



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

Lecture 32: TIMING CLOSURE (PART 1)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Introduction

- The layout of a chip must satisfy:
 - Geometric constraints (e.g. non-overlapping cells and routability)
 - Timing constraints of the design (e.g. setup and hold constraints)
- The optimization process that meets the above requirements and constraints is often called **timing closure**.
- Integrates placement and routing solutions with specialized methods to improve circuit performance.

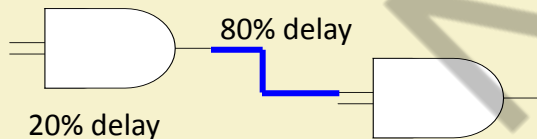
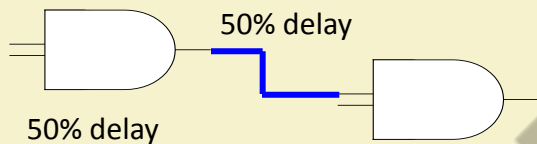
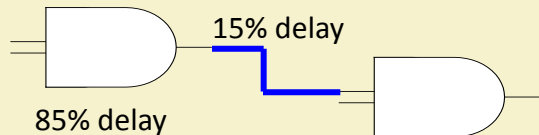
Components of Timing Closure

1. Timing-driven placement
 - Minimizes signal delays when assigning locations to circuit elements.
2. Timing-driven routing
 - Minimizes signal delays when selecting routing topologies and specific routes.
3. Physical synthesis
 - Sizing transistors or gates to decrease the delay or increase the drive strength of a gate.
 - Inserting buffers into nets to decrease propagation delays.
 - Restructuring the circuit along its critical paths.

Background

- For many years, signal propagation delay in logic gates was the main contributor to circuit delay, while wire delay was negligible.
 - Cell placement and wire routing did not affect circuit performance.
- Technology scaling post-1990 significantly increased the relative impact of wire-induced delays.
 - High-quality placement and routing have become critical for timing closure.

Background



- Mid 80 Scenario
 - ◆ Most of the input to output delay of the logic is due to gate delay.
- Mid 90 Scenario
 - ◆ Half of input to output delay of the logic is due to wire delay.
- Today's Scenario
 - ◆ Most of input to output delay of the logic is due to wire delay.

Quick Recap of Setup and Hold Times

- Timing optimization tools adjust propagation delays through circuit components, with the primary goal of satisfying timing constraints. Two ways:
 - **Setup (long-path) constraints**: Amount of time a data input signal should be stable **before** the clock edge for each storage element.
 - **Hold (short-path) constraints**: Amount of time a data input signal should be stable **after** the clock edge at each storage element.

(a) Setup Constraints

- Ensure that no signal transition occurs too late.
- Initial phases of timing closure focus on these types of constraints:

$$t_{cycle} \geq t_{combDelay} + t_{setup} + t_{skew}$$

- Checking whether a circuit meets setup constraints requires estimating how long signal transitions will take to propagate from one storage element to the next.
 - Typically uses **Static Timing Analysis**.

- What is **Static Timing Analysis**?
 - Propagates **actual arrival times** (AAT) and **required arrival times** (RAT) to the terminals of every gate or cell.
 - Can quickly identify timing violations, and diagnose them by tracing out critical paths in the circuit that are responsible for these timing failures.
 - Models propagation of signal transitions with the worst possible delay.
 - Typically excludes **false paths** from the analysis.

- For every timing point x in the circuit netlist, the *timing slack* is computed as:

$$\text{SLACK}(x) = \text{RAT}(x) - \text{AAT}(x)$$

- Positive slack means timing has been met; negative means violation.
- Guided by slack values, physical synthesis restructures the netlist to make it more suitable for high-performance layout implementation.
 - Gates lying on critical paths can be upsized to propagate signals faster.
 - Buffers may be inserted into long critical wires.
 - The netlist tree can be restructured to decrease the overall depth.

Hold-time Constraints

- Ensure that signal transitions do not occur too early.
 - Hold violations can occur when a signal path is too short, allowing a receiving flip-flop to capture the signal at the same cycle instead of the next cycle.
- Hold-time constraint is given by:
$$t_{combDelay} \geq t_{hold} + t_{skew}$$
 - Clock skew affects hold-time constraints significantly more than setup constraints. So, hold-time constraints are typically enforced after synthesizing the clock network.

END OF LECTURE 32



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

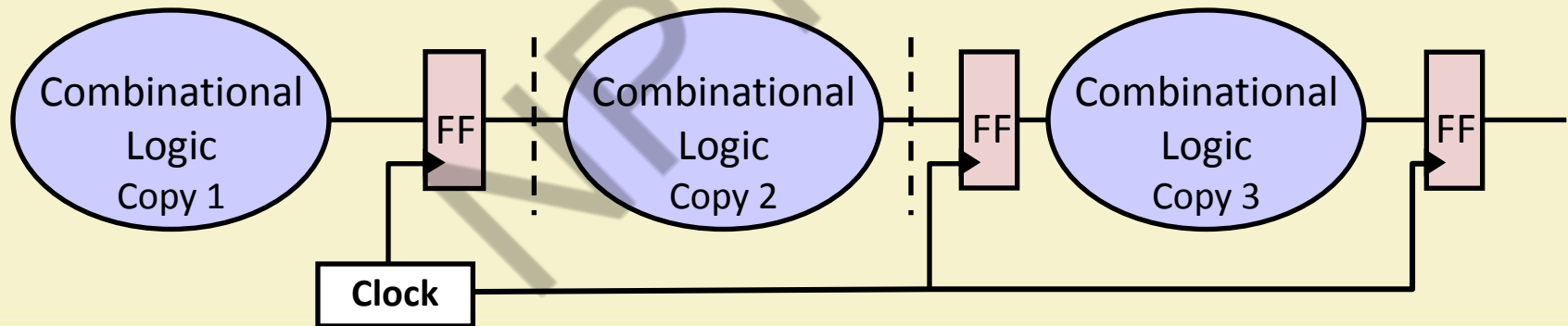
Lecture 33: TIMING CLOSURE (PART 2)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Timing Analysis and Performance Constraints

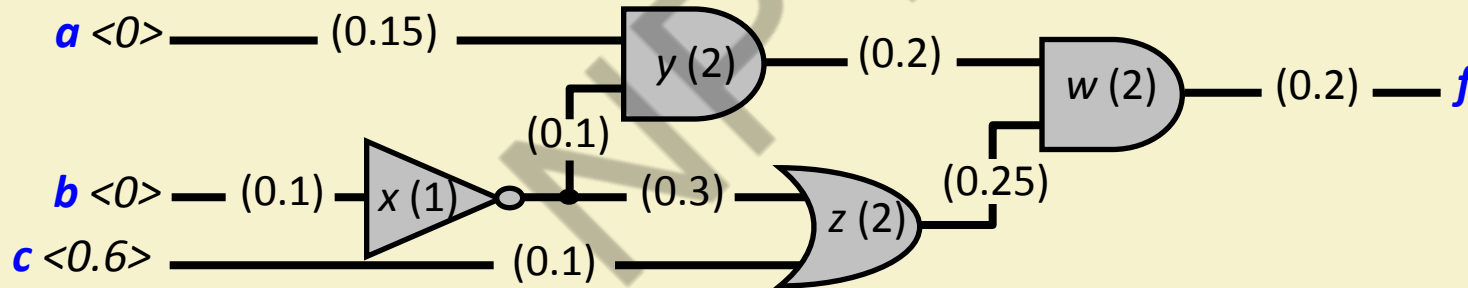
- Almost all digital ICs are synchronous Finite State Machines (FSM).
 - Transitions occur at a set clock frequency.
 - A sequential circuit, *unrolled in time*:



- The maximum clock frequency for a given design depends upon:
 - **Gate delays**, which are the signal delays due to gate transitions.
 - **Wire delays**, which are the delays associated with signal propagation along wires.
 - **Clock skew**.
- Need to quickly estimate sequential circuit timing:
 - Perform static timing analysis (STA).
 - Assume clock skew is negligible, postpone until after clock network synthesis.

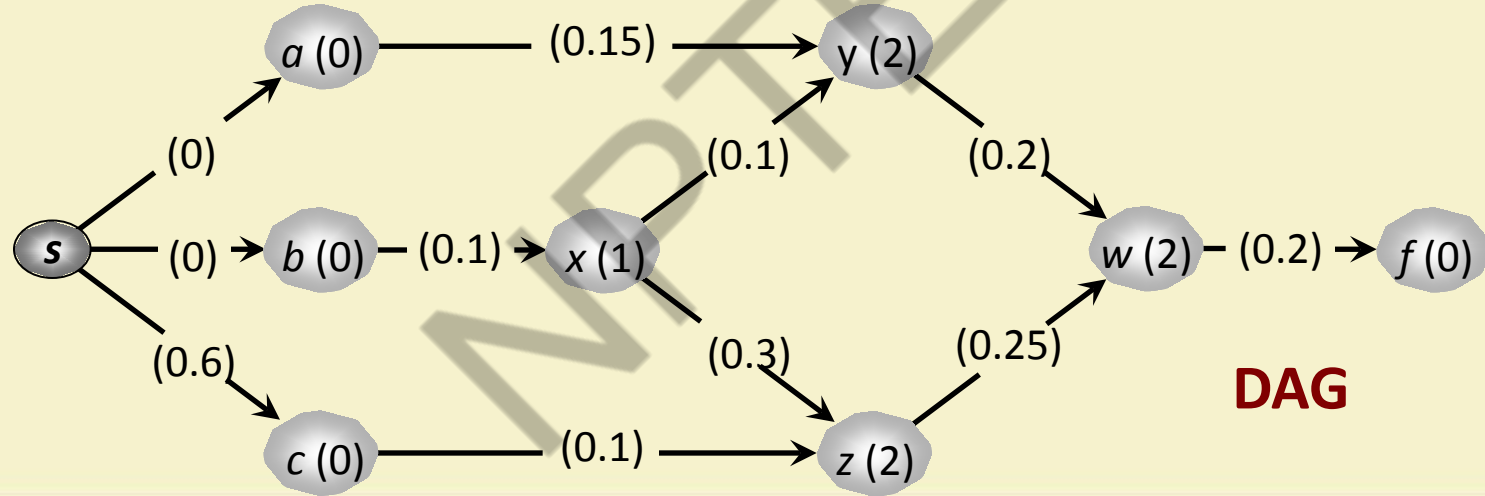
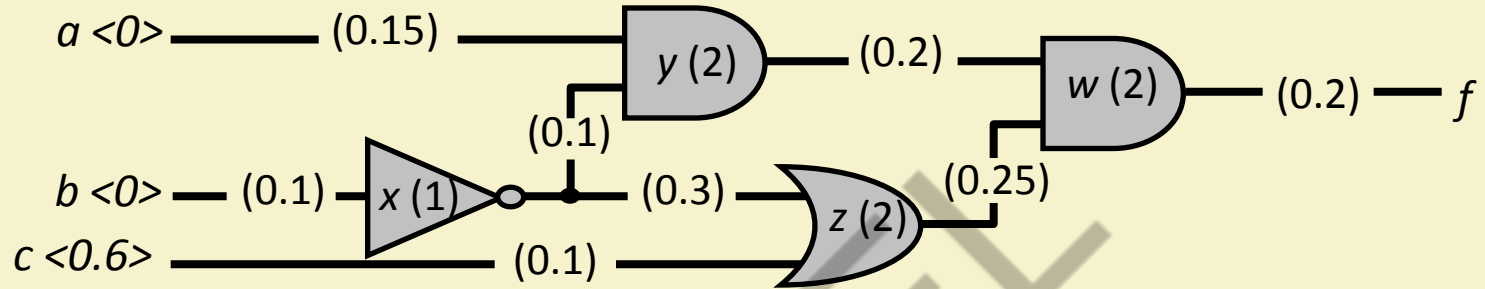
Static Timing Analysis

- We represent a combinational logic netlist as a directed acyclic graph (DAG).
- The inputs are annotated with times 0, 0 and 0.6 time units respectively, at which signal transitions occur relative to the start of the clock cycle.
- The gate and wire delays are also shown.



DAG Representation

- The graph has one vertex for each input and output, as well as one vertex for each logic gate.
- A source node **s** is introduced with a directed edge to each input.
- Vertices corresponding to logic gates are labeled with the respective gate delays.
- Directed edges from the source to the inputs are labeled with transition times, and directed edges between gate vertices are labeled with wire delays.



DAG

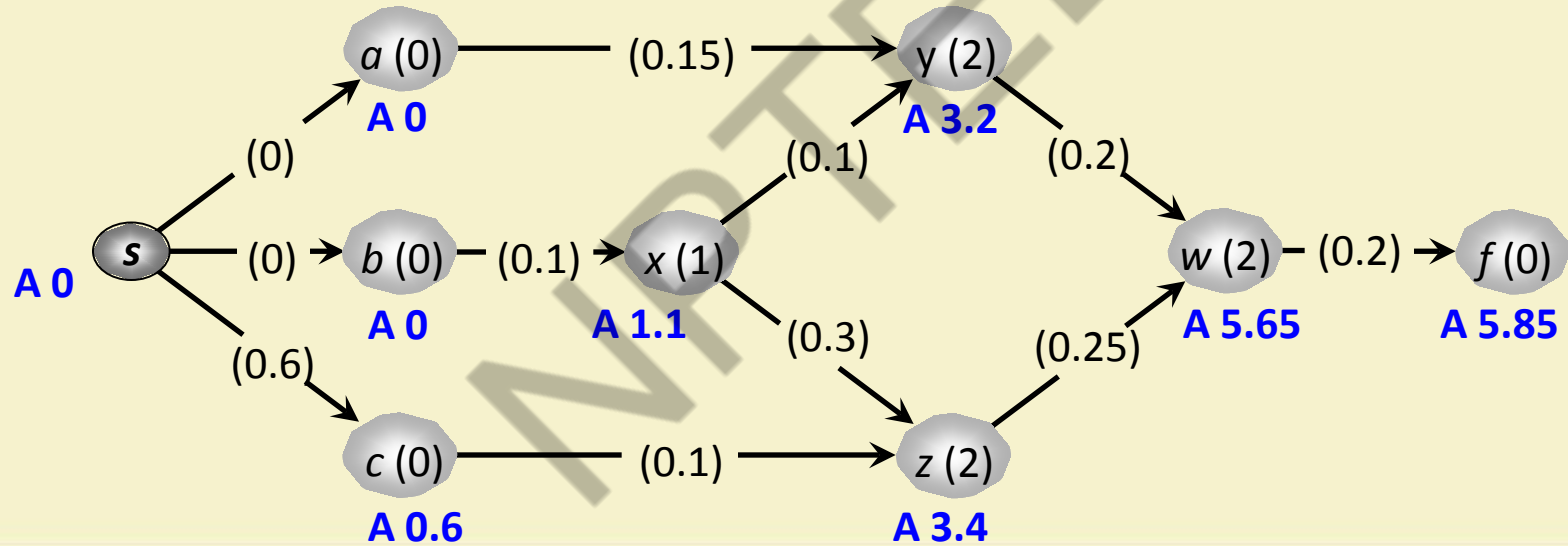
Actual Arrival Time (AAT)

- The AAT of a given node $v \in V$, denoted as $AAT(v)$, is defined as the latest transition time at v measured from the beginning of the clock cycle.
 - By convention, $AAT(v)$ records the arrival time at the **output side** of node v .
 - In the previous example, $AAT(x) = 0.1 + 1 = 1.1$, $AAT(y) = 1.1 + 0.1 + 2 = 3.2$
- Formal definition:

$$AAT(v) = \max_{u \in FI(v)} (AAT(u) + t(u, v))$$

where $FI(v)$ is the set of all nodes from which there exists a directed edge to v , and $t(u, v)$ is the delay corresponding to the (u, v) edge.

- All AAT values in the DAG can be computed in $O(|V| + |E|)$ time.
 - Linear in number of gates are edges.
- This linear scaling of runtime makes STA applicable to modern designs with hundreds of millions of gates.



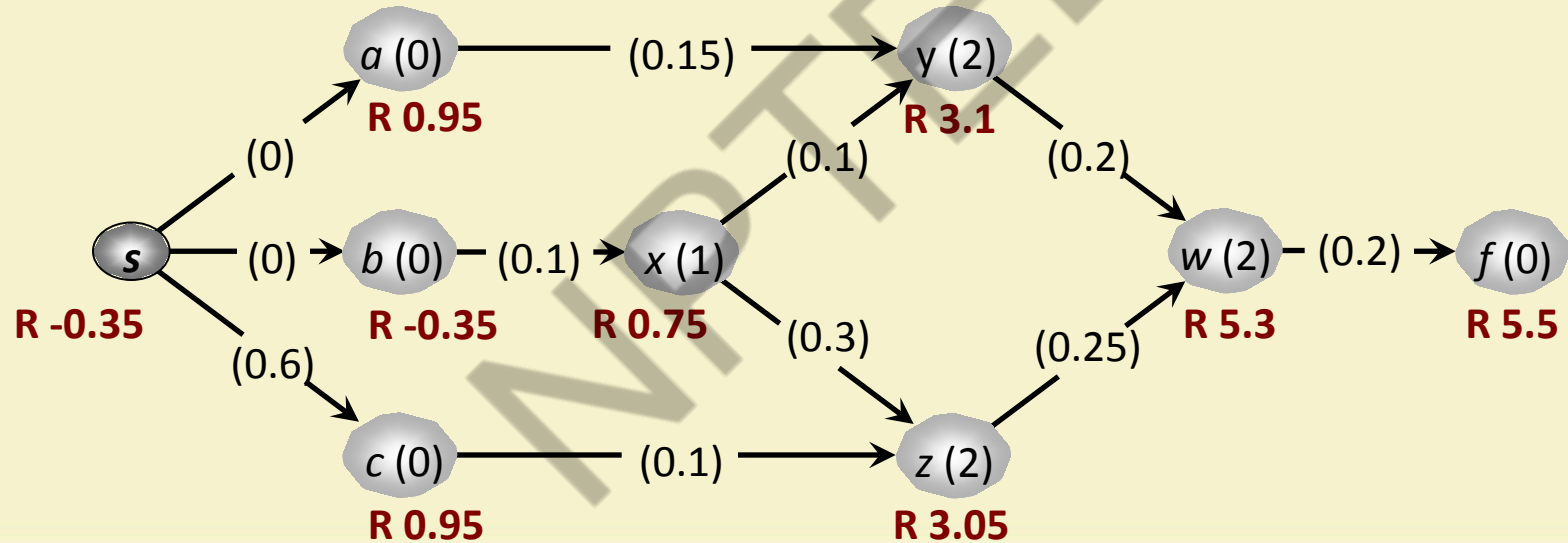
Required Arrival Time (RAT)

- The RAT of a given node $v \in V$, denoted as $RAT(v)$, is defined as the time by which the latest transition at a given node v must occur in order for the circuit to operate correctly within a given clock cycle.
 - Unlike AATs, which are determined from multiple paths from upstream inputs and flip-flop outputs, RATs are determined from multiple paths to downstream outputs and flip-flop inputs.
- Formal definition:

$$RAT(v) = \min_{u \in FO(v)} (RAT(u) - t(u, v))$$

where $FO(v)$ is the set of all vertices with a directed edge from v .

- It is assumed that the RAT values for the outputs are given.
- For the example, suppose that **$RAT(f) = 5.5$** .



Slack Computation

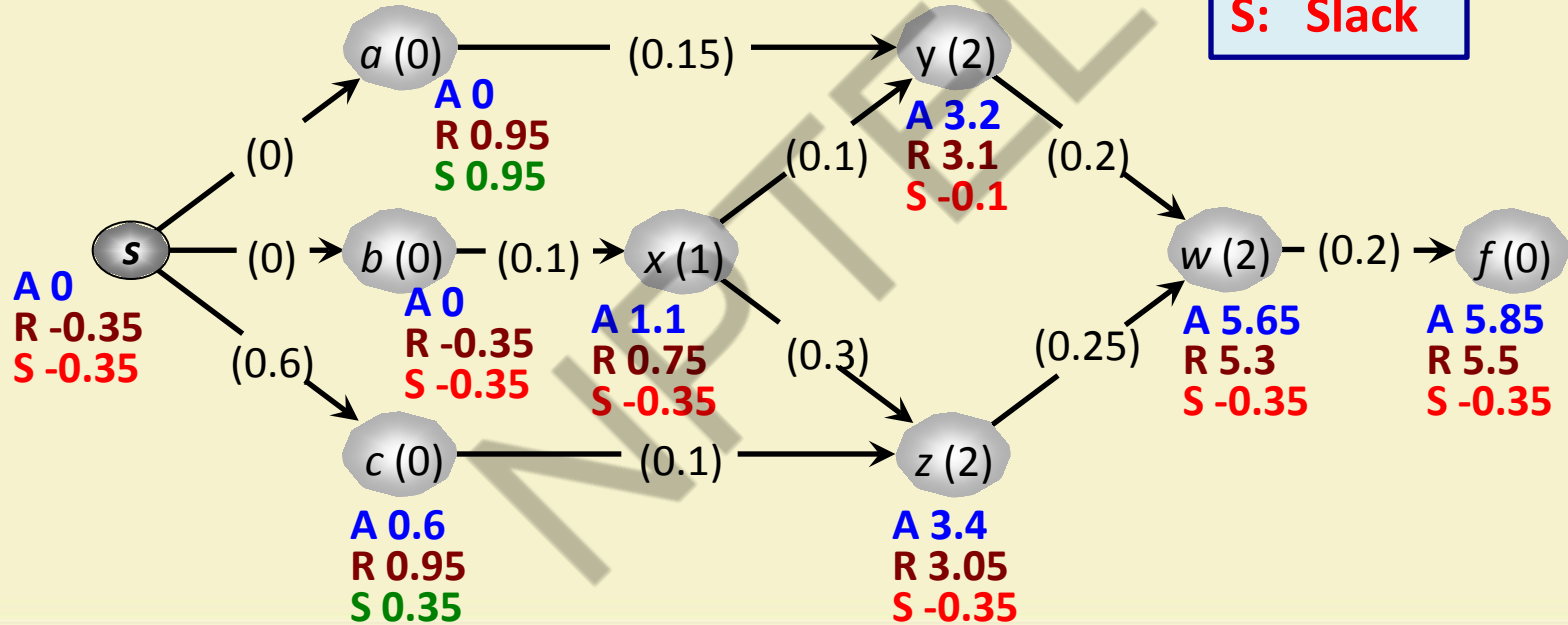
- The correct operation of the chip with respect to setup constraints (e.g. maximum path delay), requires that AAT at each node does not exceed RAT.
 - That is, for all vertices $v \in V$, we must have $AAT(v) \leq RAT(v)$.
- The slack of a node v is computed as:

$$slack(v) = RAT(v) - AAT(v)$$

- Critical paths or critical nets are signals that have negative slack.
- Non-critical paths or non-critical nets have positive slack.

Final Result with Slacks Computed

A: AAT
R: RAT
S: Slack



Current Practice

- In modern designs, separate timing analyses are performed for the cases of *rise delay* (rising transitions) and *fall delay* (falling transitions).
- Signal integrity extensions to STA consider changes in delay due to switching activity on neighboring wires of the path under analysis.
 - For signal integrity analysis, the STA tool keeps track of windows (intervals) of AATs and RATs.
 - Typically executes multiple timing analysis iterations before these timing windows stabilize.
- Statistical STA is a generalization of STA where gate and wire delays are modeled by random variables and represented by probability distributions.

Drawbacks of STA

1. Assumption of a clock.

Not applicable to asynchronous subsystems.

2. Assumption that all paths are sensitizable.

Optimization tools waste considerable runtime and chip resources (e.g. power, area, speed) satisfying *phantom* constraints.

- *False paths*, which are never activated.
- *Multi-cycle paths*, where signal transitions do not need to finish within one clock cycle.

END OF LECTURE 33





IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

Lecture 34: TIMING CLOSURE (PART 3)

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Delay Budgeting with the Zero-Slack Algorithm

- In timing-driven physical design, both gate and wire delays must be optimized to obtain a timing-correct layout.
- There exists a dilemma:
 - Timing optimization requires knowledge of capacitive loads, and hence the actual wire length.
 - Wire lengths are unknown until placement and routing are completed.
- Timing budgets are used to establish delay and wire length constraints for each net, for guiding placement and routing to a timing-correct result.
 - Best-known approach to timing budgeting is the ***Zero Slack Algorithm***.

Basic Idea

- Some notations:
 - Consider a netlist consisting of logic gates v_1, v_2, \dots, v_n
 - Consider a set of nets e_1, e_2, \dots, e_m , where e_i is the output net of gate v_i .
 - Let $t(v)$ and $t(e)$ denote gate delay and wire delay, respectively.

- The ZSA takes the netlist as input, and tries to decrease positive slacks of all nodes to zero by increasing $t(v)$ and $t(e)$ values.
- These increased delay values together constitutes the **Timing Budget** $TB(v)$ of node v , which should not be exceeded during placement and routing.

$$TB(v) = t(v) + t(e)$$

- If $TB(v)$ is exceeded, then the place-and-route tool typically:
 - (i) decrease the wirelength of e , or (ii) changes the size of gate v .
 - The delay impact of a wire or gate size change can be estimated using the Elmore delay model.

- If most arcs (branches) of a timing path are within budget, then the path may meet its timing constraints even if some arcs exceed their budgets.
 - Thus, another approach to satisfying the timing budget is *rebudgeting*.
- The zero slack algorithm shall be explained with the help of an illustrative example.

Basic Steps in ZSA

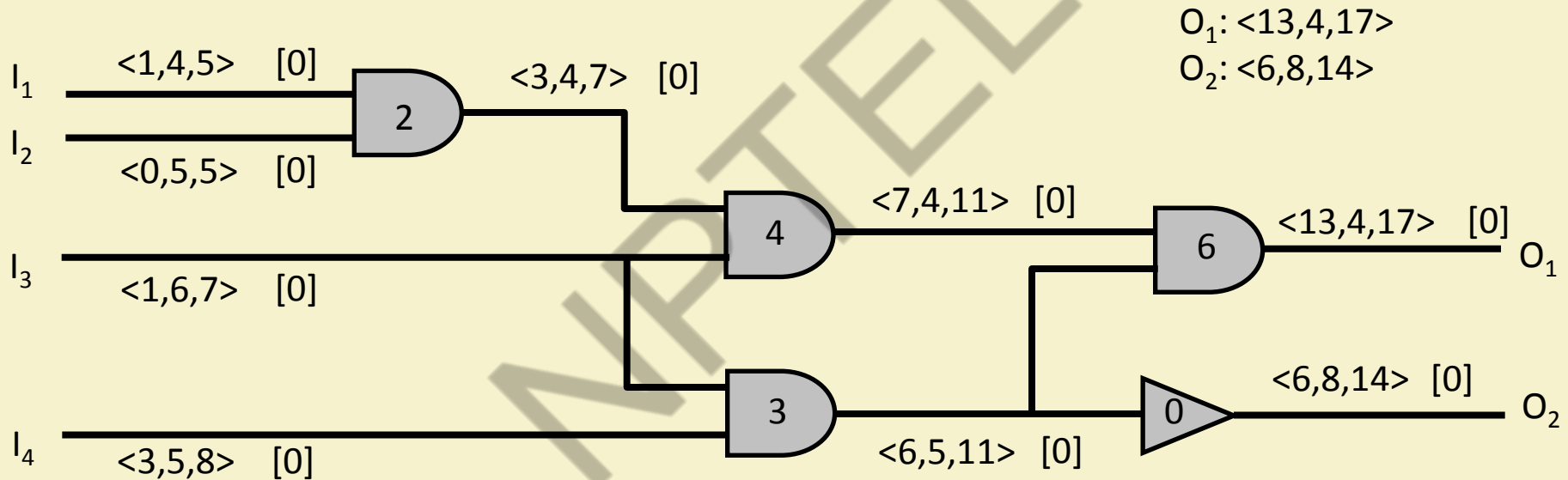
1. Determine the initial slacks of all the nodes, and select a node v_{min} with minimum positive slack $slack_{min}$.
2. Find a path of vertices that dominates $slack_{min}$, i.e. any change in the delays in vertices along the path will cause $slack_{min}$ to change.
3. Evenly distribute the slack by increasing $TB(v)$ for each vertex v in the path. Each budget increment will decrement the slack value of a vertex.

By repeating the process, the slack of each node in V will end up at zero.

The resulting timing budgets at all nodes are the final output of ZSA.

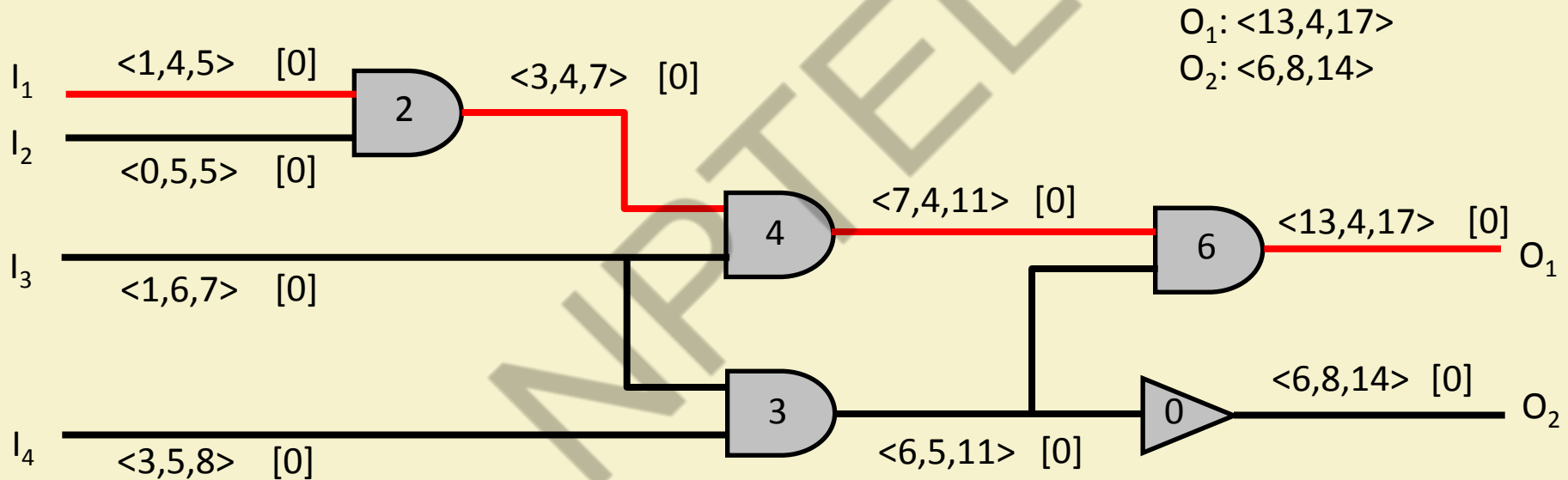
Example

- Use the zero-slack algorithm to distribute slack
- Format: $\langle AAT, Slack, RAT \rangle, [timing\ budget]$



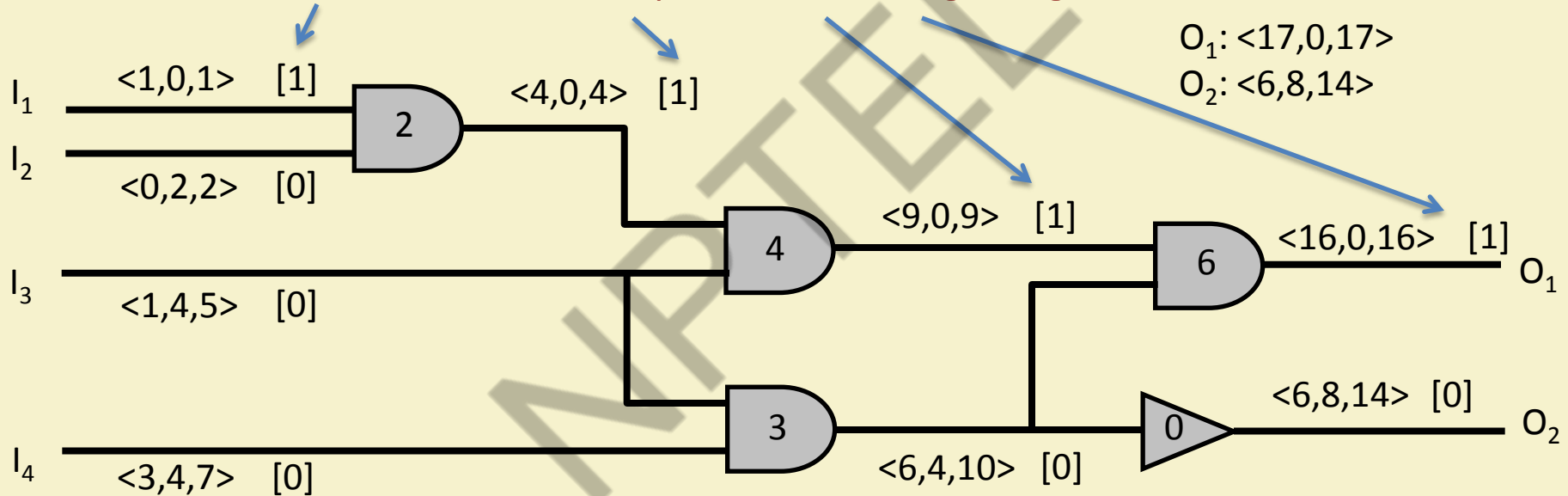
Example

- Find the path with the minimum non-zero slack (MARKED IN RED).



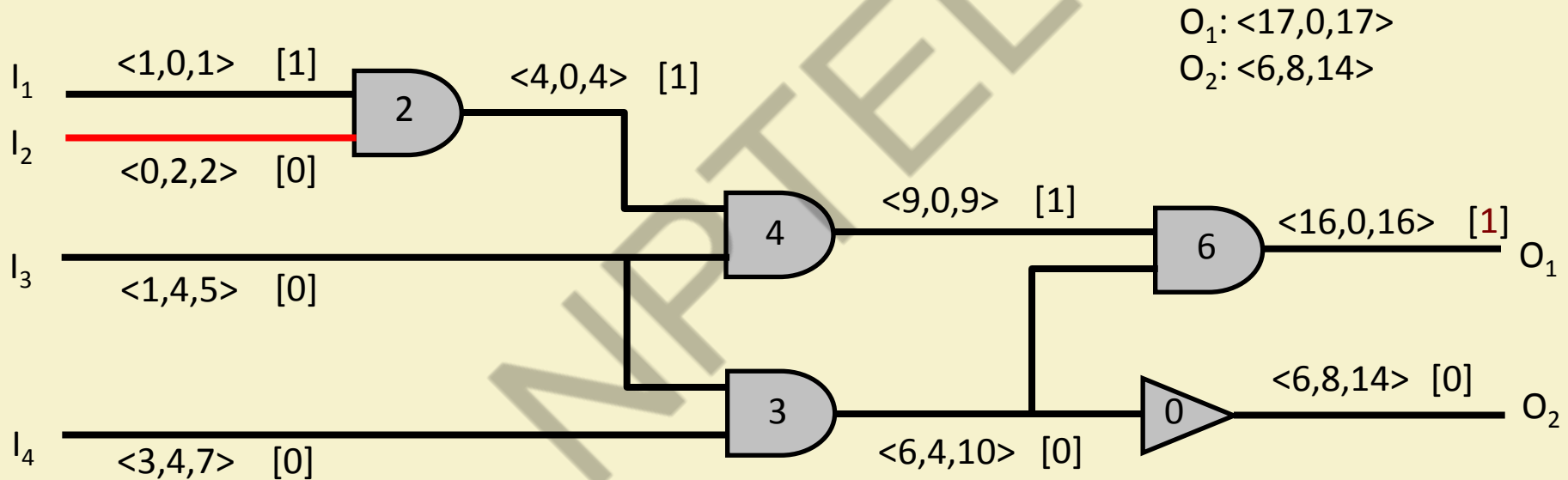
Example

- Find the path with the minimum non-zero slack.
- Distribute the slacks and update the timing budgets.



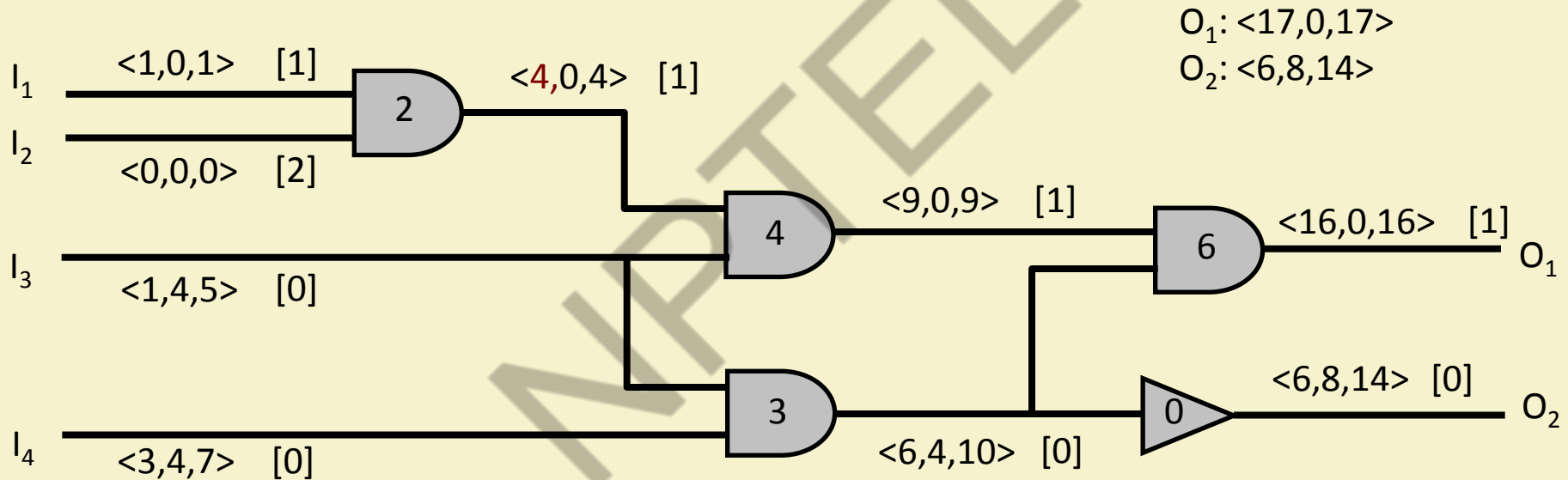
Example

- Find the path with the minimum non-zero slack.
- Distribute the slacks and update the timing budgets.



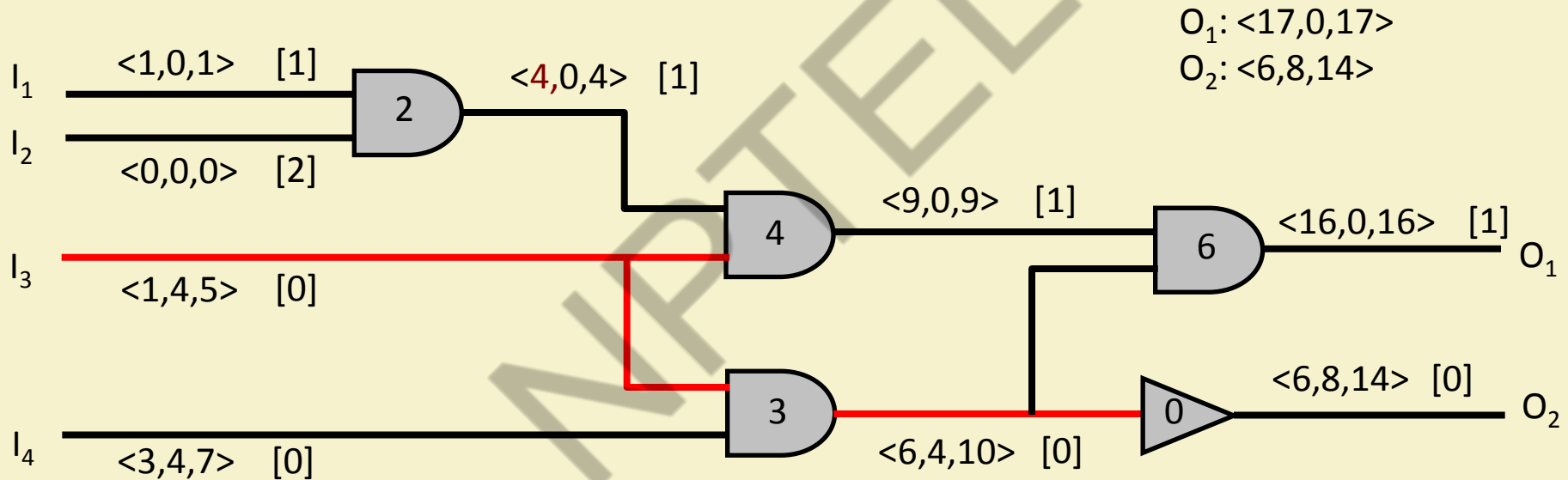
Example

- Find the path with the minimum non-zero slack.
- Distribute the slacks and update the timing budgets.



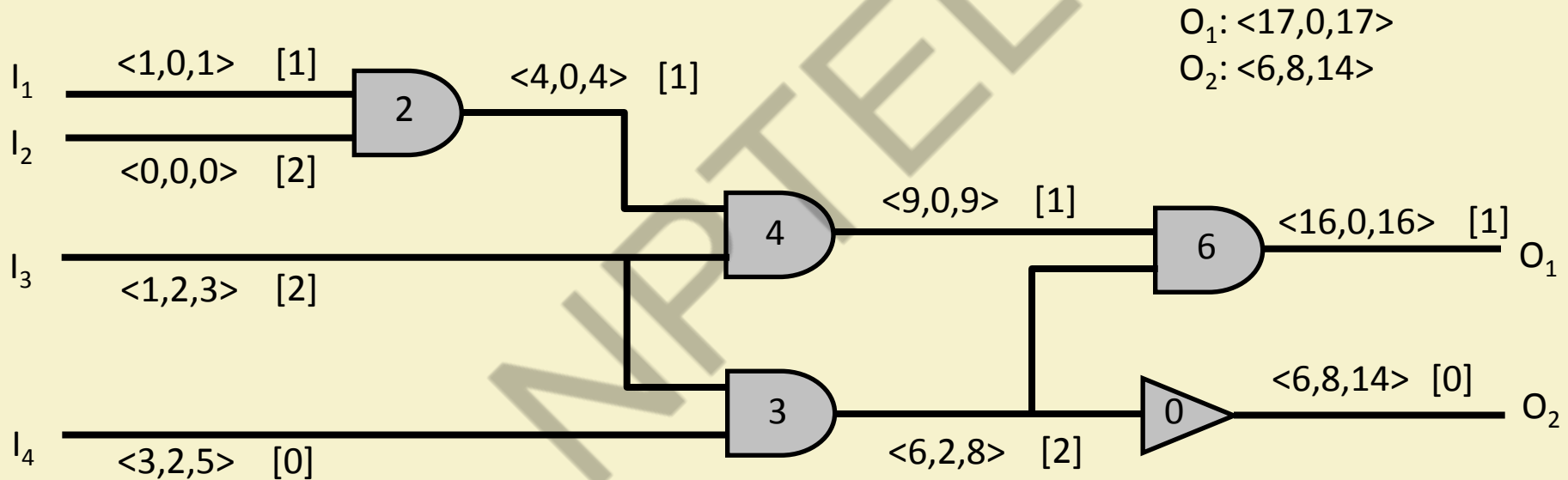
Example

- Find the path with the minimum non-zero slack.
- Distribute the slacks and update the timing budgets.



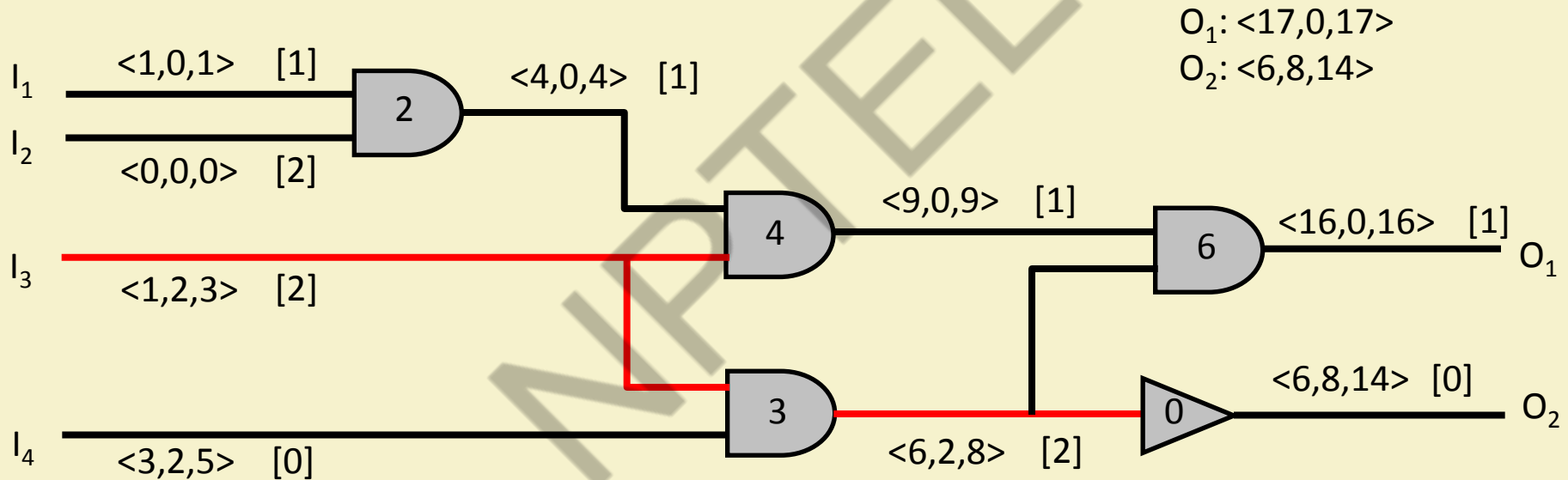
Example

- Find the path with the minimum non-zero slack.
- Distribute the slacks and update the timing budgets.



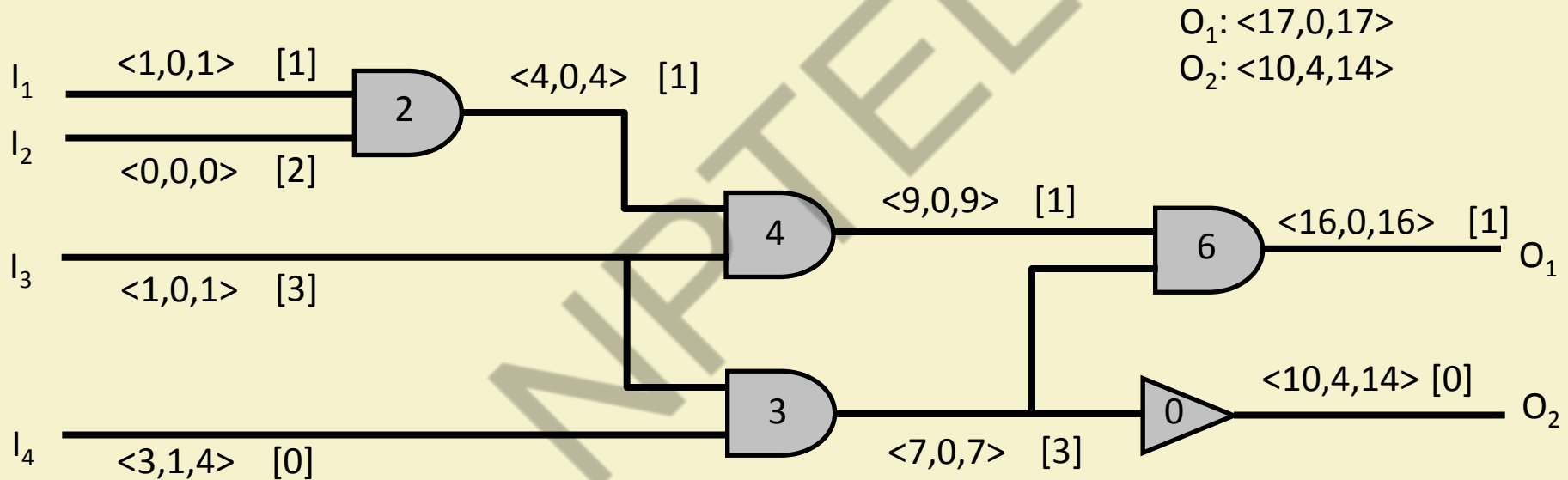
Example

- Find the path with the minimum non-zero slack.
- Distribute the slacks and update the timing budgets.



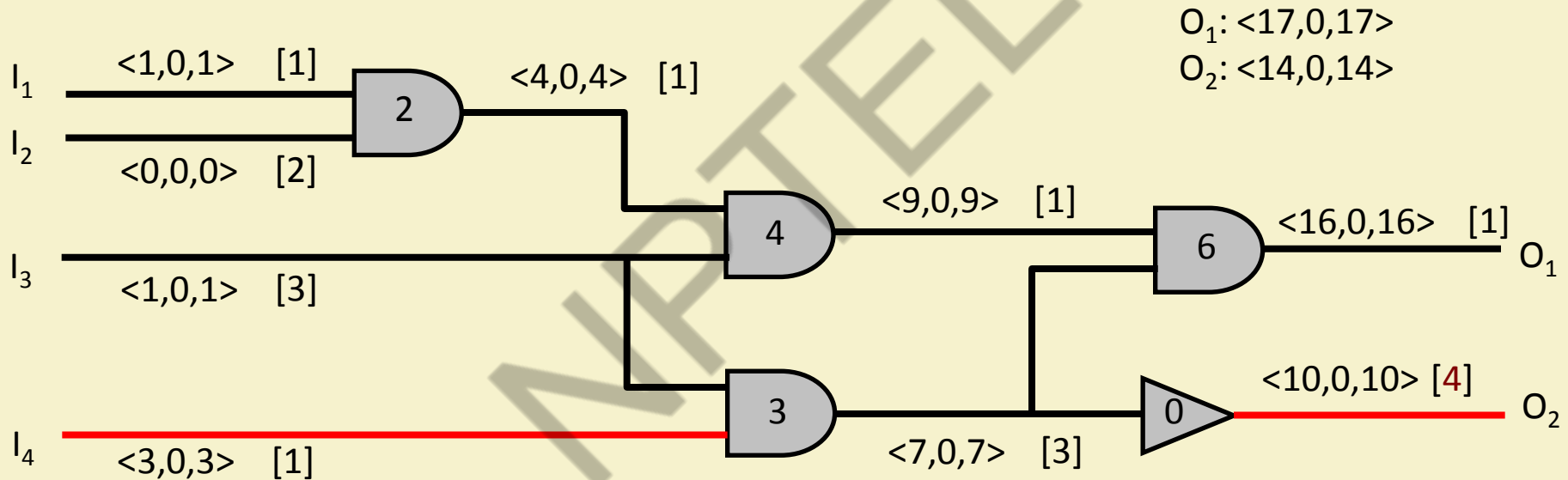
Example

- Find the path with the minimum non-zero slack.
- Distribute the slacks and update the timing budgets.



Example

- Find the path with the minimum non-zero slack.
- Distribute the slacks and update the timing budgets.



A Modification: Early Mode Analysis

- ZSA uses late-mode analysis with respect to setup constraints, i.e. the latest times by which signal transitions can occur for the circuit to operate correctly.
- Correct operation also depends on satisfying hold-time constraints on the earliest signal transition times.
- Early-mode analysis considers these constraints.

How it Works?

- To correctly analyze this timing constraint, the earliest actual arrival time of signal transitions at each node must be determined.
- The required arrival time of a sequential element in early mode is the time at which the earliest signal can arrive and still satisfy the library-cell hold-time requirement.
- For each gate v , $AAT_{EM}(v) \geq RAT_{EM}(v)$ must be satisfied.
 - $AAT_{EM}(v)$ is the earliest actual arrival time of a signal transition at gate v
 - $RAT_{EM}(v)$ is the required arrival time in early mode at gate v

- The early-mode slack can be defined as:

$$slack_{EM}(v) = AAT_{EM}(v) - RAT_{EM}(v)$$

- When adapted to early-mode analysis, ZSA is also called the near zero-slack algorithm.
 - The modified algorithm seeks to decrease $TB(v)$ by decreasing $t(v)$ or $t(e)$, so that all nodes have minimum early-mode timing slacks.
 - Since $t(v)$ and $t(e)$ cannot be negative, node slacks may not necessarily all become zero.

To Summarize

- In practice, if the delay of a node does not satisfy its early-mode timing budget, the delay constraint can be satisfied by adding additional delay (padding) to appropriate components.
 - The additional delay can violate late-mode timing constraints.
- Thus, a circuit should be first designed with ZSA and late-mode analysis. Early-mode analysis may then be used to confirm that early-mode constraints are satisfied, or to guide circuit modifications to satisfy such constraints.

END OF LECTURE 34





IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

Lecture 35: TIMING CLOSURE (PART 4)

PROF. INDRANIL SENGUPTA

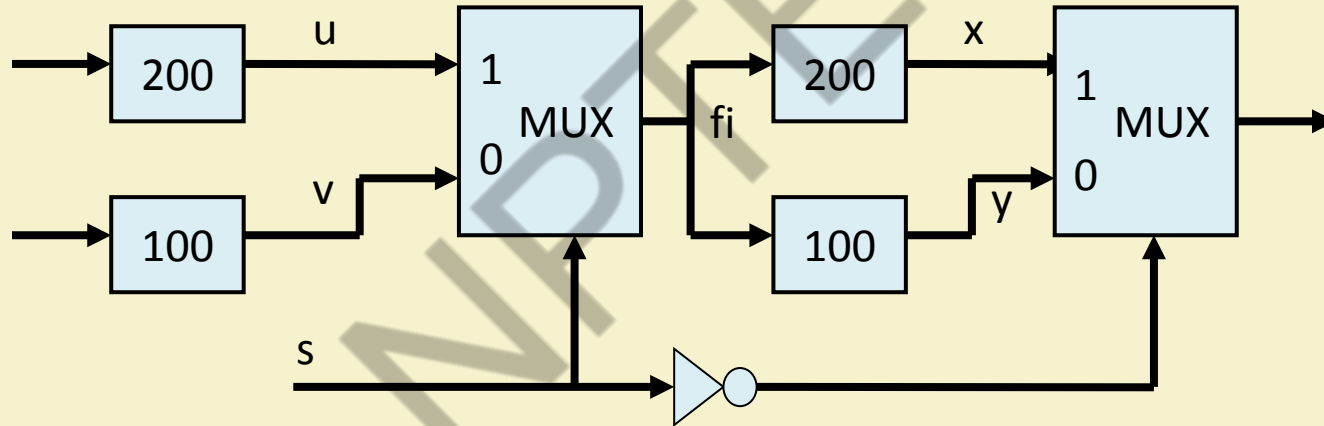
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

False Paths

- Paths that physically exist in a design but are not logic /functional paths.
- These paths never get sensitized under any input conditions.
- An example is shown on the next slide.

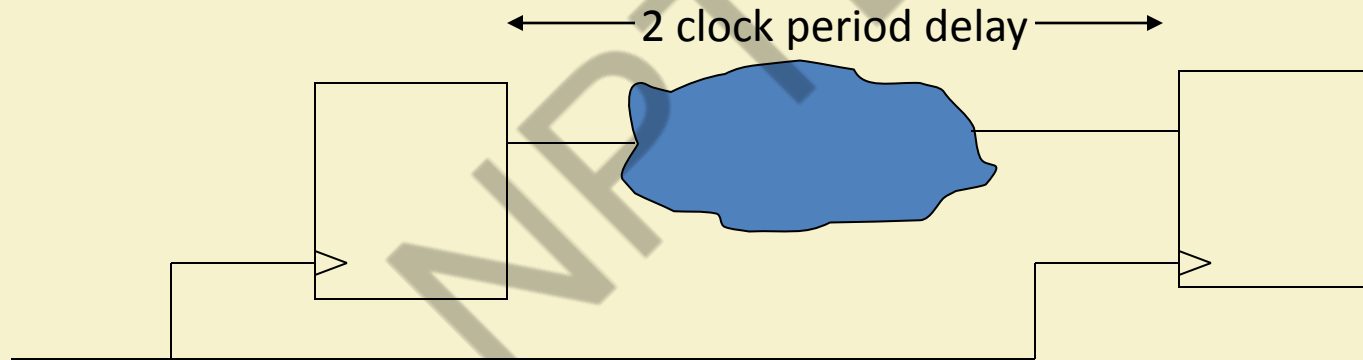
An example:

The path of length 400 is never exercised.



Multi-cycle Paths

- Data paths that require more than one clock period for execution.

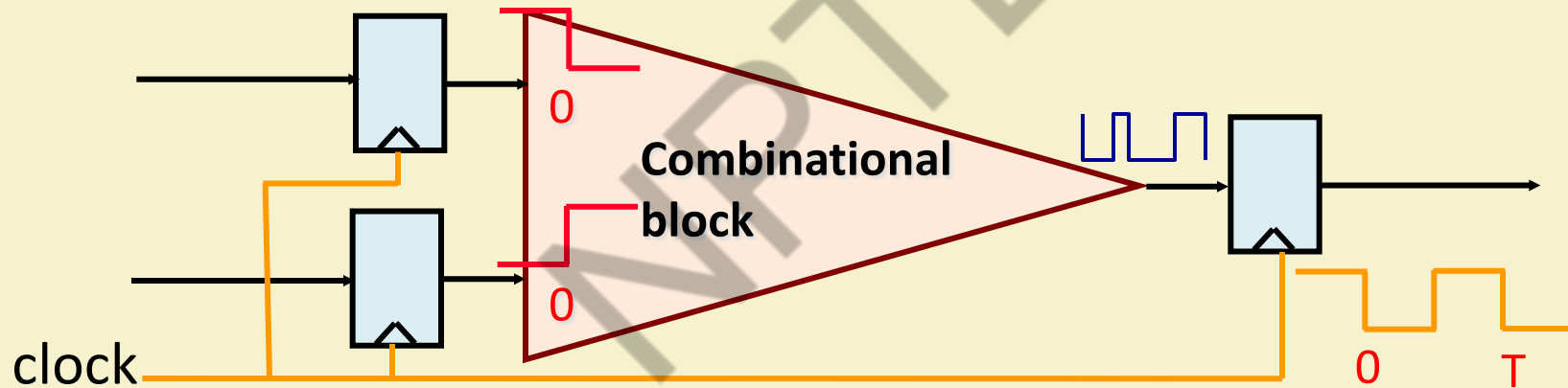


Timing Analysis Problems

- We want to determine the *true* critical paths of a circuit in order to:
 - Determine the minimum cycle time for which the circuit will function.
 - Identify critical paths from performance optimization – do not try to optimize the *wrong* (non-critical) paths
- Implications:
 - Do not want *false paths* (produced by static delay analysis).
 - Delay model is worst case model.

Functional Timing Analysis

- Estimate when the output of a given circuit gets stable.



Why Timing Analysis?

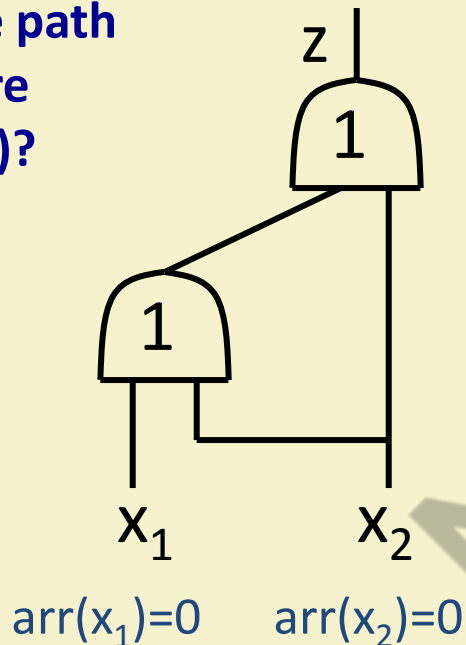
- Timing verification
 - Verifies whether a design *meets* a given timing constraint.
 - Example: cycle-time constraint
- Timing optimization
 - Needs to identify *critical* portion of a design for further optimization.
 - Critical path identification
- In both cases, higher the accuracy, the better.

Timing Analysis - Basics

- Naïve approach - Simulate all input vectors with SPICE
 - Accurate, but too expensive.
- Gate-level timing analysis
 - 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
 $\text{arr}(z)$?

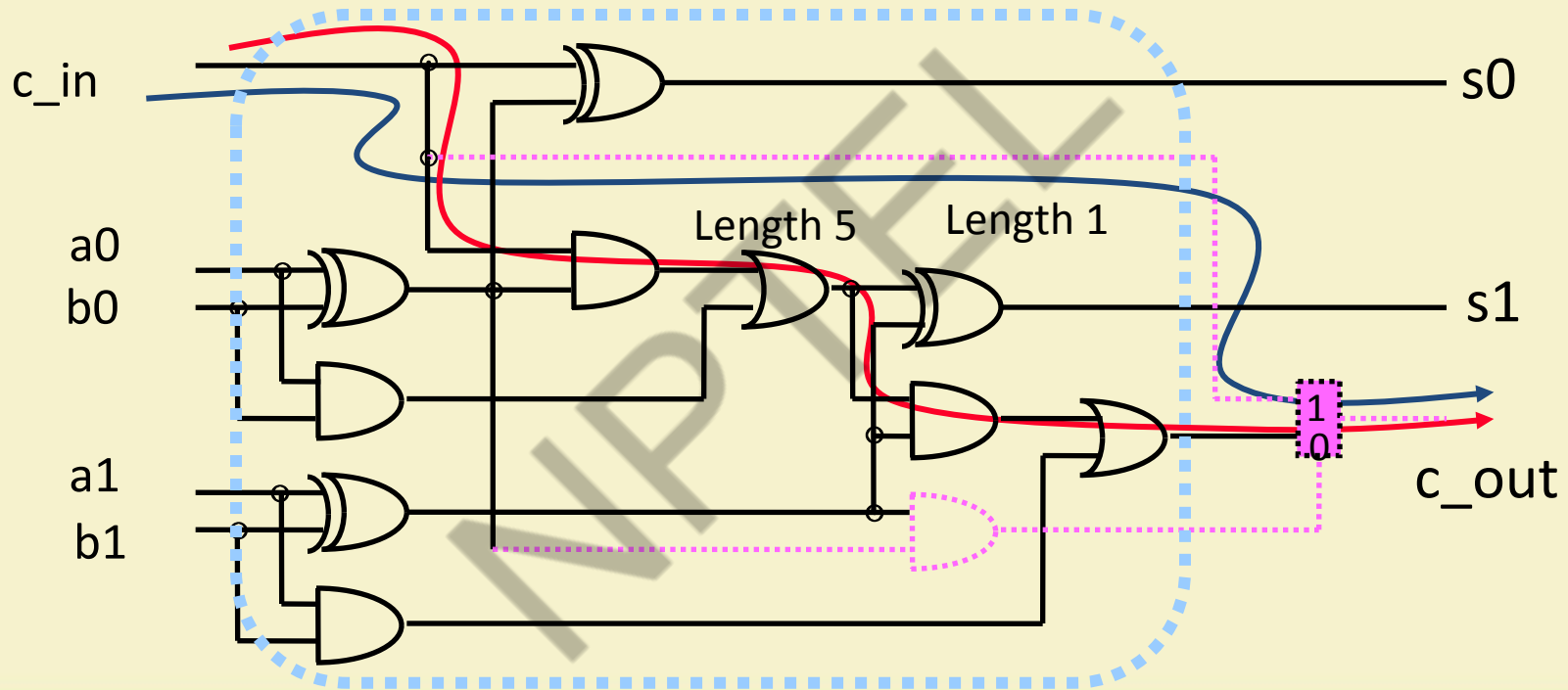


- A naive approach is topological analysis.
 - Easy longest-path problem
 - Linear in the size of a network
- Not all paths can propagate signal events.
 - False paths
 - If all longest paths are false, topological analysis gives delay overestimate.

Functional timing analysis = false-path-aware timing analysis

- Compute false-path-aware arrival time

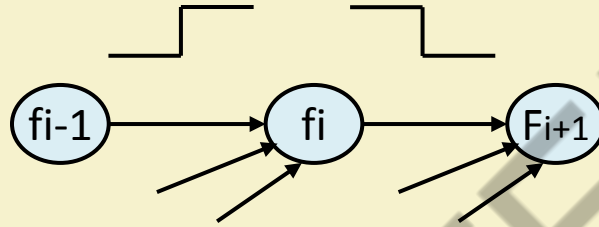
Example: 2-bit Carry-skip Adder



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.

Possible Approach :: Boolean Difference

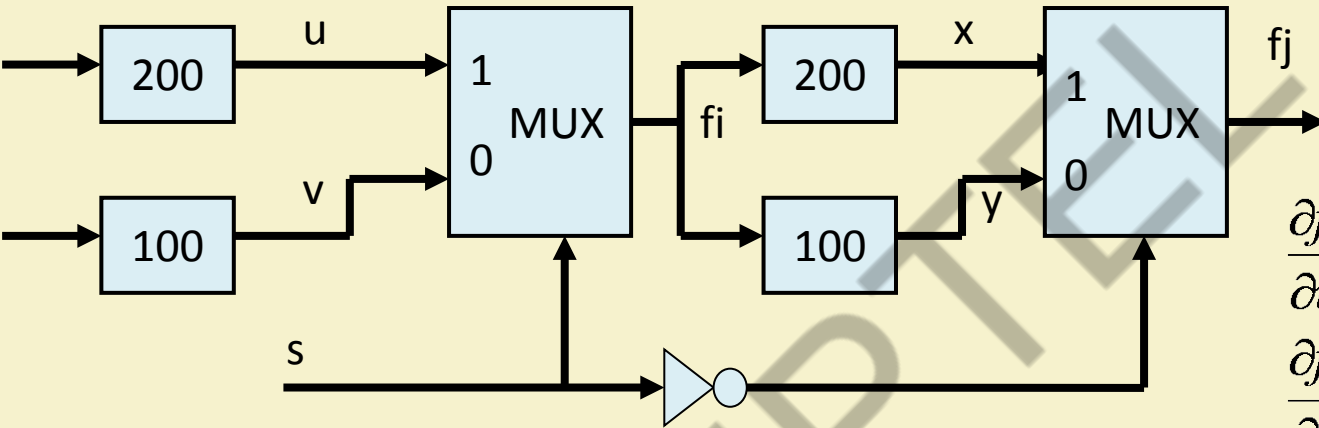


- Path $P = \{f_0, f_1, f_2, \dots, f_n\}$

$\frac{\partial f_i}{\partial f_{i-1}}$ gives conditions under which node f_i is “sensitive” to node f_{i-1}

- So output P is sensitive to f_0 if $\prod_{i=1}^n \frac{\partial f_i}{\partial f_{i-1}} \neq 0$

Example :: Static False Path



$$f_i = su + \bar{s}v$$

$$f_j = \bar{s}x + sy$$

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

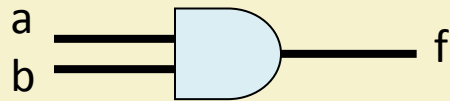
$$\frac{\partial f_j}{\partial x} = \bar{s}$$

Hence, $\frac{\partial f_i}{\partial u} \cdot \frac{\partial f_j}{\partial x} = 0$

The path is not sensitizable and hence 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.
 - 0 is a controlling value for AND gate; 1 is non-controlling value for AND gate.
- Controlling / non-controlling value is merely a specialization of the Boolean difference to simple gates.

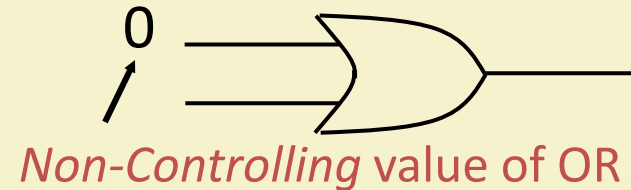
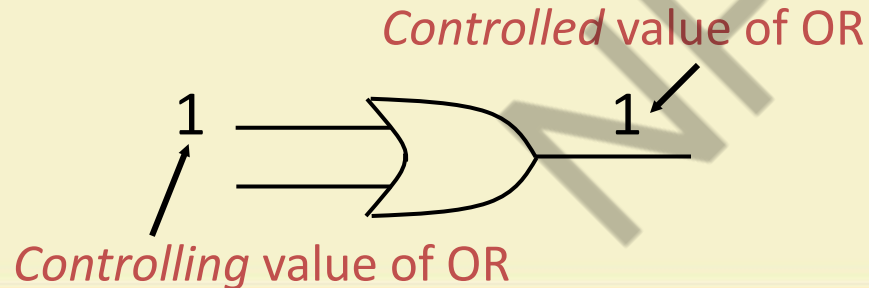
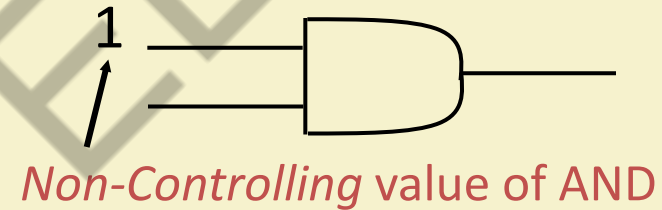
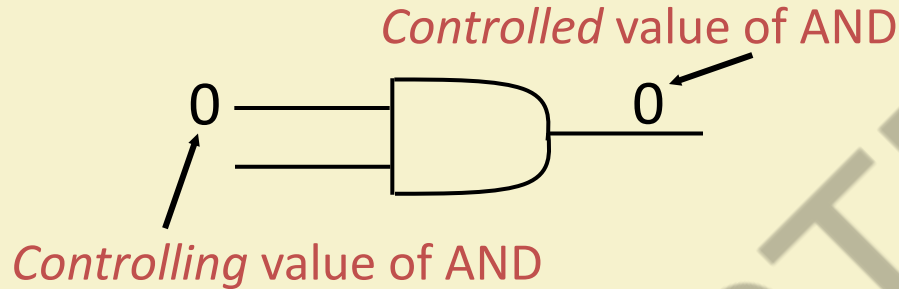


$$\frac{\partial f}{\partial a} = b$$



$$\frac{\partial g}{\partial a} = \bar{b}$$

Controlling/Non-Controlling Values



END OF LECTURE 35



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

Lecture 36: TIMING CLOSURE (PART 5)

PROF. INDRANIL SENGUPTA

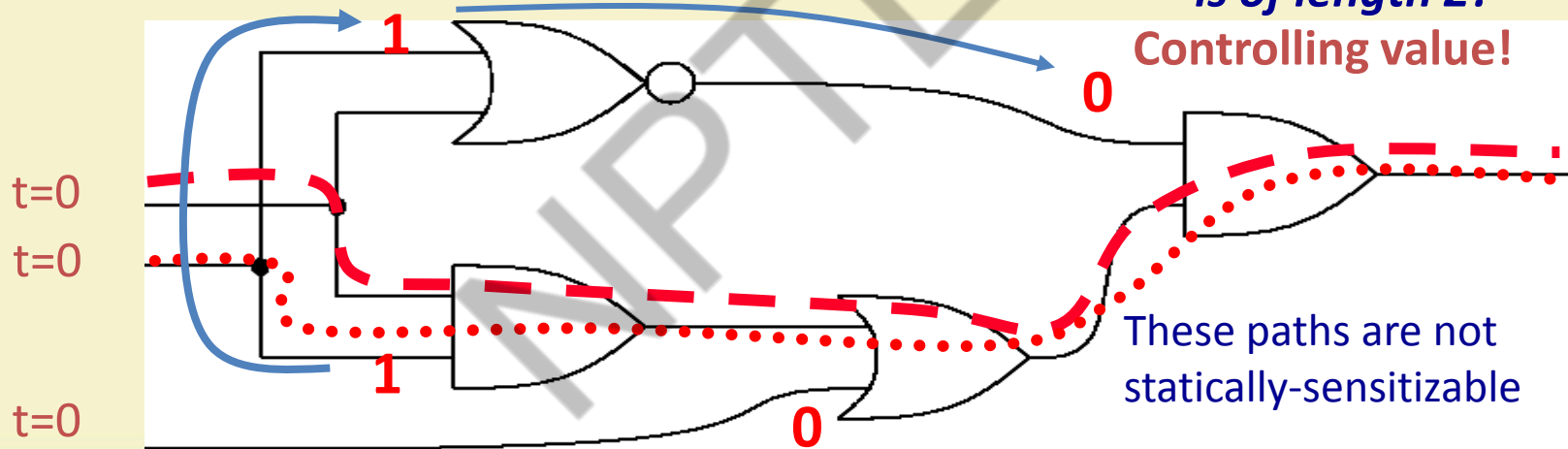
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

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.

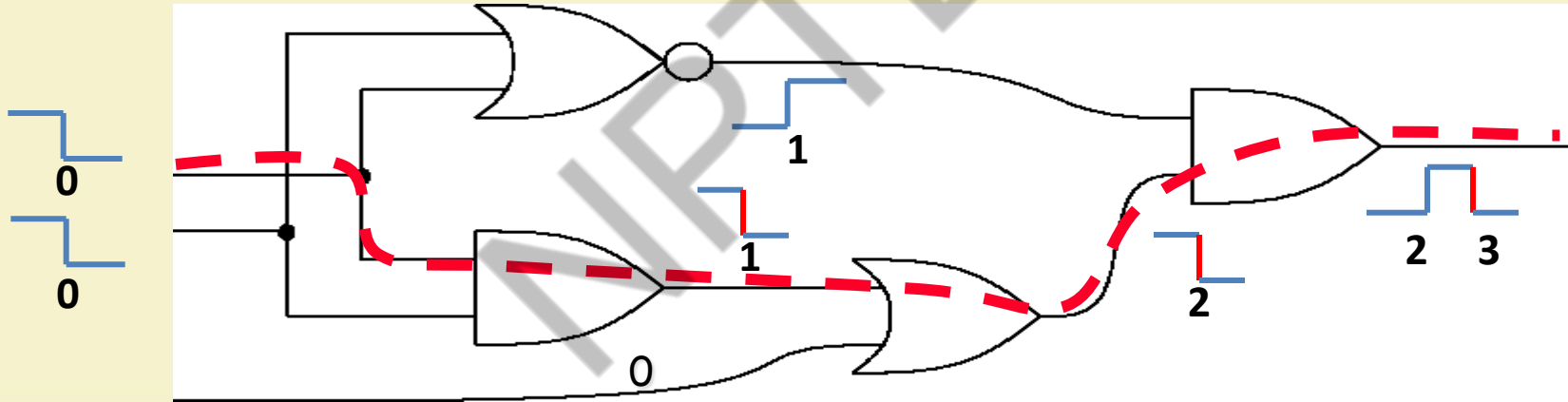
The longest true path is of length 2?

Controlling value!



Static Sensitization (contd.)

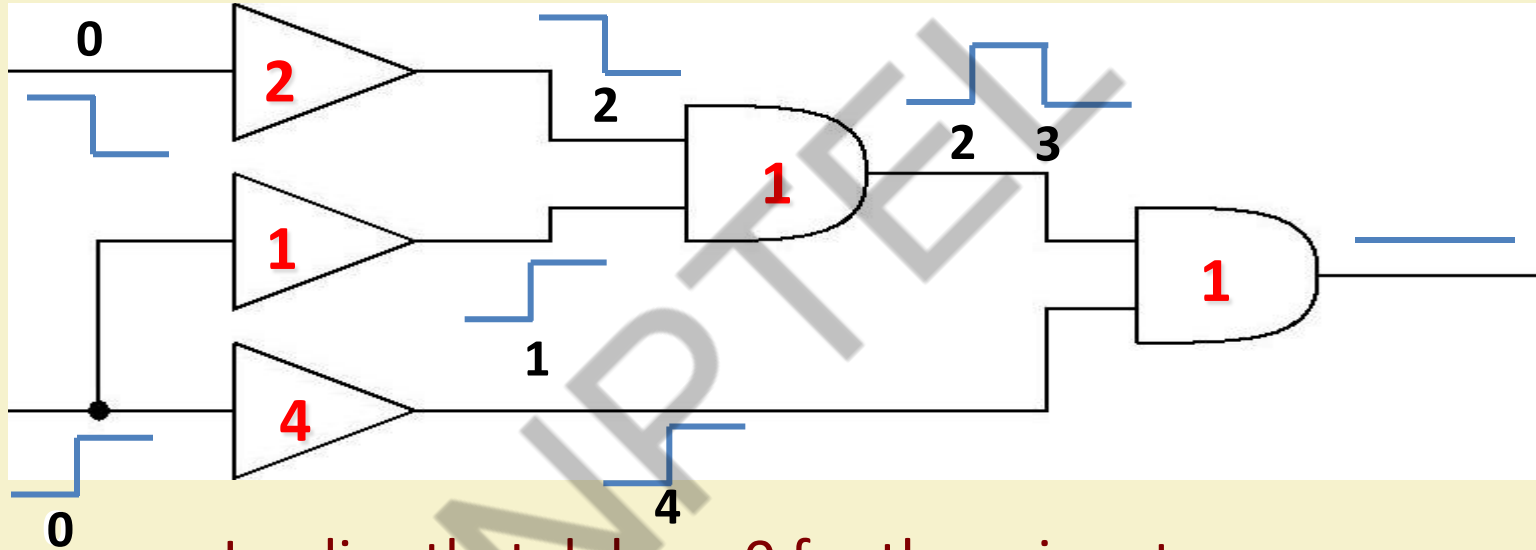
- The (dashed) path is responsible for delay!
- Delay underestimation by static sensitization (delay = 2 when true delay = 3)
 - incorrect condition



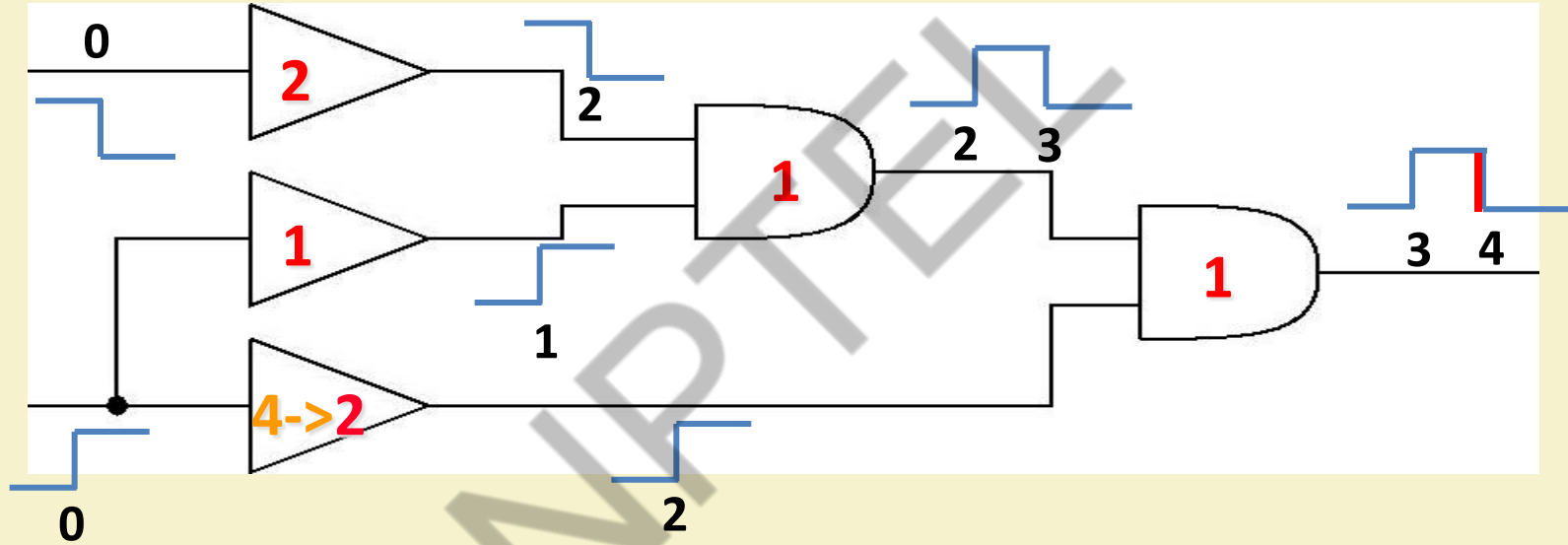
What is Wrong with Static Sensitization?

- 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!



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

What is Wrong with Timing Simulation?

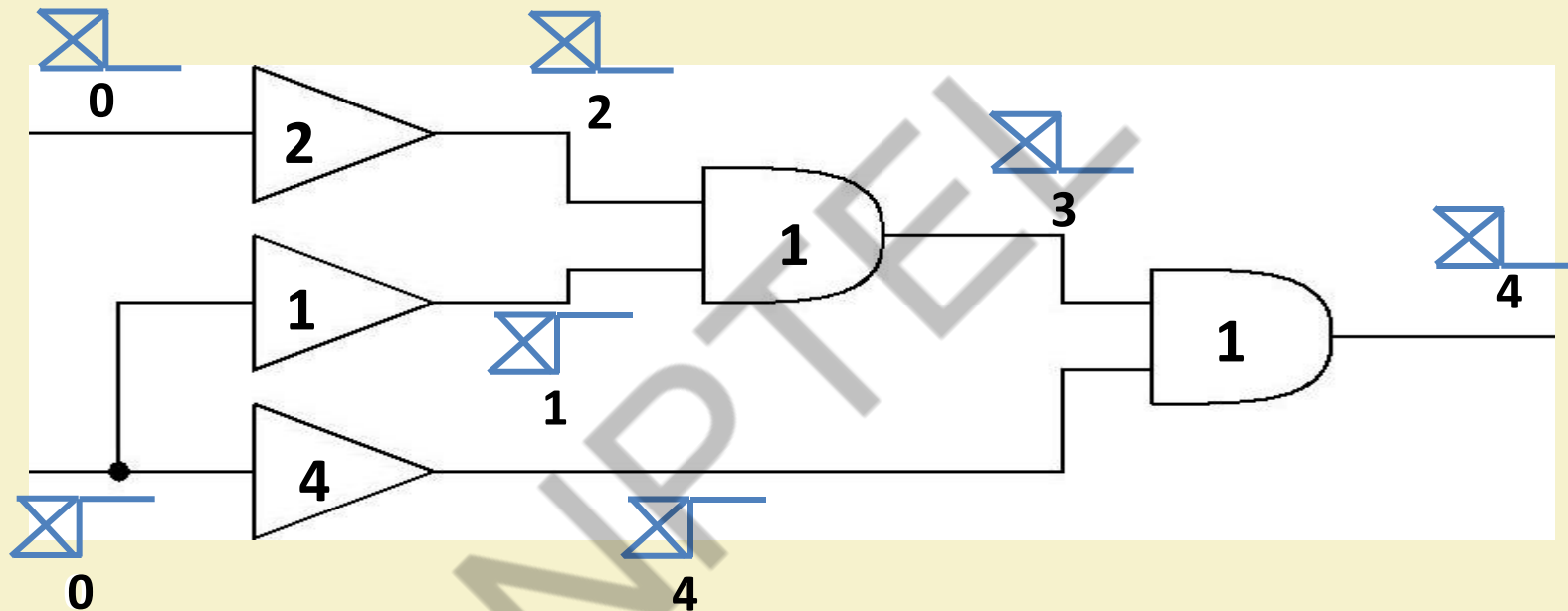
- If gate delays are reduced, delay estimates can increase.
- Not acceptable since
 - Gate delays are just upper-bounds, actual delay is in $[0, d]$.
 - Delay uncertainty due to manufacturing.
 - We are implicitly analyzing a family of circuits where gate delays are within the upper-bounds.


Monotone Speedup Property

- Definition: For any circuit C , if
 - a) C' is obtained from C by reducing some gate delays, and
 - b) $\text{delay_estimate}(C') \leq \text{delay_estimate}(C)$,then delay_estimate has **Monotone Speedup** property.

Timing simulation does not have this property.

Timing Simulation Revisited



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

What we just saw ...

- Timed 3-valued (0,1,X) simulation
 - called X-valued simulation.
- Monotone speedup property is *satisfied*.

SAT Based False Path Analysis

- Satisfiability (SAT) solvers are used for solving a wide range of problems.
- Modern SAT solvers run very fast and can handle a large number of variables.
- Basically, given a Boolean function F in product-of-sum form, a SAT solver tries to find some assignment of the variables for which $F = 1$.

The SAT Formulation

Decision problem:

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

Idea:

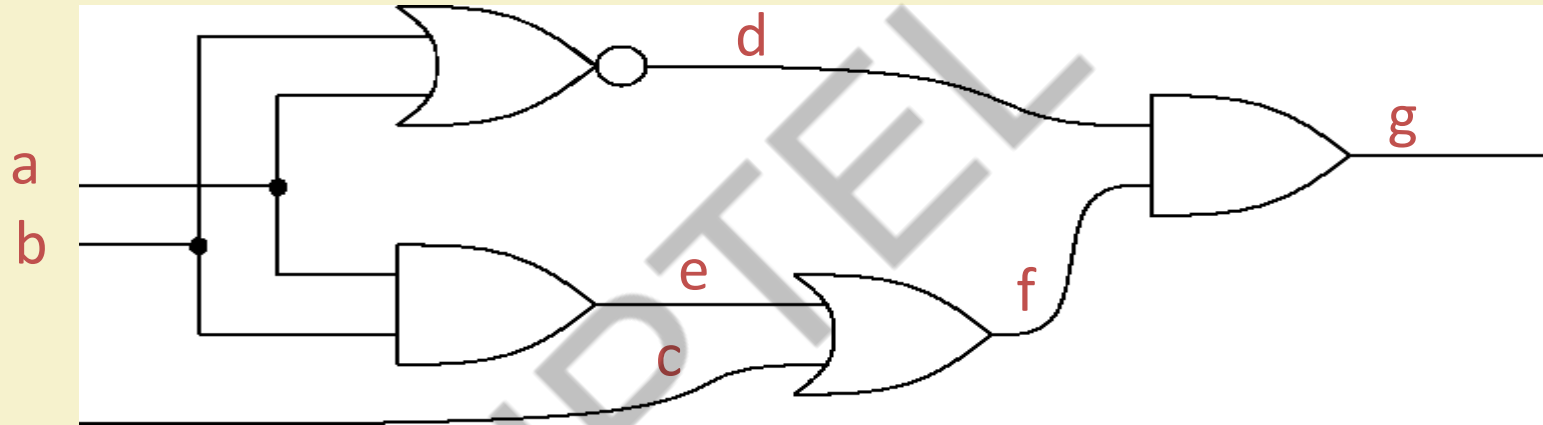
1. Characterize the set of all input vectors $S(T)$ that make the output stable no later than $t = T$.
2. Check if $S(T)$ contains S = all possible input vectors.

This check is solved as a SAT problem:

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

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

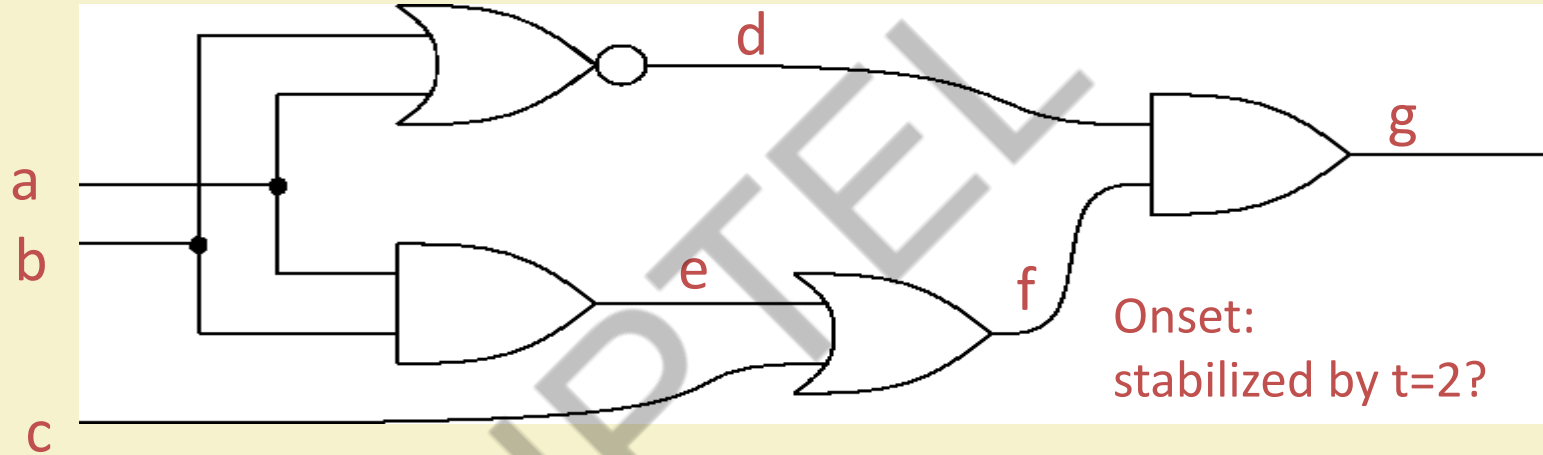
Example



Assume all the PIs arrive at $t = 0$, all gate delays = 1.

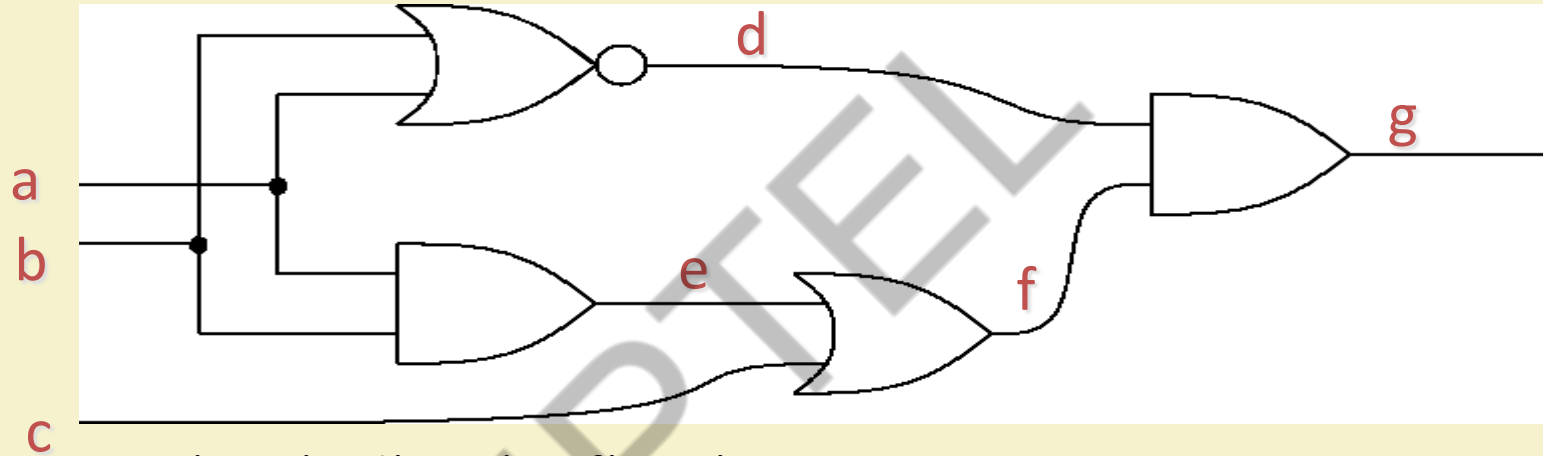
Is the output stable at time $t > 2$?

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



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

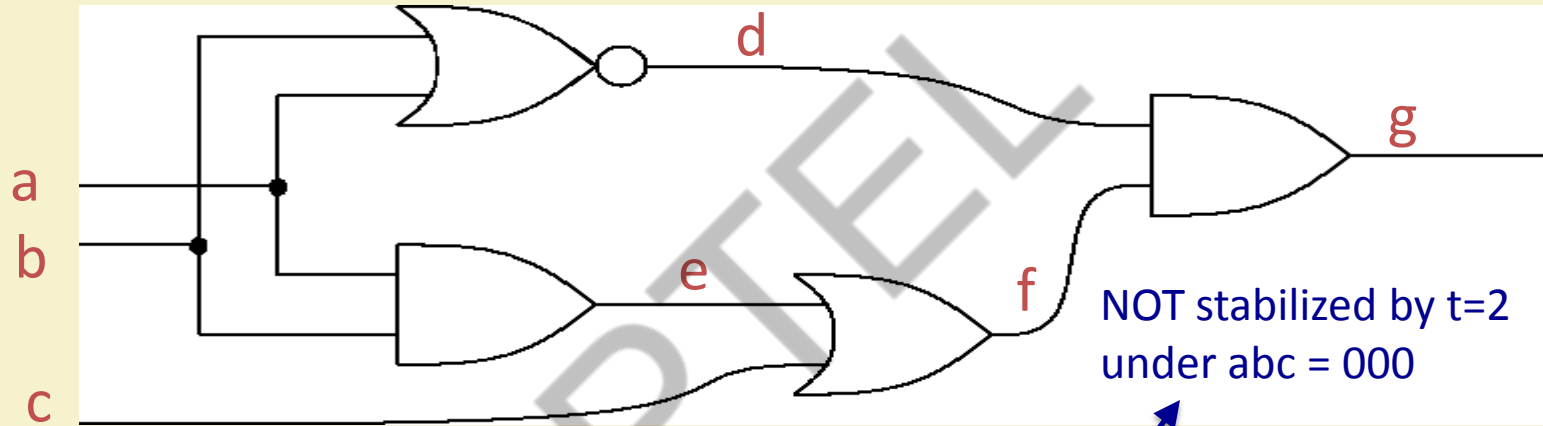
$g(0, t=2)$: the set of input vectors under which
 g gets stable to value = 0 no later than $t=2$



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

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

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



NOT stabilized by $t=2$
under $abc = 000$

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

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

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

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.

END OF LECTURE 36





IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

Lecture 37: TIMING DRIVEN PLACEMENT

PROF. INDRANIL SENGUPTA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Timing Driven Placement (TDP)

- TDP optimizes circuit delay, either to satisfy all timing constraints, or to achieve the greatest possible clock frequency.
- It uses the results of STA to identify critical nets and attempts to improve signal propagation delay through those nets.
- TDP minimizes one or both of the following:

a) Worst negative-slack (WNS)

b) Total negative slack (TNS)

$$WNS = \min_{\tau \in T} (slack(\tau))$$

$$TNS = \sum_{\tau \in T, slack(\tau) < 0} slack(\tau)$$

where T is the set of timing endpoints (i.e. primary outputs, or inputs to flip-flops).

Techniques for Timing-Driven Placement

- Algorithmic techniques for TDP can be categorized as *net-based*, *path-based*, or *integrated*.
- Two types of net-based techniques:
 1. Delay budgeting, which assigns upper bounds to the timing or length of individual nets.
 2. Net weighting, which assign higher priorities to critical nets during placement.
- Path-based techniques seek to shorten or speedup all timing-critical paths rather than individual nets.
 - More accurate but does not scale to large designs because number of paths can grow exponentially with number of gates (e.g. multiplier).

- Both path-based and net-based approaches rely on support within the placement algorithm, and require a dedicated infrastructure for incremental calculation of timing statistics and parameters.
- Integrated techniques typically use constraint-driven mathematical formulation in which STA results are incorporated as constraints and possibly in the objective function.
- In practice, some industrial flows do not incorporate timing-driven methods during initial placement because timing information can be quite inaccurate until locations are available.
 - Instead, subsequent placement iterations, especially during detailed placement, perform timing optimizations.

Net Based Techniques

- These approaches impose either quantitative priorities that reflect timing criticality (**net weights**), or upper bounds on the timing of nets in the form of net constraints (**delay budgets**).
- Net weights are more effective at the early design stages, while delay budgets are more meaningful if timing analysis is more accurate.

(a) Net Weighting

- A traditional placer optimizes total wirelength and routability.
- To account for timing, a placer can minimize the total weighted wirelength, where each net is assigned a net weight.
 - The higher the net weight is, the more timing-critical the net is.
- Net weights can be assigned either statically or dynamically to improve the timing.

Static Net Weights

- They are computed before placement and do not change.
- They are usually based on slack: the more critical the net (i.e. smaller slack), greater is the weight.
- Static net weights can be either discrete:

$$w = \begin{cases} \omega_1 & \text{if } slack > 0 \\ \omega_2 & \text{if } slack \leq 0 \end{cases} \quad \text{where } \omega_1 > 0, \omega_2 > 0, \text{ and } \omega_2 > \omega_1$$

- Or they can be continuous:

$$w = \left(1 - \frac{slack}{t} \right)^\alpha$$

where t is the longest path delay and α is a criticality exponent.

- Alternatively, net weights can be assigned based on sensitivity, as:

$$w = w_o + \alpha(\text{slack}_{\text{target}} - \text{slack}) \cdot s_w^{\text{SLACK}} + \beta \cdot s_w^{\text{TNS}}$$

where w_o is the original net weight

slack is the computed slack value of the net

$\text{slack}_{\text{target}}$ is the target slack of the design

s_w^{SLACK} is the slack sensitivity to the weight of the net

s_w^{TNS} is the TNS sensitivity to the net weight

α and β are constants bounds on the net weight change that control the tradeoff between WNS and TNS.

TNS: Total Negative Slack
WNS: Worst Negative Slack

Dynamic Net Weights

- They are computed during placement iterations and keep an updated timing profile.
- This can be more effective than static net weights, since they are computed before placement, and can become outdated when net lengths change.
- Estimated slack of a net at iteration k can be computed as:

$$slack_k = slack_{k-1} - s_L^{DELAY} \cdot \Delta L$$

where ΔL is the change in wirelength between iterations $(k-1)$ and k

$slack_k$ is the slack at iteration k

s_L^{DELAY} is the delay sensitivity to the wirelength

- After the timing information has been updated, the net weights should be adjusted accordingly.
 - This incremental method of weight modification is based on previous iterations.
- The net criticality at iteration k is computed as:

$$v_k = \begin{cases} \frac{1}{2}(v_{k-1} + 1) & \text{if among the top 3\% of critical nets} \\ \frac{1}{2}v_{k-1} & \text{otherwise} \end{cases}$$

- And then the net weight is updated as:

$$w_k = w_{k-1} \cdot (1 + v_k)$$

Integrated Technique using Linear Programs

- Unlike net-based methods, where the timing requirements are mapped to net weights or net constraints, path-based methods directly optimize the design's timing.
 - As the number of paths can grow quickly, this method is much slower than net-based approaches.
- To improve scalability, timing analysis may be captured by a set of constraints and an optimization objective.
 - For example, in a linear programming framework.

- In the context of timing-driven placement, a linear program (LP) minimizes a function of slack (e.g. TNS), subject to two main types of constraints:
 1. *Physical constraints*, which define the locations of the cells.
 2. *Timing constraints*, which define the slack requirements.
- In addition, some electrical constraints may also be incorporated.

Physical Constraints:

- Given a set of cells V and the set of nets E , we define the notations:
 - x_v and y_v denote the center of cell $v \in V$
 - V_e denotes the set of cells connected to net $e \in E$
 - $left(e)$, $right(e)$, $bottom(e)$, and $top(e)$ respectively denote the coordinates of the left, right, bottom, and top boundaries of e 's bounding box
 - $\delta_x(v,e)$ and $\delta_y(v,e)$ denote pin offsets from x_v and y_v for v 's pin connected to e

- Then, for all $v \in V_e$:
$$\begin{aligned} \text{left}(e) &\leq x_v + \delta_x(v, e) \\ \text{right}(e) &\geq x_v + \delta_x(v, e) \\ \text{bottom}(e) &\leq y_v + \delta_y(v, e) \\ \text{top}(e) &\geq y_v + \delta_y(v, e) \end{aligned}$$

Every pin of a given net e must be contained within e 's bounding box.

- Then, e 's *half-parameter wire-length* (HPWL) is defined as

$$L(e) = \text{right}(e) - \text{left}(e) + \text{top}(e) - \text{bottom}(e)$$

Timing Constraints:

- For timing constraints, let
 - $t_{GATE}(v_i, v_o)$ be the gate delay from an input pin v_i to the output pin v_o for cell v
 - $t_{NET}(e, u_o, v_i)$ be net e 's delay from cell u 's output pin u_o to cell v 's input pin v_i
 - $AAT(v_j)$ be the arrival time on pin j of cell v
- For every input pin v_i of cell v , the arrival time at v_i is the arrival time at the previous output pin u_o of cell u plus the net delay:

$$AAT(v_i) = AAT(u_o) + t_{NET}(u_o, v_i)$$

- For every output pin v_o of cell v , the arrival time at v_o should be greater than or equal to the arrival time plus gate delay of each input v_i . That is, for each input v_i of cell v ,

$$AAT(v_o) \geq AAT(v_i) + t_{GATE}(v_i, v_o)$$

- For every pin τ_p in a sequential cell τ , the slack is computed as the difference between the required arrival time $RAT(\tau_p)$ and actual arrival time $AAT(\tau_p)$,

$$slack(\tau_p) \leq RAT(\tau_p) - AAT(\tau_p)$$

- Upper bound all pin slacks by zero (or a small positive value),

$$slack(\tau_p) \leq 0$$

Objective Functions:

a) Optimize the total negative slack (TNS) $\max : \sum_{\tau_p \in Pins(\tau), \tau \in T} slack(\tau_p)$

where $Pins(\tau)$ is the set of pins of cell τ , and T is the set of all sequential elements or endpoints.

b) Optimize the worst negative slack (WNS) $\max : WNS$
where $WNS \leq slack(\tau_p)$ for all pins.

c) Optimize some combination of wirelength and slack
where E is the set of all nets, α is a constant between 0 and 1 that trades off WNS and wirelength, and $L(e)$ is the HPWL of net e . $\min : \sum_{e \in E} L(e) - \alpha \cdot WNS$

END OF LECTURE 37

