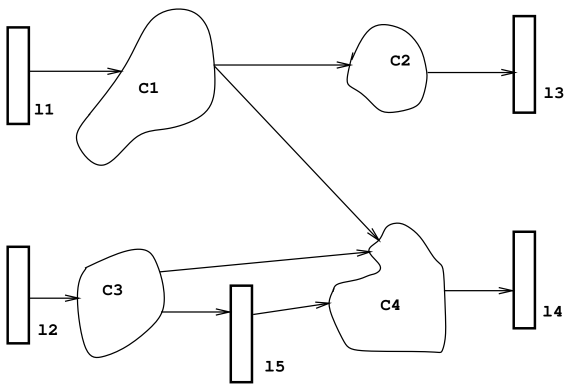
If (k_i, k) is an edge between k_i and k,
$R_{k_i,k} = R_k - D_k$

Hence, the required time of output of node k is $R_k = \min\{R_{k,k_j} | k_j \in \text{fanout}(k)\}$

Slacks: Slack at the output of node k is
$S_k = R_k - A_k$
Since $R_{k_i,k} = R_k - D_k$
$S_{k_i,k} = R_{k_i,k} - A_{k_i}$
$S_{k_i,k} + A_{k_i} = S_k + A_k - D_k$
Since $A_k = \max\{A_{k_j}\} + D_k$
$S_{k_i,k} = S_k + \max\{A_{k_j}\} - A_{k_i}, \quad k_j, k_i \in \text{fanin}(k)$
$S_k = \min\{S_{k,k_j}\} \quad k_j \in \text{fanout}(k)$

Note that each edge in the graph has a slack and a required time

Sequential networks



Arrival times known at l_1 and l_2
Required times known at l_3, l_4, and l_5.

Delay analysis gives arrival and required times (hence slacks) for C_1, C_2, C_3, C_4

Note: Latch l_5 may be transparent

t_1 is the beginning of latch transparent time
Arrival time at l_5_output =
Max{Arrival Time at l_5_input, t_1}
$+D_{\text{latch}}$(hold)

Required time at l_5_input =
Min{Required Time at l_5_output, t_2}
$-D_{\text{latch}}$(set_up)

Static critical paths

Min-Max problem:

Minimize Max$\{-S_i, 0\}$

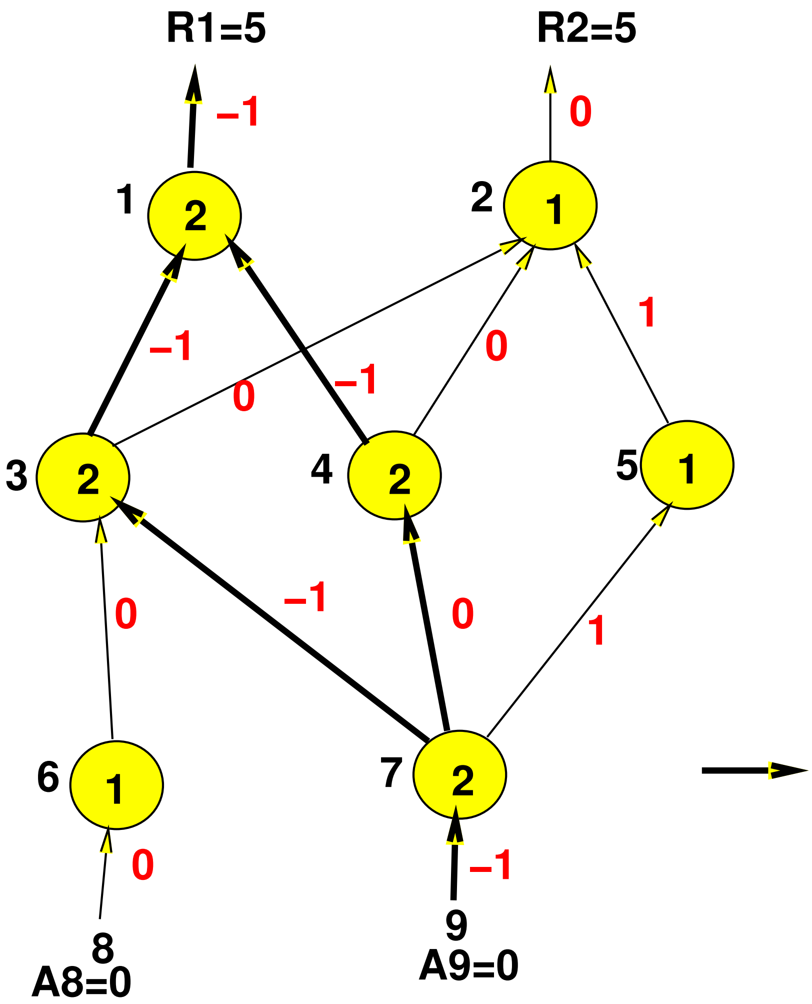
A **static critical path** of a Boolean network is a path $P = \{i_1, i_2, ..., i_p\}$ where

$S_{i_k, i_{k+1}} < 0$

Note: If 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

S1 = –1 R3 = 3
S2 = 0 R7 = 1
S31 = –1 R9 = –1
S73 = –1
S97 = –1

S42 = 0
S52 = 1
S74 = –1
S97 = –1

critical path edges

$$S_{k_i,k} = S_k + \max_{k_j}\{A_{k_j}\} - A_{k_i} \quad k_i, k_j \in \text{fanin}(k)$$
$$S_k = \min\{S_{k,k_j}\} \quad k_j \in \text{fanout}(k)$$

Timing analysis problems

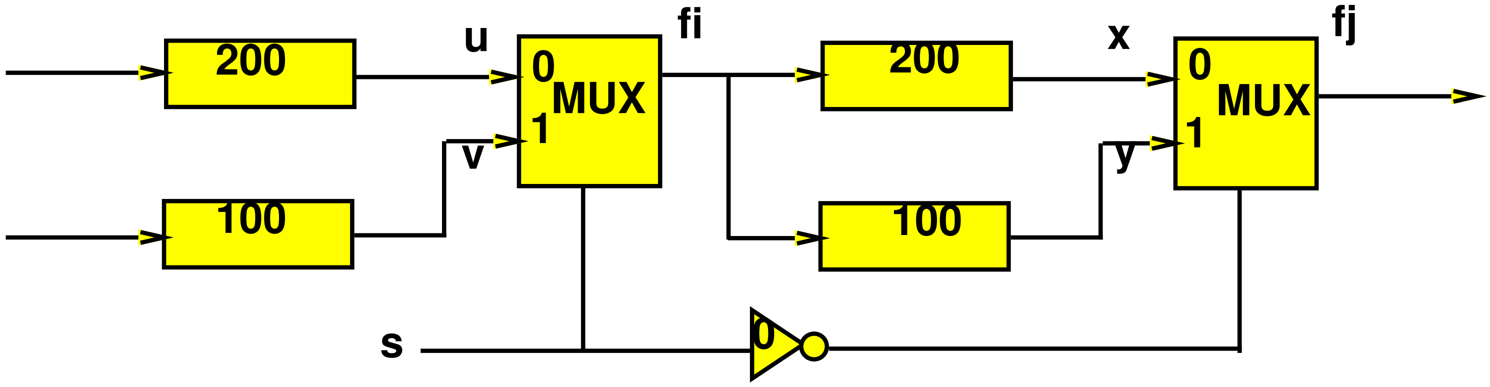
- We want to determine the true critical paths of a circuit in order to:
  - ◇ To determine the minimum cycle time that the circuit will function
  - ◇ To identify critical paths for performance optimization - don't want to try to optimize the wrong (non-critical) paths.

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 $\leq D_i^M$

False paths

Static analysis is fast but leads to **false paths**
Path of length 400 is never "exercised"



Approaches:

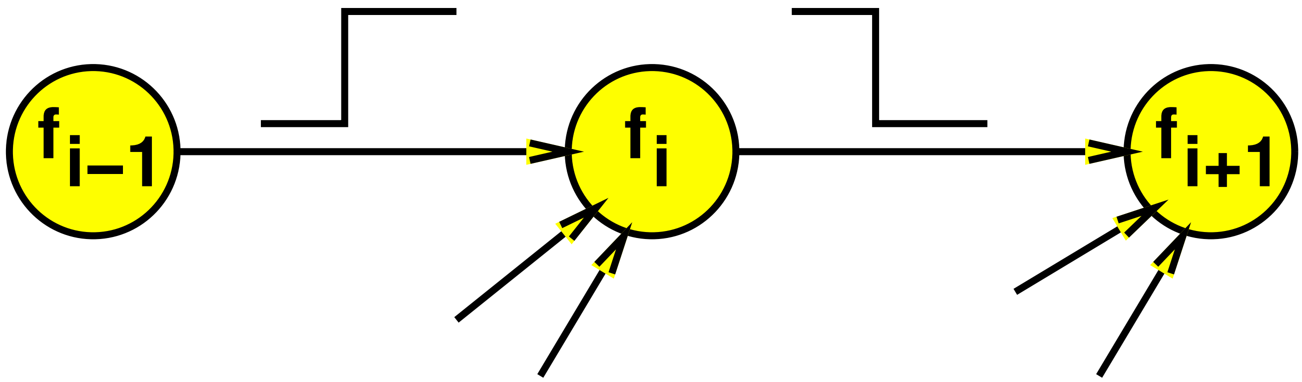
◇ Mark orthogonal pairs
(may be wrong, can't find all possibilities,
is the method correct?)

◇ Throw out "non-sensitizable" (false) paths

- Circuit delay = Length of longest path?
  Not a good enough bound (too pessimistic)
- Circuit delay = Time of last output change

This leads to functional timing analysis for false paths

First attempt: Boolean difference



Intuition:
Check for "static false path":

Path $P = \{f_0, f_1, f_2, ..., f_n\}$
$\frac{\partial f_i}{\partial f_{i-1}}$ gives conditions under which node f_i is "sensitive" to node f_{i-1}
So output of P is sensitive to f_0 if

$$\prod_{i=1}^{n} \frac{\partial f_i}{\partial f_{i-1}} \neq 0$$

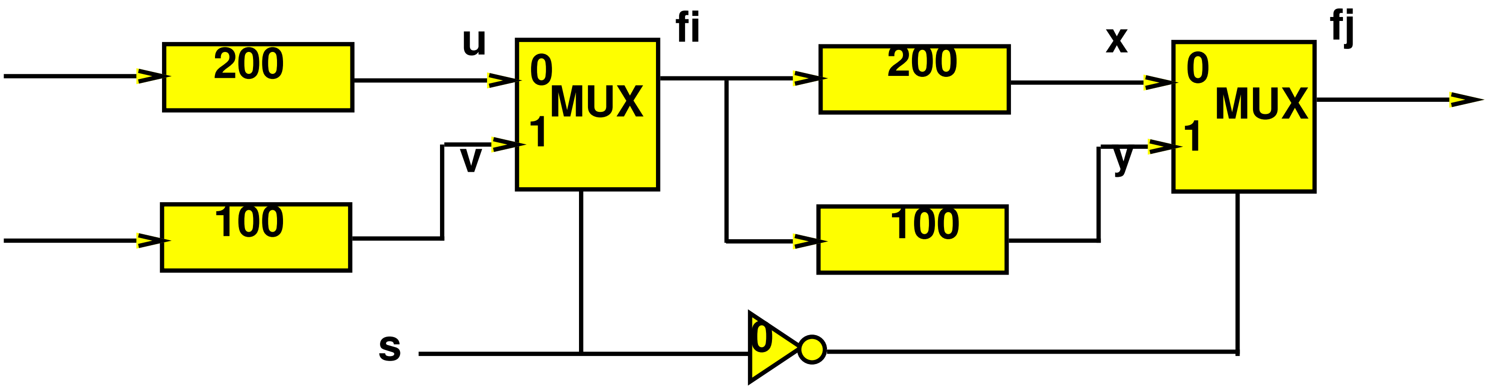
Recall Boolean difference:

$$\frac{\partial f}{\partial x} = f_x \oplus f_{\overline{x}}$$

Example: $f = x\overline{y} + \overline{xz}$

$$\frac{\partial f}{\partial x} = \overline{y}z + y\overline{z}$$

Example: Static false path



$f_i = su + \overline{s}v$ and $f_j = \overline{s}x + sy$

$$\frac{\partial f_i}{\partial u} = (s+v)(s+\overline{v}) + \overline{sv}(\overline{s}v) = s$$

$$\frac{\partial f_j}{\partial x} = \overline{s}$$

Hence, $\dfrac{\partial f_i}{\partial u} . \dfrac{\partial f_j}{\partial x} = 0$

Thus (by previous condition) any path

$$P = \{f_0, ..., u, f_i, ..., x, f_j, ...\}$$

is not "statically sensitizable" and is "false"

Definitions

Given a **simple gate** (i.e. AND, OR, NAND, NOR), a **controlling** value on an input determines the output of the gate independent of the other inputs

Given a **simple gate** (i.e. AND, OR, NAND, NOR), a **non-controlling** value on an input cannot determine the output of the gate independent of the other inputs

Example: 0 is a controlling value for AND gate. 1 is non-controlling value for AND gate

Note: Controlling / non-controlling value is merely a specialization of the Boolean difference to simple gates

Static sensitization

Simple Gates:

Let path $P = \{f_0, f_1, ..., f_i\}$

A side-input to a gate f_i along P is any input other than f_{i-1}

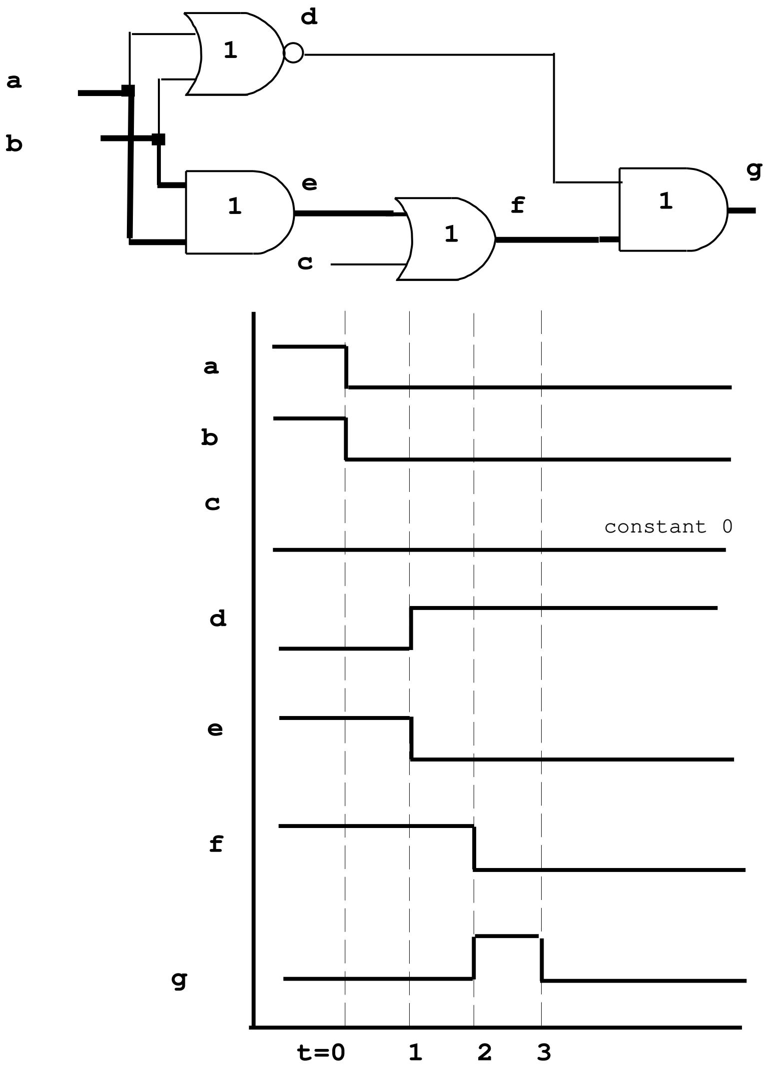
An event is a transition from 0 to 1 or 1 to 0

Path P is **statically sensitizable** if there exists a primary input vector under which every side-input is set to a non-controlling value

A path is a "statically false path" if it is not statically sensitizable (see previous example)

Static sensitization and false paths

- Static sensitization is wrong!



Paths shown in bold are not statically sensitizable, but delay of circuit is 3

Why static sensitization fails

Static sensitization fails because it considers only the final value on each side-input. It does not consider values on side-inputs at the moment the event propagates from f_{i-1} through node f_i

For example, in previous circuit when determining static sensitization of path $\{b, e, f, g\}$ we assume side-input a of gate e is at final non-controlling value of 1. This is not necessary for the path to be sensitizable

Dynamic sensitization

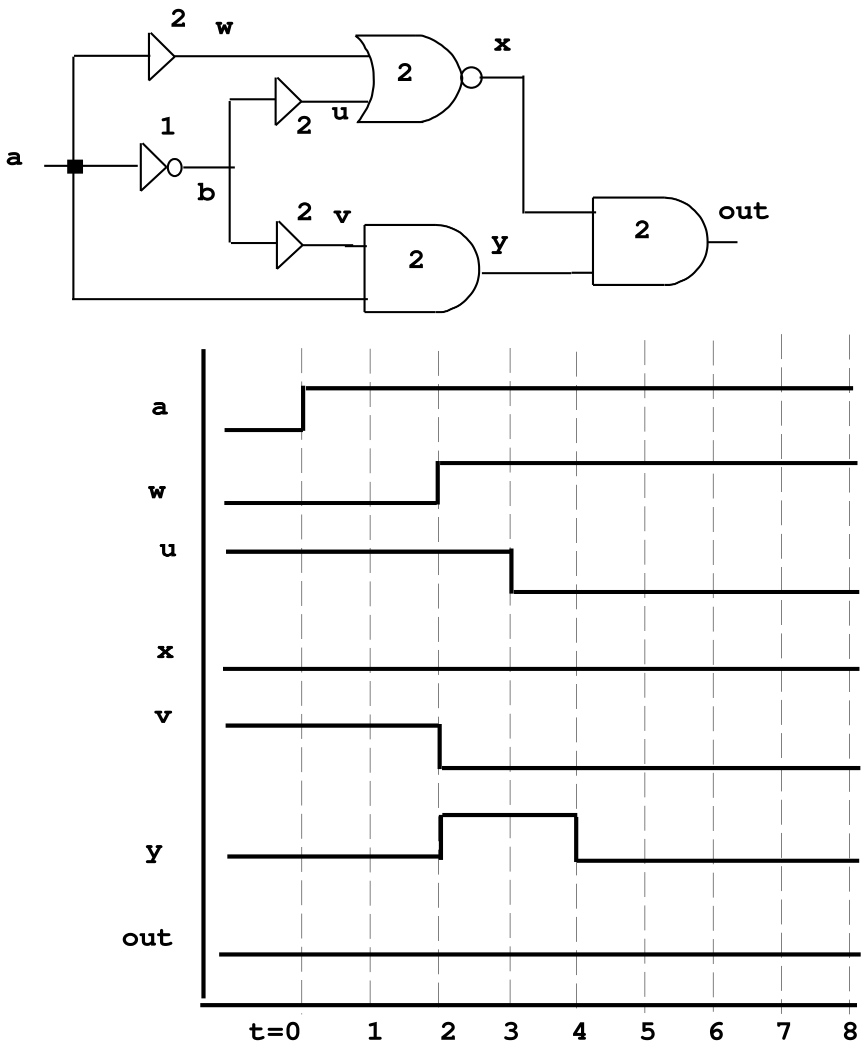
Extend static sensitization condition to include the time that the event occurs

Let τ_{i-1} denote the time that input f_{i-1} is available to node f_i. Note

$$\tau_{i-1} = \sum_{k=1}^{i-1} D(f_{k-1}, f_k)$$

A path $P = \{f_0, f_1, f_2, ..., f_n\}$ is **dynamically sensitizable** by a vector pair v_0, v_1, where v_0 is applied at time $t = -\infty$ and v_1 is applied at time $t = 0$ such that each side input to f_i is non-controlling value at time τ_{i-1}

Example: dynamic sensitization

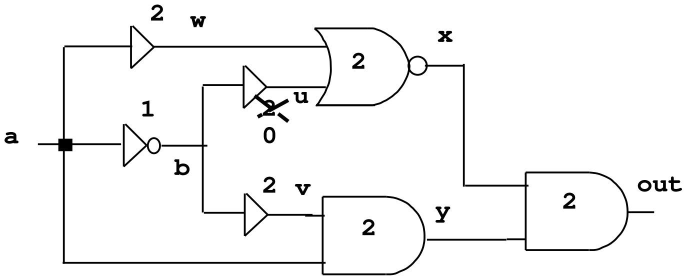


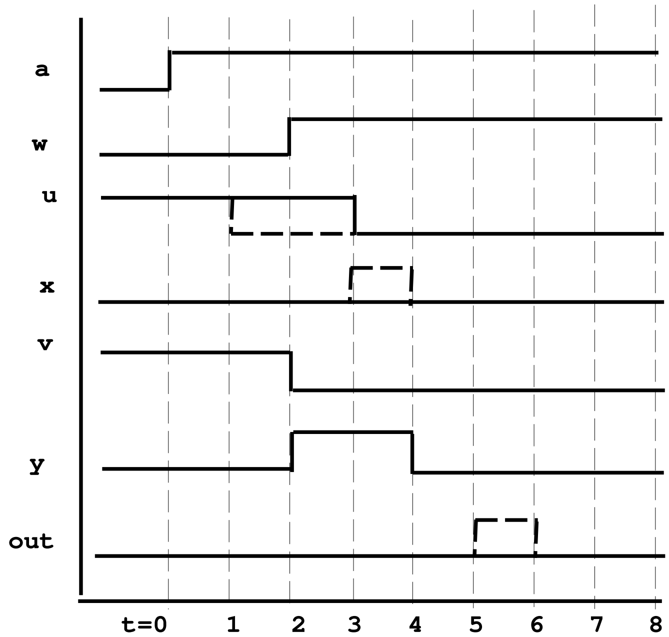
Dynamic sensitization works for given (fixed) delay assignment for single network

But does it work if some delays are decreased?

Dynamic sensitization failure

Speed up delay of buffer between b and u from 2 to 0





Delay estimate increases from 0 to 6 !
Dynamic sensitization invalid when delays are reduced

Constraints on delay analysis

Goal is to eliminate false paths from consideration but we must guarantee:

1. Delay is not underestimated
   It is okay (but undesirable) to overestimate delay (static analysis (longest topological path) does this, leading to longer clock period)

2. timing analysis estimate must remain valid under **any allowable** assignment of delays
   although for each gate/connection, a maximum delay is specified, the actual delay depends on many factors (input waveform shape, operating temparature, process variations, etc.) Thus the delay on a gate/connection is really a range between $[D_{\text{min}}, D_{\text{max}}]$. *Often* $D_{\text{min}} = 0$ *(unbounded delay model)*

3. the **monotone speedup property** should hold
   the delay estimate is valid for any assignment where $D^c \leq D^c_{\text{max}}$, *i.e. can't increase delay estimate by speeding up gates/connections in the network*

Correct delay analysis techniques - A Local History

Single vector analysis:

- [McGeer, Brayton 88]: Defined **monotone speedup** criterion. Defined **viability** analysis on paths in networks with **complex symmetric** gates. Proved viability analysis obeys monotone speedup. Could not prove viability estimate was exact bound on delay of complex symmetric gate circuit
- [Chen, Du 89]: Worked on **simple gates** for exact timing analysis. Simpler formulation (called Chen-Du or floating delay model). Returns **same delay** estimate as viability. Applied by [Devadas, Keutzer, Malik 91] to efficient delay estimation (multi-path analysis).

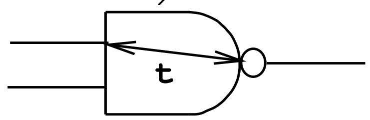
Two (and multi) vector analysis:

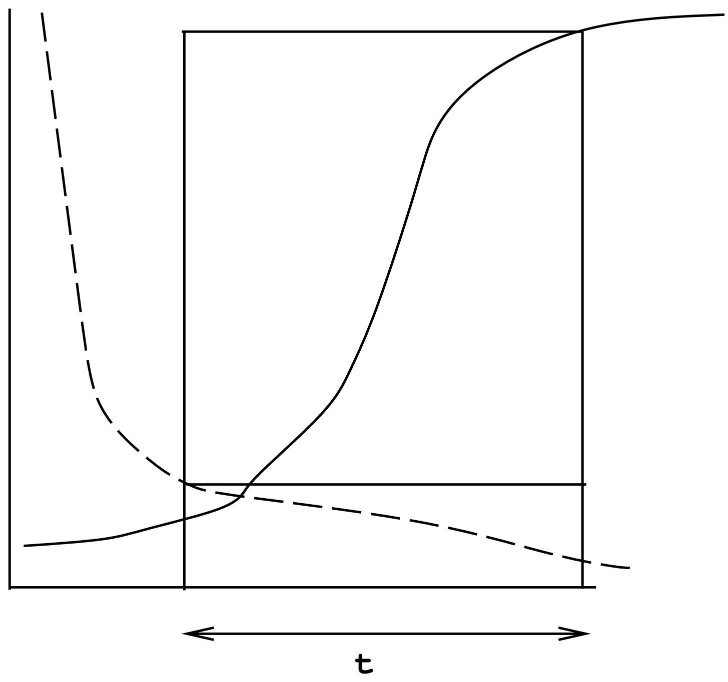
- [McGeer, Saldanha, Brayton, Sangiovanni 92 - in Reader]: Introduced notion that timing analysis is linked with delay model. Proved **exact** timing analysis criterion for **complex** (possibly asymmetric) gates under common delay models
- [Lam, Brayton, Sangiovanni 93]: Applied multi-vector delay analysis to **sequential** circuits

Many other papers in literature

Delays and delay models

What is the delay of a gate? (actually the delay of a connection, since all delays on gates may be modeled by delays on connections)





The delay t is a computed quantity:

- ◇ Computation done via a **delay model**
- ◇ Model captures delays and uncertainties

Framework for delay models and analysis

Use a ternary algebra

$$\mathcal{T} = \{0, 1, X\}$$

X represents two quantities:

⋄ Indeterminate state (signal neither 0 nor 1)

⋄ Unknown state (signal either 0 or 1)

Algebra: Standard Boolean algebra over 0, 1 plus

$$\begin{aligned} \textbf{NOT } X &= X \\ X \textbf{ OR } 0 &= X \\ 1 \textbf{ AND } X &= X \\ X \textbf{ OR } 1 &= 1 \\ X \textbf{ AND } 0 &= 0 \end{aligned}$$

Note: $(X \overline{\oplus} X) = (X \oplus X) = X$

Ternary algebra over complex functions:
Introduce a partial order $\preceq$ over $\mathcal{T}$

$i \preceq i \qquad \forall i \in \mathcal{T}$
$i \preceq X \qquad \forall i \in \mathcal{T}$

$\preceq$ represents increasing instability, *i.e.* $a \preceq b$ means b is more unstable than a

$\preceq$ extends to vectors
$\vec{a} \preceq \vec{b} \Leftrightarrow a_k \preceq b_k \qquad \forall k$

Note correspondence of ternary vectors to cubes (a 2 is an X, and $\preceq$ is isomorphic to $\subseteq$)

Ternary evaluation on complex gates

Let $g(y_1, y_2, ..., y_n)$ denote the function of gate G with n inputs, and let Y_i denote the value of input y_i of G

Then:
$g(Y_1, Y_2, ..., Y_n) = 1 \Leftrightarrow$
$g(Z_1, Z_2, ..., Z_n) = 1 \forall Z_1 Z_2 ... Z_n \preceq Y_1 Y_2 ... Y_n$

Equivalently,
$g(Y_1, Y_2, ..., Y_n) = 1 \Leftrightarrow$
$Y_1 Y_2 ... Y_n \preceq p$ for some prime p of $g(y_1, y_2, ... y_n)$

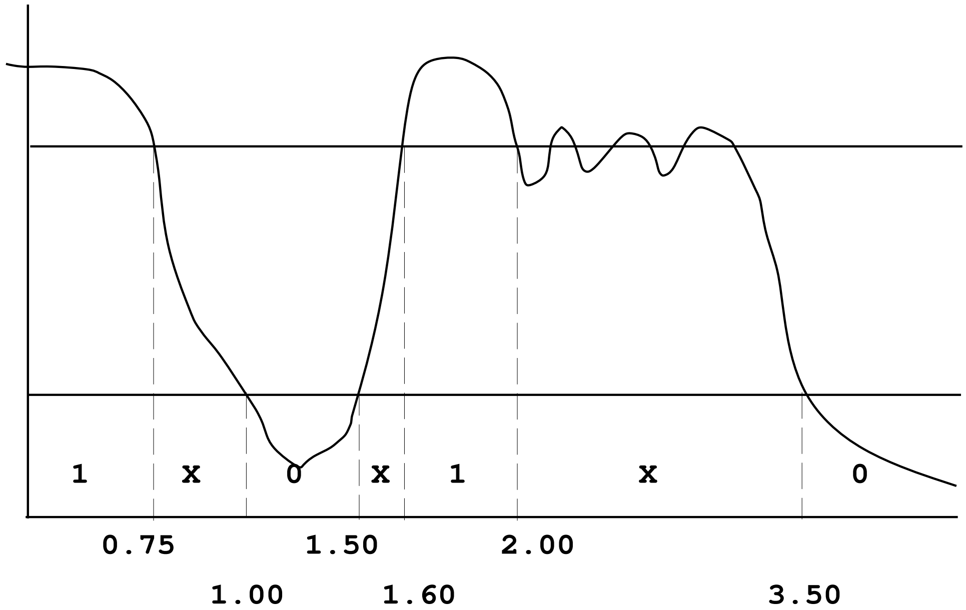
Similarly,
$g(Y_1, Y_2, ..., Y_n) = 0 \Leftrightarrow$
$Y_1 Y_2 ... Y_n \preceq q$ for some prime q of $\overline{g}(y_1, y_2, ... y_n)$

Hence
$g(Y_1, Y_2, ..., Y_n) = X \Leftrightarrow Y_1 Y_2 ... Y_n$ is $\not\preceq$ a prime of $g(y_1, y_2, ... y_n)$ and $\not\preceq$ prime of $\overline{g}(y_1, y_2, ... y_n)$

Note: Gate output is constant if input values are contained in a prime in the on-set or off-set. Evaluation is independent of gate structure. This is the correct generalization of controlling values for simple gates.

Ternary waveforms

$\mathcal{W} : \mathcal{R} \Rightarrow 0, 1, X$



Notation:
$\mathcal{W}_I$ denotes waveform $\mathcal{W}$ restricted to interval I
$\mathcal{W}_I \mathcal{W}'_J$ denotes concatenation of $\mathcal{W}_I$ and $\mathcal{W}'_J$

Example waveforms are:

$1_{[-\infty,0.75)} X_{[0.75,1.00]} 0_{(1.00,1.50)} X_{[1.50,1.60]}$

$1_{(1.60,2.00)} X_{[2.00,3.50]} 0_{(3.50,\infty]}$

Ternary waveforms

$\mathcal{W} : \mathcal{R} \Rightarrow 0, 1, X$ is **ternary waveform** if and only if:

- ⋄ **total:** $\mathcal{W}(t)$ is defined for all $t \in \mathcal{R}$
- ⋄ **continuous:** $t_1 < t_2$ and $\mathcal{W}(t_1) \neq \mathcal{W}(t_2)$ implies $\exists t_0$, $t_1 < t_0 < t_2$ such that $\mathcal{W}(t_0) = X$

By convention, signal values are X over closed intervals, and 0 and 1 over open intervals

Waveform algebra terms: Input waveform: waveform on an input variable

Waveform vector: vector of input waveforms

Waveform space $\mathbf{W}^n$: set of all possible waveform vectors

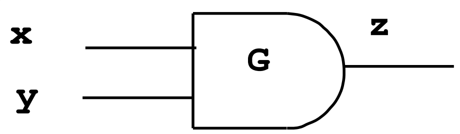
$\mathcal{W}^g$ denotes the waveform on signal g

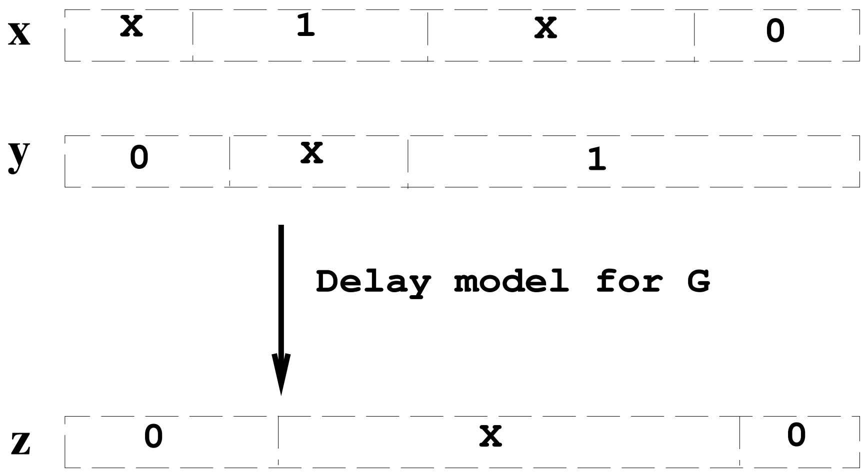
Computing waveforms

Given input waveforms, want to determine output waveform

- Simulate
  - ◇ Too many waveform vectors
  - ◇ Uncertain delays

- Compute using delay model





Delay model over waveforms

- Delay model for gate G with output $z = g(x, y)$ is a function of the form:

$$\mathcal{W}^z(t) = f(\mathcal{W}^x_{(0,t)}, \mathcal{W}^y_{(0,t)}, \mathcal{W}^z_{(0,t)})$$

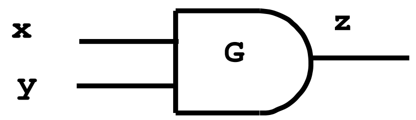
where f is the computation rule of the delay model.

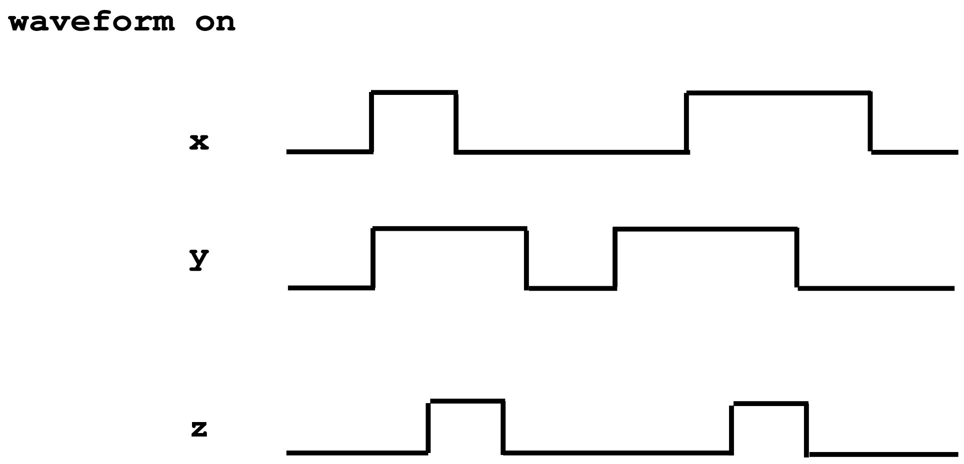
Many common delay models are easily captured in this framework...

Delay model example

- Pure binary, pure delay (PBPD)
  - ◇ Square wave model
  - ◇ Assumes no variance in delay
  - ◇ X only on intervals of measure 0
  - ◇ If $z = f(y_1, y_2, ..., y_n)$ then

$$\mathcal{W}^z(t) = f(\mathcal{W}^{y1}(t - D(z, y_1)), \ldots, \mathcal{W}^{yn}(t - D(z, y_n)))$$





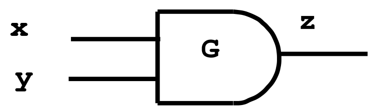
Delay model example

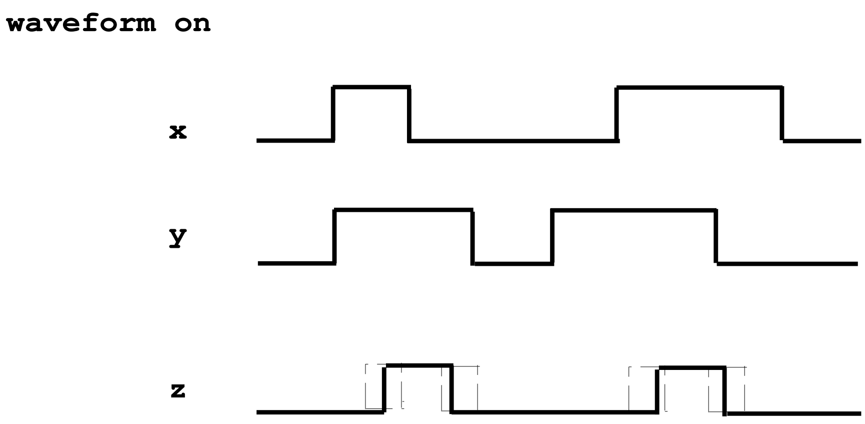
- Pure binary, variable delay (PBVD)
  - ◇ Square wave model
  - ◇ Delay variation is

$$D^{\text{min}}(z, y_j) \leq D(z, y_j) \leq D^{\text{max}}(z, y_j)$$

  - ◇ X only on intervals of measure 0
  - ◇ Same evaluation rule as PBPD
  - ◇ Feasible values of $D(z, y_j)$ determined by mixed Boolean / linear program
  - ◇ Assumes variance in delay is static

This model is also called **transition delay** model





XBD Delay model

- Extended bounded delay model (XBD)
  - ◇ Slope delay model
  - ◇ X on finite intervals
  - ◇ Complex evaluation rule given $D^{\mathsf{min}}(z, y_j) \leq D^{\mathsf{max}}(z, y_j)$
  - ◇ Models static and dynamic delay variations

- Extended bounded delay-0 model (XBD0)
  - ◇ XBD model with $D^{\mathsf{min}}(z, y_j) = 0$ for all z, y_j
  - ◇ Model for single vector criterion for delay analysis (*e.g.* viability analysis, floating delay)

Methods for computing waveforms

Two approaches:

(1) Simulate:
- ⋄ Too many waveform vectors

(2) Implicit computation:
- ⋄ Use delay model evaluation rule
- ⋄ Convert to Boolean domain

Sensitization and delay calculation:
Let the characteristic Boolean function of waveform vectors satisfying a condition C be denoted χ^C

Example: $\chi^{\{G(t)=1\}}$ is the Boolean function whose on-set contains only the vectors such that $G(t) = 1$

Delay of a gate G with output z:
Smallest t such that $Z_{(t,\infty)}$ is a stable binary constant (*typically, we restrict input waveform vectors to be stable on* $[0, \infty)$)

Set of waveform vectors such that the delay of $z \leq t$:

$$\chi^{\{Z_{(t,\infty)}=1_{(t,\infty]}\}} + \chi^{\{Z_{(t,\infty]}=0_{(t,\infty]}\}}$$

Least t for which this is the universal set (i.e. tautology) is the delay of G

XBD0 evaluation: a simplification

It is proved in [McGeer et.al in reader] that for the XBD0 model, each waveform is of the form ($a \in \{0,1\}$):

- $a_{[-\infty,0)} X_{[0,t]} \overline{a}_{(t,\infty]}$ or
- $a_{[-\infty,0)} X_{[0,t]} a_{(t,\infty]}$ or
- $a_{[-\infty,\infty]}$

where is is assumed for inputs, that $t = 0$ *i.e.*, only the last transition on a waveform is relevant! (*because* $D_{min} = 0$, X *immediately propagates to all points*). This means that inputs can be characterized by a single vector $a_{(0,\infty]}$, since $a_{[-\infty,0)}$ is irrelevant.

Hence for $t > 0$

$$\begin{aligned} \chi^{\{G_{(t,\infty]}=0\}} &= \chi^{\{G(t)=0\}} \\ \chi^{\{G_{(t,\infty]}=1\}} &= \chi^{\{G(t)=1\}} \end{aligned}$$

XBD0 evaluation rule:
Gate G with output $z = g(y_1, y_2, ..., y_n)$. Then,

$$Z(t) = g(Y_1(t), Y_2(t), ..., Y_n(t))$$

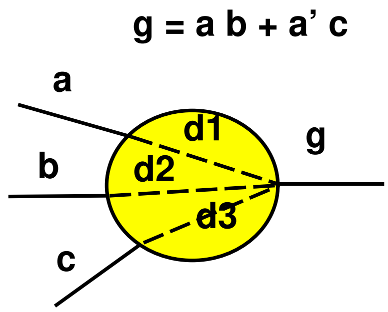
where $Y_j(t)$ denote **"effective value"** of y_j at t:

$Y_j(t) = i$ if $\mathcal{W}^{Y_j}_{(t-D^{\max}(z,y_j),t)} = i, \ i \in \{0,1\}$

$Y_j(t) = X$ otherwise

Condition follows because we only need to consider last transition on waveform.

Example: XBD0 evaluation



Want to determine $\chi^{\{G(t)=1\}}$
Primes of on-set of $g = ab + a'c + bc$

$$
\begin{aligned}
\chi^{\{G(t)=1\}} = \quad & \chi^{\{A(t-d1)=1\}}\chi^{\{B(t-d2)=1\}}\chi^{\{C(t-d3)=0,1,X\}} + \\
& \chi^{\{A(t-d1)=0\}}\chi^{\{B(t-d2)=0,1,X\}}\chi^{\{C(t-d3)=1\}} + \\
& \chi^{\{A(t-d1)=0,1,X\}}\chi^{\{B(t-d2)=1\}}\chi^{\{C(t-d3)=1\}}
\end{aligned}
$$

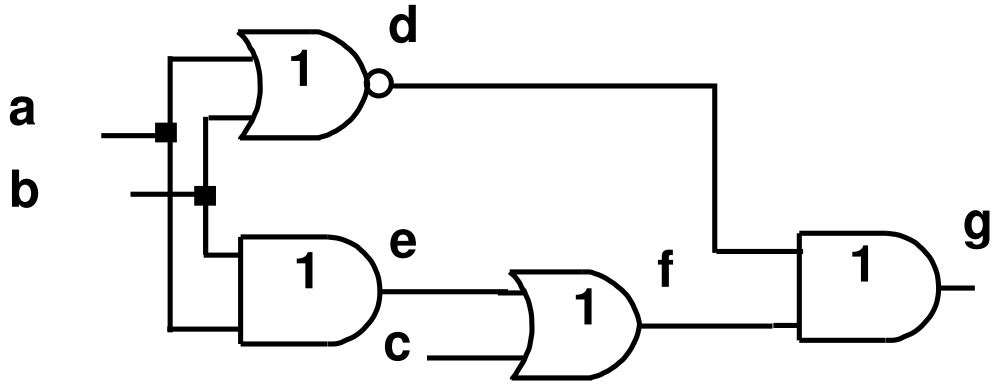
Note: $\chi^{\{A(t)=0,1,X\}} = 1$ (trivially true)

$$
\begin{aligned}
\chi^{\{G(t)=1\}} = \quad & \chi^{\{A(t-d1)=1\}}\chi^{\{B(t-d2)=1\}} + \\
& \chi^{\{A(t-d1)=0\}}\chi^{\{C(t-d3)=1\}} + \\
& \chi^{\{B(t-d2)=1\}}\chi^{\{C(t-d3)=1\}}
\end{aligned}
$$

Delay computation

To determine if path with delay $> \tau$ exists, determine if any input waveform (vector for single vector criterion) causes an output waveform to settle to final constant binary value after τ (another way is to check if $\chi^{\{F(\tau)=X\}} \neq 0$)

Example: circuit delay computation



Compute delay of circuit: (Inputs change at $t = 0$)

$\chi^{\{A(0)=1\}} = a, \chi^{\{A(0)=0\}} = \overline{a}$

$\chi^{\{B(0)=1\}} = b, \chi^{\{B(0)=0\}} = \overline{b}$

Primes of $e = \{ab\}$; Primes of $\overline{e} = \{\overline{a}, \overline{b}\}$

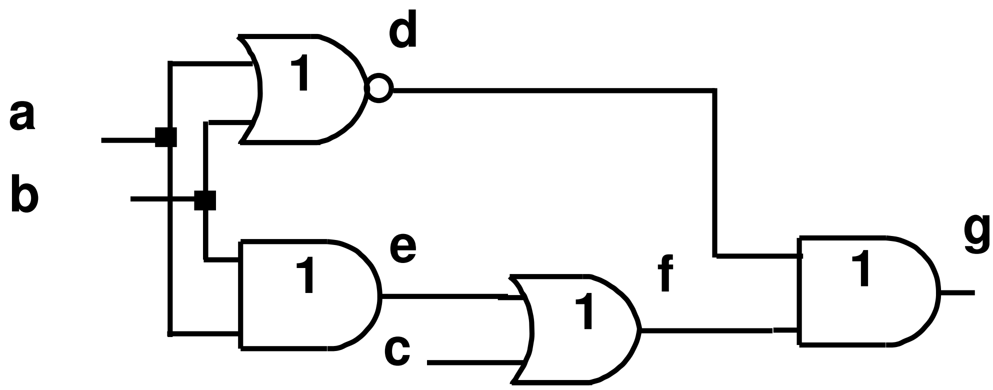
$\chi^{\{E(1)=1\}} = \chi^{\{A(0)=1\}} \chi^{\{B(0)=1\}} = ab$

$\chi^{\{E(1)=0\}} = \chi^{\{A(0)=0\}} + \chi^{\{B(0)=0\}} = \overline{a} + \overline{b}$

$\chi^{\{E(0)=0\}} = \chi^{\{E(0)=1\}} = 0$

since $\chi^{\{H(t)=v\}} = 0$ for $t < D_{max}, v \in \{0, 1\}$ for any internal signal h.

Example continued



Primes of $f = \{c, e\}$; Primes of $\overline{f} = \{\overline{c}\ \overline{e}\}$
$\chi^{\{F(1)=0\}} = 0$ (because $\chi^{\{E(0)=0\}} = 0$)
$\chi^{\{F(1)=1\}} = c + \chi^{\{E(0)=1\}} = c$
$\chi^{\{F(2)=0\}} = \overline{c}\chi^{\{E(1)=0\}} = \overline{c}(\overline{a} + \overline{b})$
$\chi^{\{F(2)=1\}} = c + \chi^{\{E(1)=1\}} = c + ab$

Also, $\chi^{\{D(2)=0\}} = a + b$
$\chi^{\{D(1)=1\}} = \chi^{\{D(2)=1\}} = \overline{a}\ \overline{b}$

Primes of $g = \{df\}$; Primes of $\overline{g} = \{\overline{d}, \overline{f}\}$
$\chi^{\{G(3)=0\}} = \overline{g}$ and $\chi^{\{G(3)=1\}} = g$
(*meaning static function value*)
since longest path to $g \leq 3$
$\chi^{\{G(2)=0\}} = \chi^{\{D(1)=0\}} + \chi^{\{F(1)=0\}} = a + b$
$\chi^{\{G(2)=1\}} = \chi^{\{D(1)=1\}}\chi^{\{F(1)=1\}} = \overline{a}\ \overline{b}c$

$\chi^{\{G(2)=X\}} = \overline{\chi^{\{G(2)=1\}} + \chi^{\{G(2)=0\}}} = \overline{a}\ \overline{b}\ \overline{c} \neq 0$

Hence circuit delay is > 2, *i.e.* 3

How to do computation efficiently?

Goal: Determine if a vector exists for which delay $\geq t$ (*guess at* t)

From XBD0 delay model, output waveform at a gate is determined only by primes of the gate and the waveforms on its input

For a given output waveform, determine all input waveforms that can cause it

Recursive application of this, yields all input waveforms that cause delay $\geq t$. If input waveforms are consistent, delay $\geq t$, else not, i.e. delay $< t$.

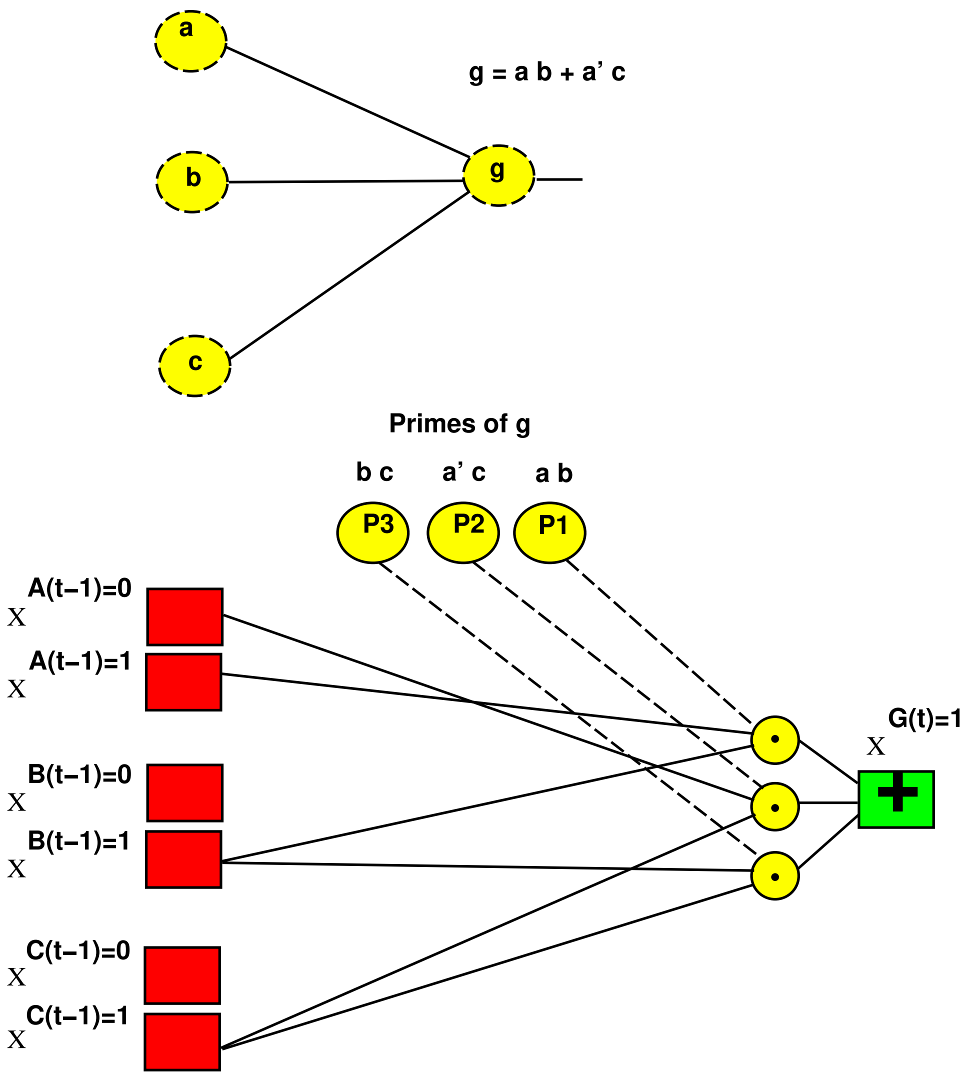
Find exact t by binary search.

Method:

(1) Build Boolean function recursively

(2) Determine satisfiability of function

Building functions recursively



Boolean satisfiability

Given a multi-level Boolean function f, does there exist a vector v such that $f(v) = 1$?

- Consider T_0 in a circuit representation:
- Express network T_0 in CNF using characteristic function of each gate.

Example: a = x y

$$F = (\overline{x} + \overline{y} + a)(x + \overline{a})(y + \overline{a})$$

Conjunction over all gates describes T_0

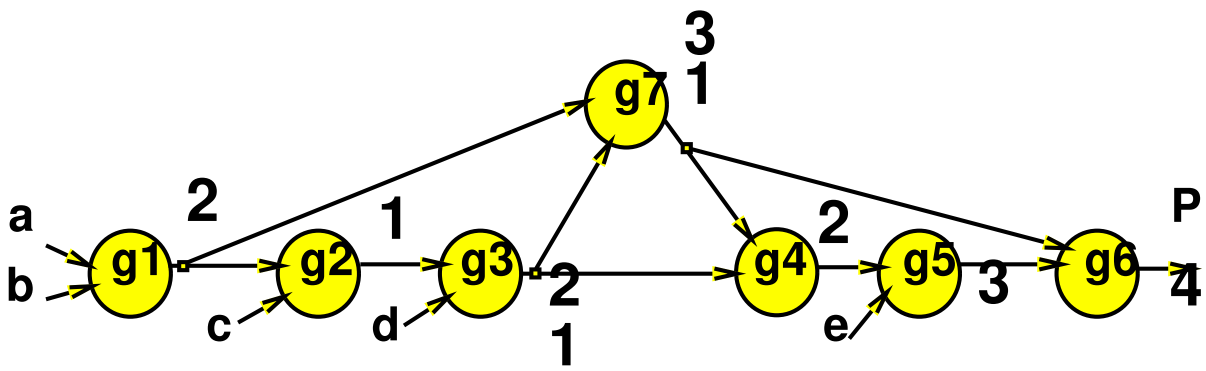
- Since entire formula is in CNF, problem is *satisfiability*

- Satisfiability solved either by BDD's or branch-and-bound search algorithm

- ATPG is a special case of satisfiability. All ATPG heuristics are incorporated into SAT package

How many functions?

Due to reconvergence may need to represent several waveforms at a node



To compute $\chi^{\{G6(4)=0\}}$ need to compute $\chi^{\{G7(3)=0\}}$ and $\chi^{\{G7(1)=0\}}$

In general, may require exponential (worst case) functions to represent waveforms at different time instances

But many functions may be avoided ...

Some pruning rules

- Check satisfiability of intermediate functions; if false, replace function by constant
  - ⋄ dramatic effect on memory requirement
- Infer functions where possible

e.g. Assume topologically longest path up to gate g is t_a. Then for all $t_r > t_a$, $\chi^{g(t_r)=1} = g$ and $\chi^{g(t_r)=0} = \overline{g}$

Further, if for some t_r, $\chi^{g(t_r)=1} = g$, then $\chi^{g(t)=1} = g$ for $t > t_r$

Results: exact floating delay

Name	Delay Estimate		CPU secs.
	Longest	Exact	Exact
C432	36.39	35.96	5
C499	22.49	22.34	8
C880	23.20	23.20	1
C1355	30.70	30.70	7
C1908	39.25	34.77	135
C2670	39.46	38.28	9
C3540	47.74	47.61	29
C5315	37.64	37.58	20
C6288	127.23	124.82†	363
C7552	67.53	64.90	30
csa.32.4	77.63	34.56	160
5xp1	16.38	14.97	2
bw	27.52	20.64	9
des	103.45	93.05	27
misex1	12.53	10.49	1
rot	31.58	29.41	10

Running times are on a DECstation RISC (38 MIPS)
Circuits are technology mapped.
Longest: Topologically longest path (static delay)
Exact: Longest sensitizable path
†: Memory limitation (50 Mbytes) exceeded

Evolution of CPU times on C2670:

1988: unit delay model, too long to complete

1989: unit delay model, few hrs.

1990: unit delay model, few secs.

1991-92: mapped delay model, few secs.

Applications, Related Work and Conclusions

Applications:

- Exact floating delay analysis

- Exact two-vector and multi-vector analysis using delay model with uncertainty range

- Supercedes all published literature on timing analysis for these problems

Related work:

- Extension of waveform analysis to XBD model (non-zero lower bounds)

- Application of waveform analysis to wave-pipelining, sequential circuit timing analysis, multi-vector analysis [Lam, Brayton, Sangiovanni 93]

Conclusions:

- Functional timing analysis well solved for combinational circuits

- Easily extended to edge-triggered sequential circuits

- Recent research on application to sequential circuits with level-sensitive latches