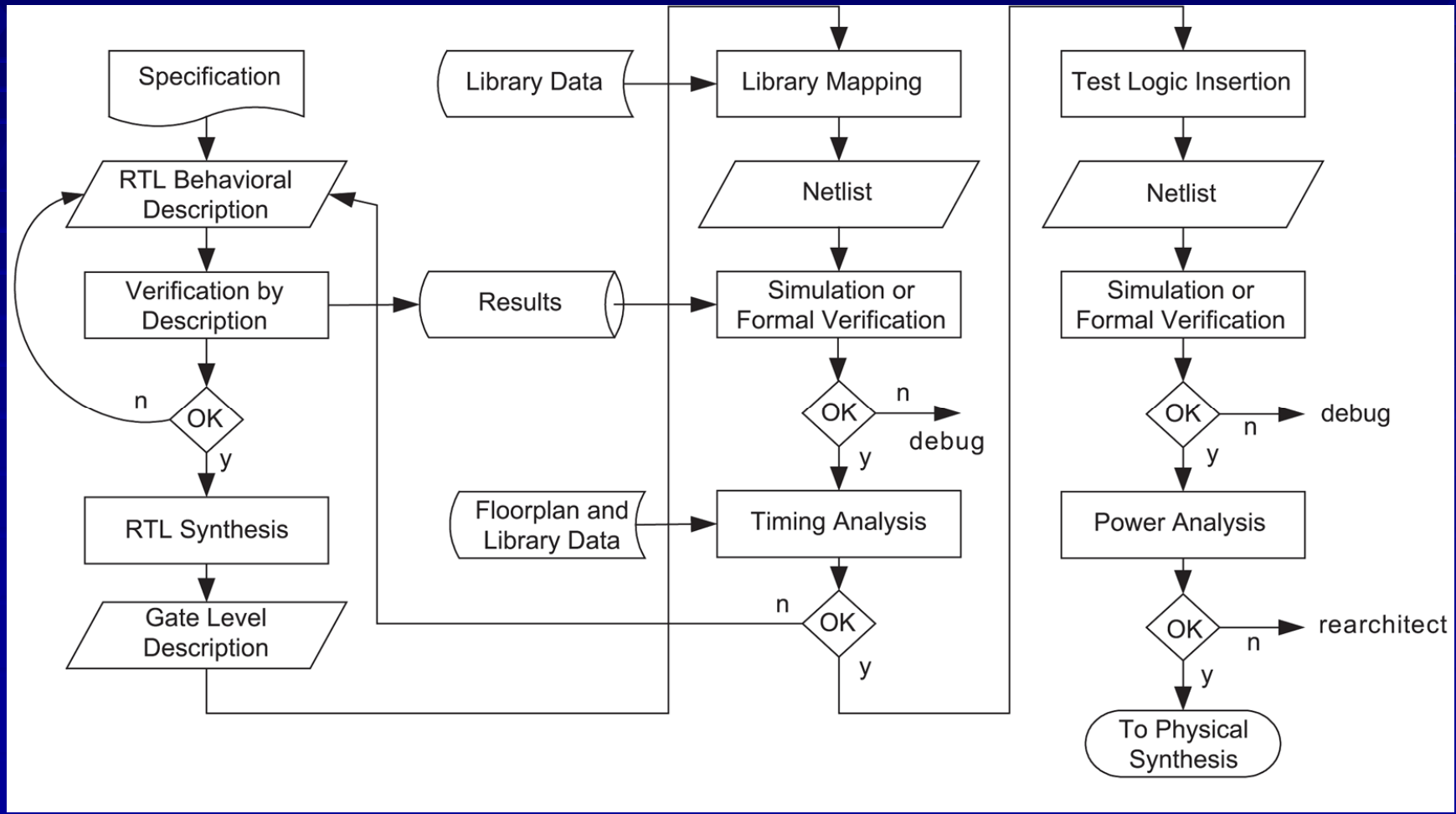
Behavioral Synthesis Design Flow ASIC

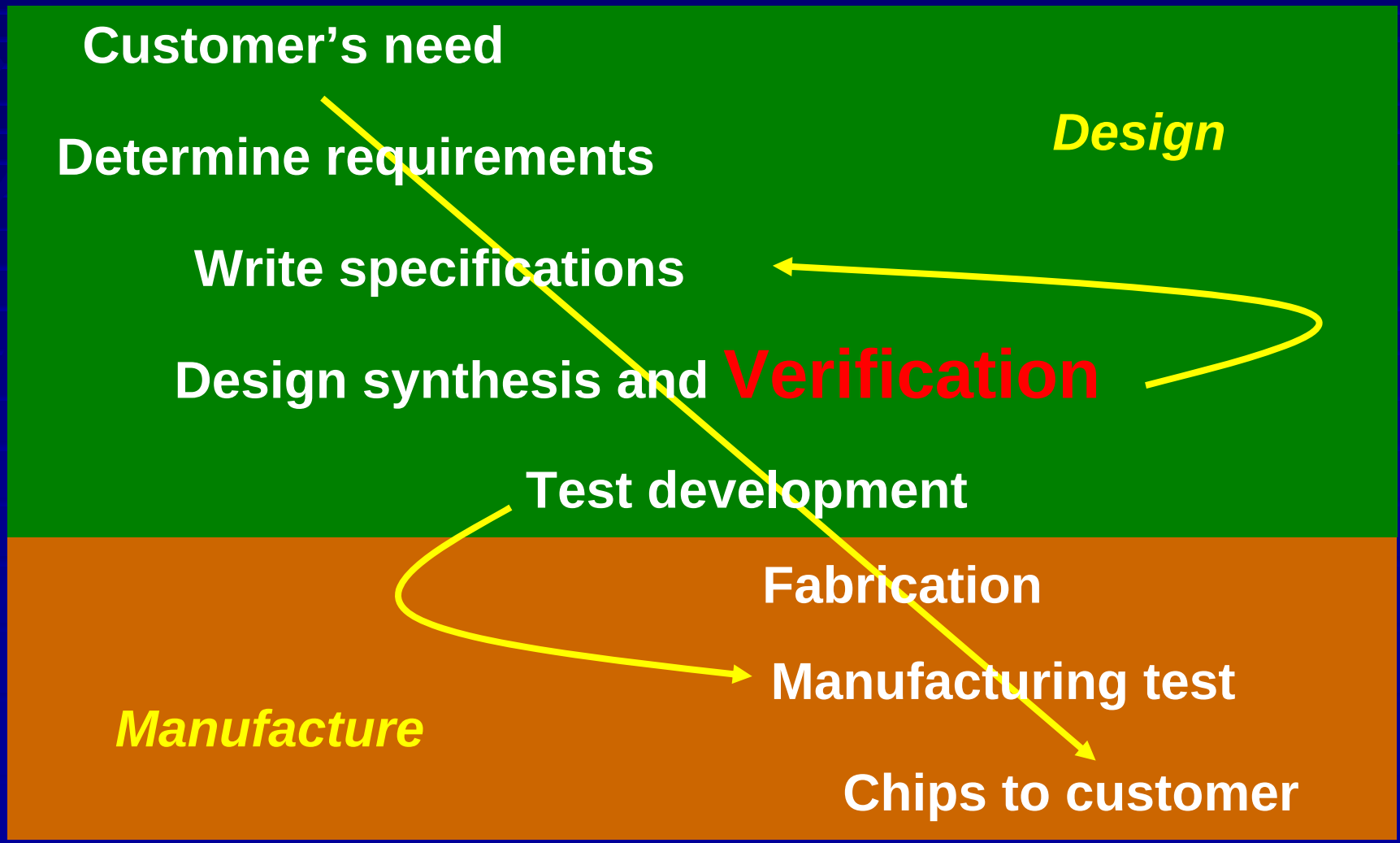


CMOS VLSI Design, Weste & Harris: Ch. 8.4

Functional or Formal Verification

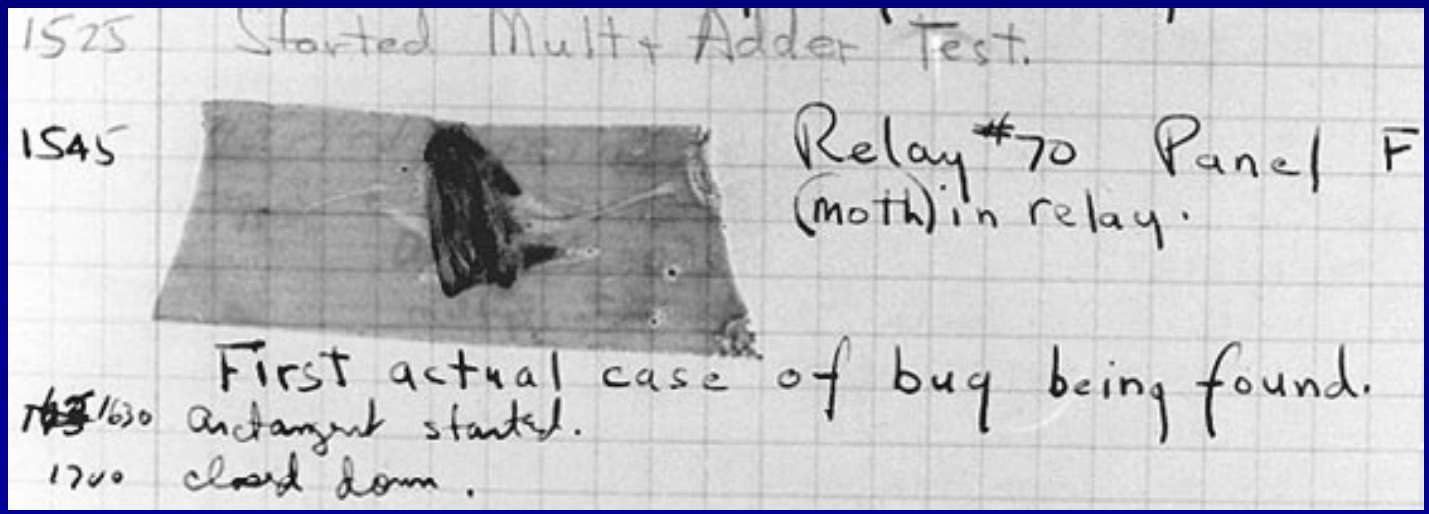
- Structural netlist performed the same as original behavioral VHDL
- Synthesizer may produce incorrect netlist from poorly written behavioral code
- Verification Strategy => Logic Test Bench
- And/Or Formal Verification: compares the logical equivalence of two descriptions
- Semantic Check: Ensuring all bus assignments march in bit width
- Structural Check: Ensure all outputs connected

VLSI Realization Process



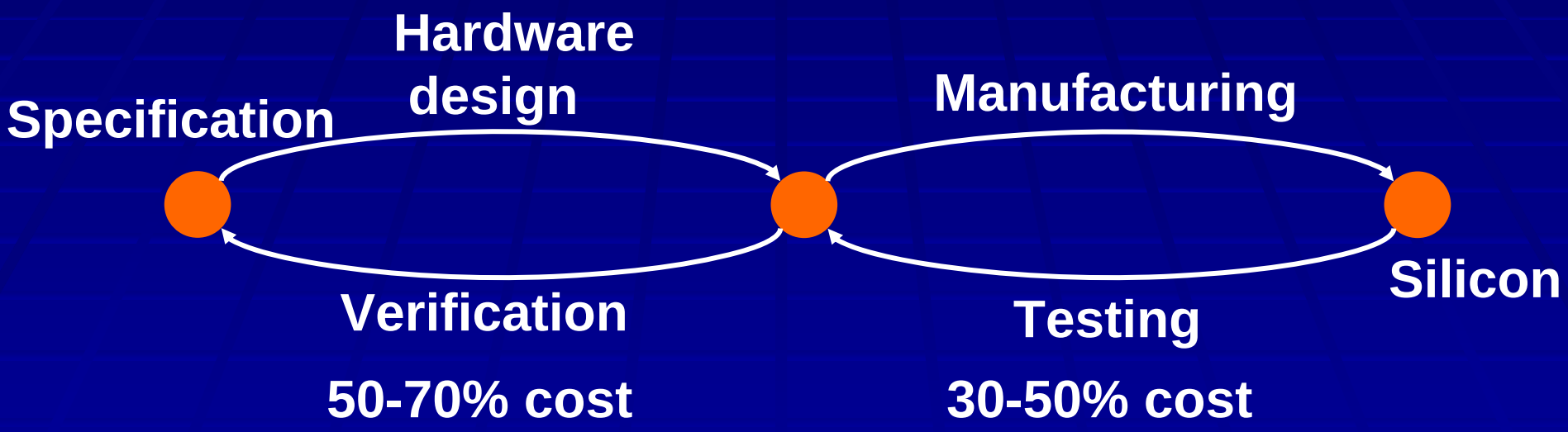
Origin of "Debugging"

Thomas Edison wrote in a letter in 1878: "*It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise—this thing gives out and* [it is] *then that "Bugs" — as such little faults and difficulties are called — show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached."* An interesting example of "debugging" was in 1945 when a computer failure was traced down to a moth that was caught in a relay between contacts (Figure 3-1).



D. Gizopoulos (Editor), *Advances in Electronic Testing: Challenges and Methodologies*, Springer, 2006, Chapter 3, "Silicon Debug," by D. Josephson and B. Gottlieb.

Verification and Testing

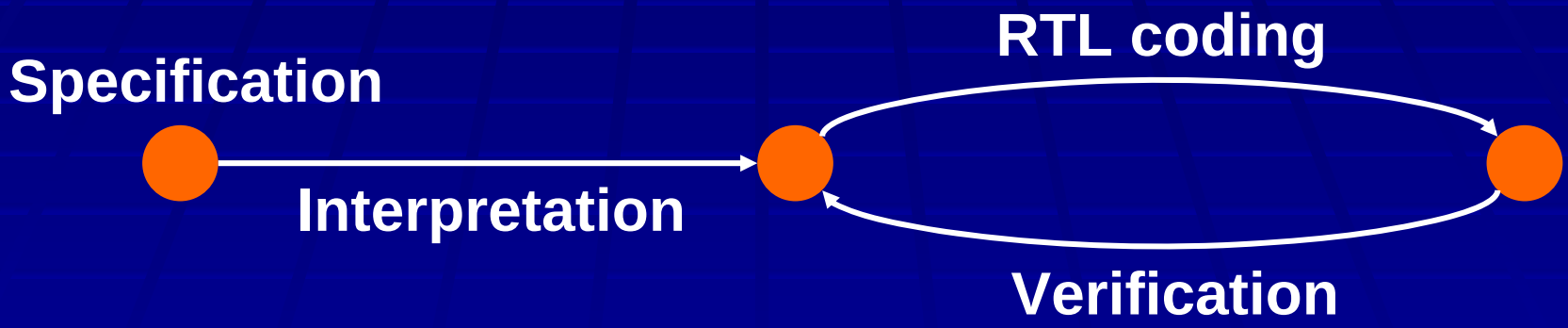


Definitions

- Verification: Predictive analysis to ensure that the synthesized design, when manufactured, will perform the given I/O function.
- Alternative Definition: Verification is a process used to demonstrate the functional correctness of a design.

What is Being Verified?

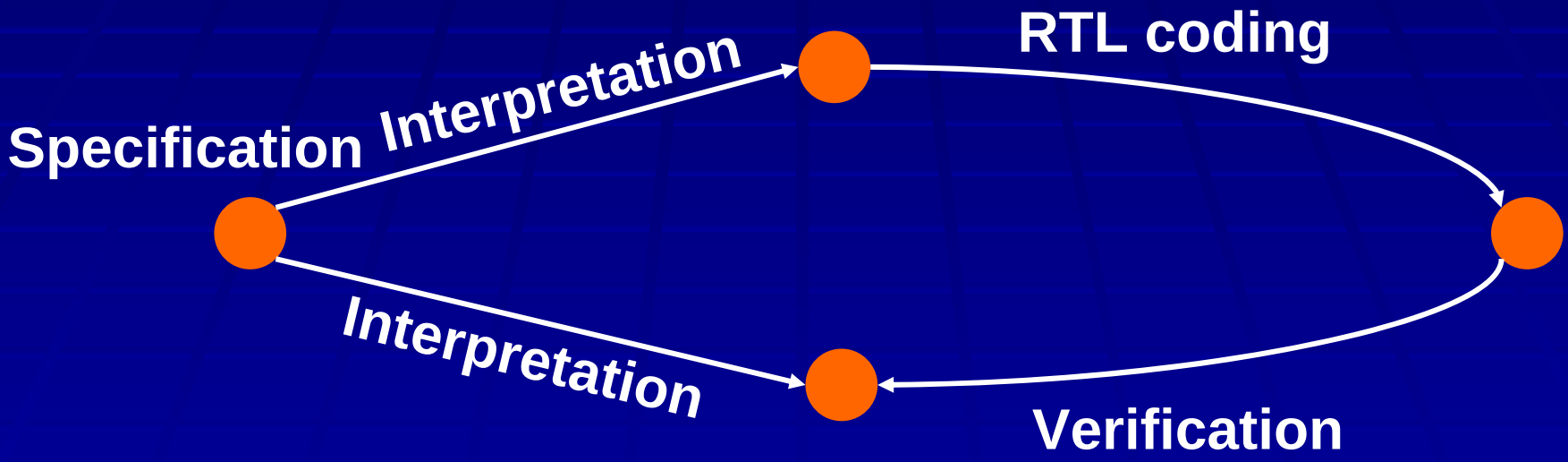
- Given a set of specification,
- Does the design do what was specified?



J. Bergeron, *Writing Testbenches: Functional Verification Of HDL Models*, Springer, 2000.

Avoiding Interpretation Error

- Use redundancy



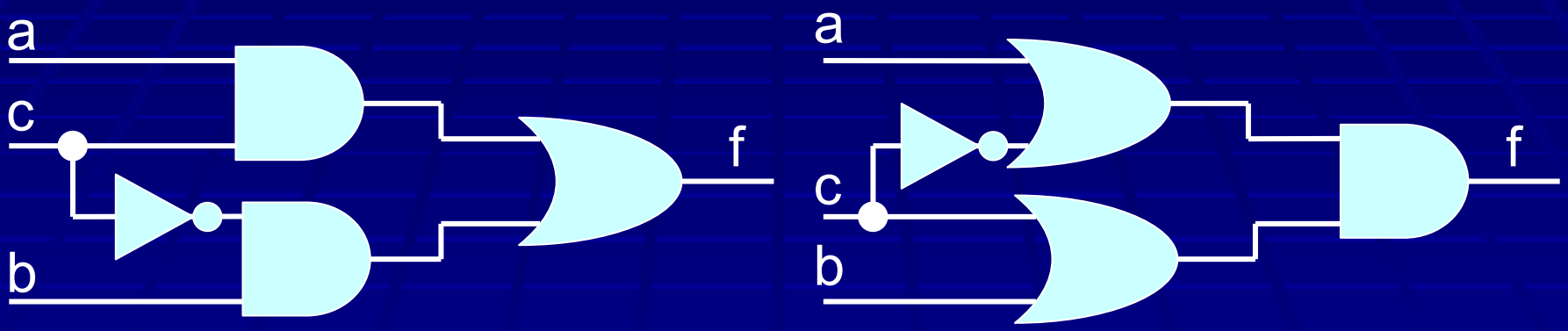
Methods of Verification

- Simulation: Verify input-output behavior for selected cases.
- Formal verification: Exhaustively verify input-output behavior:
  - Equivalence checking
  - Model checking
  - Symbolic simulation

Equivalence Checking

- Logic equivalence: Two circuits implement identical Boolean function.
- Logic and temporal equivalence: Two finite state machines have identical input-output behavior (machine equivalence).
- Topological equivalence: Two netlists are identical (graph isomorphism).
- Reference: S.-Y. Hwang and K.-T. Cheng, *Formal Equivalence Checking and Design Debugging*, Springer, 1998.

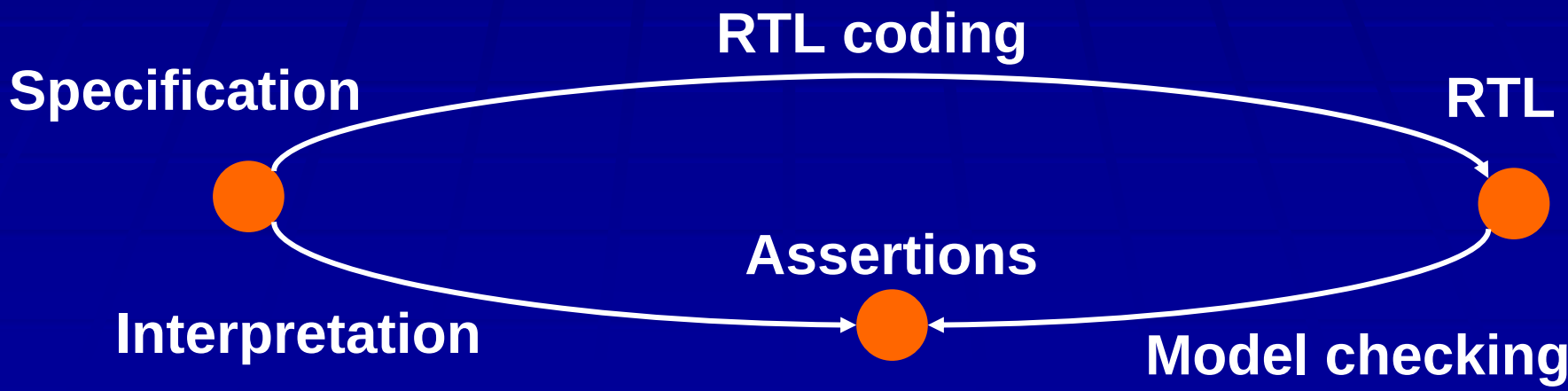
Compare Two Circuits



- Graphs isomorphic?
- Boolean functions identical?
- Timing behaviors identical?

Model Checking

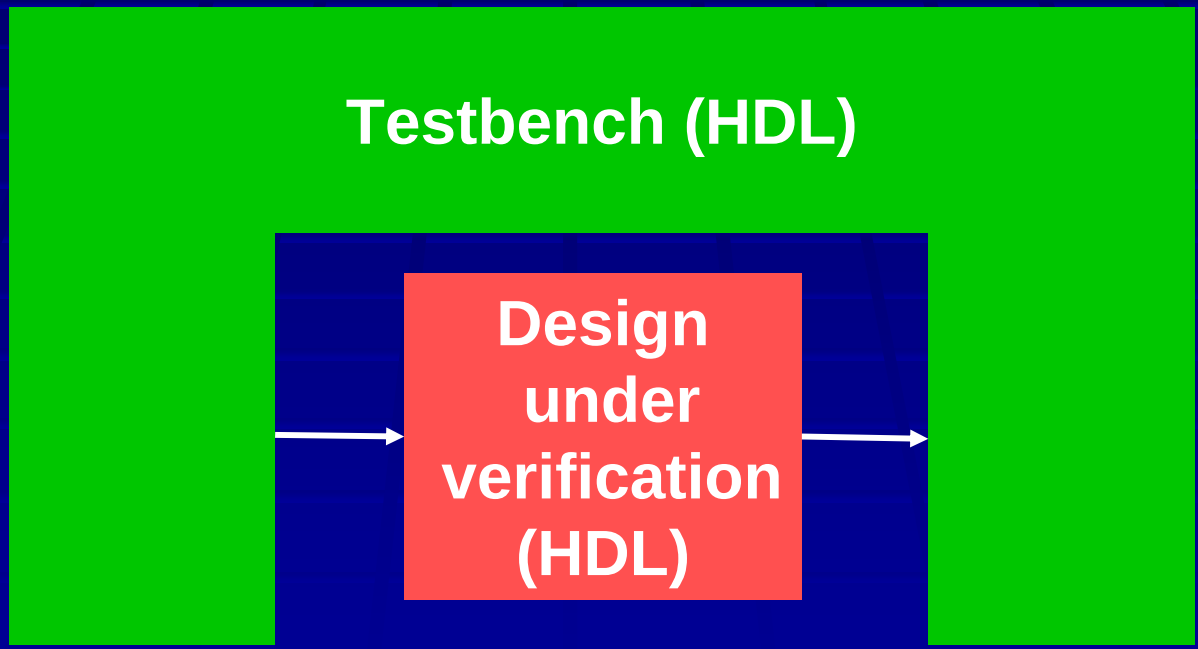
- Construct an abstract model of the system, usually in the form of a finite-state machine (FSM).
- Analytically prove that the model does not violate the properties (assertions) of original specification.
- Reference: E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*, MIT Press, 1999.



Symbolic Simulation

- Simulation with algebraic symbols rather than numerical values.
- Self-consistency: A complex (more advanced) design produces the same result as a much simpler (and previously verified) design.
- Reference: R. B. Jones, *Symbolic Simulation Methods for Industrial Formal Verification*, Springer, 2002.

Simulation: Testbench



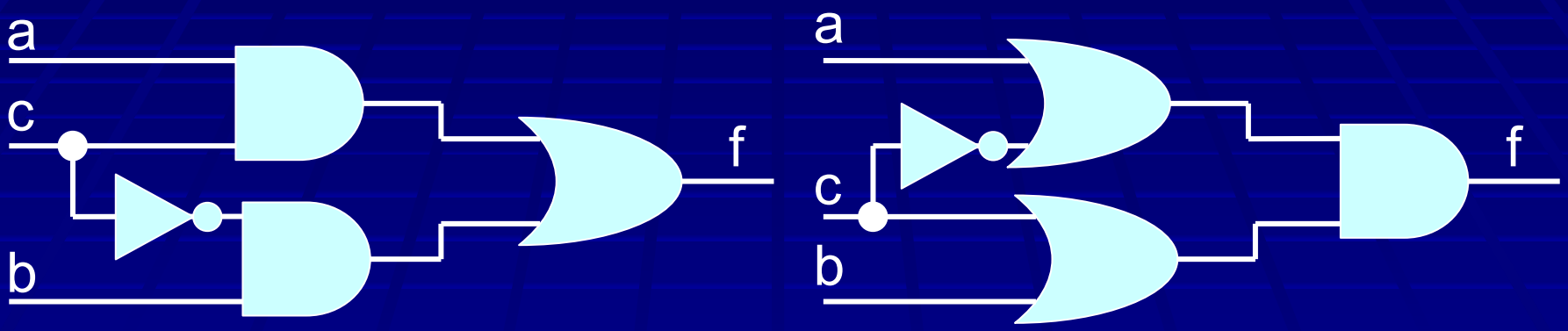
Testbench

- HDL code:
  - Generates stimuli
  - Checks output responses
- Approaches:
  - Blackbox
  - Whitebox
  - Greybox
- Metrics (unreliable):
  - Statement coverage
  - Path coverage
  - Expression or branch coverage

Equivalence Checking

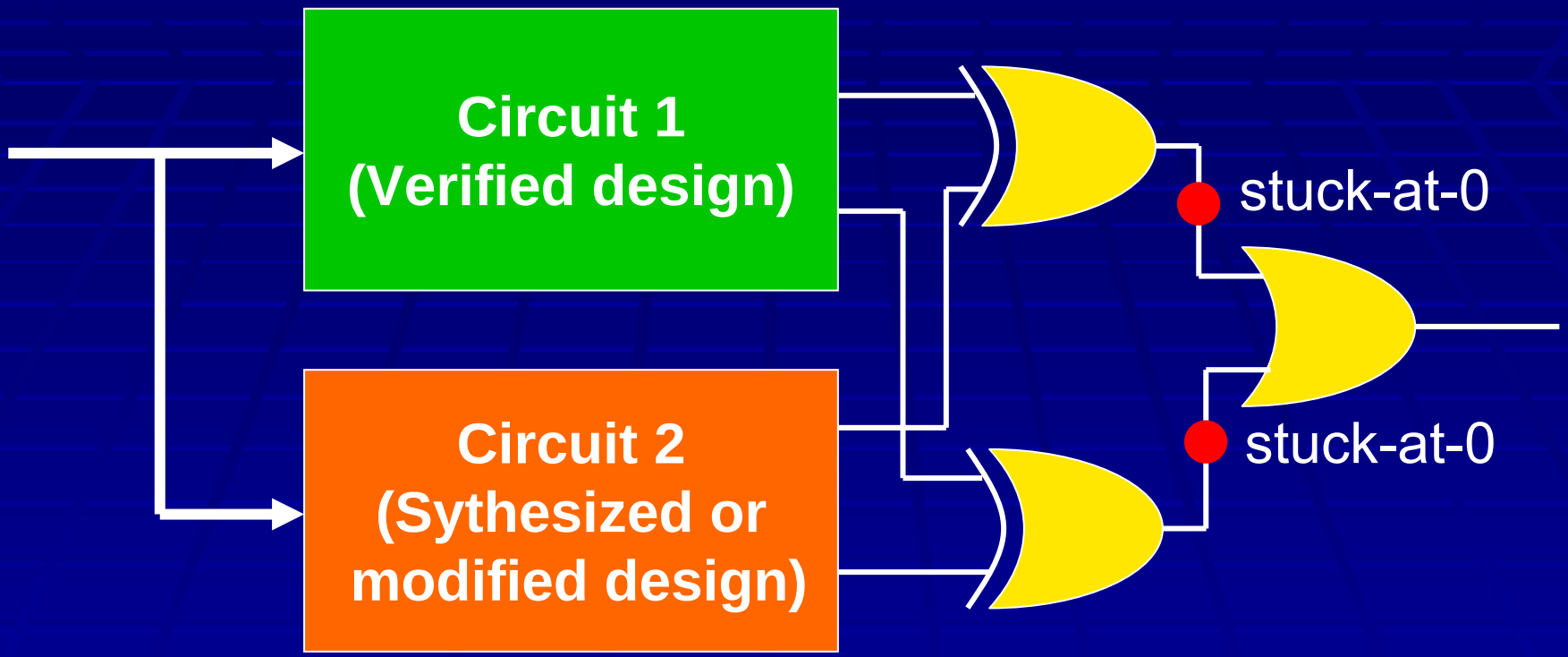
- Definition: Establishing that two circuits are functionally equivalent.
- Applications:
  - Verify that a design is identical to specification.
  - Verify that synthesis did not change the function.
  - Verify that corrections made to a design did not create new errors.

Compare Two Circuits



- Are graphs isomorphic? Yes
- Else, are Boolean functions identical? Yes
- Then, are timing behaviors identical? Yes

ATPG Approach (Miter)

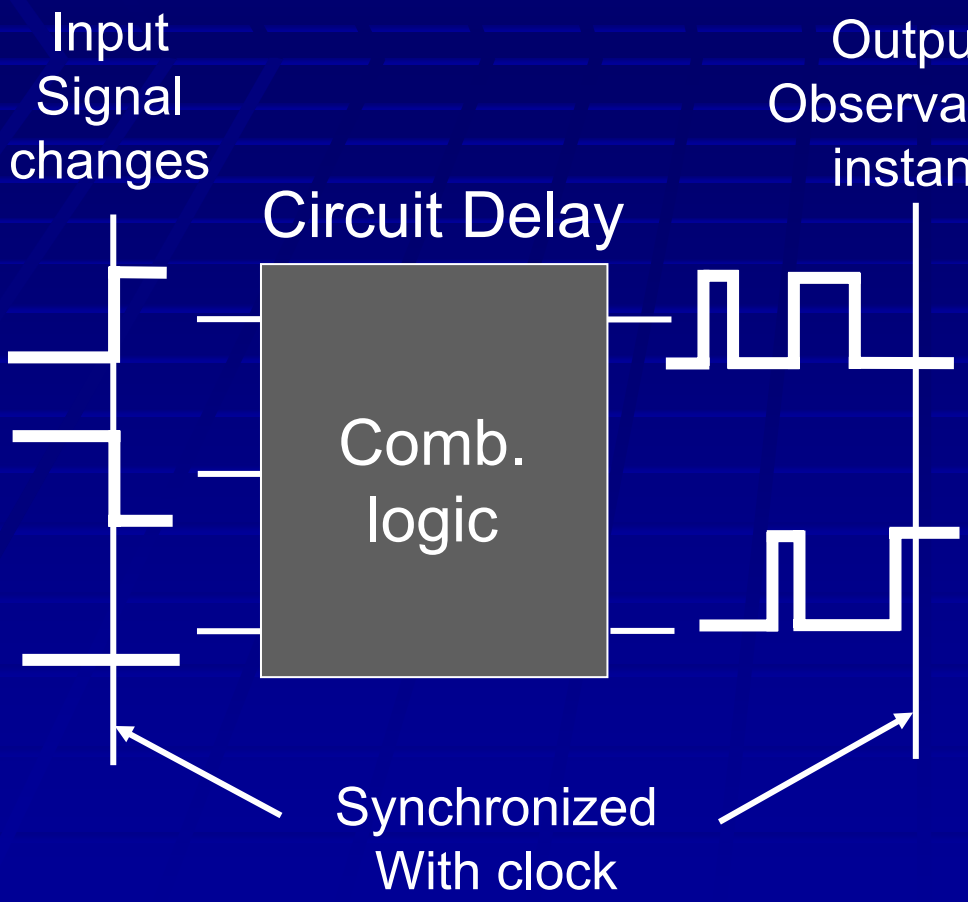


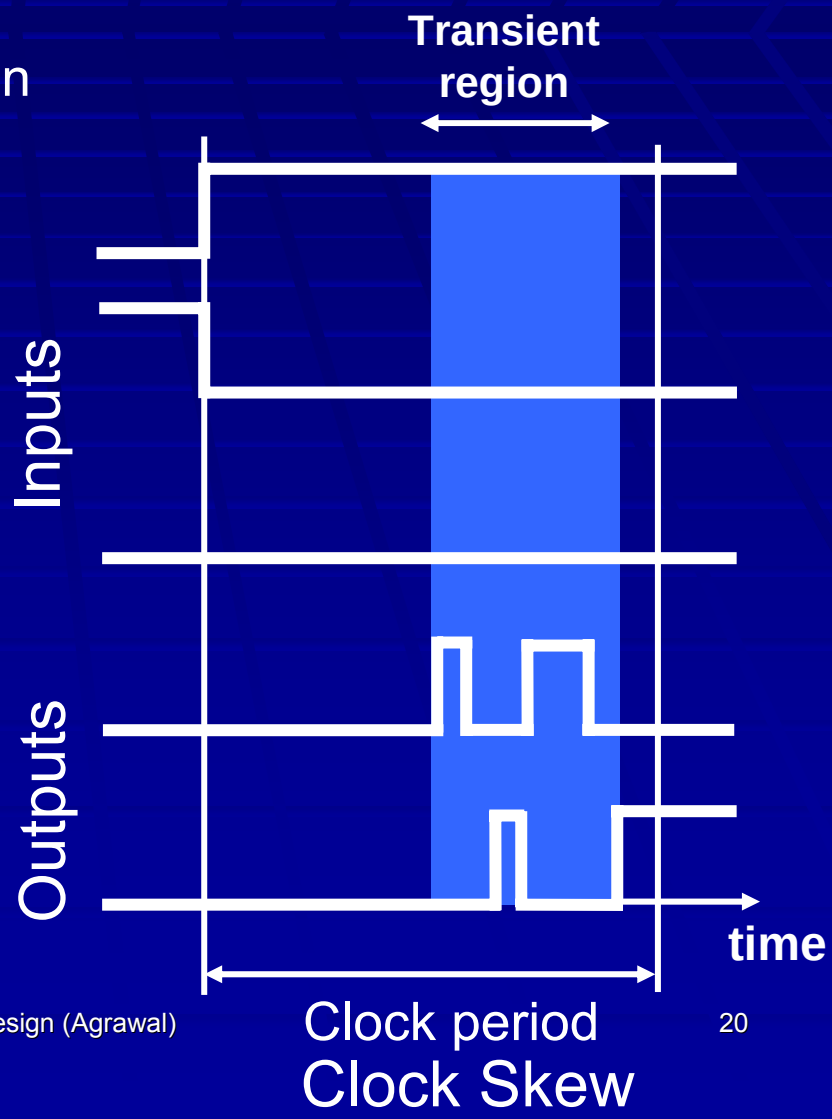
- Redundancy of a stuck-at-0 fault, checked by ATPG, establishes equivalence of the corresponding output pair.
- If the fault is detectable, its tests are used to diagnose the differences.

Timing Analysis

- Timing requirements needs to be established: i.e. the adder may add but does it add fast enough?
- Structural level: Actual cycle time has to be met
- Static Timing Analysis is used to verify timing: It verifies setup time (max-delay) and hold time (min-delay) violations
- STA suffers from *false path* problem

Digital Circuit Timing





Timing Analysis and Optimization

- Timing analysis
  - Dynamic analysis: Simulation.
  - Static timing analysis (STA): Vector-less topological analysis of circuit.
- Timing optimization
  - Performance
  - Clock design
- Other forms of design optimization
  - Chip area
  - Testability
  - Power

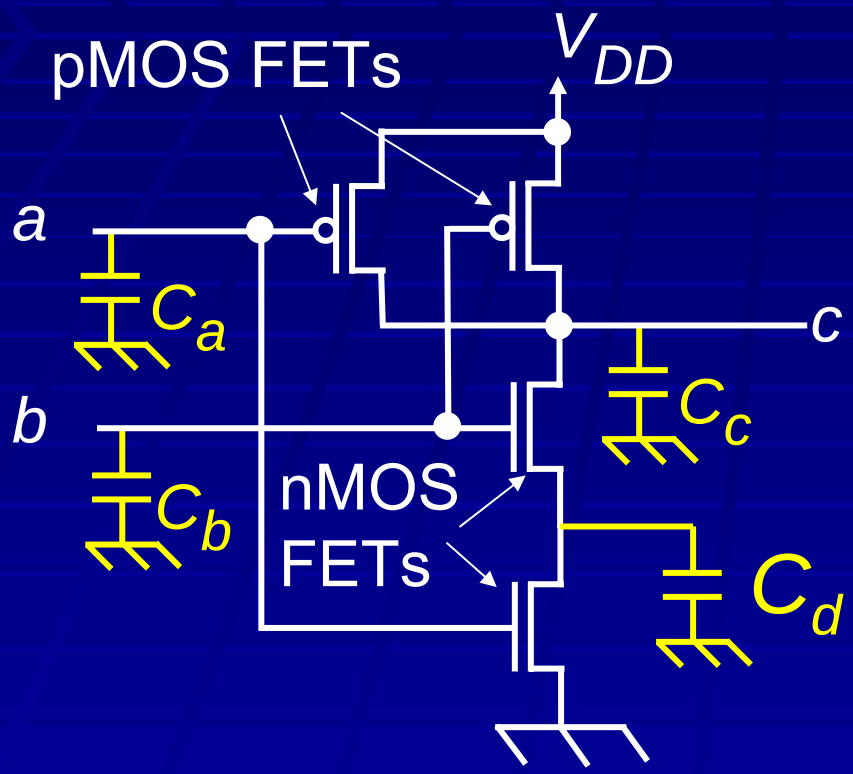
Circuit Delays

- Switching or inertial delay is the interval between input change and output change of a gate:
  - Depends on input capacitance, device (transistor) characteristics and output capacitance of gate.
  - Also depends on input rise or fall times and states of other inputs (second-order effects).
  - Approximation: fixed rise and fall delays (or min-max delay range, or single fixed delay) for gate output.
- Propagation or interconnect delay is the time a transition takes to travel between gates:
  - Depends on transmission line effects (distributed *R*, *L*, *C* parameters, length and loading) of routing paths.
  - Approximation: modeled as lumped delays for gate inputs.

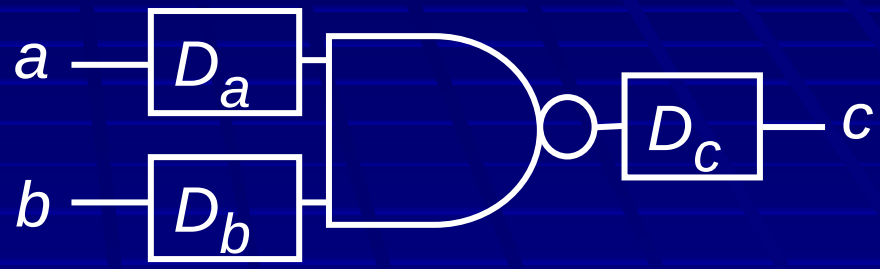
Spice

- Circuit/device level analysis
  - Circuit modeled as network of transistors, capacitors, resistors and voltage/current sources.
  - Node current equations using Kirchhoff's current law.
- Analysis is accurate but expensive
  - Used to characterize parts of a larger circuit.
- Original references:
  - L. W. Nagel and D. O. Pederson, "SPICE – Simulation Program With Integrated Circuit Emphasis," Memo ERL-M382, EECS Dept., University of California, Berkeley, Apr. 1973.
  - L. W. Nagel, *SPICE 2, A Computer program to Simulate Semiconductor Circuits*, PhD Dissertation, University of California, Berkeley, May 1975.

Logic Model of MOS Circuit



C_a, C_b, C_c and C_d are node capacitances



D_a and D_b are interconnect or propagation delays

D_c is inertial delay of gate

Spice Characterization

Input data pattern	Delay (ps)	Dynamic energy (pJ)
$a = b = 0 \rightarrow 1$	69	1.55
$a = 1, b = 0 \rightarrow 1$	62	1.67
$a = 0 \rightarrow 1, b = 1$	50	1.72
$a = b = 1 \rightarrow 0$	35	1.82
$a = 1, b = 1 \rightarrow 0$	76	1.39
$a = 1 \rightarrow 0, b = 1$	57	1.94

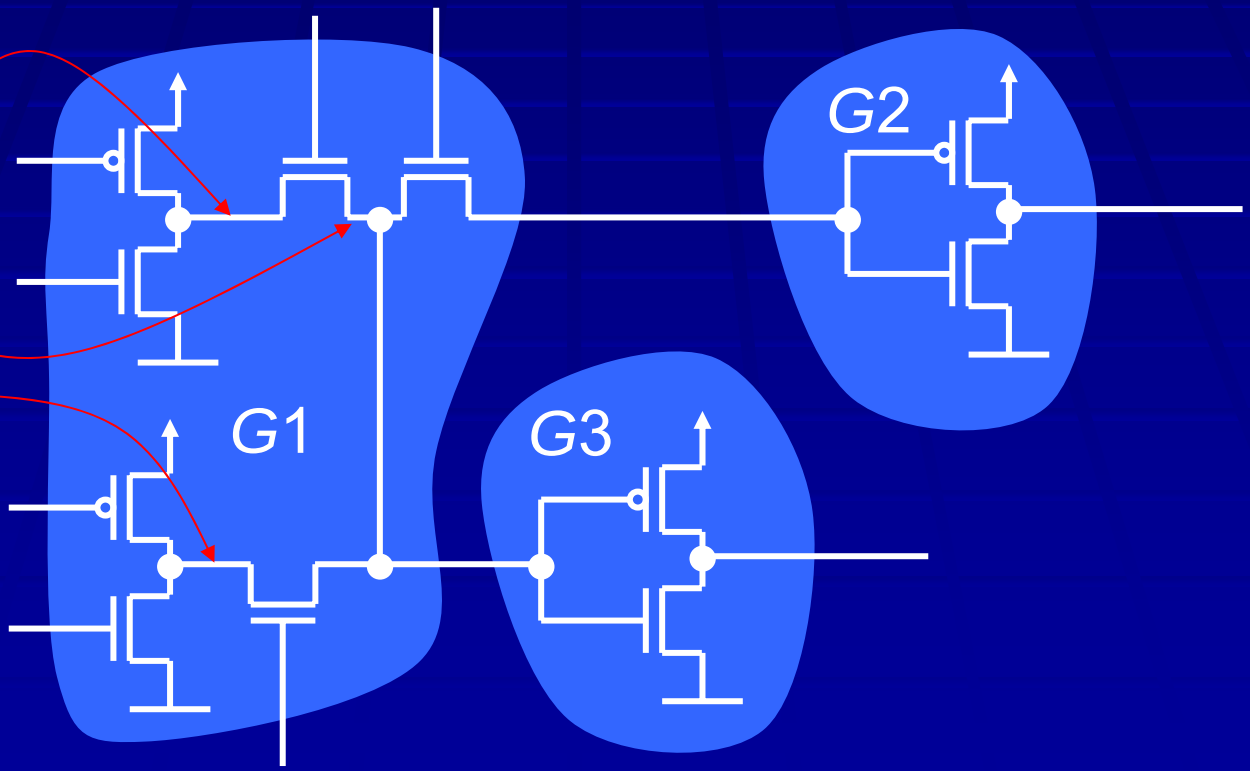
Spice Characterization (Cont.)

Input data pattern	Static power (pW)
$a = b = 0$	5.05
$a = 0, b = 1$	13.1
$a = 1, b = 0$	5.10
$a = b = 1$	28.5

Complex Gates: Switch-Level Partitions

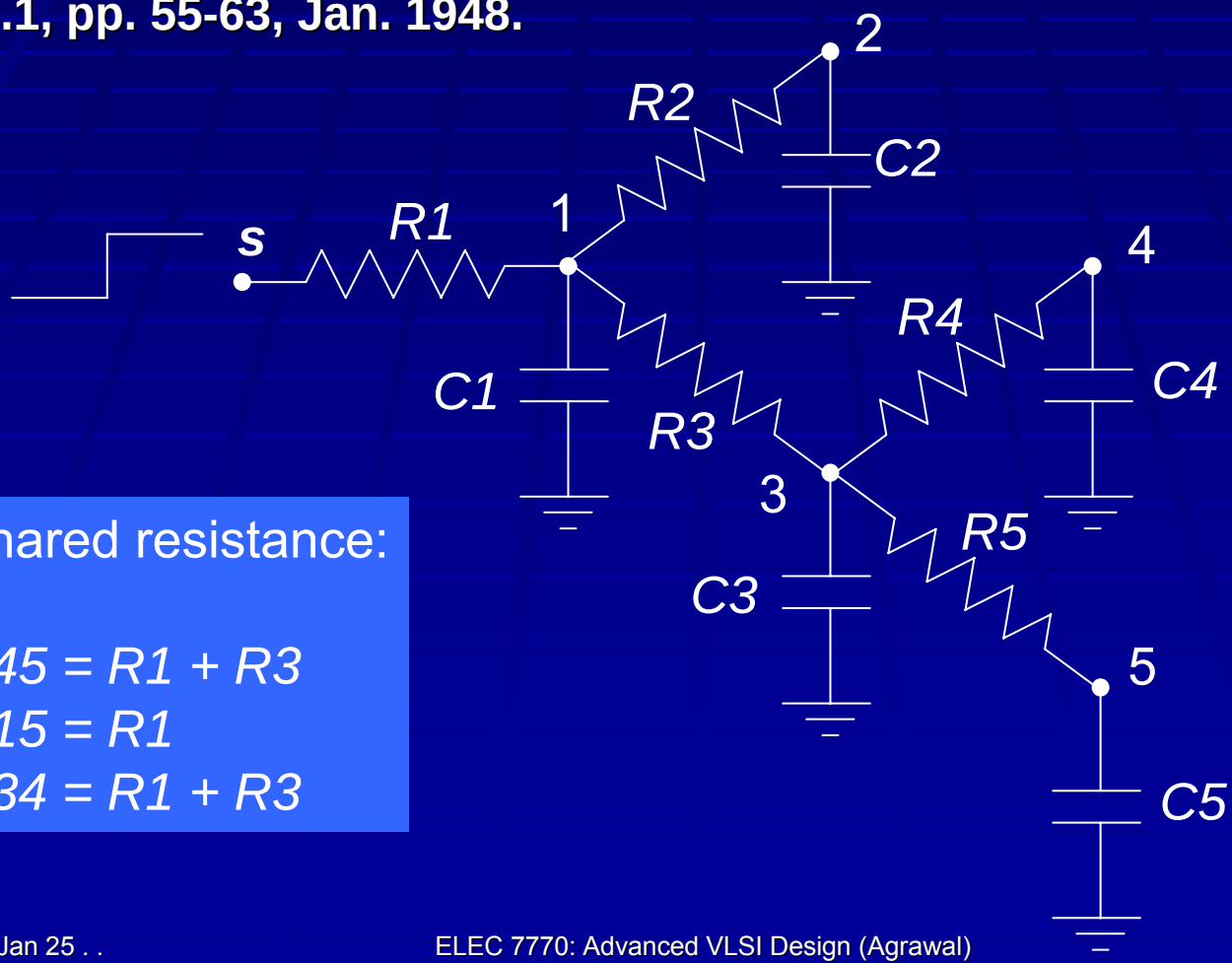
- Circuit partitioned into *channel-connected components* for Spice characterization.
- Reference: R. E. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Trans. Computers*, vol. C-33, no. 2, pp. 160-177, Feb. 1984.

Internal switching nodes not seen by logic simulator



Interconnect Delay: Elmore Delay Model

- **W. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers," *J. Appl. Phys*., vol. 19, no.1, pp. 55-63, Jan. 1948.**

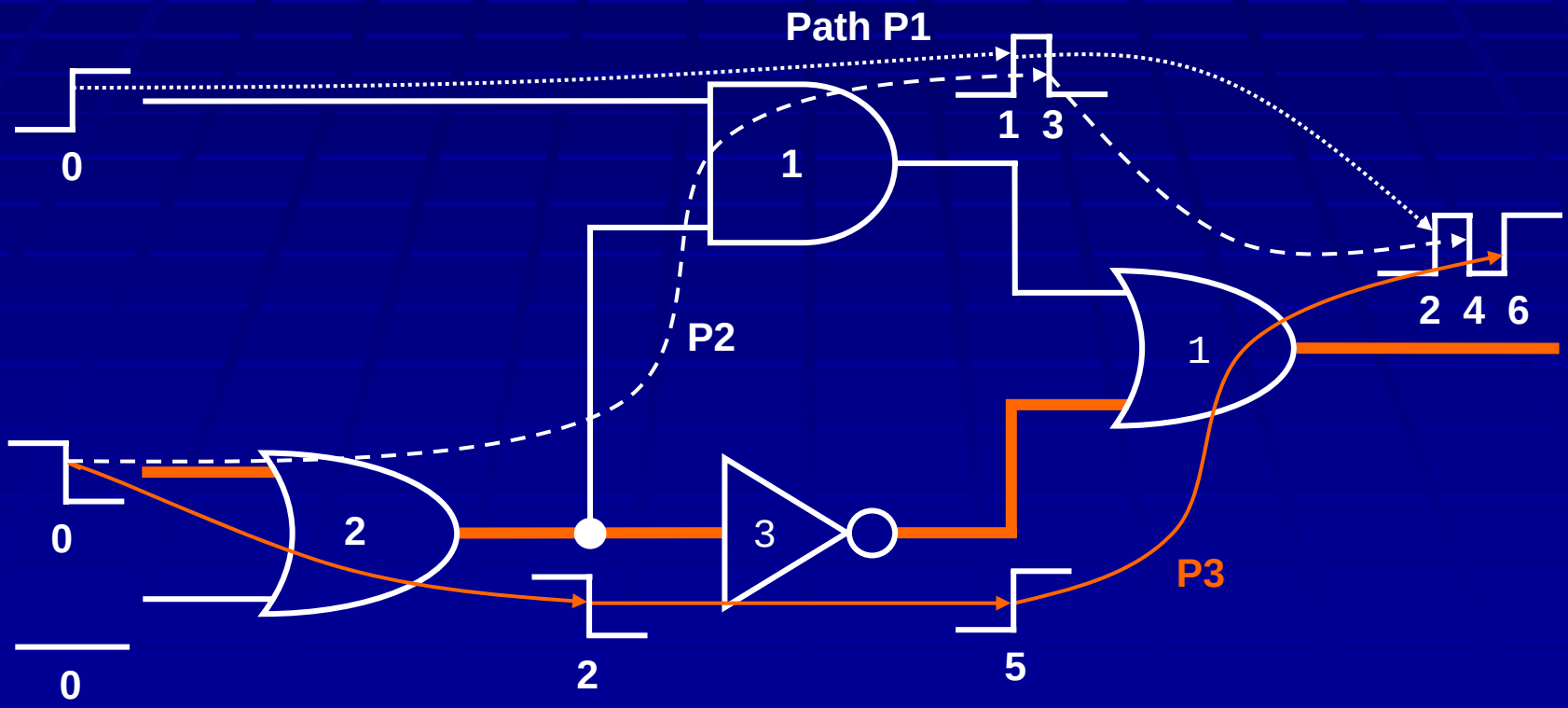


Shared resistance:

$R45 = R1 + R3$
$R15 = R1$
$R34 = R1 + R3$

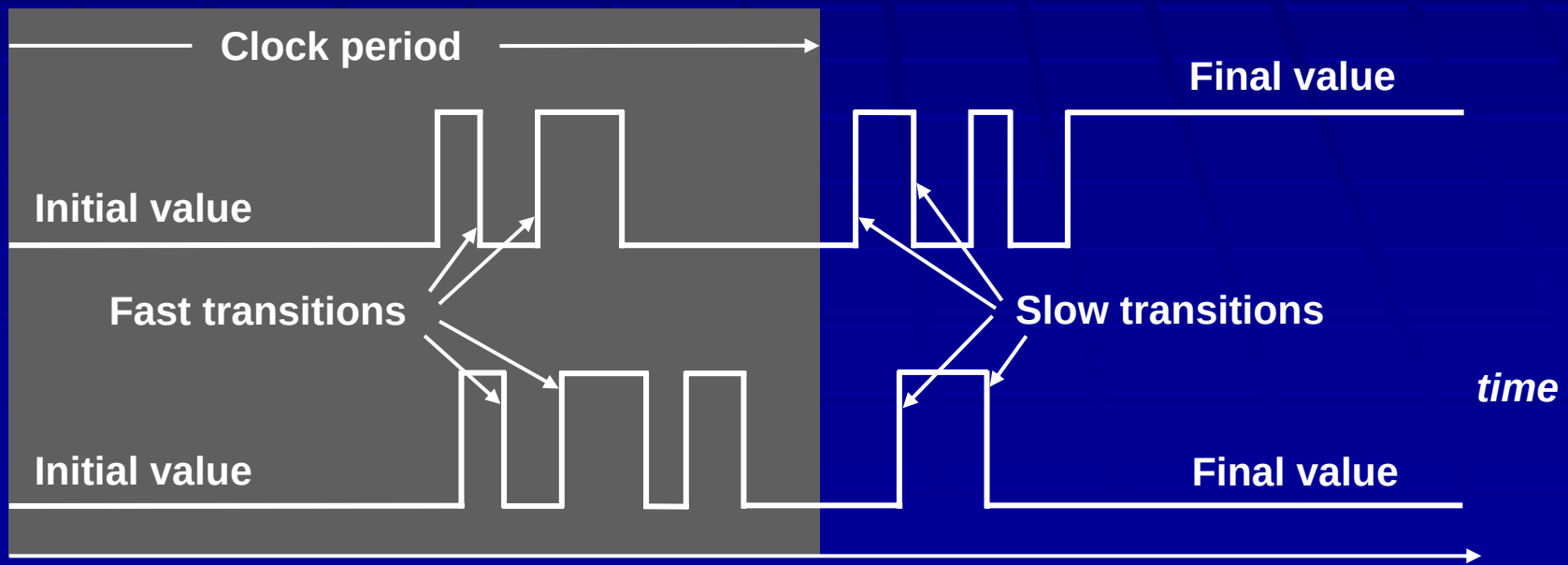
Event Propagation Delays

Single lumped inertial delay modeled for each gate
PI transitions assumed to occur without time skew



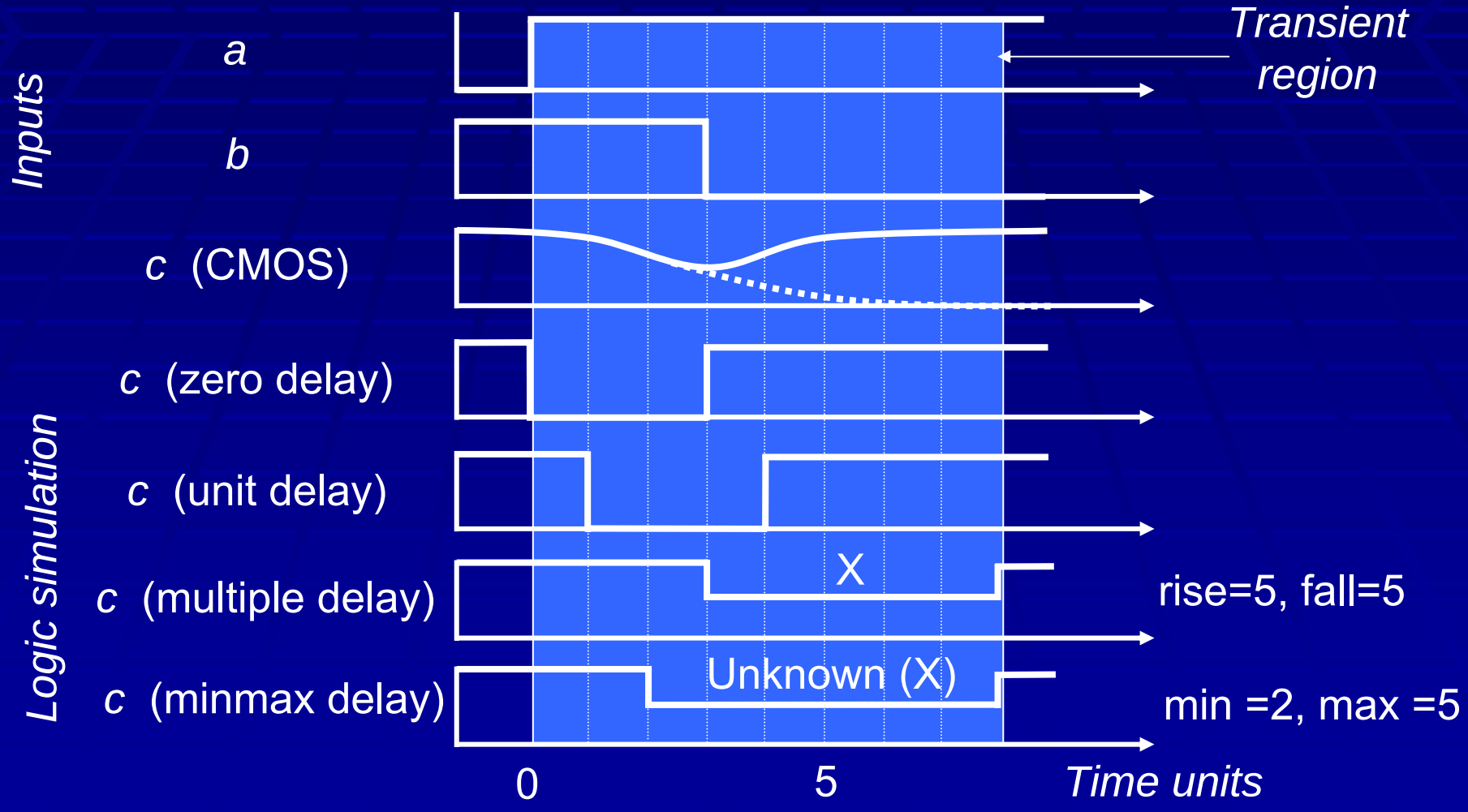
Circuit Outputs

- Each path can potentially produce one signal transition at the output.
- The location of an output transition in time is determined by the delay of the path.

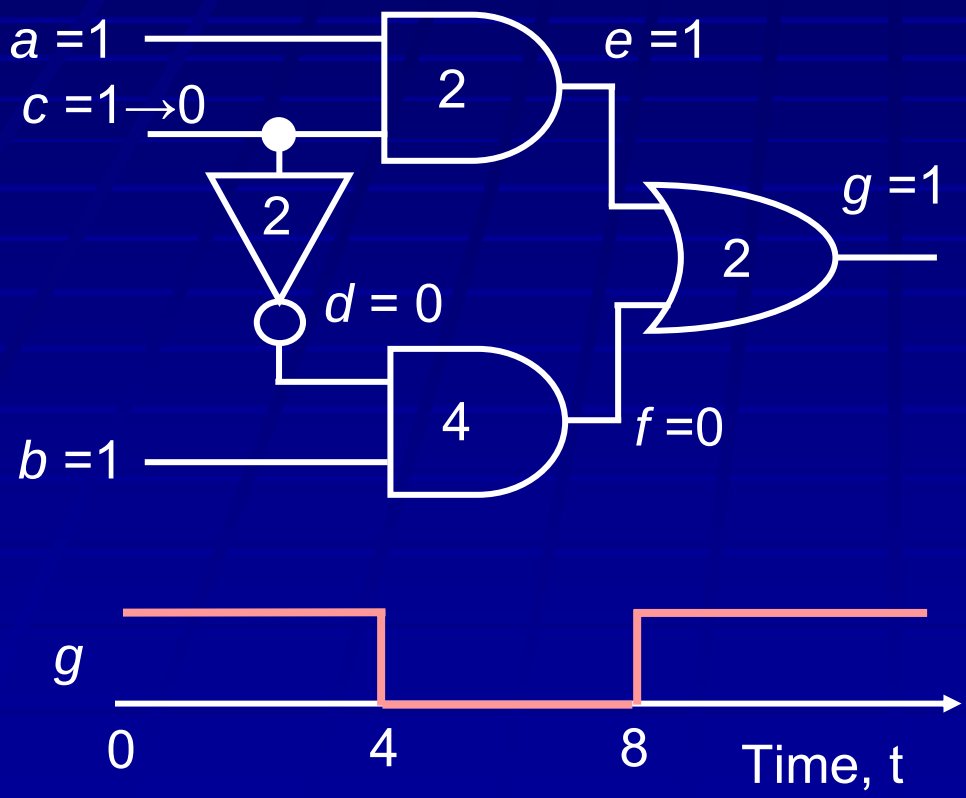


Delay and Discrete-Event Simulation

(NAND gate)

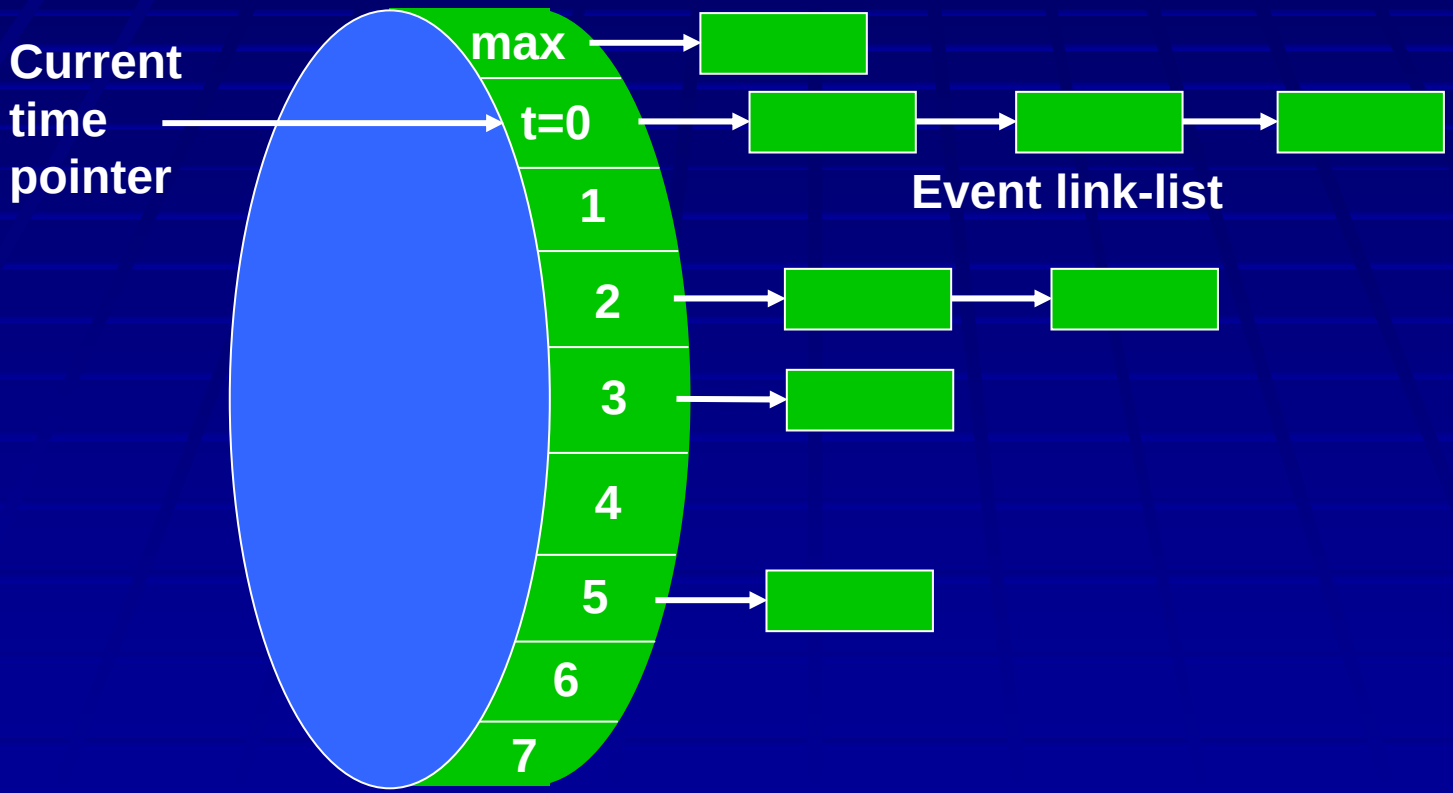


Event-Driven Simulation (Example)



Time stack	Scheduled events	Activity list
t = 0	$c = 0$	d, e
1		
2	$d = 1, e = 0$	f, g
3		
4	$g = 0$	
5		
6	$f = 1$	g
7		
8	$g = 1$	

Time Wheel (Circular Stack)

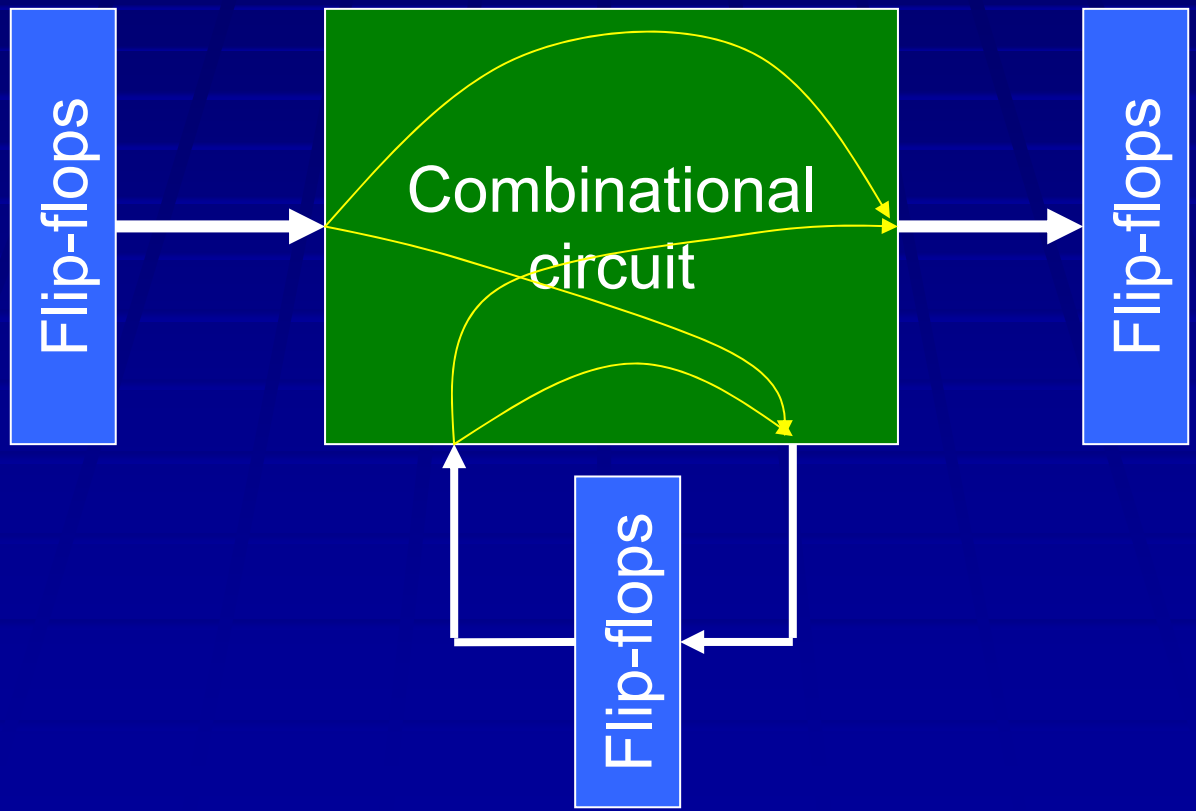


Timing Design and Delay Test

- Timing simulation:
  - Critical paths are identified by static (vector-less) timing analysis tools like *Primetime* (Synopsys).
  - Timing or circuit-level simulation using designer-generated functional vectors verifies the design.
- Layout optimization: Critical path data are used in placement and routing. Delay parameter extraction, timing simulation and layout are repeated for iterative improvement.
- Testing: Some form of at-speed test is necessary. Critical paths and all gate transition delays are tested.

Static Timing Analysis (STA)

- Finds maximum and minimum delays between all clocked flip-flops.



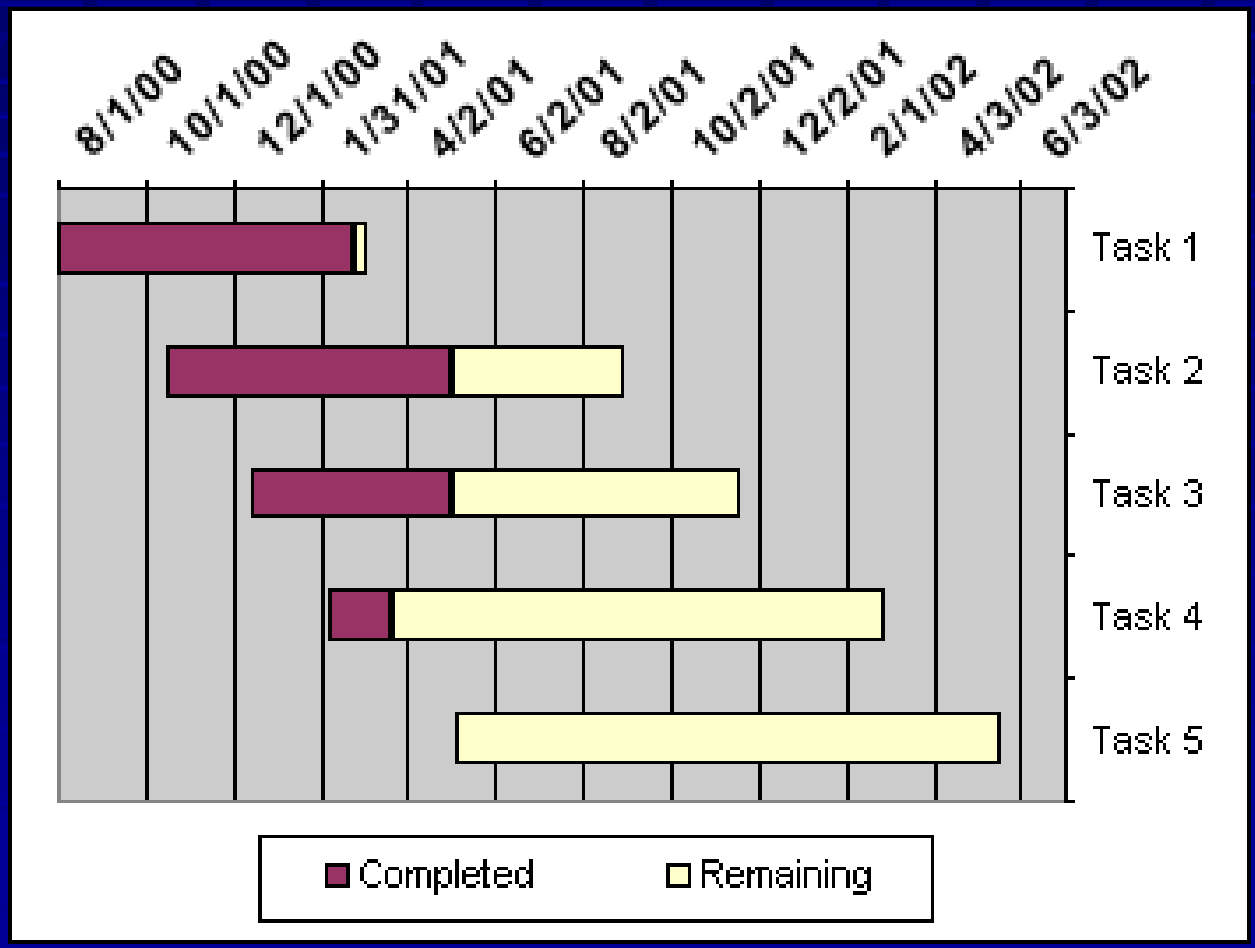
Early References

- T. I. Kirkpatrick and N. R. Clark, "PERT as an Aid to Logic Design," *IBM J. Res. Dev*., vol. 10, no. 2, pp. 135-141, March 1966.
- R. B. Hitchcock, Sr., "Timing Verification and the Timing Analysis Program," *Proc. 19th Design Automation Conf*., 1982, pp. 594-604.
- V. D. Agrawal, "Synchronous Path Analysis in MOS Circuit Simulator," *Proc. 19th Design Automation Conf*., 1982, pp. 629-635.

Basic Ideas

- Adopted from project management
  - Frederick W. Taylor (1856-1915)
  - Henry Gantt (1861-1919)
- PERT – Program Evaluation and Review Technique
- CPM – Critical Path Method

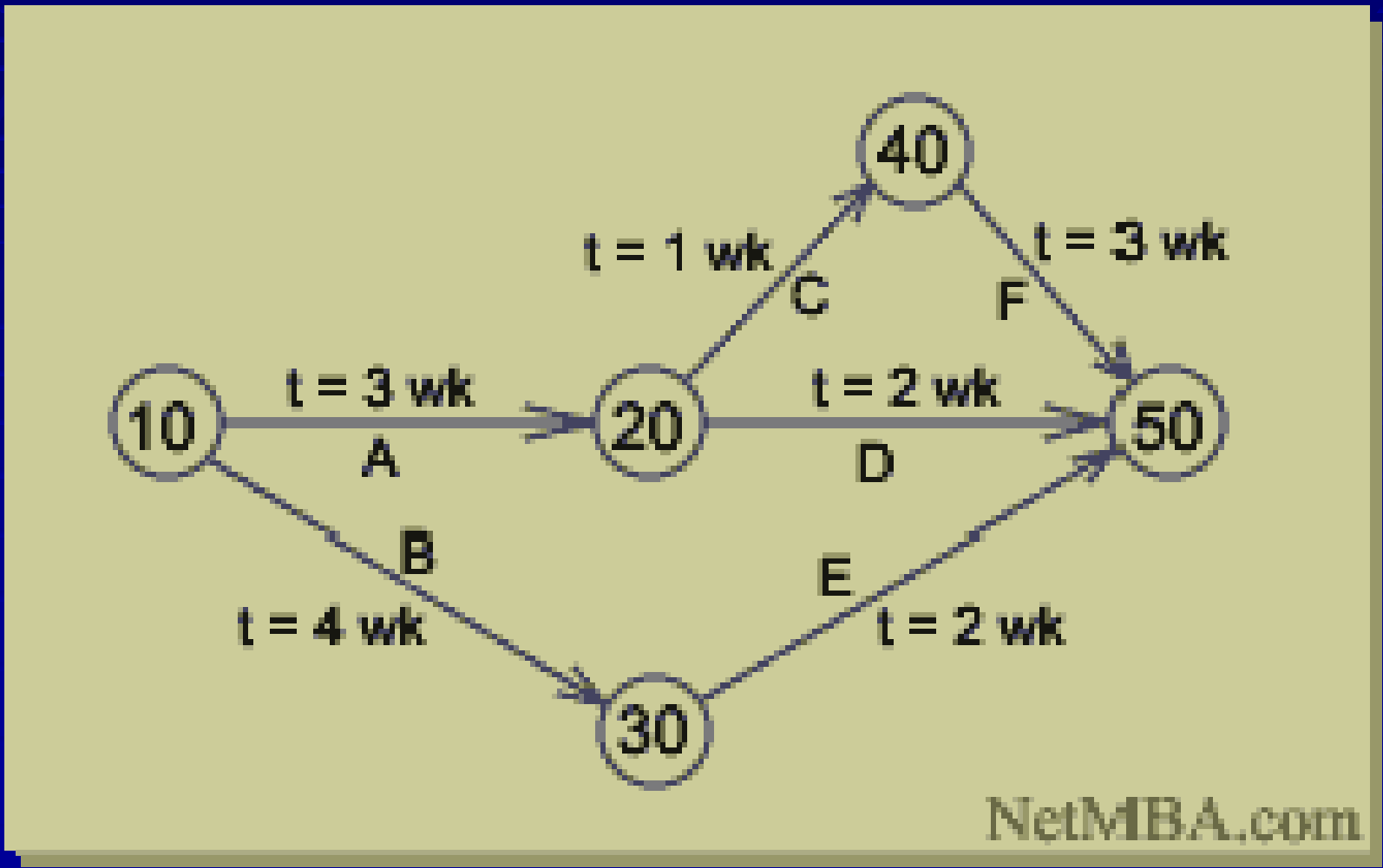
A Gantt Chart in Microsoft Excel



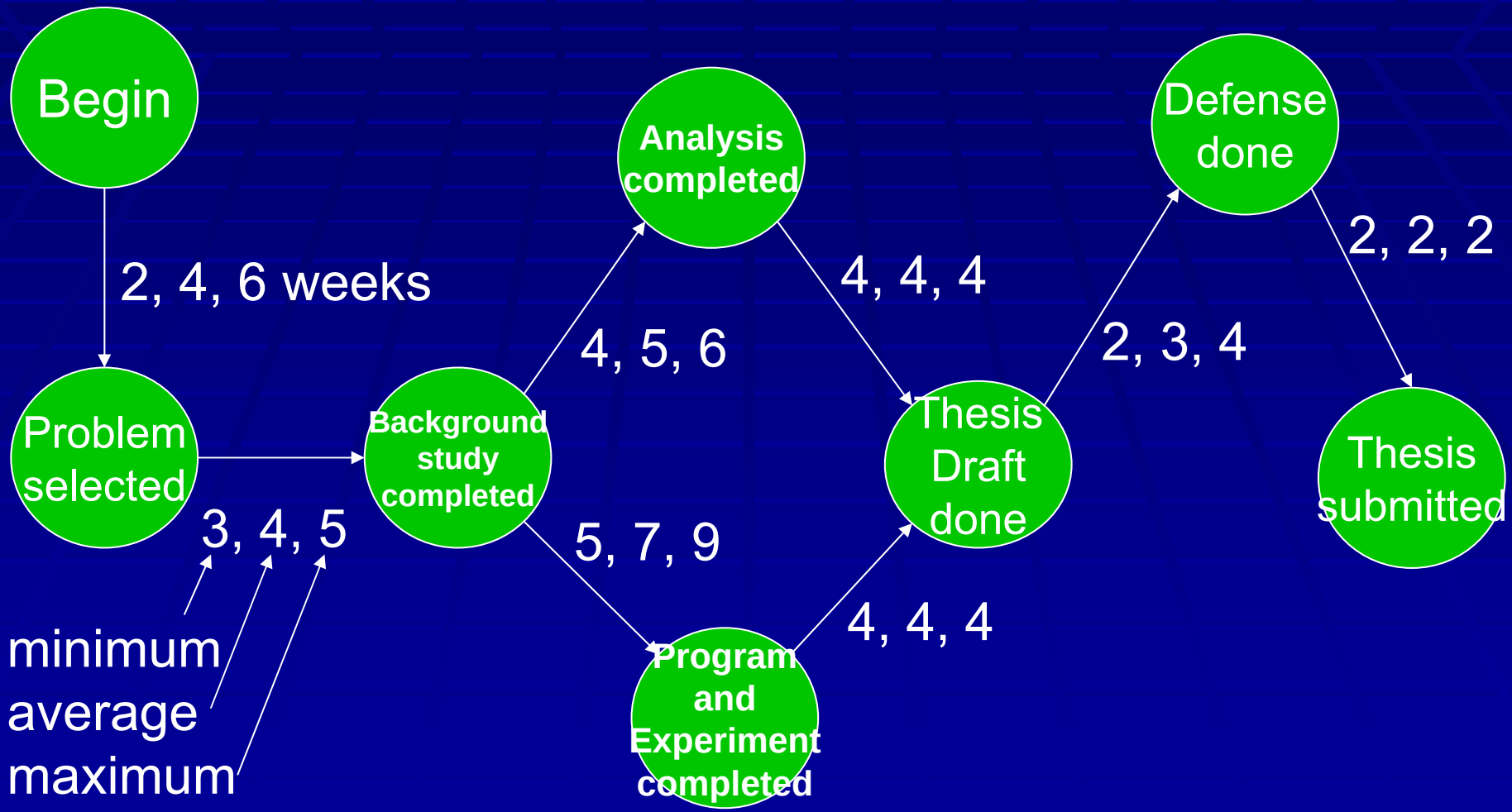
Using a Gantt Chart

- Track progress of subtasks and project.
- Assess resource needs as a function of time.

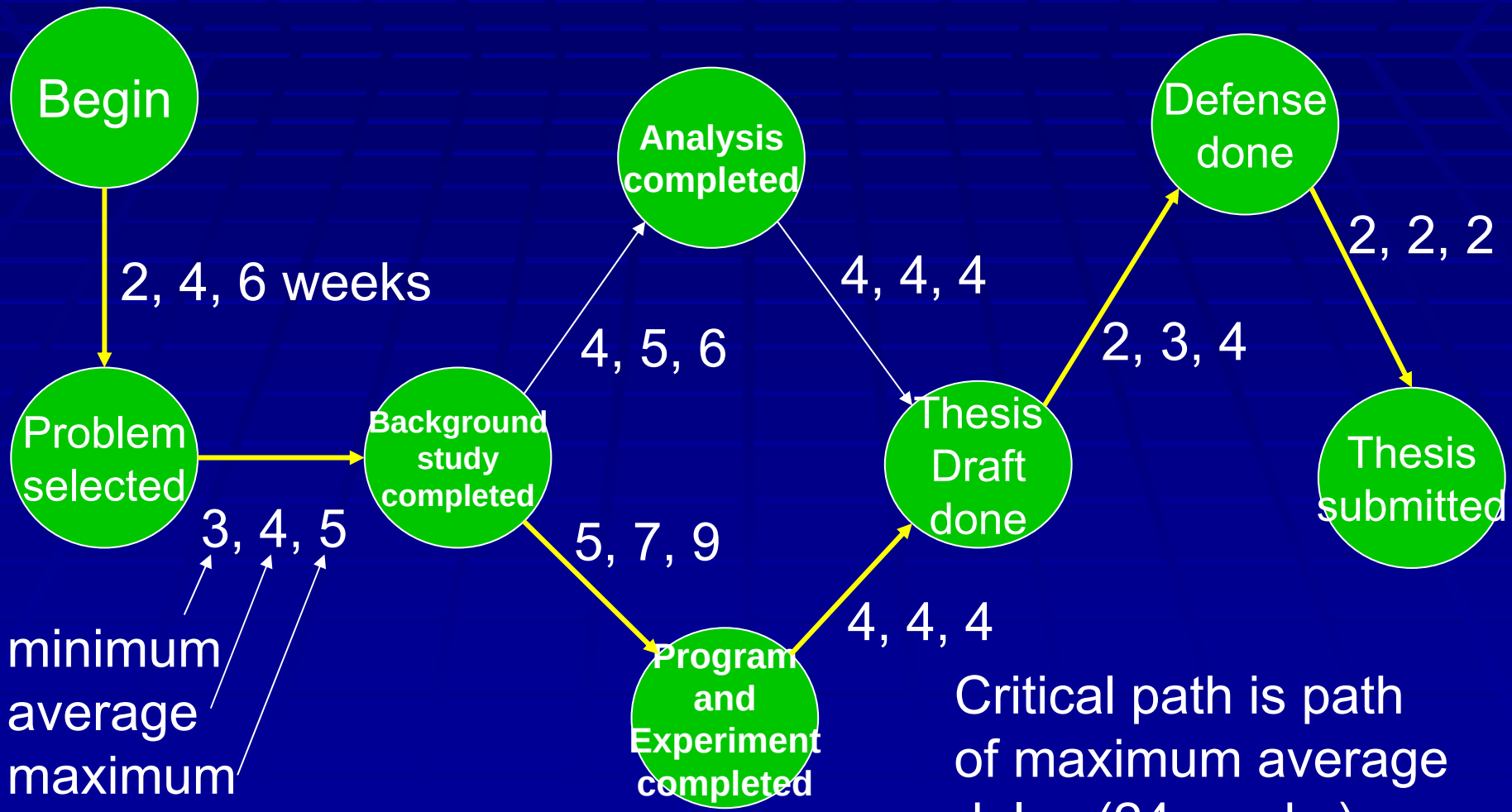
Pert Chart



Example: Thesis Research



Critical Path



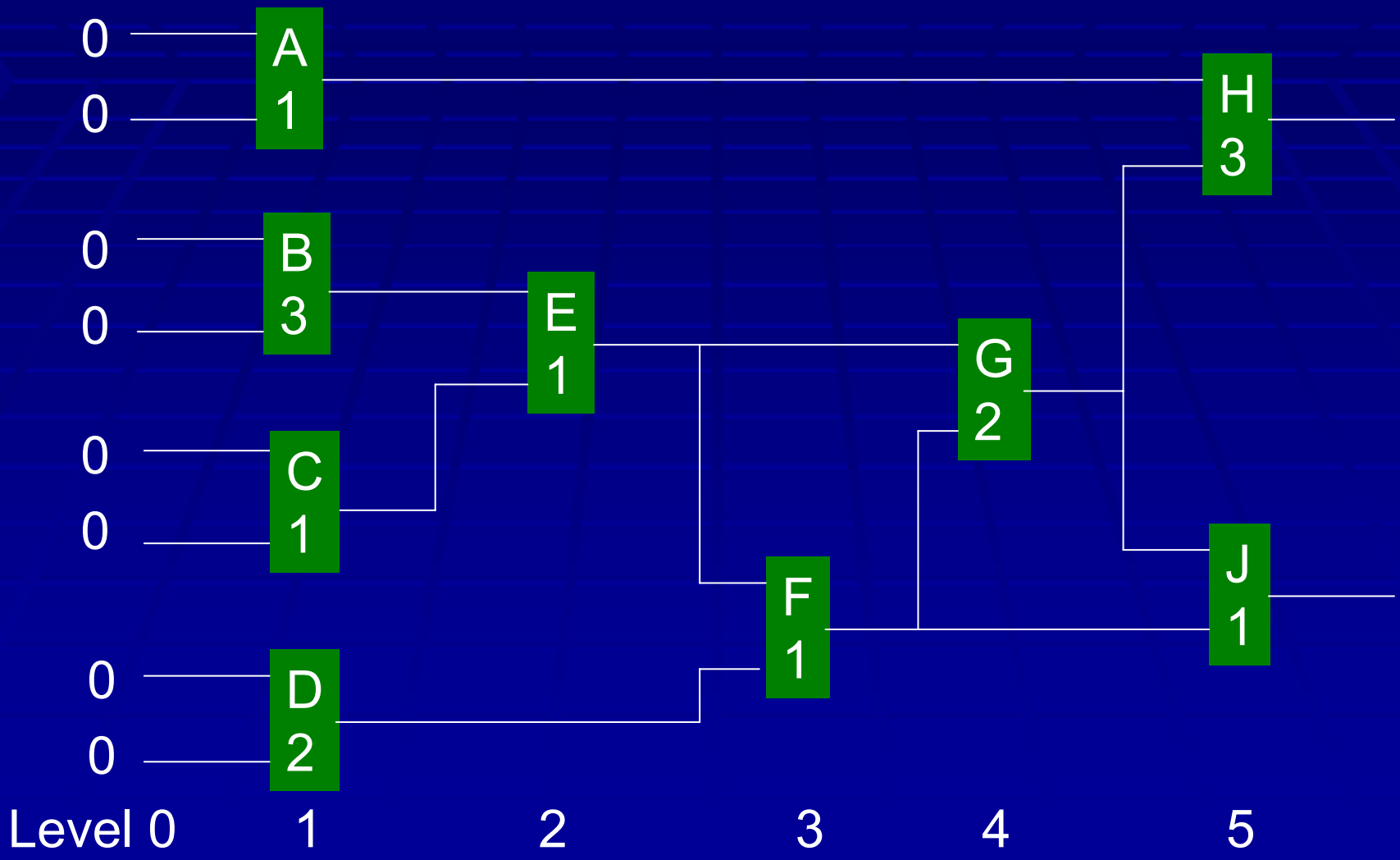
Critical path is path of maximum average delay (24 weeks).

A Basic Timing Analysis Algorithm

- Combinational logic.
- Circuit represented as an acyclic directed graph (DAG).
- Gates characterized by delays.

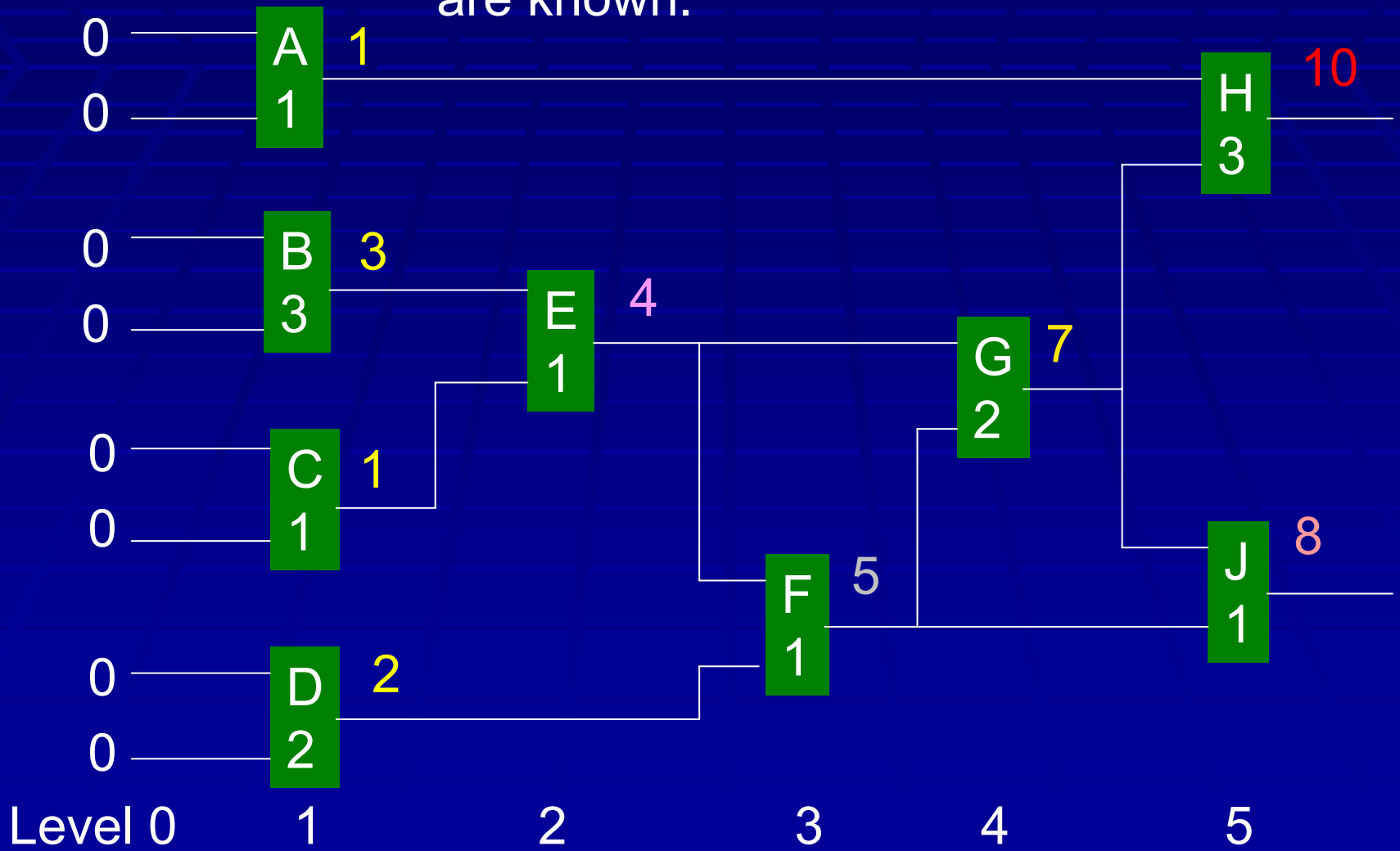
Example

Levelize graph. Initialize arrival times at primary inputs.



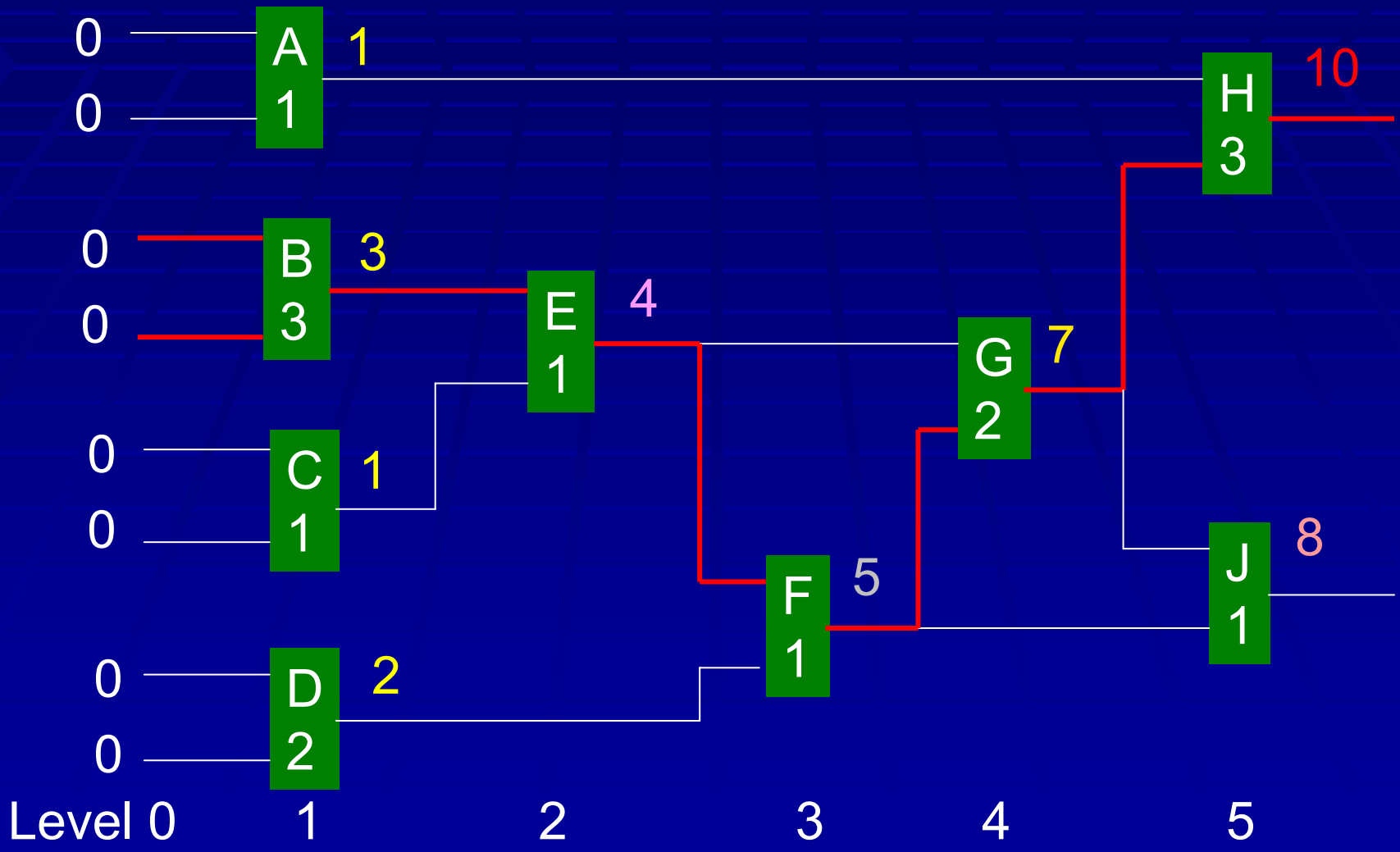
Example (Cont.)

Determine output arrival time when all input arrival times are known.

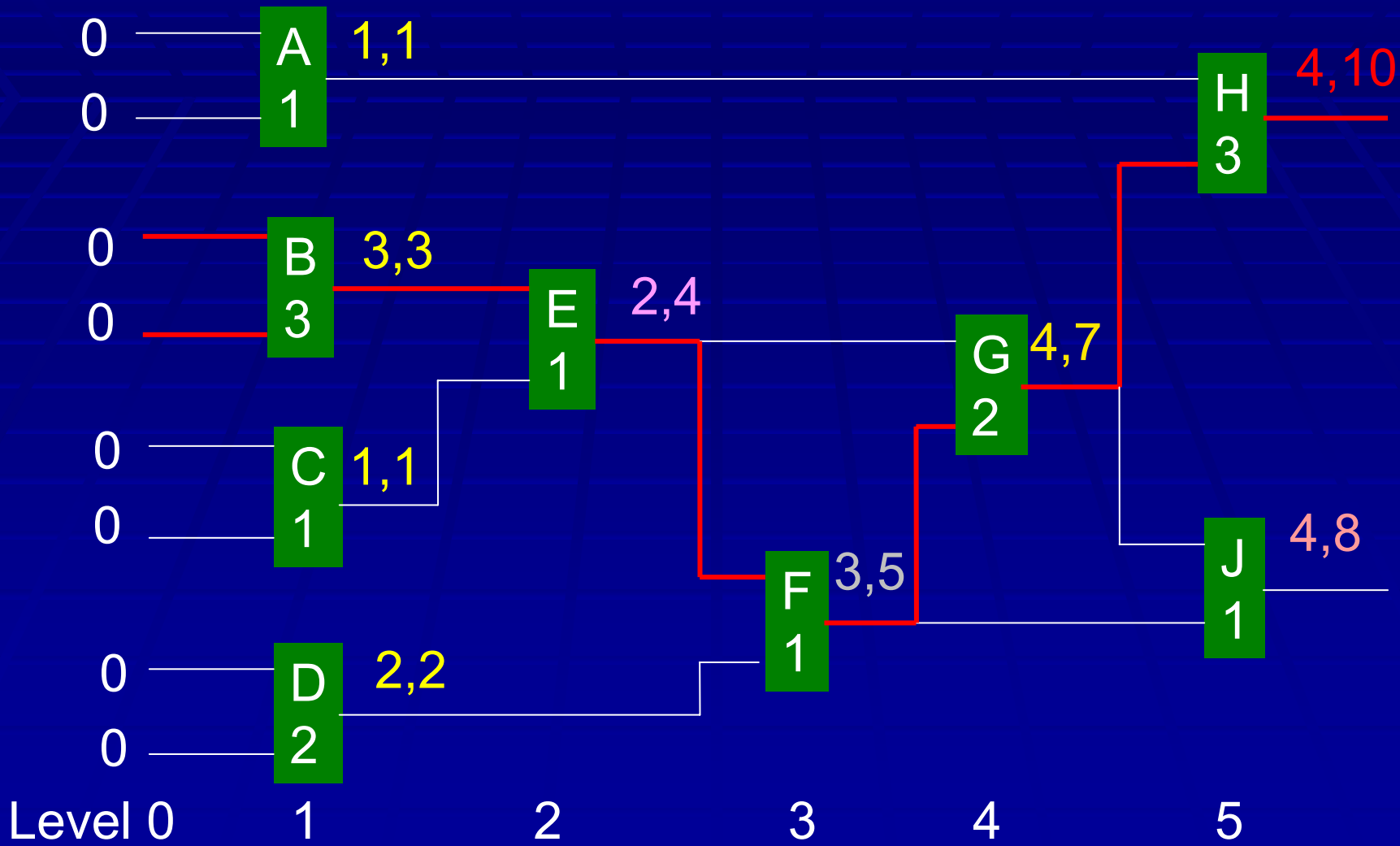


Example (Cont.)

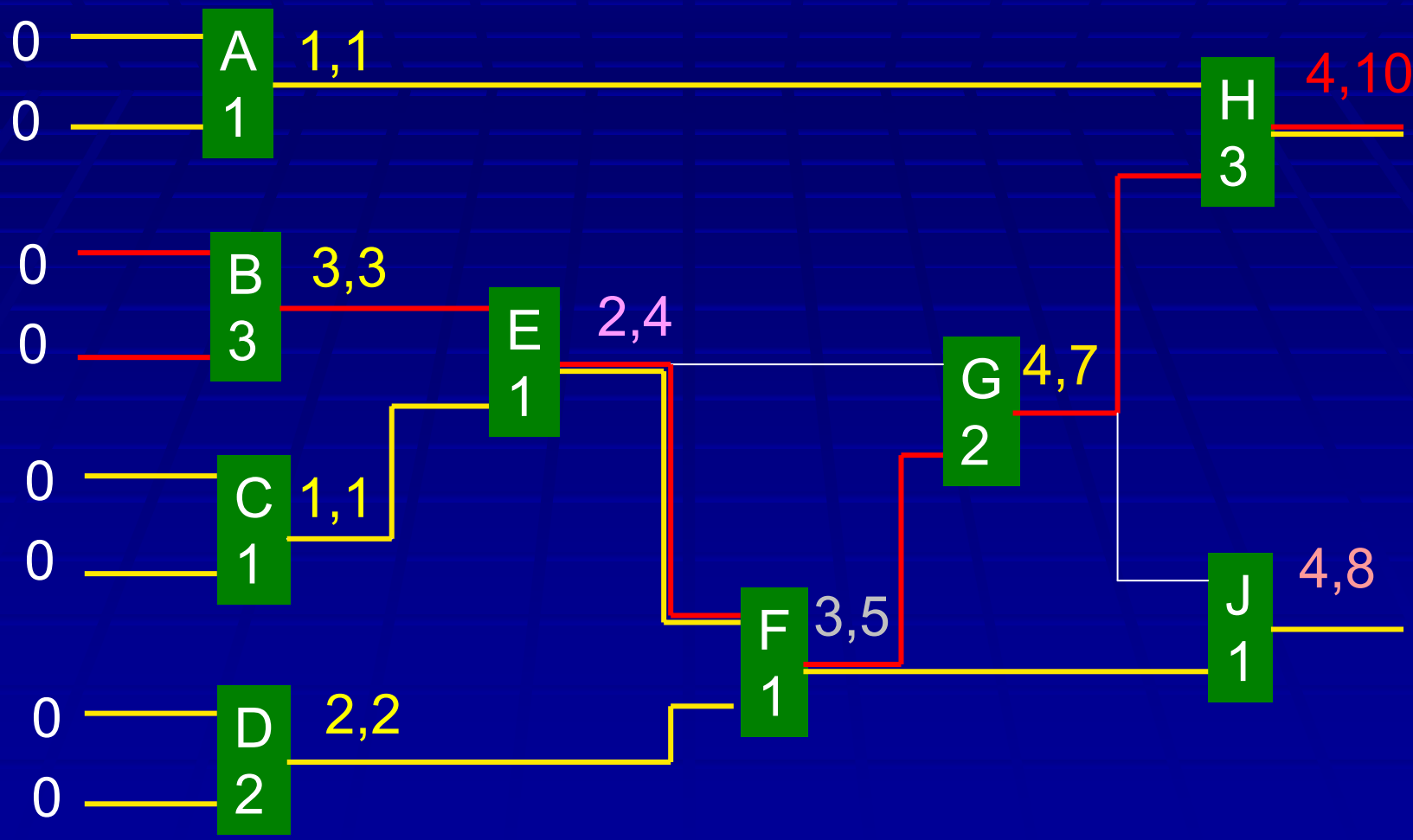
Trace critical path from the output with longest arrival time.



Finding Earliest and Longest Times



Shortest and Longest Paths



Characteristics of STA

- Linear time analysis, Complexity is O(n), n is number of gates and interconnects.
- Variations:
  - Find k longest paths:
    - S. Kundu, “An Incremental Algorithm for Identification of Longest (Shortest) **P**aths,” *Integration, the VLSI Journal,* vol. 17, no. 1, pp. 25-35, August 1994.
  - Find worst-case delays from an input to all outputs.
  - Linear programming methods.

Algorithms for Directed Acyclic Graphs (DAG)

- Graph size: n = |V| + |E|, for |V| vertices and |E| edges.
- Levelization: O(n) (linear-time) algorithm finds the maximum (or minimum) depth.
- Path counting: $O(n^2)$ algorithm. Number of paths can be exponential in n.
- Finding all paths: Exponential-time algorithm.
- Shortest (or longest) path between two nodes:
  - Dijkstra's algorithm: $O(n^2)$
  - Bellman-Ford algorithm: $O(n^3)$

References

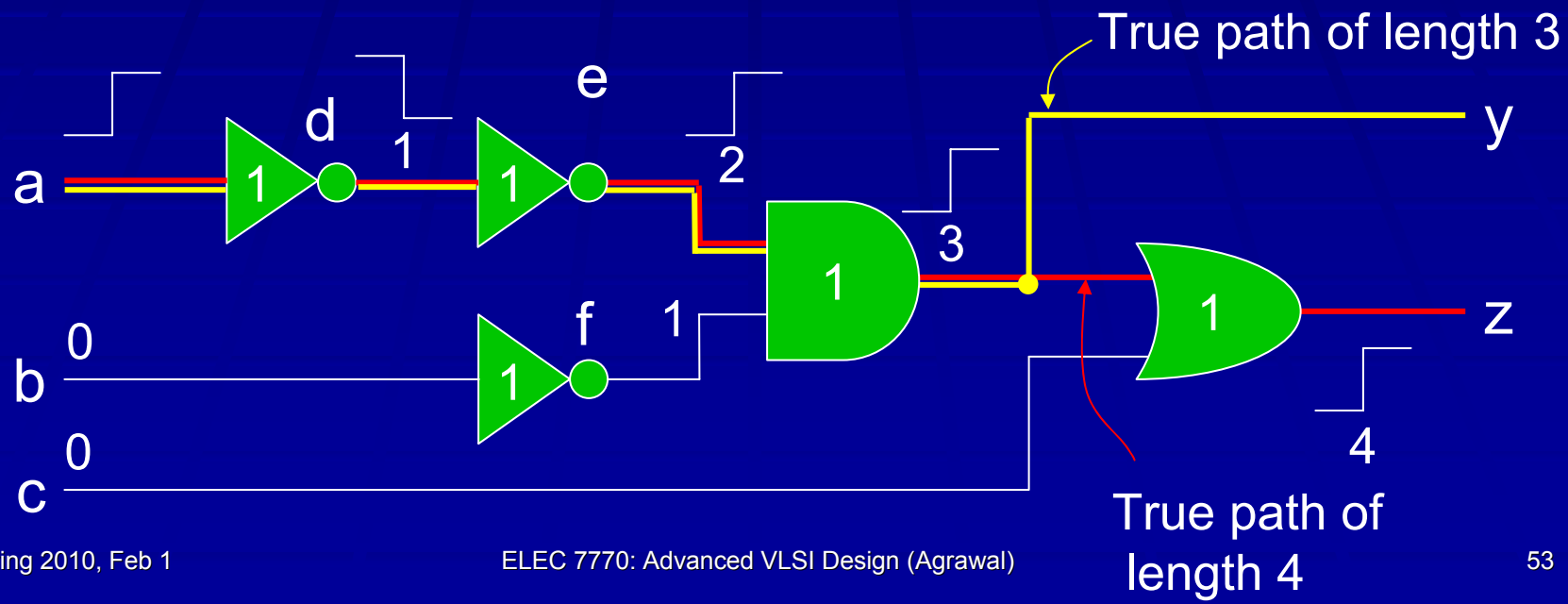
- Delay modeling, simulation and testing:
  - M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, Springer, 2000.
- Analysis and Design:
  - G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.
  - N. Maheshwari and S. S. Sapatnekar, *Timing Analysis and Optimization of Sequential Circuits*, Springer, 1999.
- PrimeTime (Static timing analysis tool):
  - H. Bhatnagar, *Advanced ASIC Chip Synthesis, Second Edition*, Springer, 2002

Proof of Correctness

- Static timing analysis proves the timing correctness. That is, the circuit is guaranteed to work at the clock rate determined by the critical path.
- But the circuit may also work correctly at faster speeds.
- Because the critical path identified by STA (static timing analysis) may be a “false path”.
- *STA overestimates the delay of the circuit.*

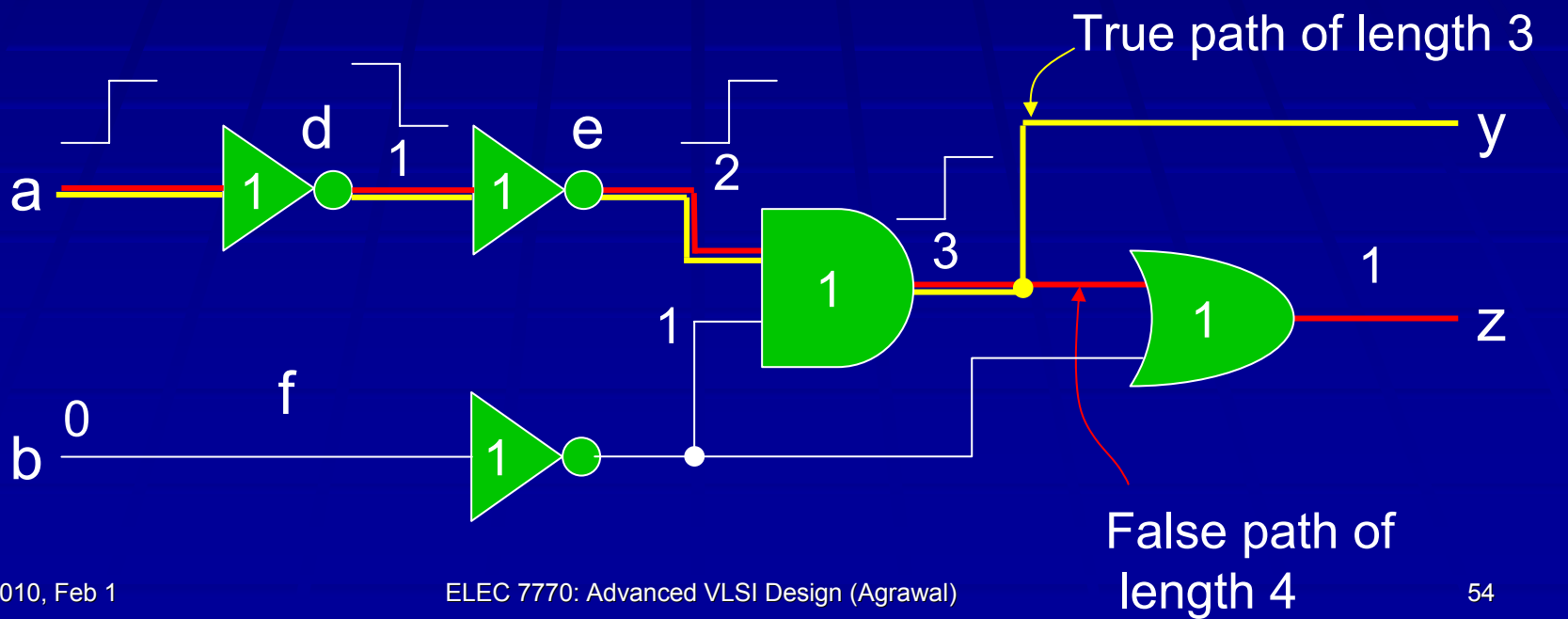
False and True Paths

- A false path cannot propagate an event and hence cannot affect the timing of the circuit. False paths are dynamically unsensitizable.
- Dynamically sensitizable path (true path): All off-path inputs must settle down to their non-controlling values when the event propagates through the path.



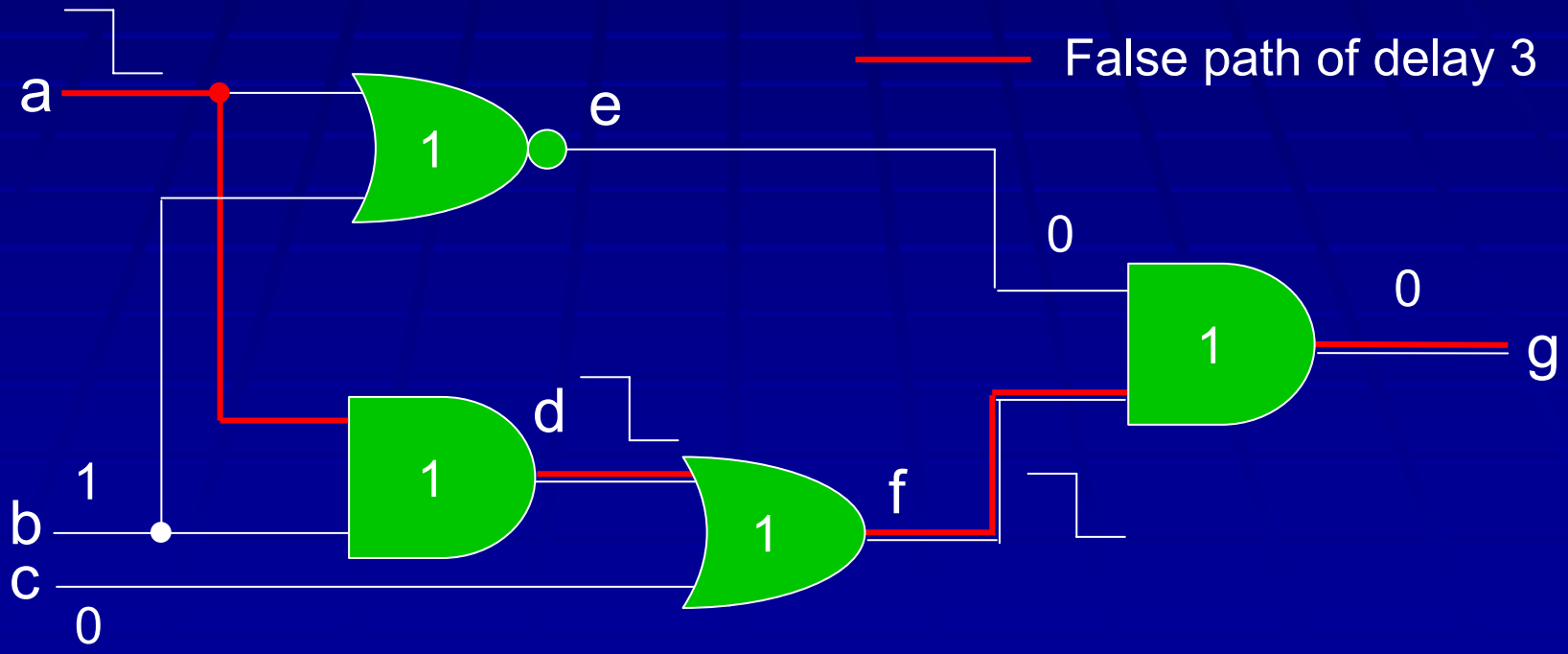
Static Sensitization of Path

- Static sensitization of path: All off-path inputs can be simultaneously set to their non-controlling values.
- Longest path in the following example is statically unsensitizable. Such paths are often referred to, though not correctly (why?), as false paths.



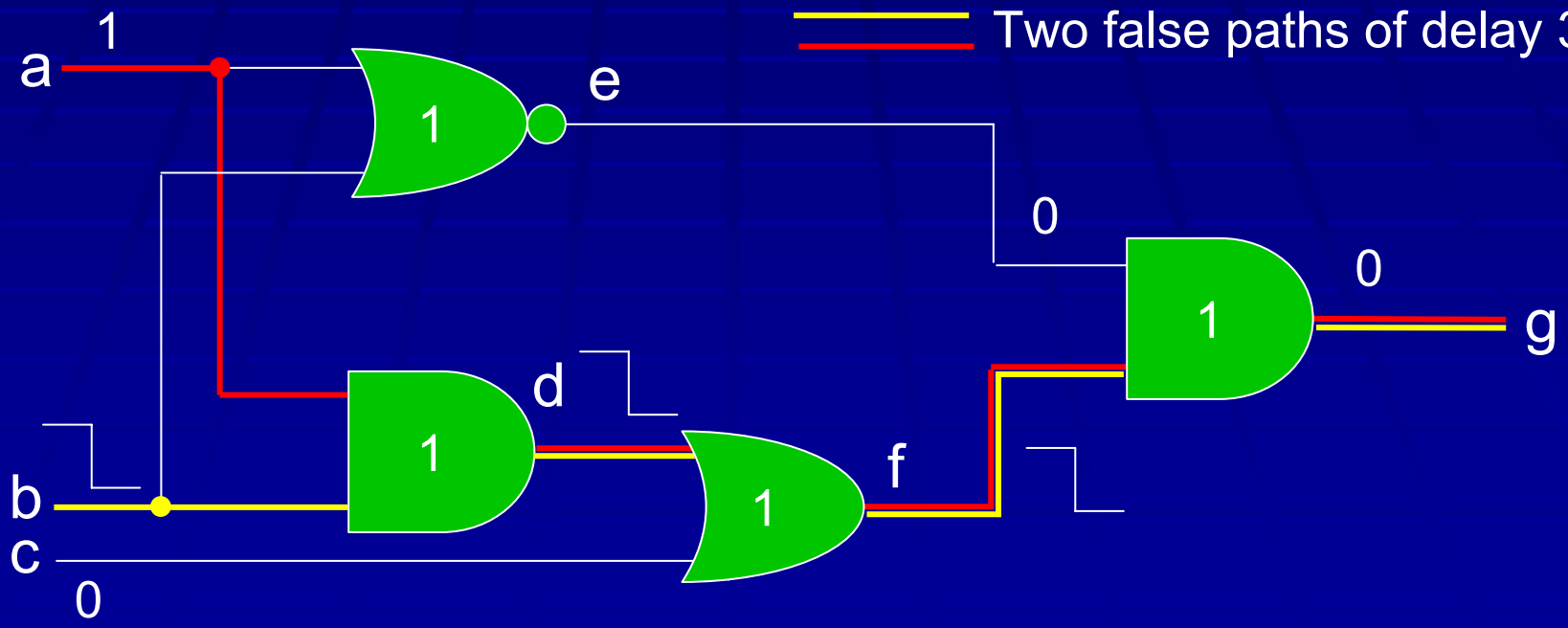
An Example

- Statically unsensitizable (false) path.
- P. C. McGeer and R. K. Brayton, *Integrating Functional and Temporal Domains in Logic Design*, Springer, 1991.



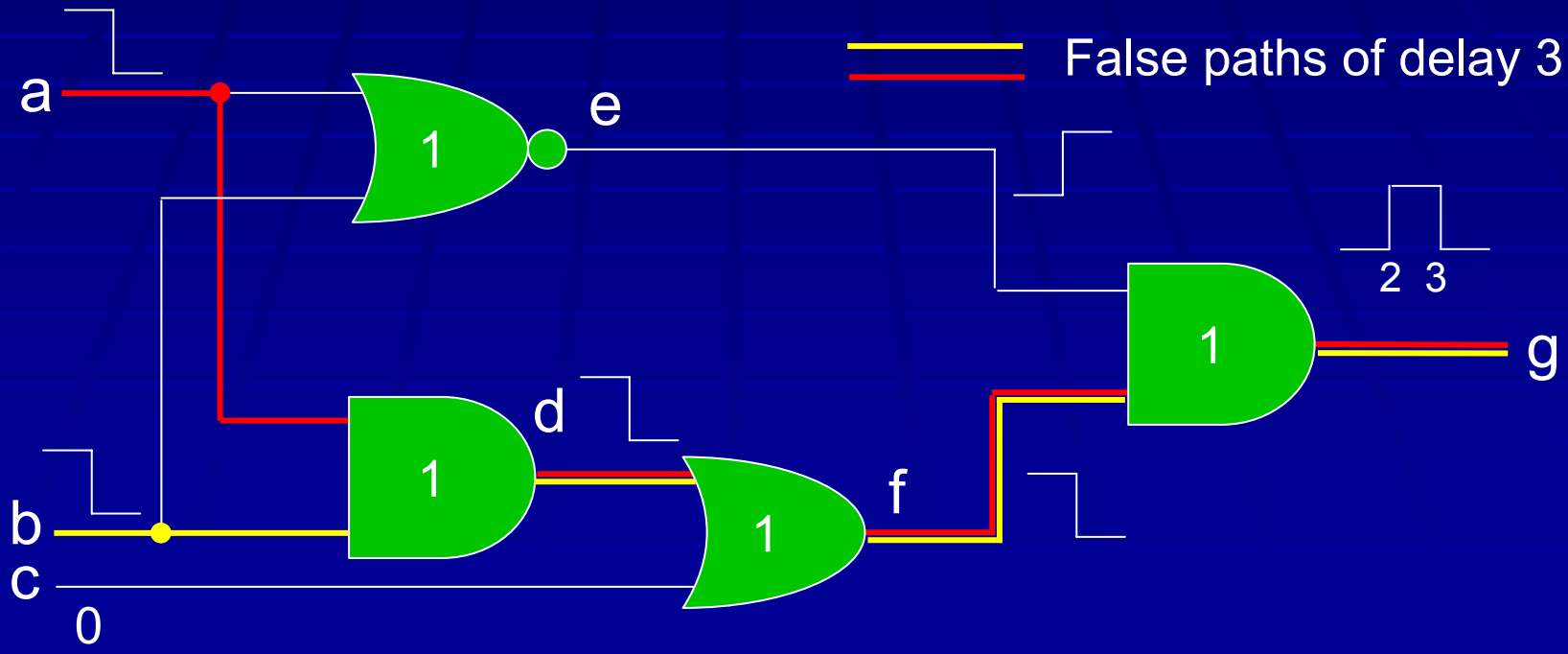
Example (Cont.)

- Another statically unsensitizable false path.
- P. C. McGeer and R. K. Brayton, *Integrating Functional and Temporal Domains in Logic Design*, Springer, 1991.

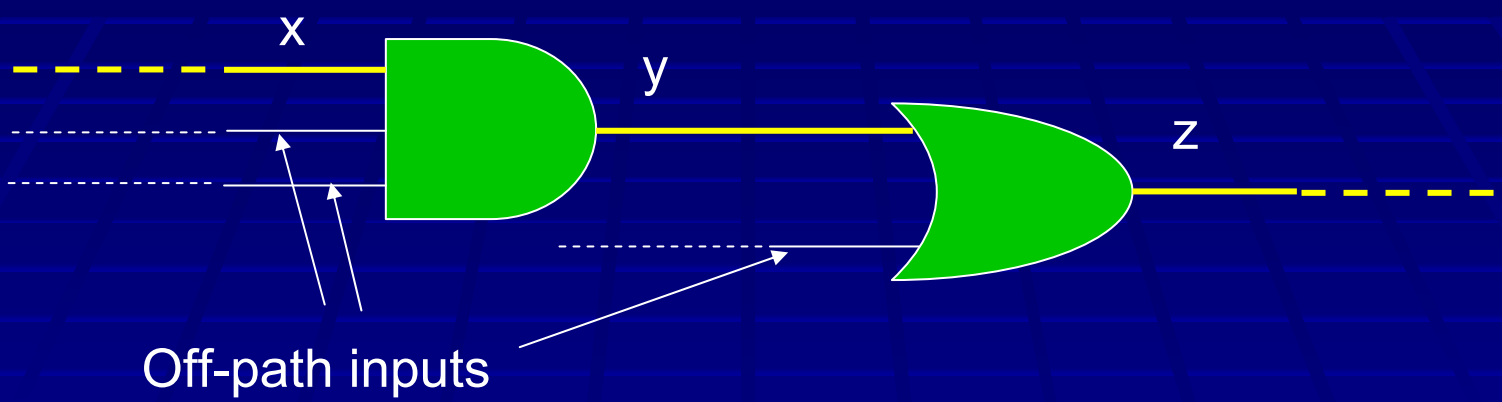


Example (Cont.)

- Two paths are dynamically sensitizable and will affect the timing if both are together faulty.
- P. C. McGeer and R. K. Brayton, *Integrating Functional and Temporal Domains in Logic Design*, Springer, 1991.



Static Sensitization Condition

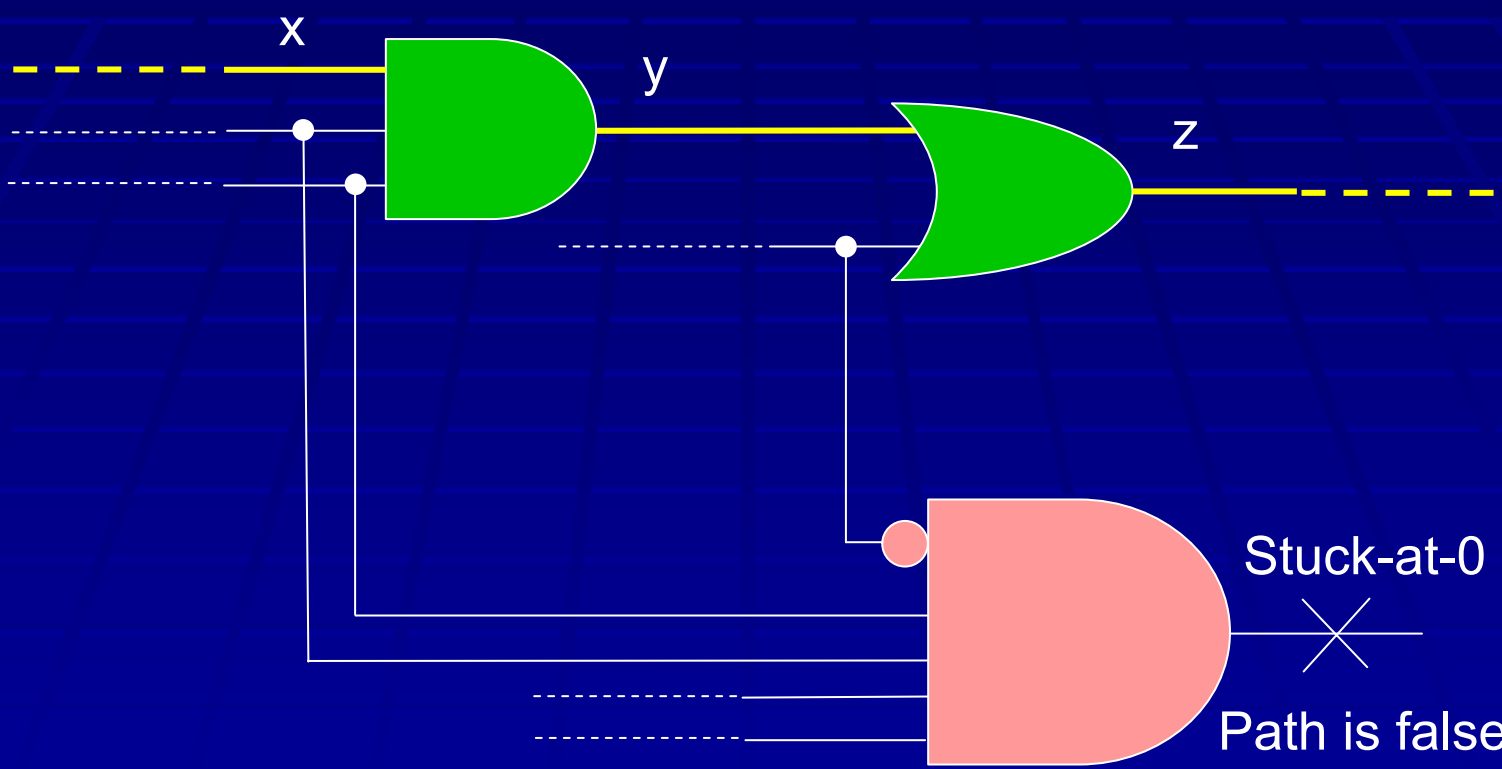


There must exist an input vector (PI) that satisfies the following conditions:

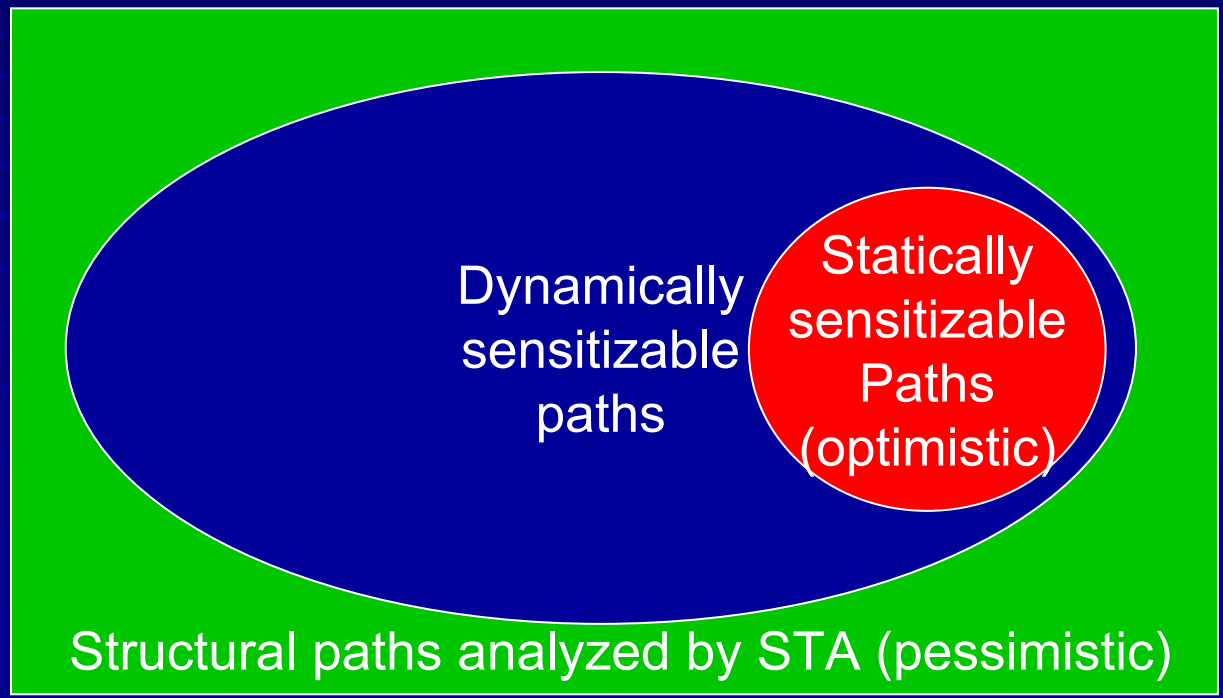
$\partial y/\partial x = 1, \quad \partial z/\partial y = 1, \ldots$

Where $\partial y/\partial x = y(x=1, PI) \oplus y(x=0, PI)$ is Boolean difference

An ATPG Method



Optimism and Pessimism



Theorem 1

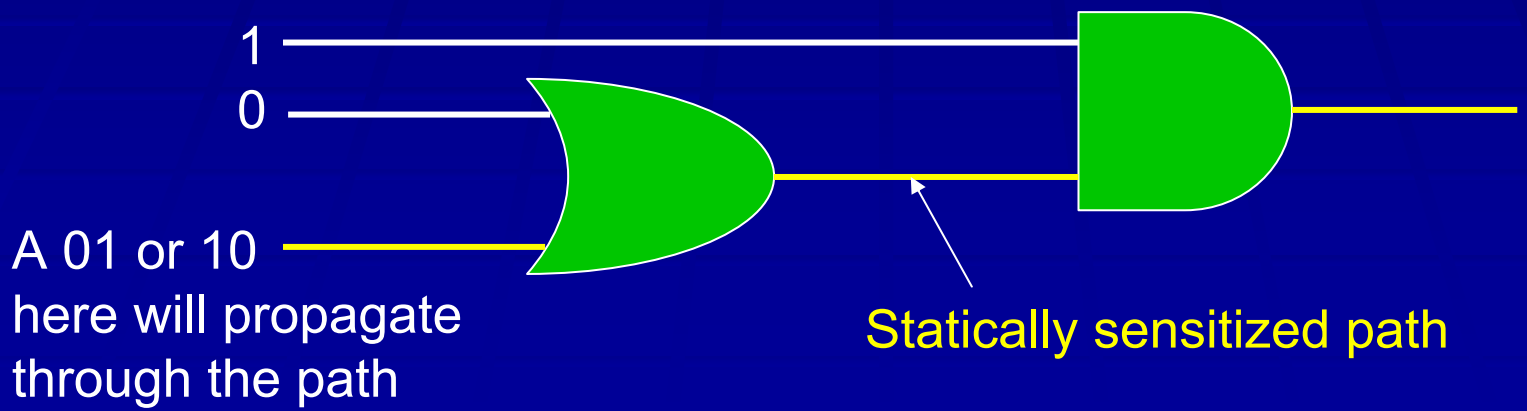
- Every statically sensitizable path is dynamically sensitizable.
- Proof: Since a vector exists to sensitize the path, if that vector does not specify the path input, then toggling the primary input at the origin of the path will propagate an event through the path.
- P. C. McGeer and R. K. Brayton, *Integrating Functional and Temporal Domains in Logic Design*, Springer, 1991, p. 35.

Theorem 2

- The longest path in a circuit is dynamically sensitizable iff it is statically sensitizable.
- Proof: Because this is the longest path, all off-path inputs will settle to their sensitizing values at the inputs of any gate before the on-path event propagates through that gate.
- P. C. McGeer and R. K. Brayton, *Integrating Functional and Temporal Domains in Logic Design*, Springer, 1991, p. 37.

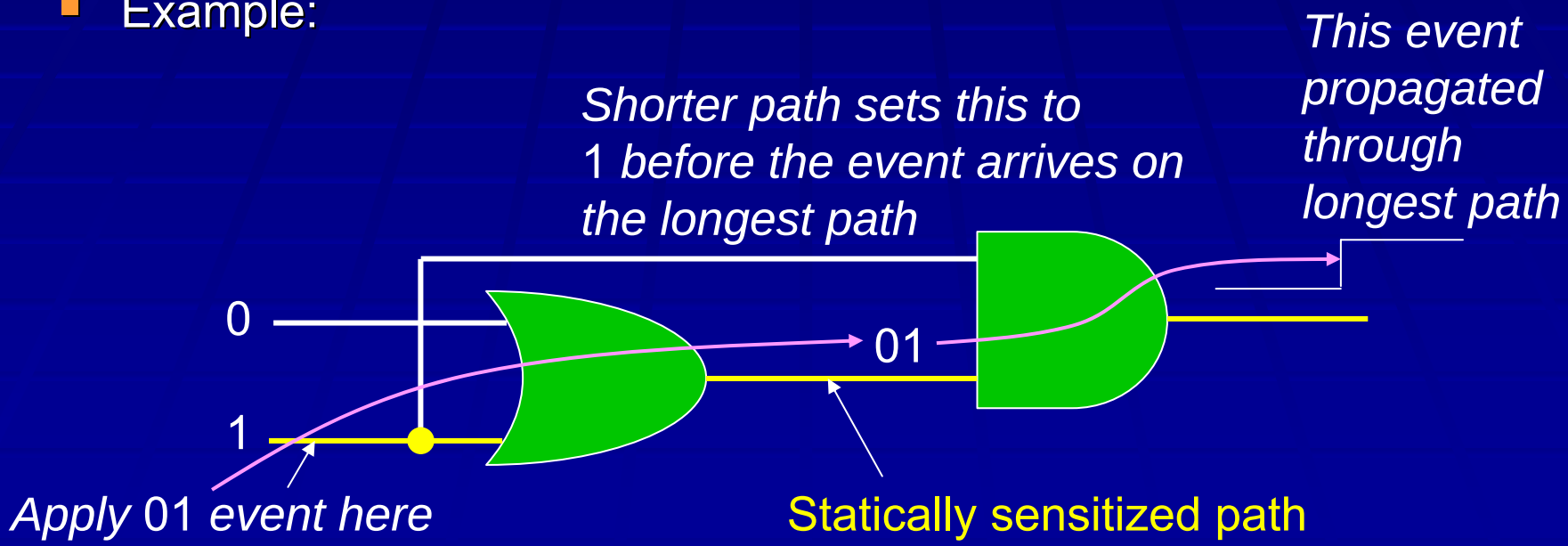
Proof of Theorem 2

- Case 1: Static sensitization does not specify the value at the path origin.
- Toggling the path origin will propagate an event through the path causing dynamic sensitization.
- Example:



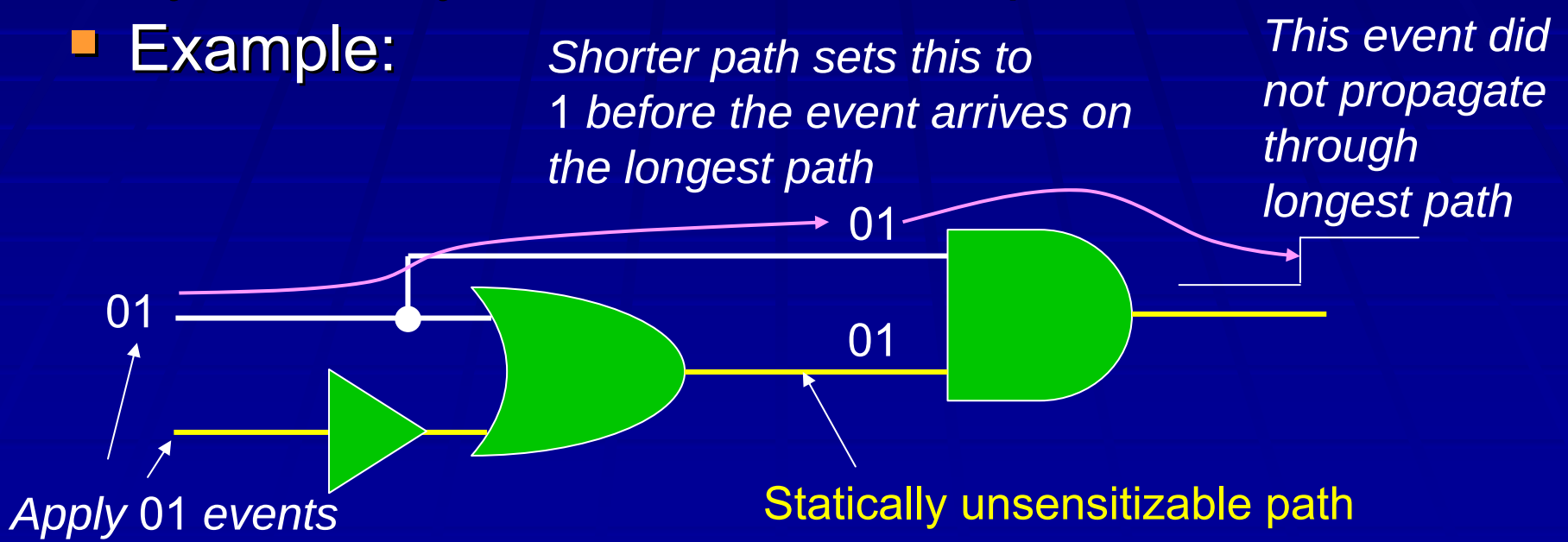
Proof of Theorem 2 (Cont.)

- Case 2: Static sensitization specifies the value at the path origin.
- Toggling the path origin will propagate an event through the path causing dynamic sensitization because the event on the longest path will see all gates sensitized through shorter paths.
- Example:

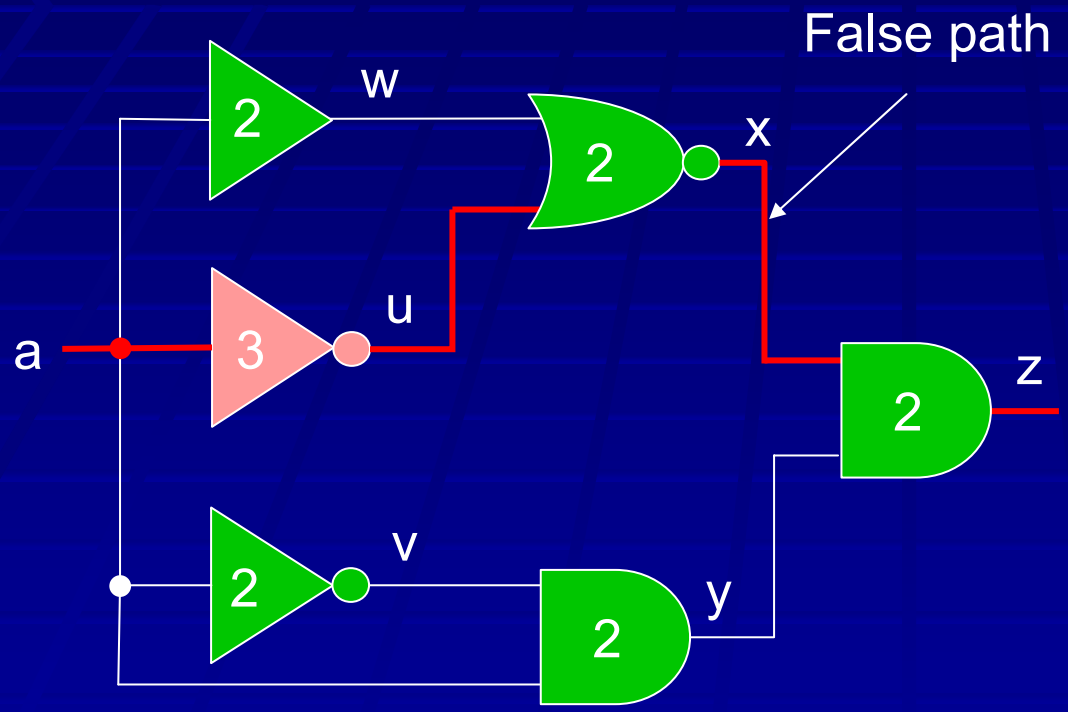


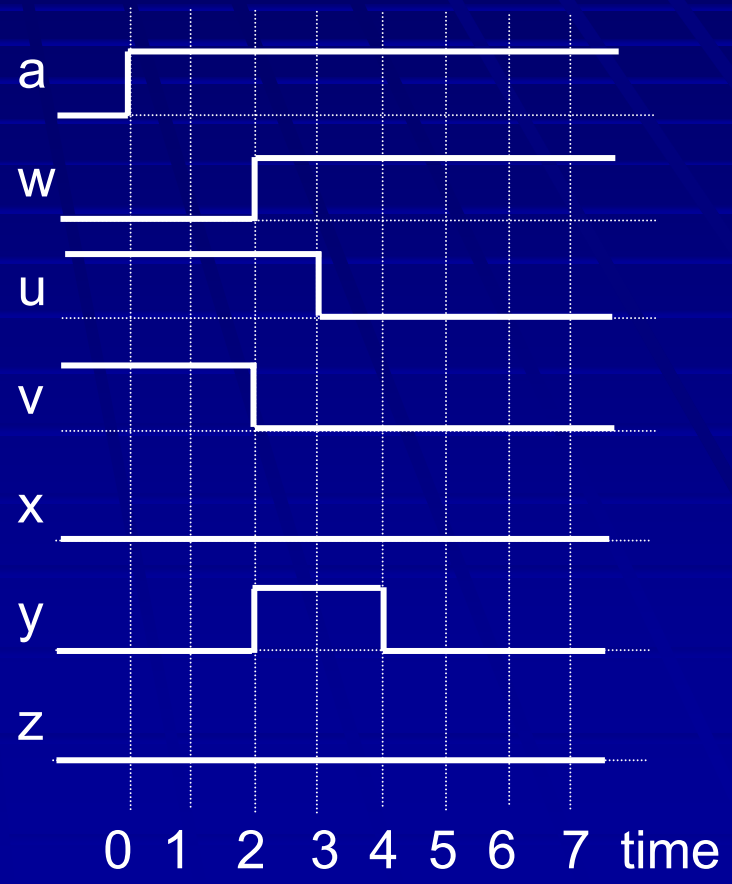
Proof of Theorem 2 (Cont.)

- Case 3: Longest path is statically unsensitizable.
- Toggling the path origin will not propagate any event through the path. Toggling other input only dynamically sensitizes shorter path.
- Example:

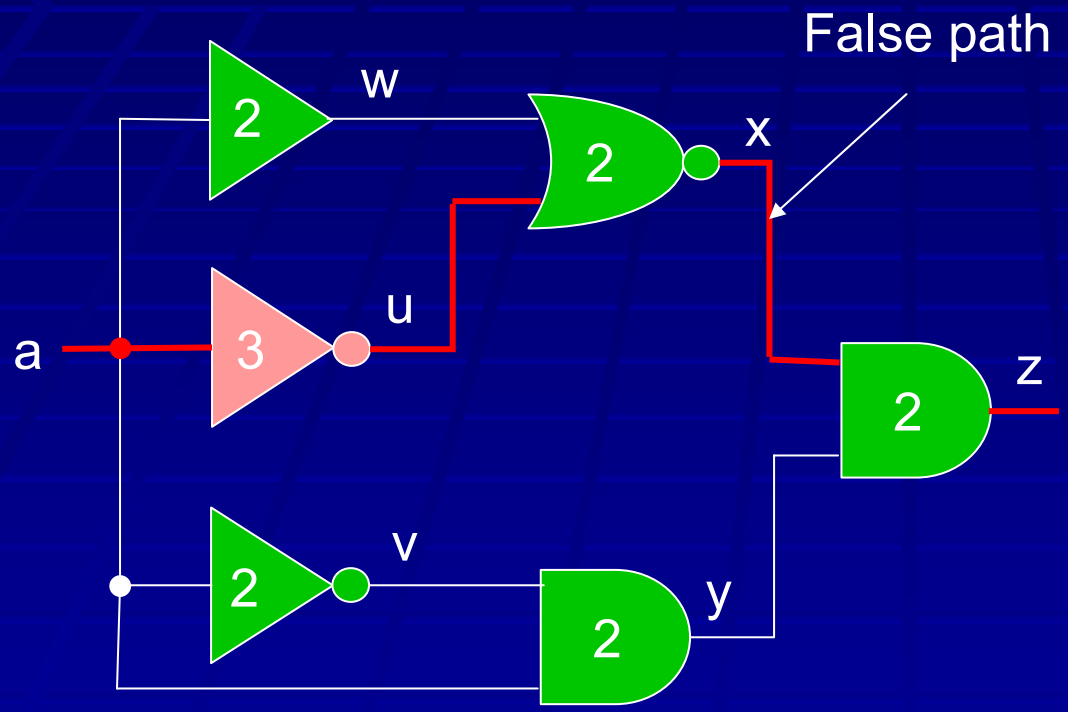


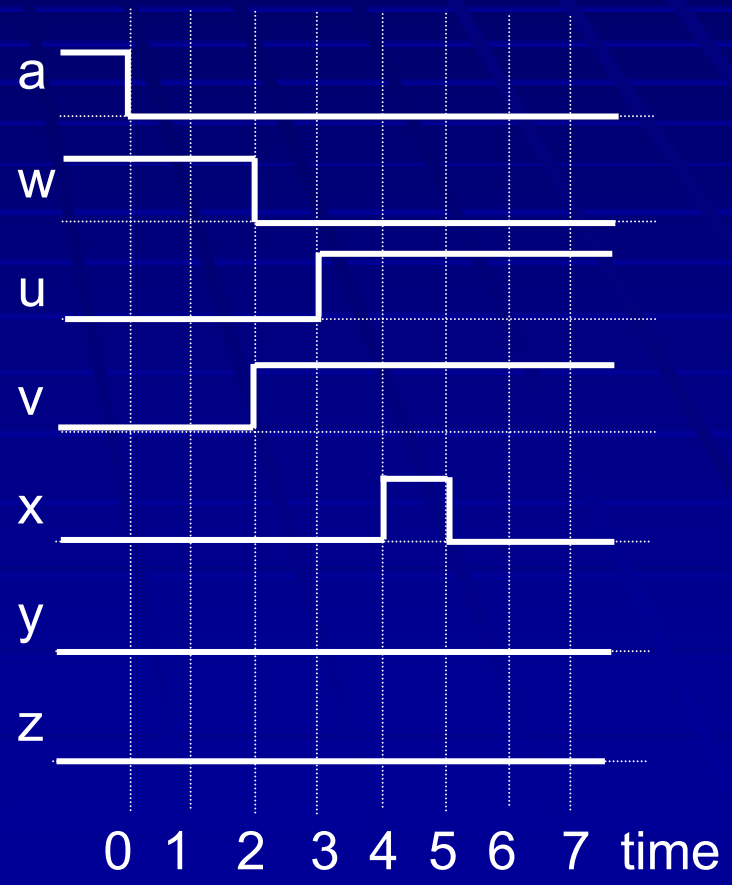
Speeding Up a Circuit





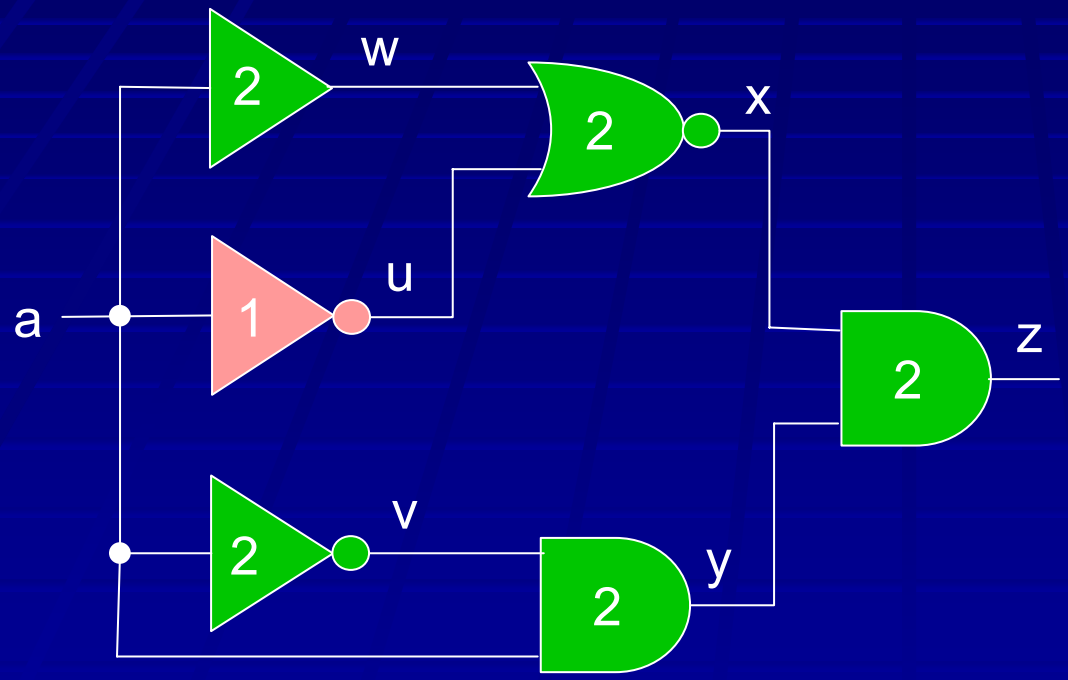
Speeding Up a Circuit

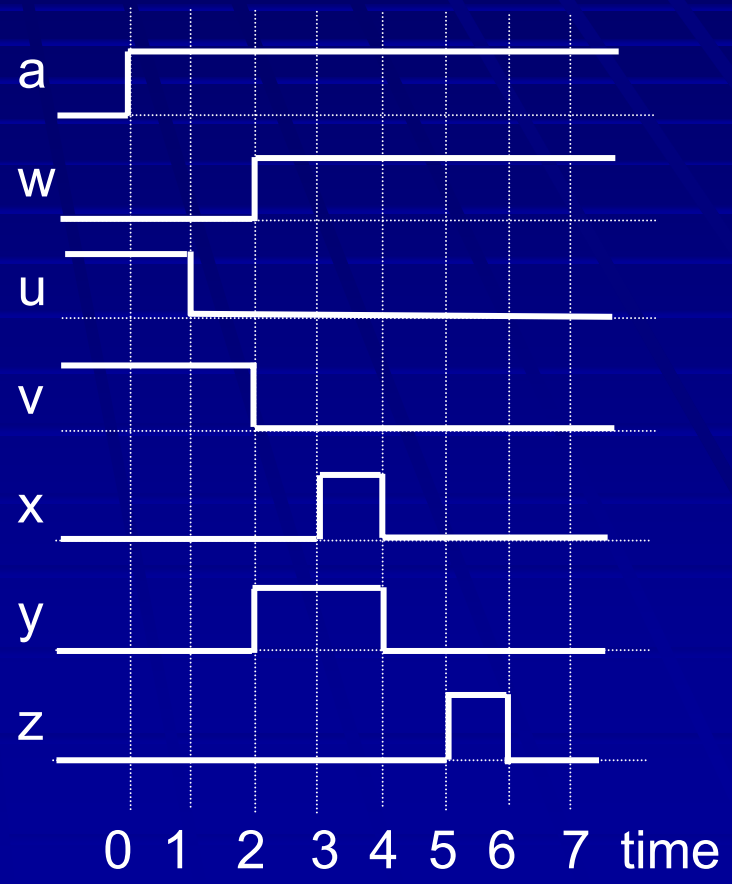




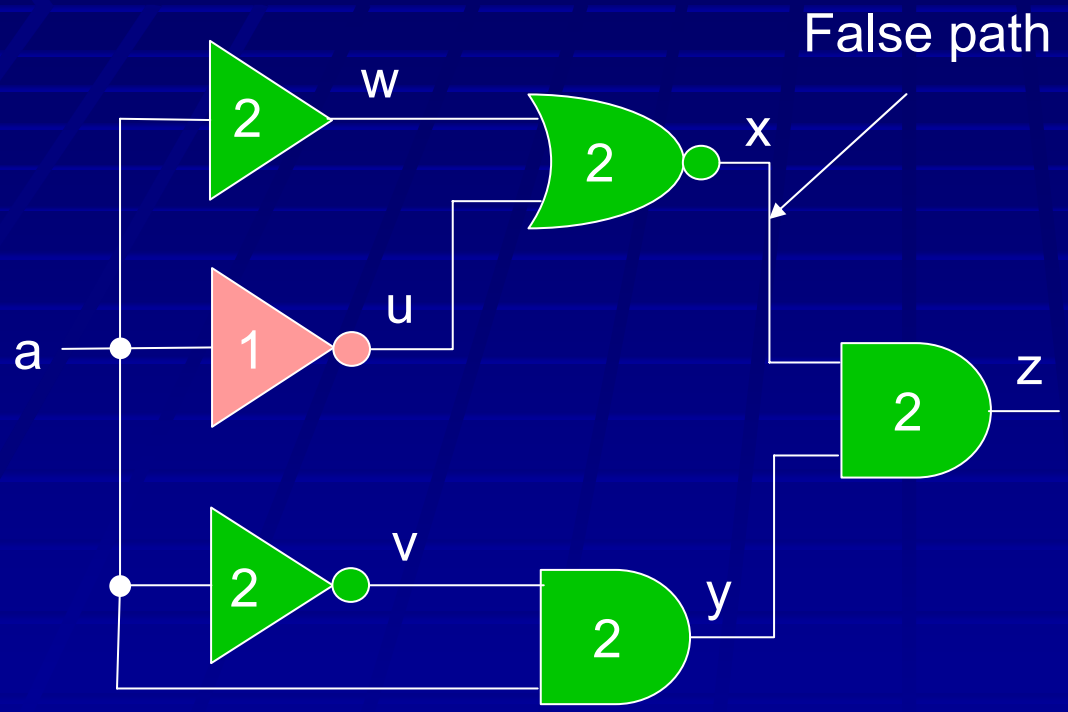
Speeding Up a Circuit

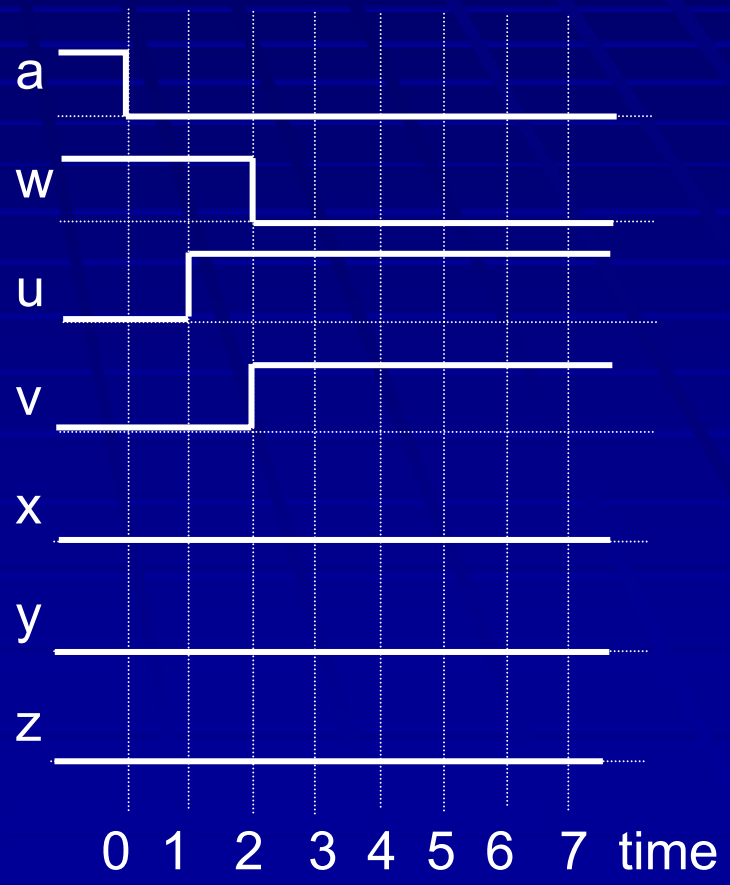
Reducing the delay of a false path can increase circuit delay.





Speeding Up a Circuit

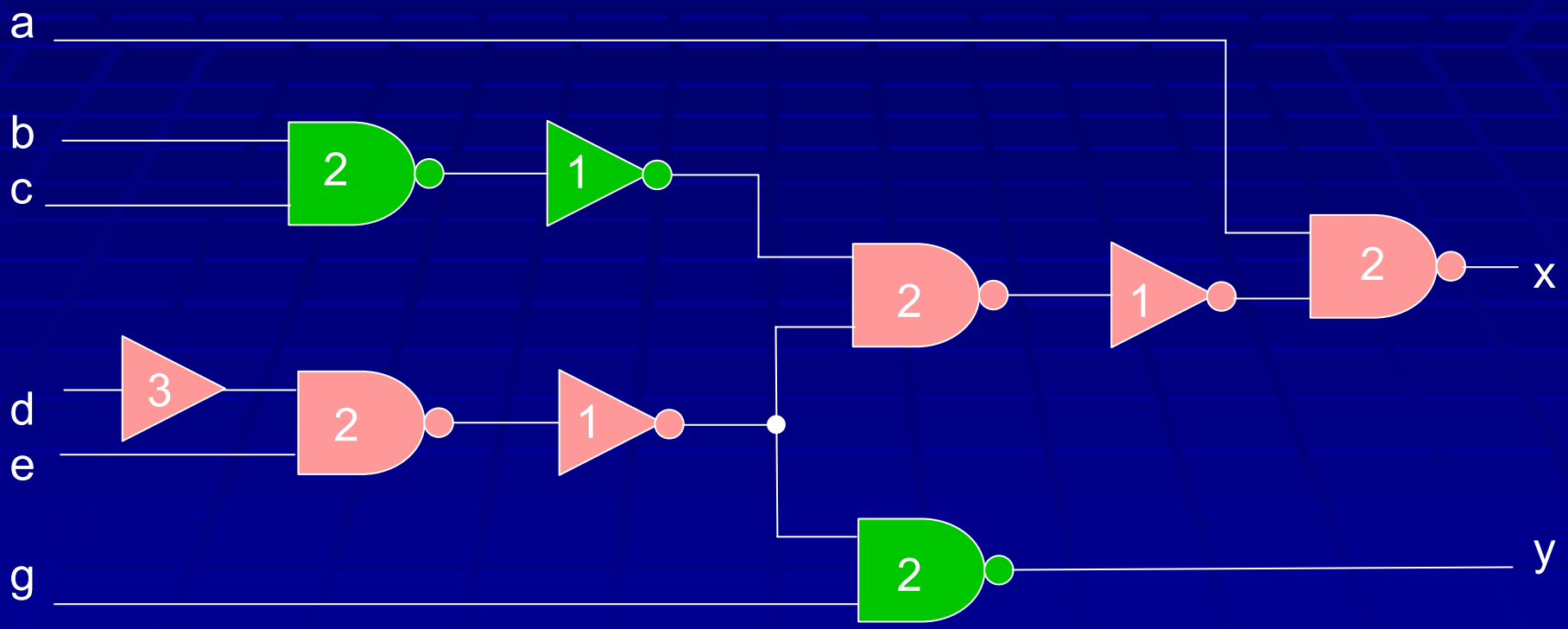




A Delay Optimization Algorithm

- REDUCE_DELAY (Circuit graph (V, E), ε)
  Repeat {
    Compute critical paths and critical delay Δ
    Set output data ready time to Δ
    Compute slacks
    U = vertex subset with slack < ε
    W = select vertices in U
    Apply transformation to vertices in W
  } until (no transformation can reduce Δ)
  }
- G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994, p. 427.

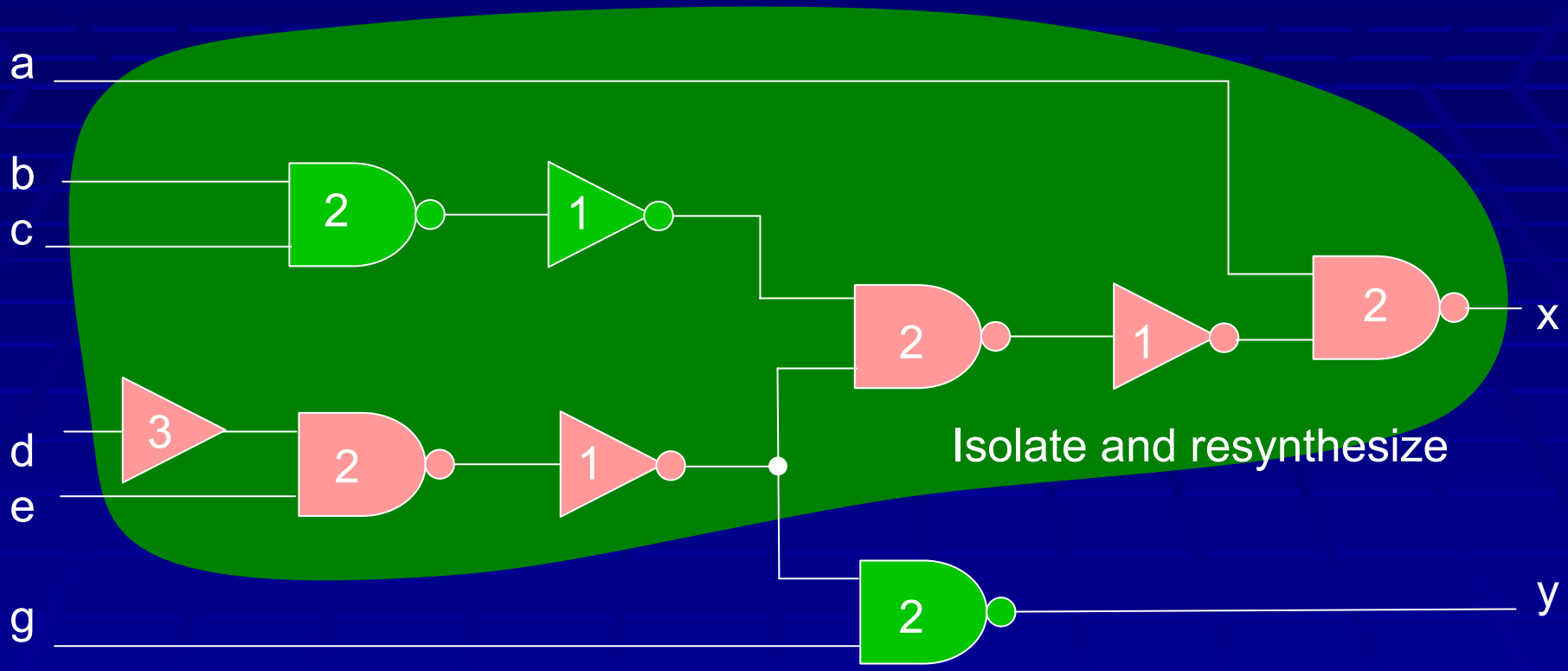
Example of a Transformation (1)



$\Delta = 11$

$x = a' + b' + c' + d' + e'$

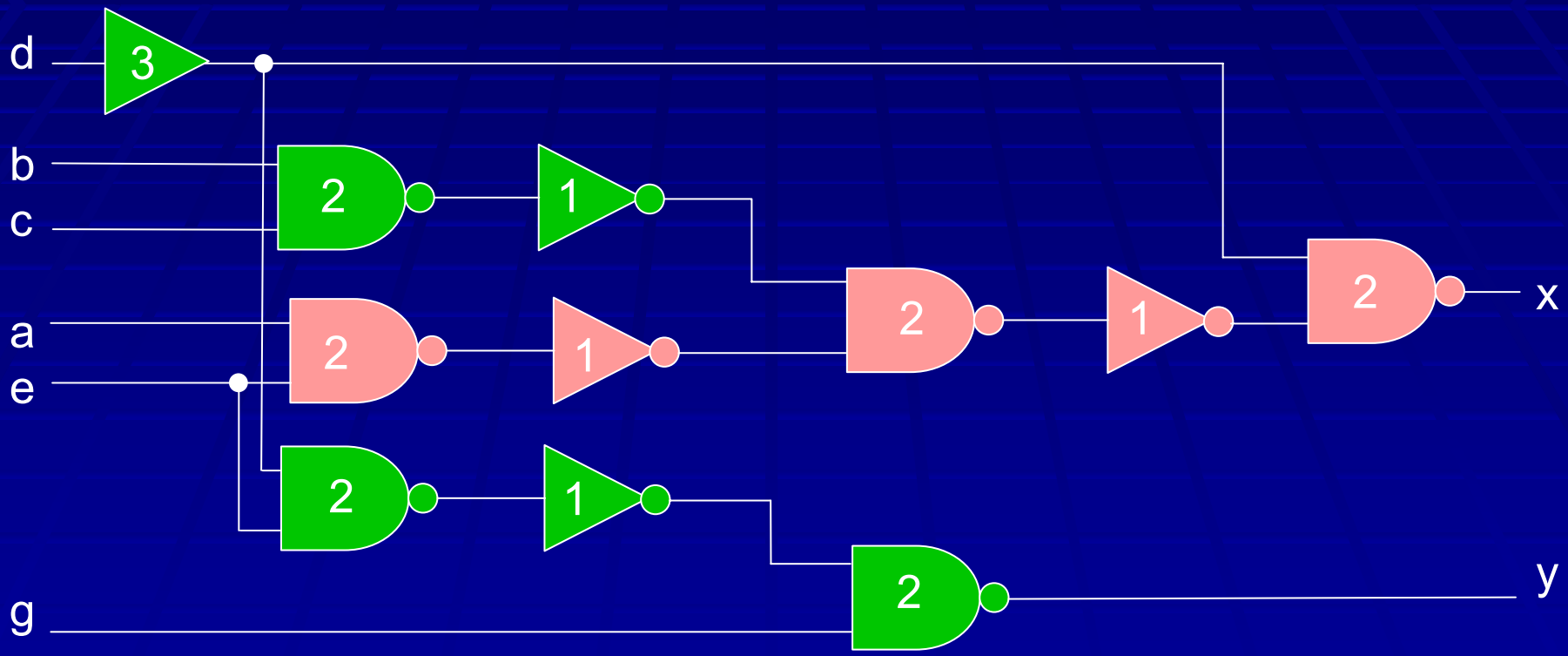
Example of a Transformation (2)



$\Delta = 11$

$x = a' + b' + c' + d' + e'$, all inputs are symmetric.

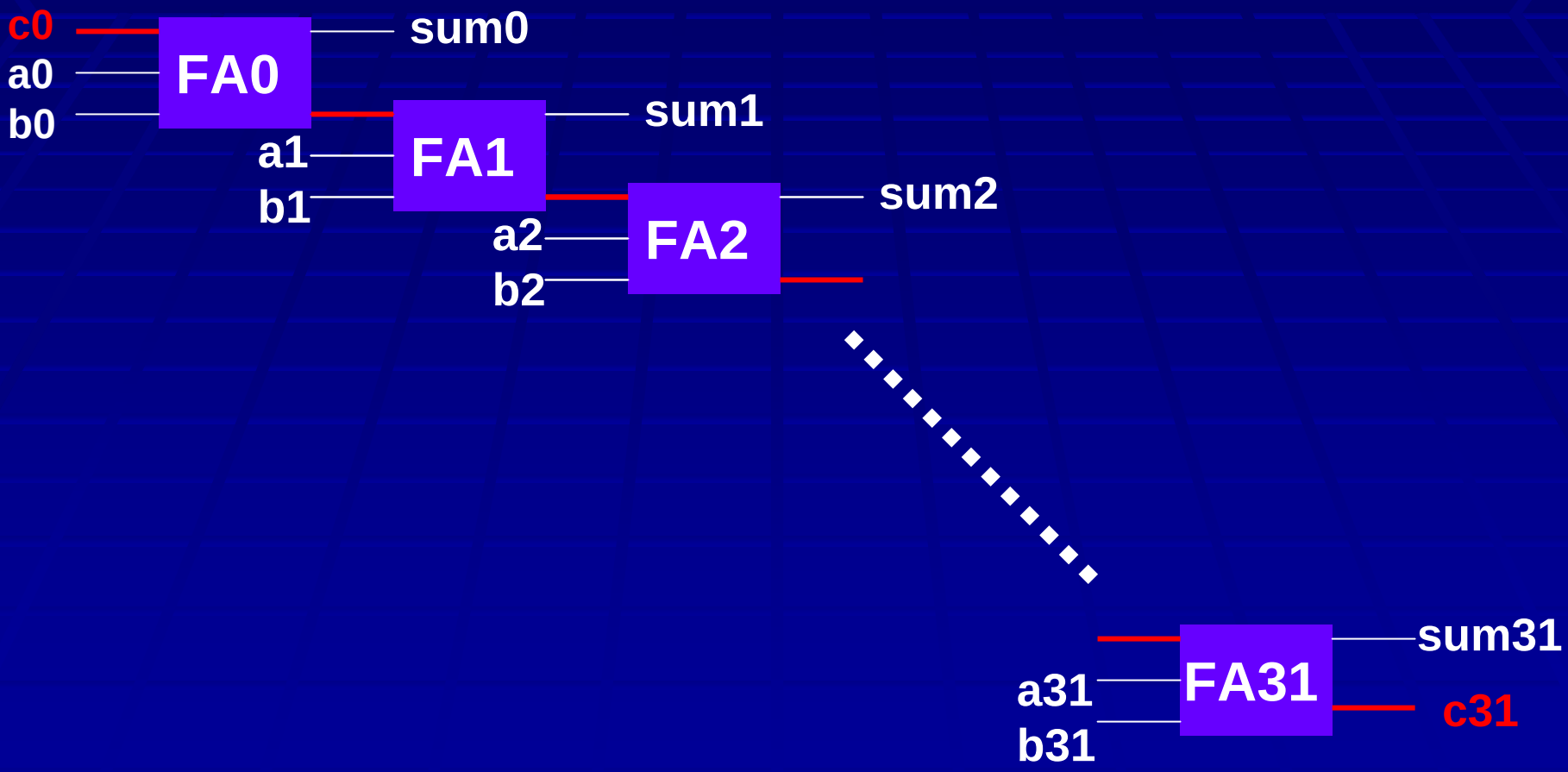
Example of a Transformation (3)



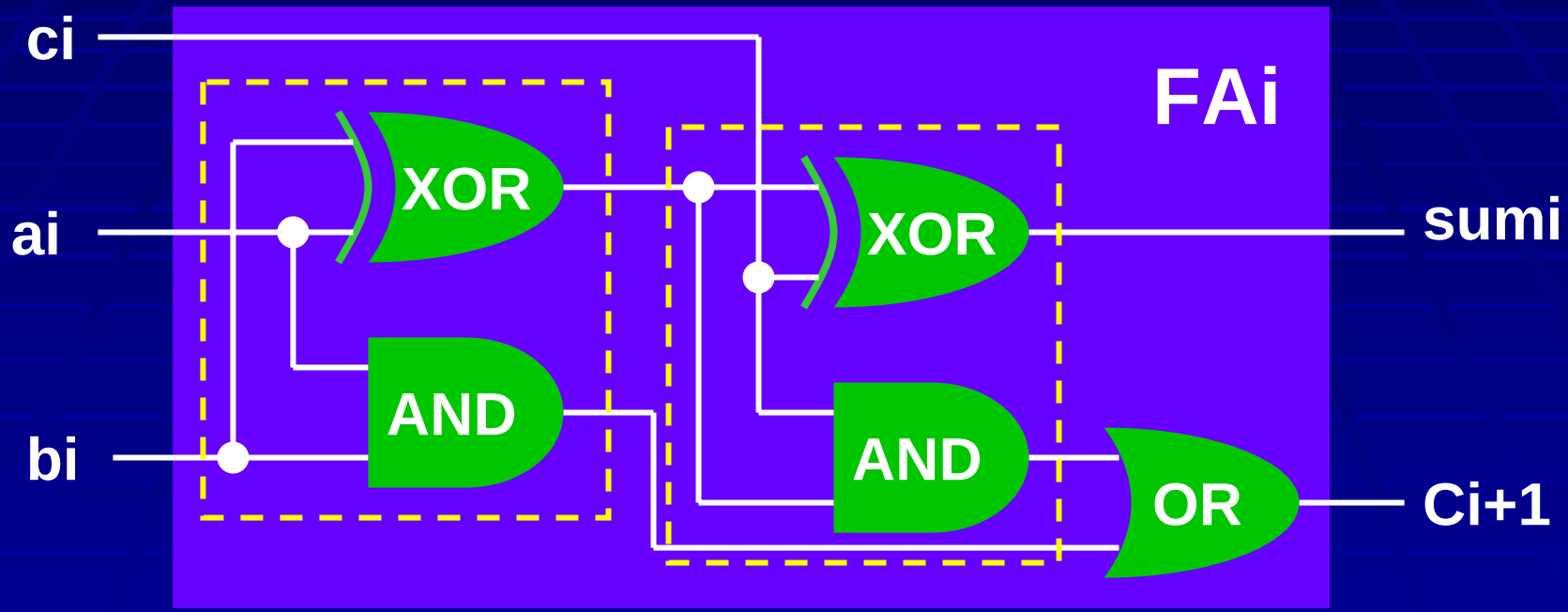
$\Delta = 8$

x = a' + b' + c' + d' + e', a and d are interchanged.

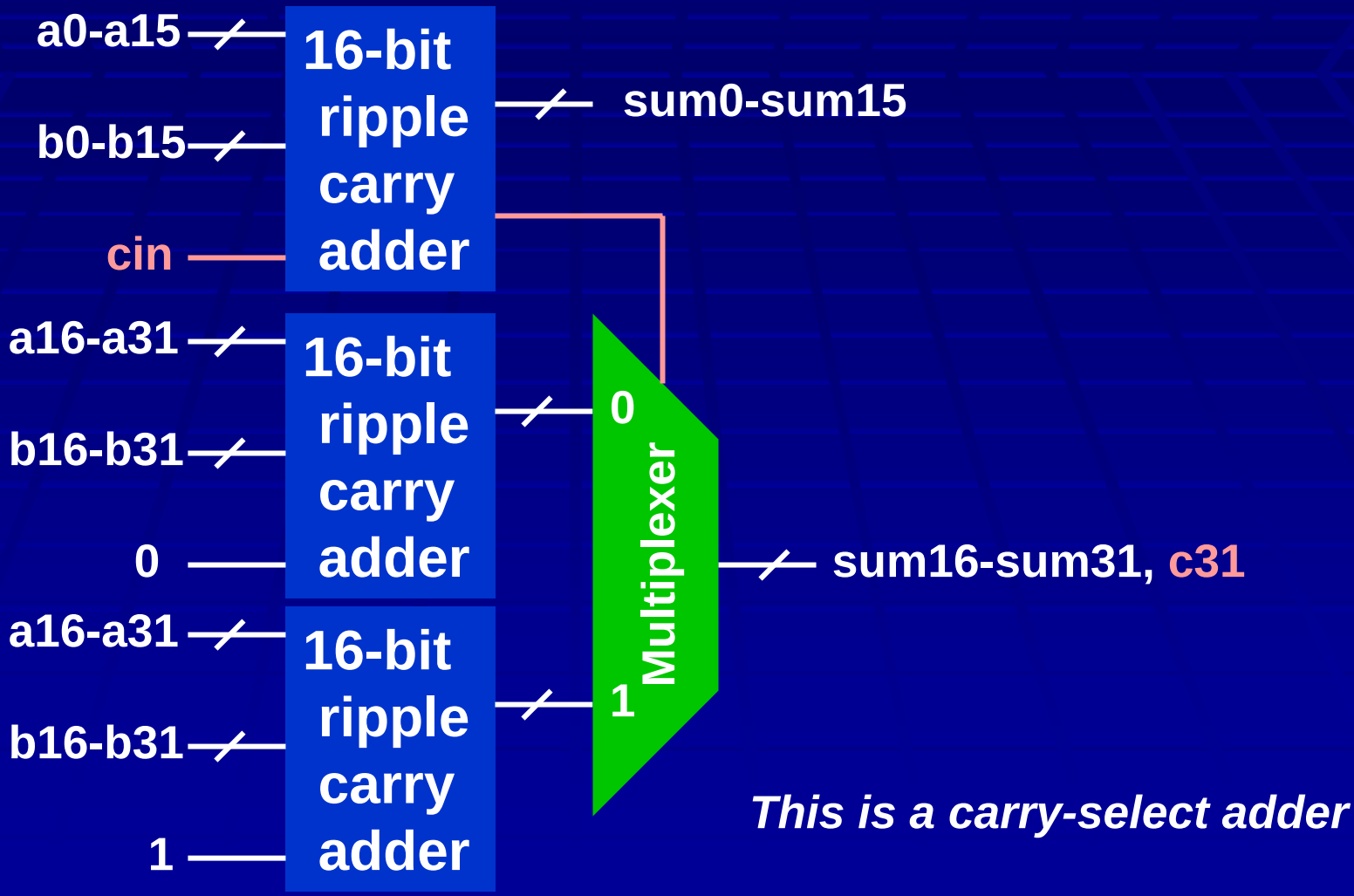
32-bit Ripple-Carry Adder



One-bit Full-Adder Circuit



Speeding Up the Adder



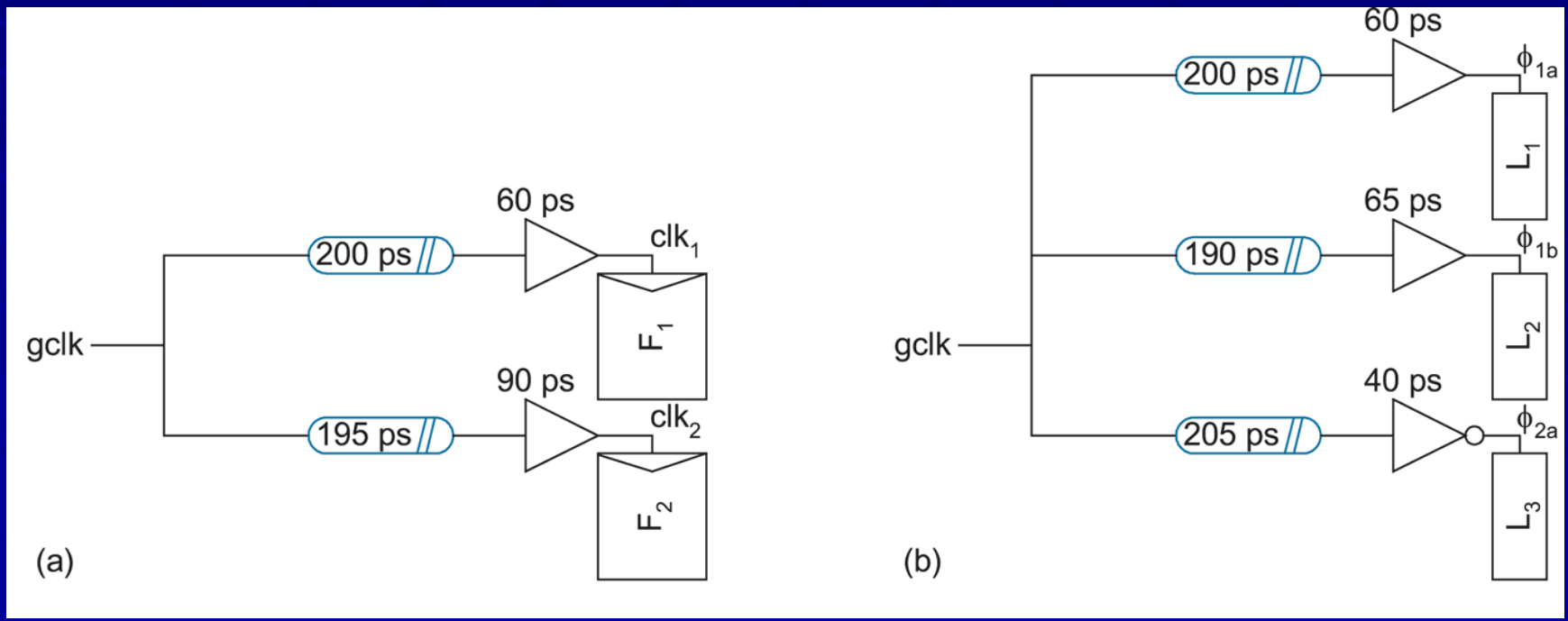
Clock Skew

Logical Clock: No skew
Physical Clock: with skew

t_{skew} ϕ1a,ϕ1b = 5 ps

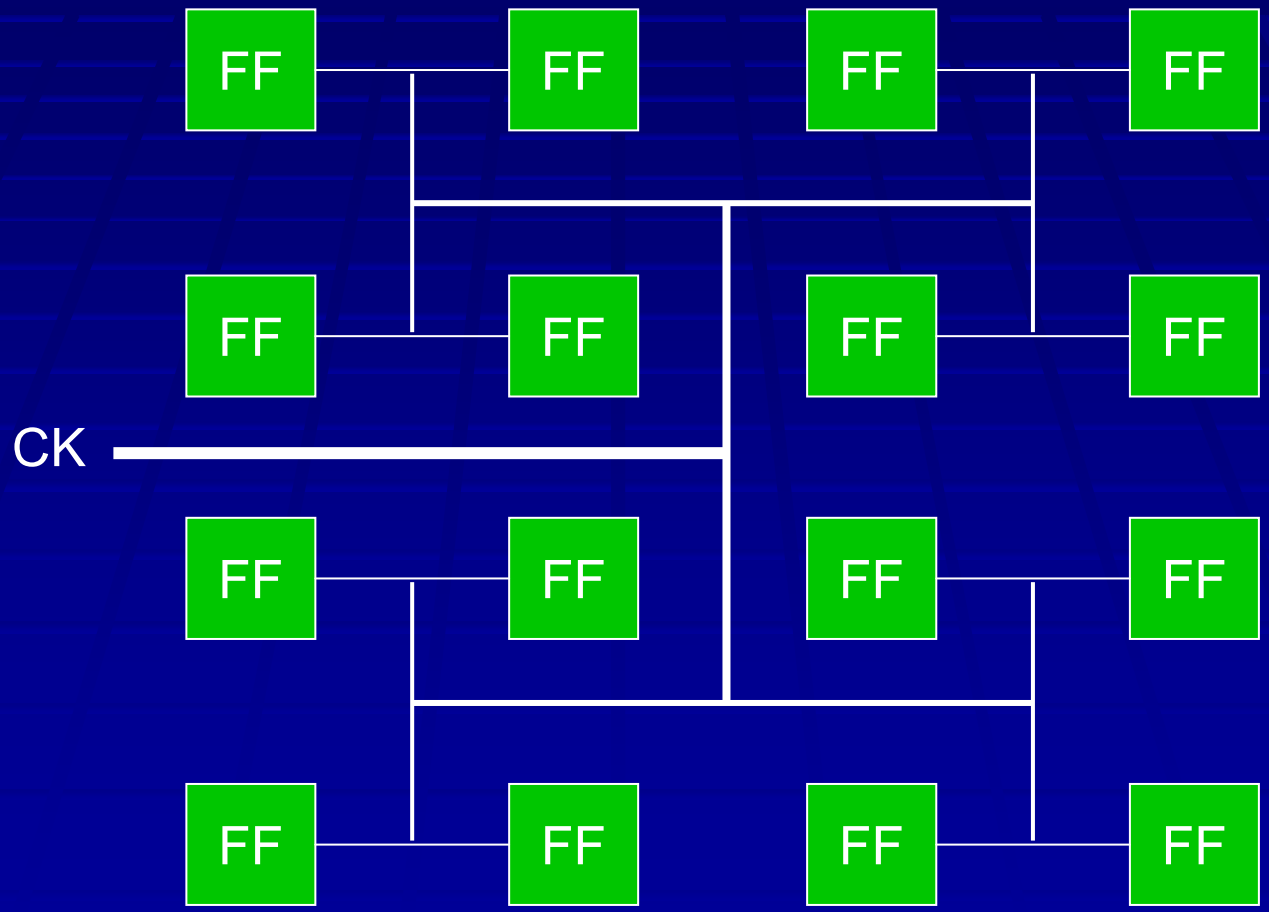
t_{skew} ϕ1a,ϕ2a = 15 ps

t_{skew} ϕ1b,ϕ2a = 10 ps

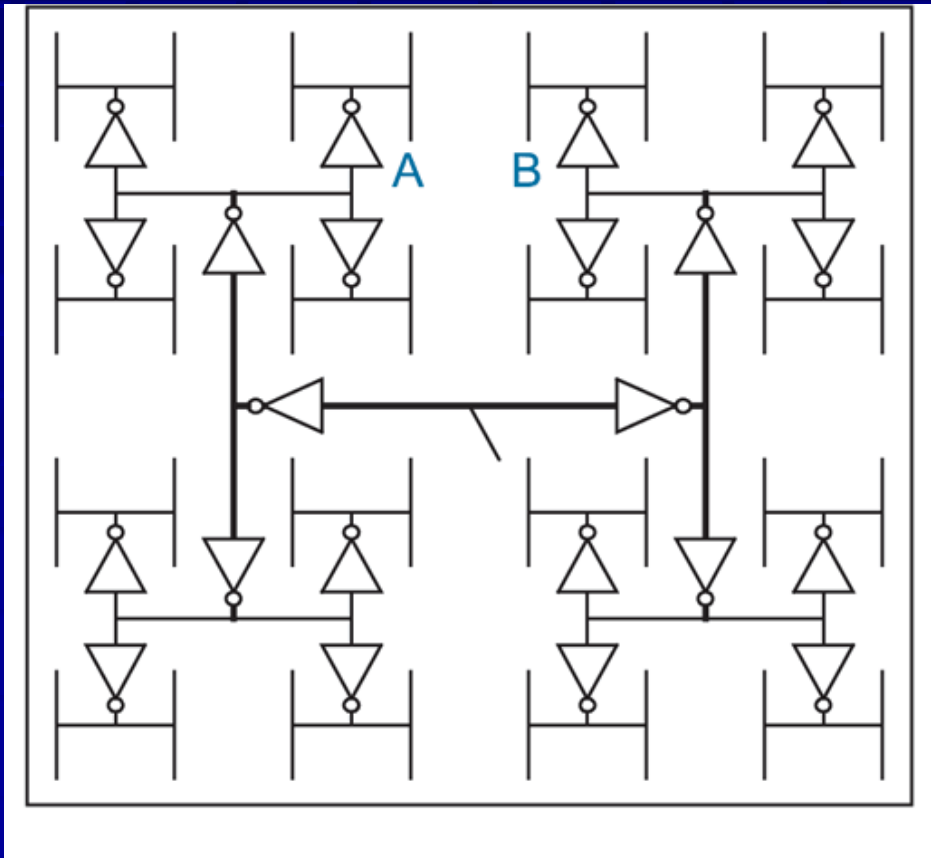


clk1 arrives 25ps before clk2

Zero-Skew Clock Routing



Zero-Skew Clock Routing



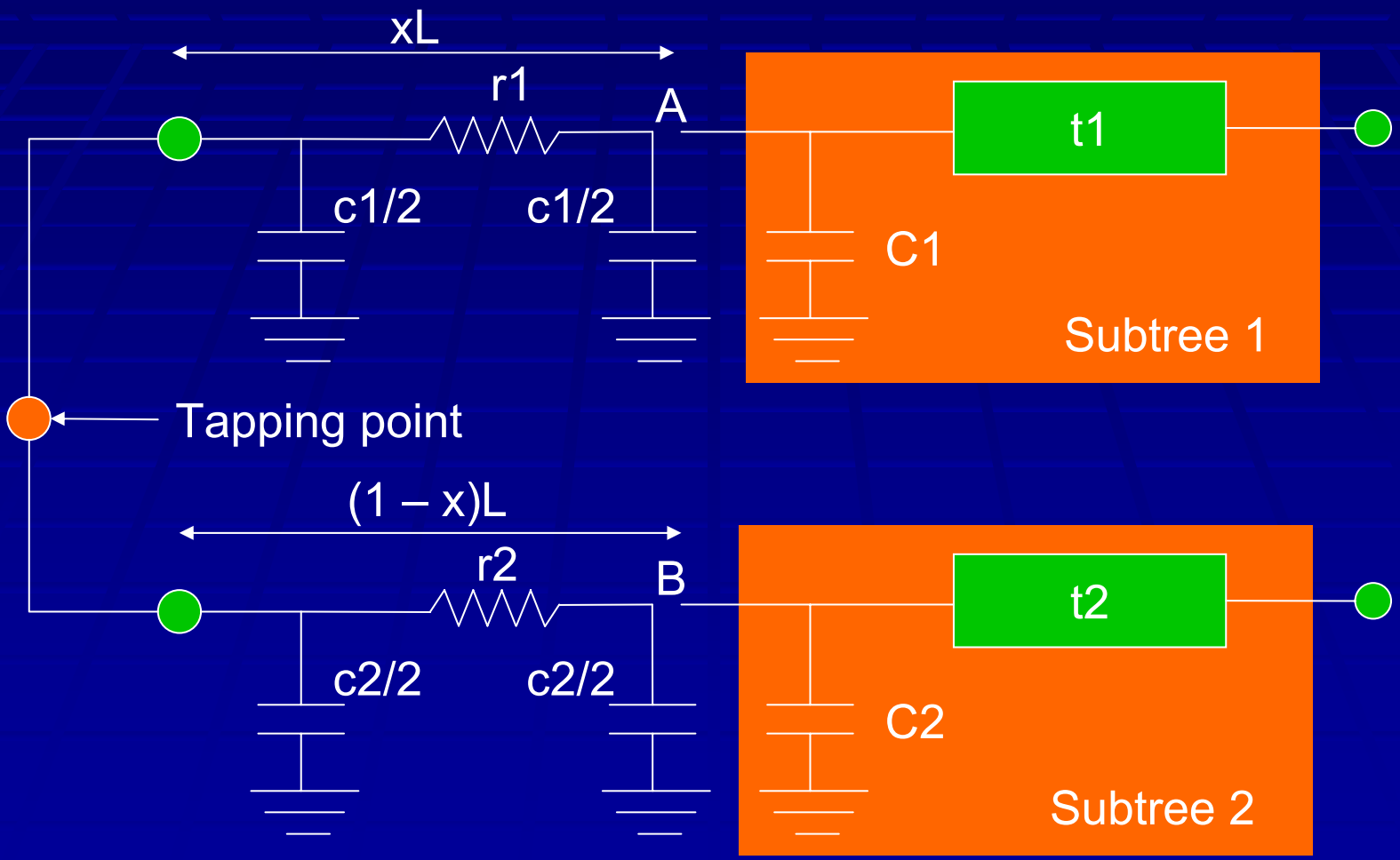
Zero-Skew: References

- H-Tree
  - A. L. Fisher and H. T. Kung, “Synchronizing Large Systolic Arrays,” *Proc. SPIE*, vol. 341, pp. 44-52, May 1982.
  - A. Kahng, J. Cong and G. Robins, “Hig-Performance Clock Routing Based on Recursive Geomrtric Matching,” *Proc. Design Automation Conf*., June 1991, pp. 322-327.
  - M. A. B. Jackson, A. Srinivasan and E. S. Kuh, “Clock Routing for High-Performance IC’s,” *Proc. Design Automation Conf*., June 1990, pp. 573-579.

Zero-Skew Routing

- Build clock tree bottom up:
  - Leaf nodes are all equal loading flip-flops.
  - Two zero-skew subtrees are joined to form a larger zero-skew subtree.
  - Entire clock tree is built recursively.
- R.-S. Tsay, "An Exact Zero-Skew Clock Routing Algorithm," *IEEE Trans. CAD*, vol. 12, no. 2, pp. 242-249, Feb. 1993.
- J. Rubenstein, P. Penfield and M. A. Horowitz, "Signal Delay in RC Tree Networks," *IEEE Trans. CAD*, vol. 2, no. 3, pp. 202-211, July 1983.

Balancing Subtrees (1)



Balancing Subtrees (2)

- Subtrees 1 and 2 are each balanced (zero-skew) trees, with delays t1 and t2 to respective leaf nodes.
- Total capacitances of subtrees are C1 and C2, respectively.
- Connect points A and B by a minimum-length wire of length L.
- Determine a tapping point x such that wire lengths xL and (1 – x)L produce zero skew.

Balancing Subtrees (3)

- Use Elmore delay formula:

  $r1(C1 + c1/2) + t1 = r2(C2 + c2/2) + t2$

- Substitute:
  - $r1 = axL$, $r2 = a(1 – x)L$
  - $c1 = bxL$, $c2 = b(1 –x)L$

  $abL^2x + aL(C1+C2)x = (t2 – t1) + aL(C2+bL/2)$

- Then solve for x:

$$x = \frac{(t2 – t1) + aL\,(C2 + bL/2)}{aL(bL + C1 + C2)}$$

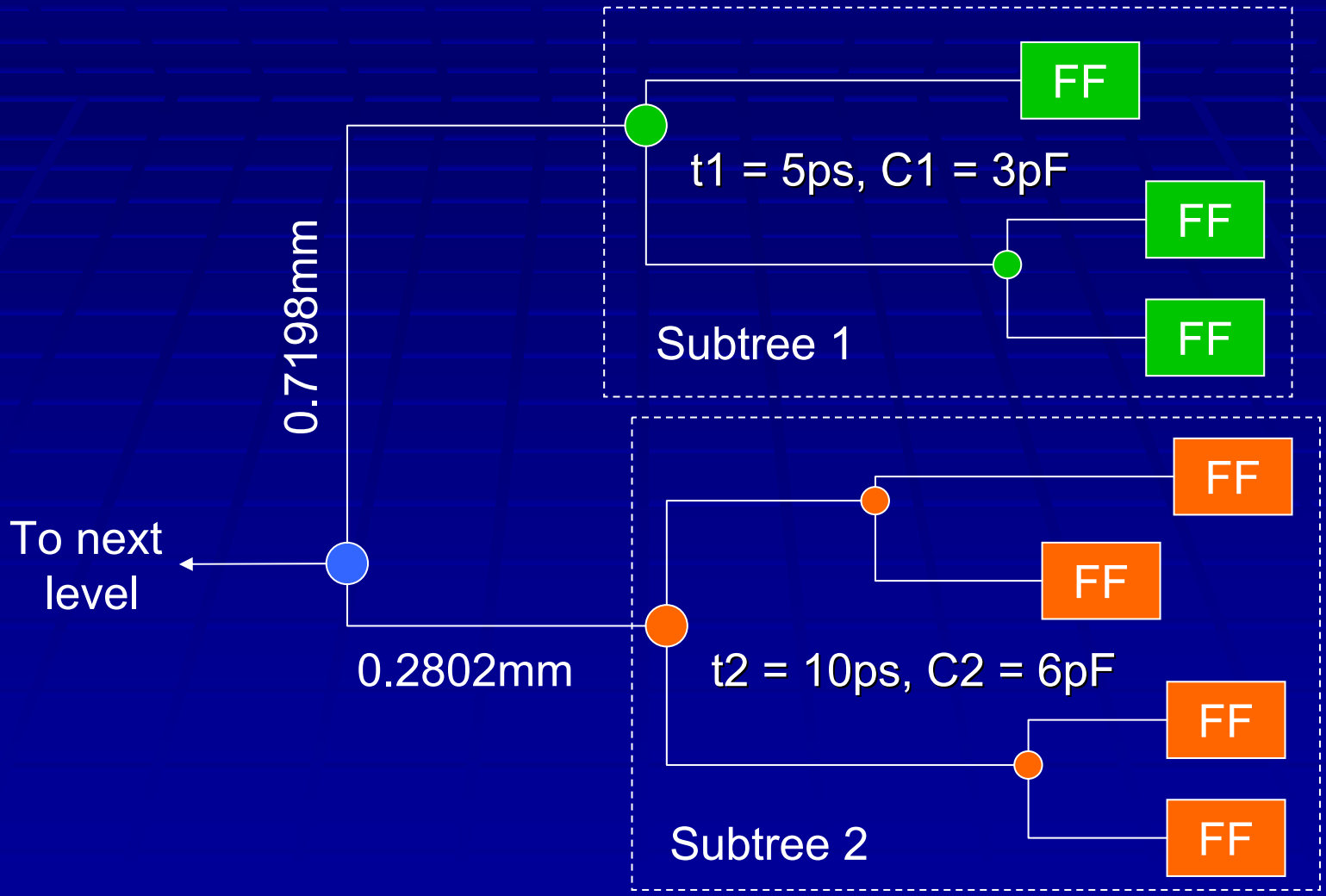
Balancing Subtrees Example 1

- Subtree parameters:
  - Subtree 1: t1 = 5ps, C1 = 3pF
  - Subtree 2: t2 = 10ps, C2 = 6pF
- Interconnect:
  - L = 1mm
  - Wire parameters: a = 100Ω/cm, b = 1pF/cm
- Tapping point:

$$x = \frac{(t2 - t1) + aL\,(C2 + bL/2)}{aL\,(bL + C1 + C2)} = \frac{(10-5) + 100\times0.1(6 + 1\times0.1/2)}{100\times0.1(1\times0.1+3+6)}$$

$$= (5 + 60.5)/(10\times9.1) = 0.7198$$

Example 1


FF
t1 = 5ps, C1 = 3pF
FF
FF
Subtree 1
0.7198mm
To next level
0.2802mm
FF
FF
t2 = 10ps, C2 = 6pF
FF
FF
Subtree 2

Balancing Subtrees, x > 1

- Tapping point set at root of tree with larger loading (C2, t2).
- Wire to the root of other tree is elongated to provide additional delay. Wire length L found as follows:
- Set x = 1 in $abL^2x + aL(C1+C2)x = (t2 - t1) + aL(C2+bL/2)$

  i.e., $L^2 + (2C1/b)L - 2(t2 - t1)/(ab) = 0$
- Wire length is given by:

$$L = \frac{[(aC1)^2+2ab(t2 - t1)]^{1/2} - aC1}{a\ b}$$

- R.-S. Tsay, "An Exact Zero-Skew Clock Routing Algorithm," *IEEE Trans. CAD*, vol. 12, no. 2, pp. 242-249, Feb. 1993.

Balancing Subtrees Example 2

- Subtree parameters:
  - Subtree 1: t1 = 2ps, C1 = 1pF
  - Subtree 2: t2 = 15ps, C2 = 10pF
- Interconnect:
  - L = 1mm
  - Wire parameters: a = 100Ω/cm, b = 1pF/cm
- Tapping point:

$$x = \frac{(t2 - t1) + aL\,(C2 + bL/2)}{aL\,(bL + C1 + C2)} = \frac{(15-2) + 100\times0.1(10 + 1\times0.1/2)}{100\times0.1(1\times0.1+1+10)}$$

$$= (13 + 100.5)/(10\times11.1) = 1.0225$$

Example 2, x = 1.0225

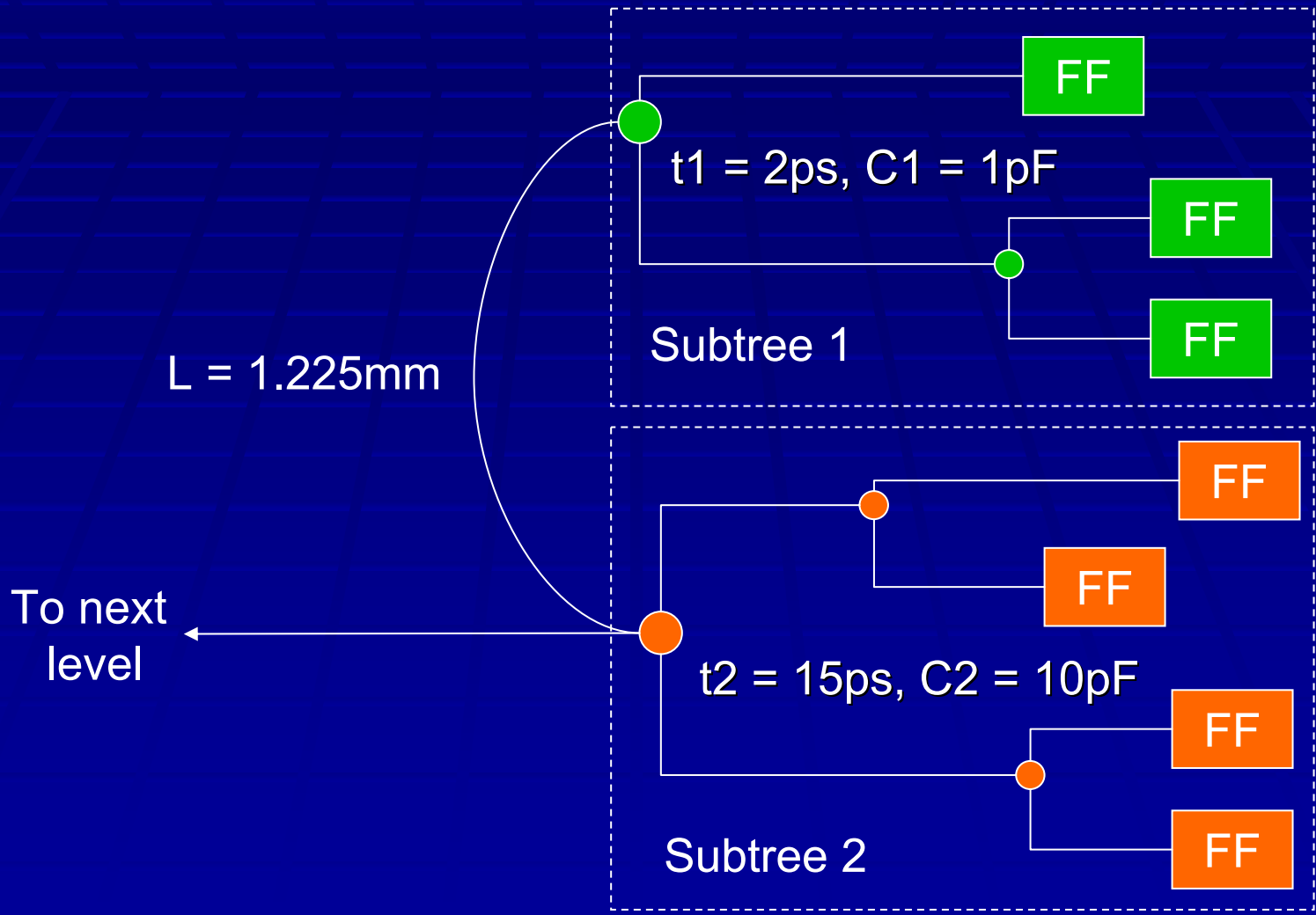
Setting x = 1.0,

$$L = \frac{[(aC1)^2+2ab(t2 - t1)]^{1/2} - aC1}{a\ b}$$

$$= \frac{[(100\times1)^2+2100\times1(15 - 2)]^{1/2} - 100\times1}{100\times1}$$

$$= 0.1225\text{cm}$$

Example 2, L = 1.255mm



Balancing Subtrees, x < 0

- Tapping point set at root of tree with smaller loading (C1, t1).
- Wire to the root of other tree is elongated to provide additional delay. Wire length L found as follows:
- Set x = 0 in $abL^2x + aL(C1+C2)x = (t2 - t1) + aL(C2+bL/2)$

  i.e., $L^2 + (2C2/b)L - 2(t1 - t2)/(ab) = 0$
- Wire length is given by:

$$L = \frac{[(aC2)^2+2ab(t1 - t2)]^{1/2} - aC2}{a\ b}$$

- R.-S. Tsay, "An Exact Zero-Skew Clock Routing Algorithm," *IEEE Trans. CAD*, vol. 12, no. 2, pp. 242-249, Feb. 1993.

Balancing Subtrees Example 3

- Subtree parameters:
  - Subtree 1: t1 = 15ps, C1 = 10pF
  - Subtree 2: t2 = 2ps, C2 = 1pF
- Interconnect:
  - L = 1mm
  - Wire parameters: a = 100Ω/cm, b = 1pF/cm
- Tapping point:

$$x = \frac{(t2 - t1) + aL\,(C2 + bL/2)}{aL\,(bL + C1 + C2)} = \frac{(2-15) + 100\times0.1(1 + 1\times0.1/2)}{100\times0.1(1\times0.1+1+10)}$$

$$= (-13 + 10.5)/(10\times11.1) = -0.0225$$

Example 3, x = – 0.0225

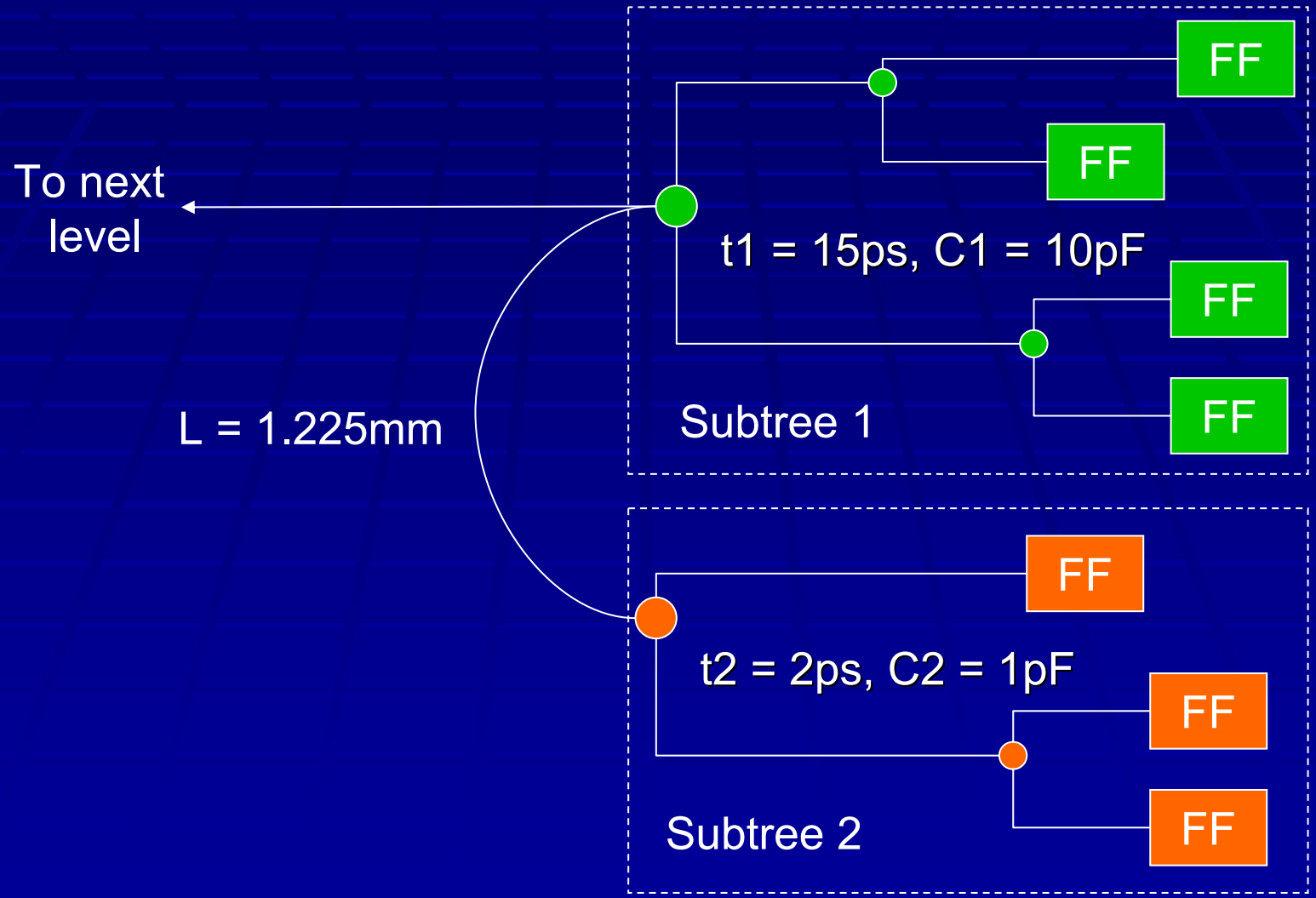
Setting x = 0.0,

$$L = \frac{[(aC2)^2+2ab(t1 - t2)]^{1/2} - aC2}{a\ b}$$

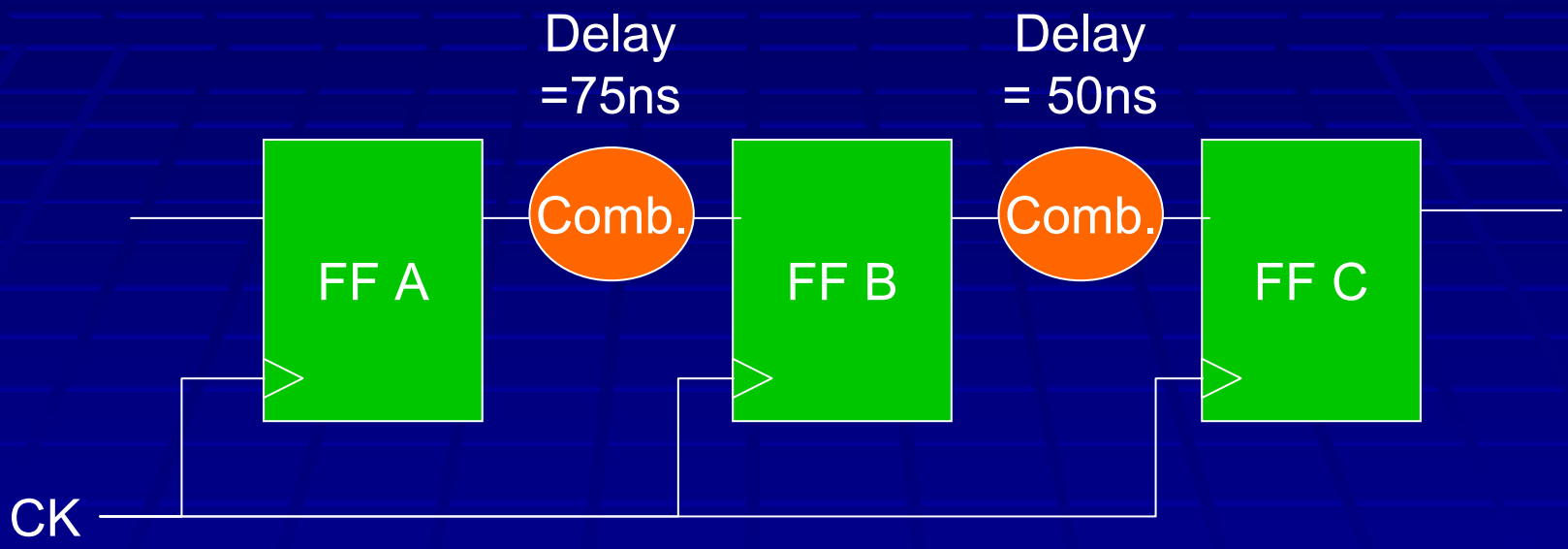
$$= \frac{[(100\times1)^2+2100\times1(15 - 2)]^{1/2} - 100\times1}{100\times1}$$

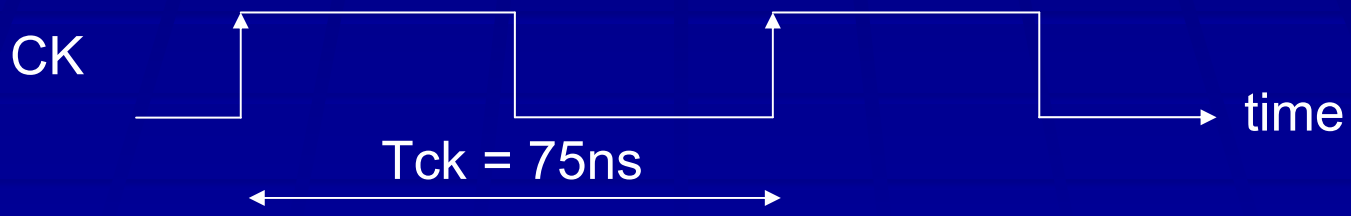
$$= 0.1225\text{cm}$$

Example 3, L = 1.255mm



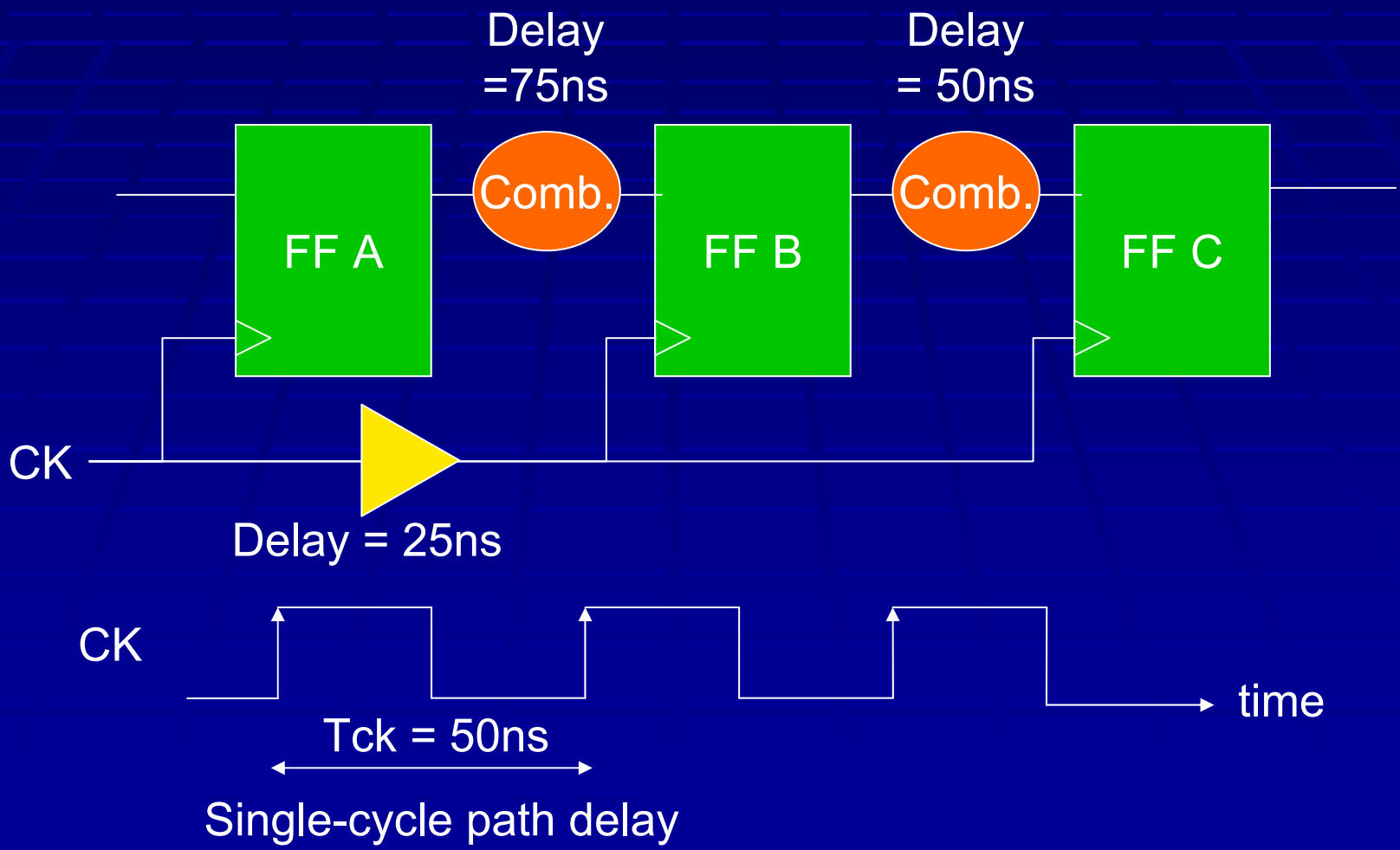
Zero-Skew Design





Single-cycle path delay

Nonzero-Skew Design



Local Clock Gaters

