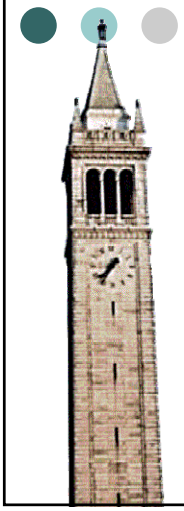


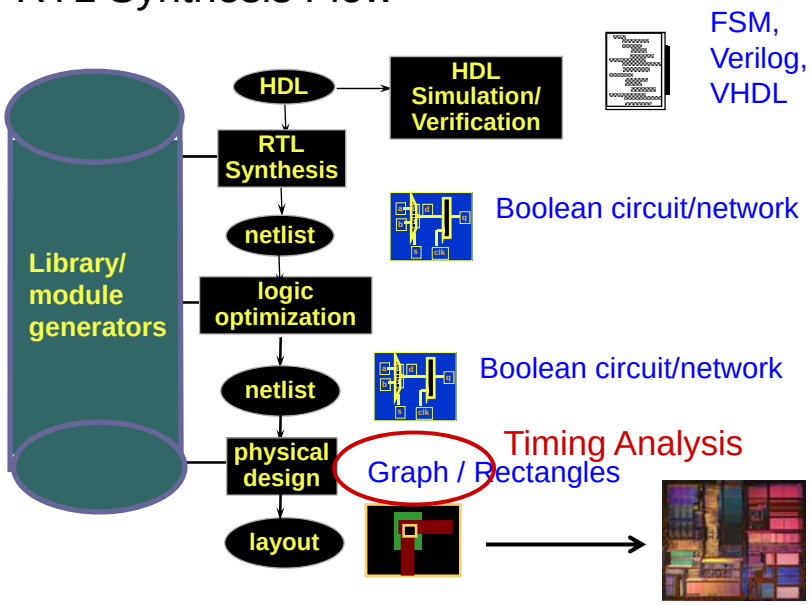
Fundamental Algorithms for System Modeling, Analysis, and Optimization

Edward A. Lee, Jaijeet Roychowdhury, Sanjit A. Seshia, Stavros Tripakis
UC Berkeley
EECS 144/244
Spring 2013

Lecture: Timing Analysis, Retiming

Thanks to Kurt Keutzer for several slides

RTL Synthesis Flow



Timing Analysis / Verification

Verifying a property about **system timing**

Arises in many settings:

- Integrated circuits
- Embedded software
- Distributed embedded systems
- Biological systems
- …

Here we focus on circuits.

Chapter 15 of Lee-Seshia book contains a more broad discussion.

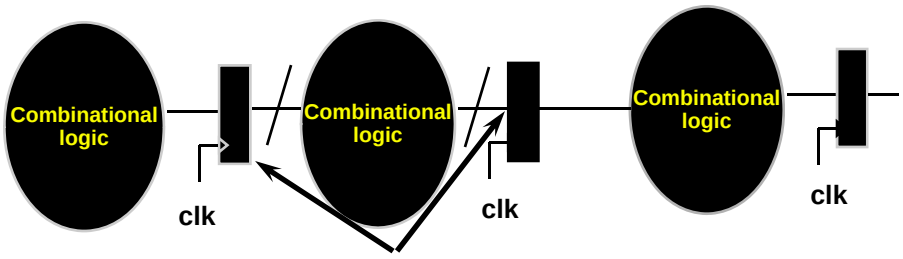
Timing Analysis for Digital ICs

(Clock) Speed is one of the major performance metrics for digital circuits

Timing Analysis = the process of verifying that a chip meets its speed requirement

- E.g., 1 GHz means that next-state function must be computed within 1 ns

Static Timing Analysis for Circuits



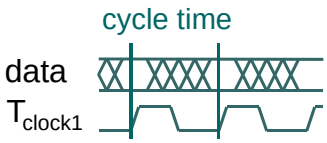
Determine fastest permissible clock speed (e.g. 1 GHz) by determining delay of longest path from register to register (e.g. 1ns.)

Cycle Time - Critical Path Delay

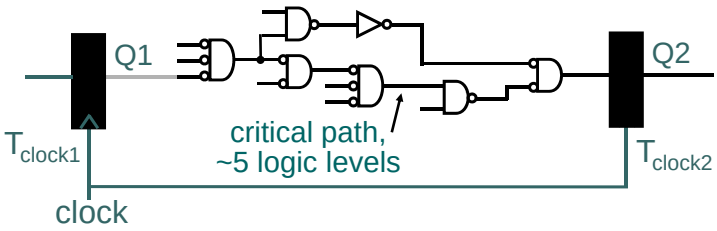
Cycle time (T) cannot be smaller than longest path delay (T_{max})

Longest (critical) path delay is a function of:

Total gate, wire delays



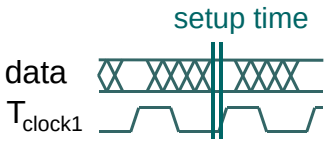
$T_{max} \leq T$



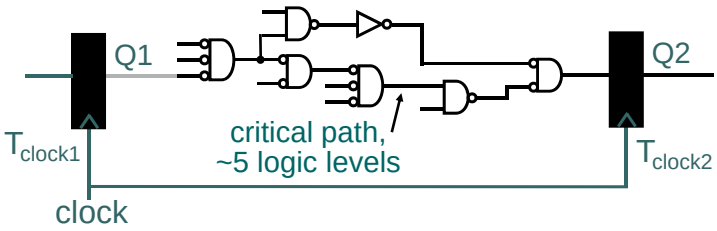
Cycle Time - Setup Time

For FFs to correctly latch data, input data must be stable during:

Setup time (T_{setup}) *before* clock arrives



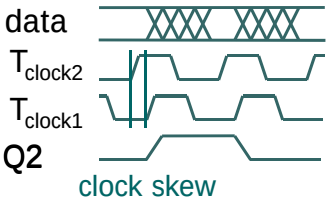
$T_{max} + T_{setup} \leq T$



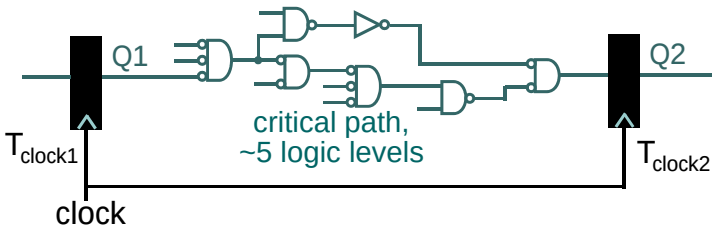
Cycle Time - Clock-skew

If clock network has unbalanced delay – clock skew

Cycle time is also a function of clock skew (T_{skew})



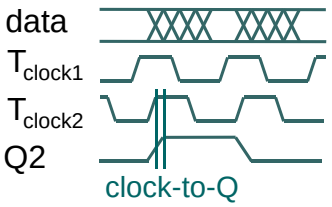
$T_{max} + T_{setup} + T_{skew} \leq T$



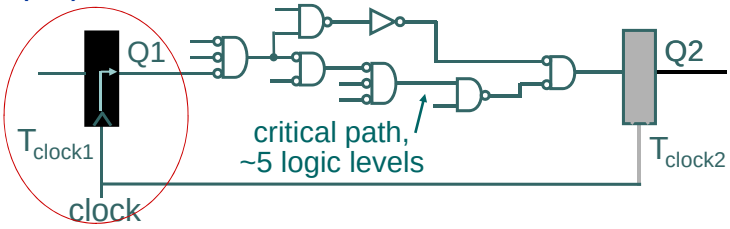
Cycle Time - Clock to Q

Cycle time is also a function of propagation delay of FF ($T_{clk\text{-}to\text{-}Q}$)

$T_{clk\text{-}to\text{-}Q}$: time from arrival of clock signal till change at FF output)



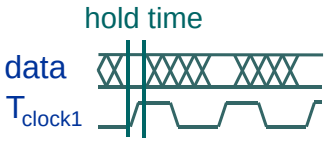
$$T_{max} + T_{setup} + T_{skew} + T_{clk-to-Q} \leq T$$



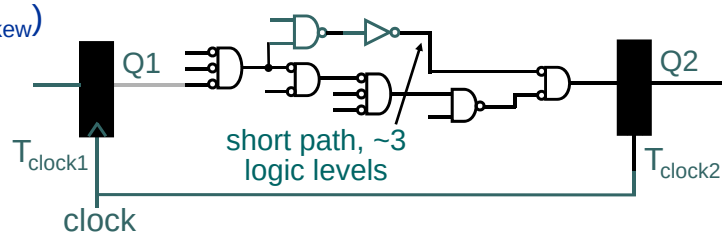
Min Path Delay - Hold Time

For FFs to correctly latch data, input data must be stable during Hold time (T_{hold}) *after* clock arrives

Determined by delay of shortest path in circuit (T_{min}) and clock skew (T_{skew})



$$T_{min} \geq T_{hold} + T_{skew}$$



Summary

Need to be able to compute

T_{min}: delay of "fastest" path in the circuit

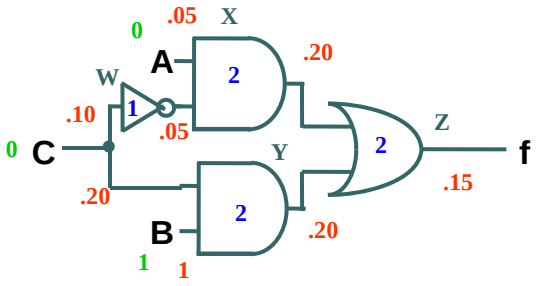
T_{max}: delay of "slowest" (critical) path in the circuit

Modeling Timing in a Combinational Circuit

Arrival time in green

Interconnect delay in red

Gate delay in blue



What's the right mathematical object to use to represent this physical object?

Modeling - 1

Use a labeled *directed* graph

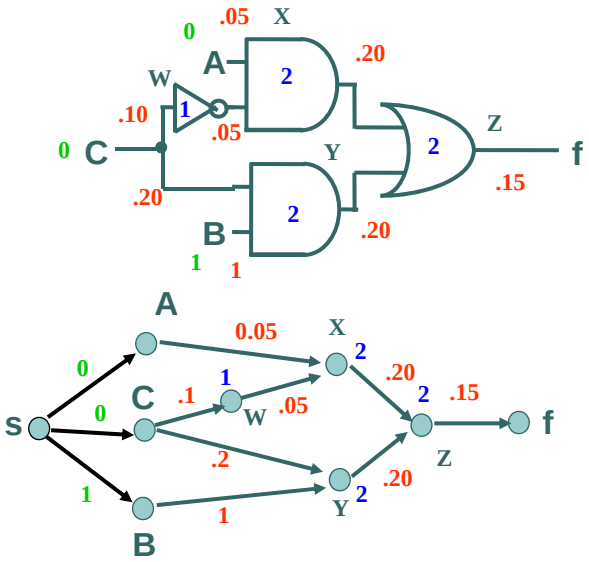
G = <V,E>

Vertices represent gates, primary inputs and primary outputs

Edges represent wires

Labels represent delays

What about arrival times?

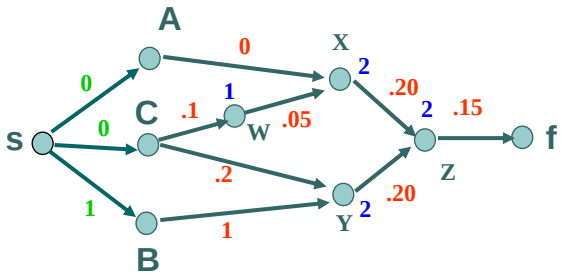


Modeling - 2

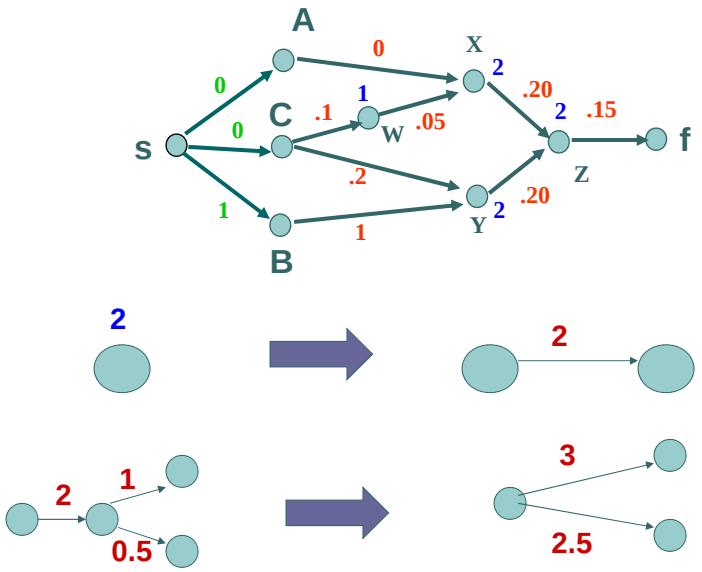
Find longest/shortest paths in a *directed* graph *G = <V,E>*

What sort of directed graph do we have?

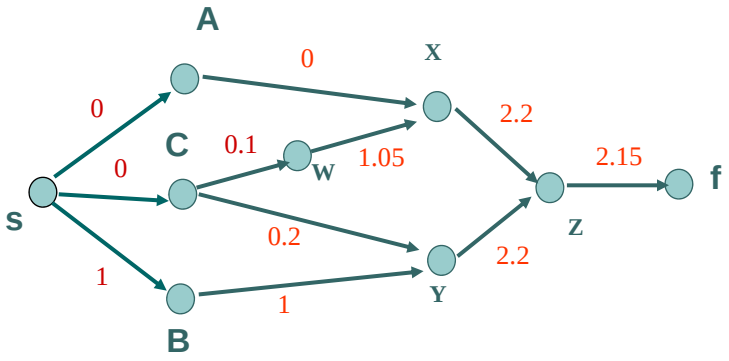
Is this in the standard form for a longest/shortest path problem?



Split Nodes into Edges



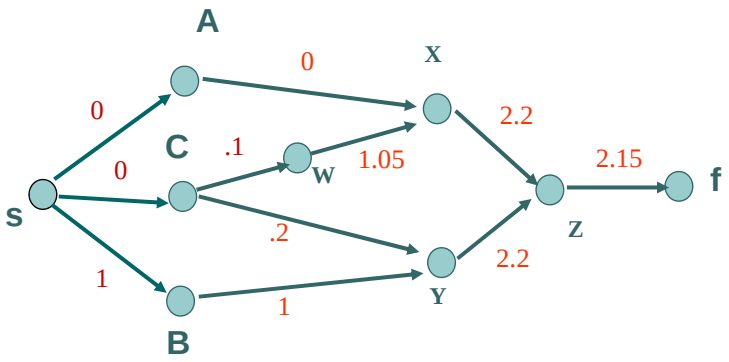
DAG with Weighted Edges



Problem: Find the longest (critical) path from source s to sink f.

Naïve Approach: Enumerate Paths

How many paths in this example?
In the worst case?



Problem:
Find the longest path from source s to sink f.

Algorithm 1: Longest path in a DAG

Critical Path Method **[Kirkpatrick 1966, IBM JRD]**
(dynamic programming)

Let w(u,v) denote weight of edge from u to v
Steps:
1. Topologically sort vertices (graph is acyclic)
   order: $v_1, v_2, \ldots, v_n$ $v_1 = s$, $v_n = ?$
2. For each vertex v, compute
   d(v) = length of longest path from source s to v
   $d(v_1) = 0$
   For i = 2..n
   $d(v_i) = \max_{\text{all incoming edges } (u, v_i)} d(u) + w(u,v_i)$

Algorithm 1: Longest path in a DAG

Critical Path Method **[Kirkpatrick 1966, IBM JRD]**

Let w(u,v) denote weight of edge from u to v

Steps:

1. Topologically sort vertices

   order: $v_1, v_2, \ldots, v_n$ $\quad v_1 = s, \; v_n = f$

2. For each vertex v, compute

   d(v) = length of longest path from source s to v

   $d(v_1) = 0$

   For i = 2..n

   $d(v_i) = \max_{\text{all incoming edges } (u, v_i)} d(u) + w(u,v_i)$

Time Complexity?
O(m+n)

Run the CPM on our example

Graph vs. Circuit

Delay of a combinational circuit depends on

- Circuit topology (graph model)
- Delay model (e.g., fixed vs. variable)
- Boolean behavior of gates

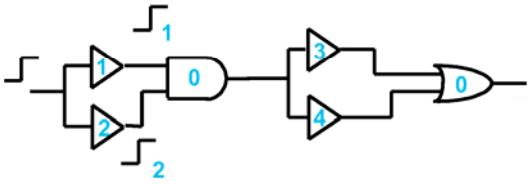
We have only considered circuit topology so far.

Longest/shortest path found on the graph can be very pessimistic:

- Paths can be FALSE
- Delay values are BOUNDS

False Paths (consider Transition Mode)

A path is false if it cannot be responsible for the delay of a circuit

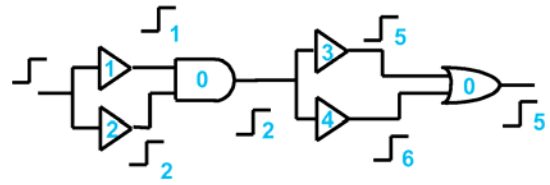


Graph model implies path of length 6

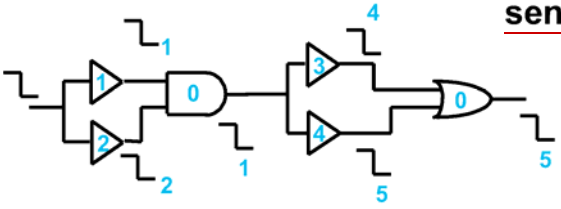
y: 22

False Paths

A path is false if it cannot be responsible for the delay of a circuit



Length of sensitized path = 5



Graph model implies path of length 6

y: 23

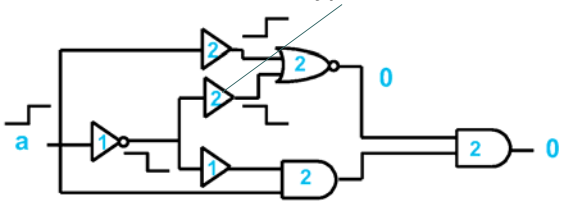
False vs. True Paths

TRUE path = one that can be responsible for the delay of a circuit

Need techniques to find whether a path is TRUE or FALSE

The Fixed Delay Model: Constant delay for each gate (or wire)

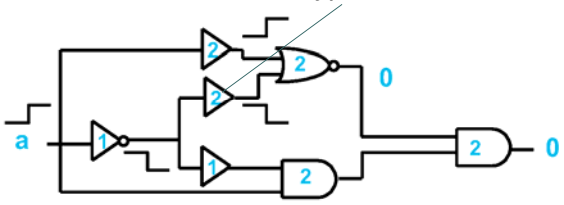
Typo: this should be **1**



Transition delay is 0, for both input transitions.

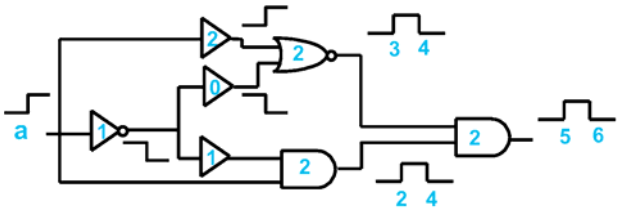
Paradoxical Behavior with the Transition Model?

Typo: this should be **1**



Transition delay is 0, for both input transitions.
Consider the "faster" circuit

non-monotonic w.r.t. time delays!



Problems with Fixed Delay + Transition Model

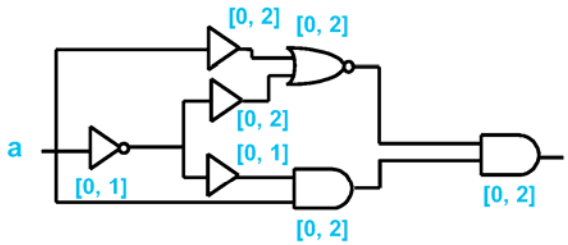
1. Transition model can be tricky to reason about
2. Fixed gate delays are unrealistic, due to manufacturing process variations

More realistic delay model: *Lower and upper bounds*

Perform timing analysis for a whole family of circuits that share the same lower/upper bounds

Fixed Delays → Bounded Delays

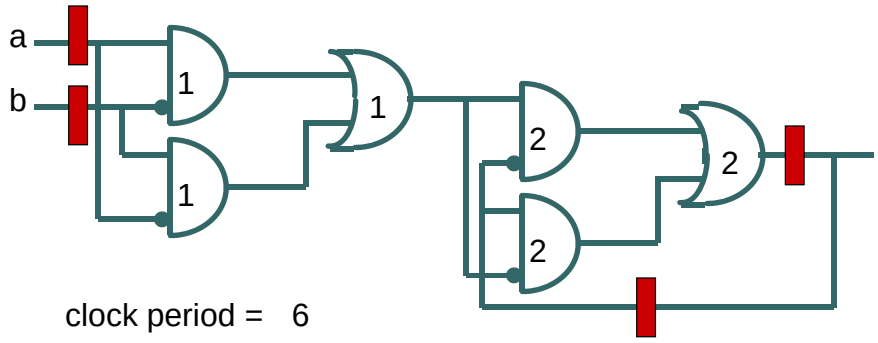
Want algorithms that report the **critical path delay** of the **slowest circuit** in the circuit family



Paper: *Computation of Floating Mode Delay in Combinational Circuits: Theory and Algorithms*, Devadas, Keutzer, Malik, 1993 (on bspace)

RETIMING

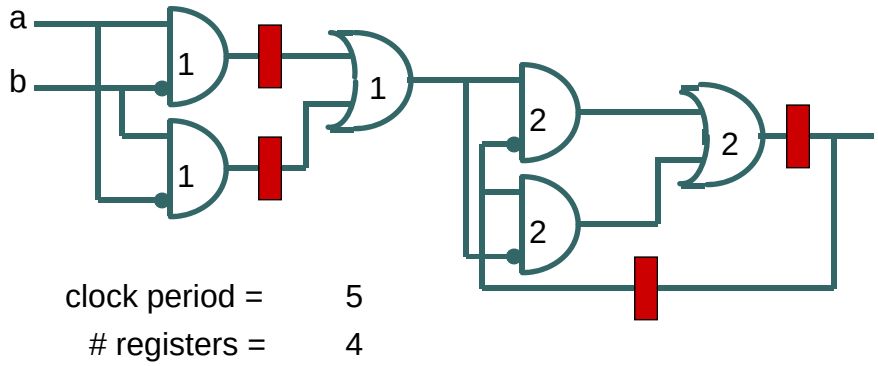
Retiming Tradeoffs



clock period = 6
registers = 4

[Shenoy, 1997]

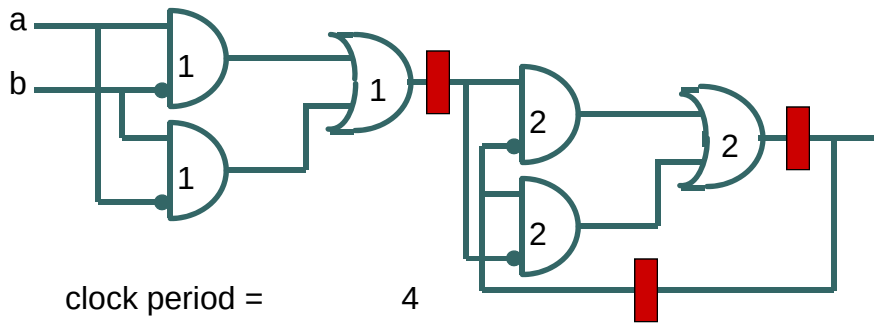
Retiming Tradeoffs



clock period = 5
registers = 4

[Shenoy, 1997]

Retiming Tradeoffs

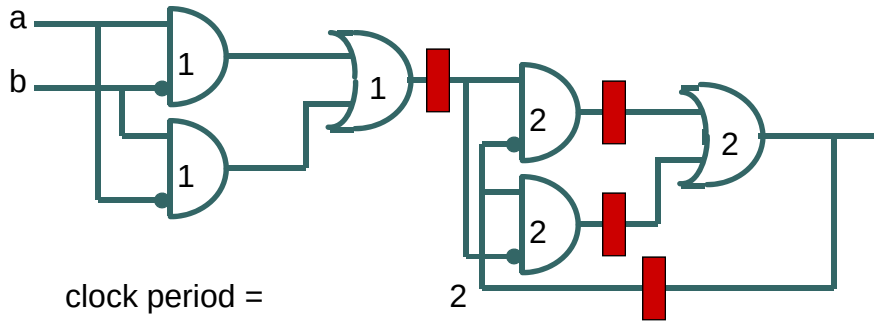


clock period = 4

registers = 3

[Shenoy, 1997]

Retiming Tradeoffs



clock period = 2

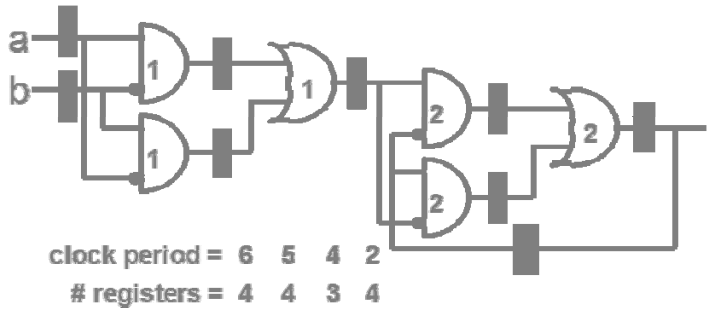
registers = 4

[Shenoy, 1997]

Goals of Retiming

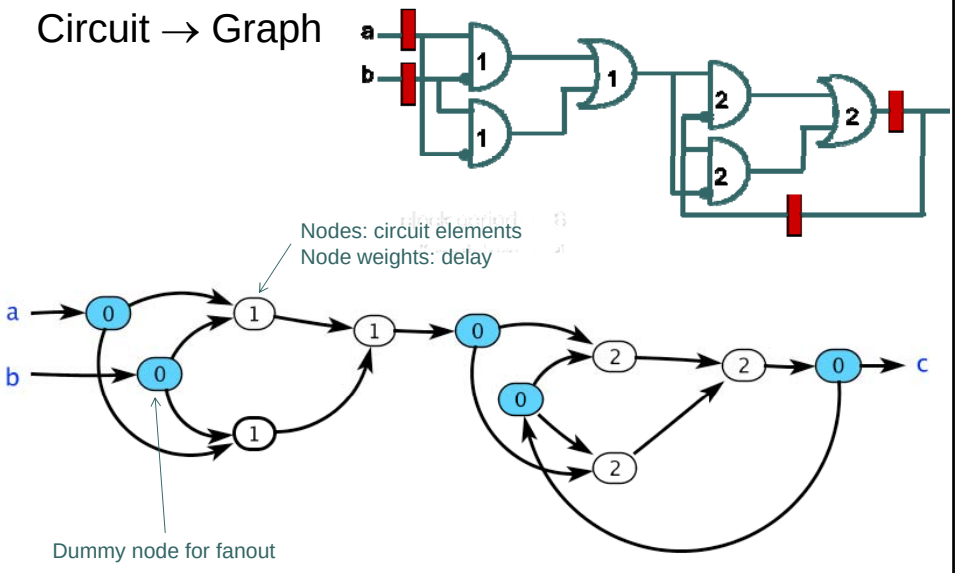
Possible goals:

- Minimize clock period (min-period retiming)
- Minimize number of registers (min-area retiming)
- Minimize number of registers for a target clock period (constrained min-area retiming)

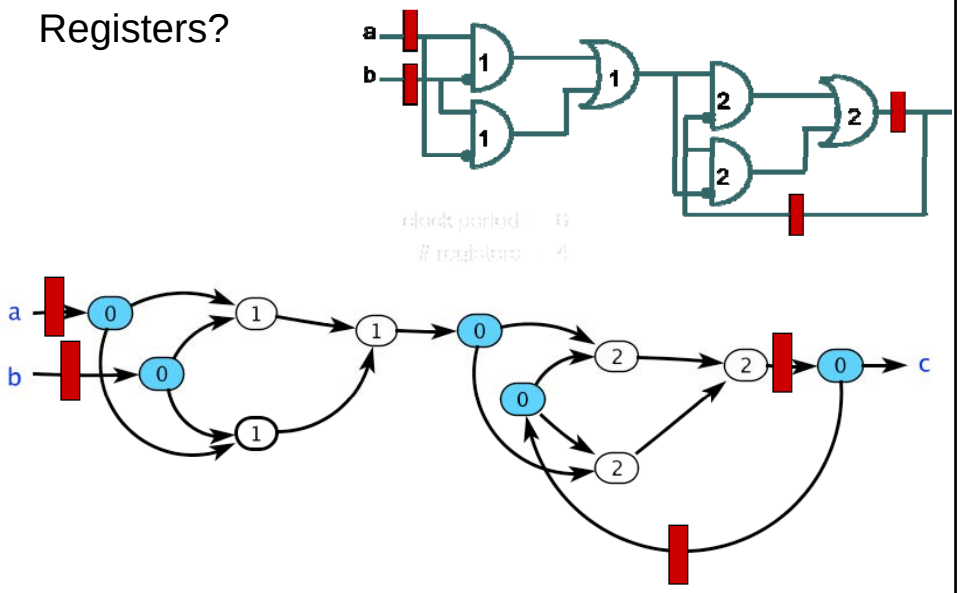


[Shenoy, 1997]

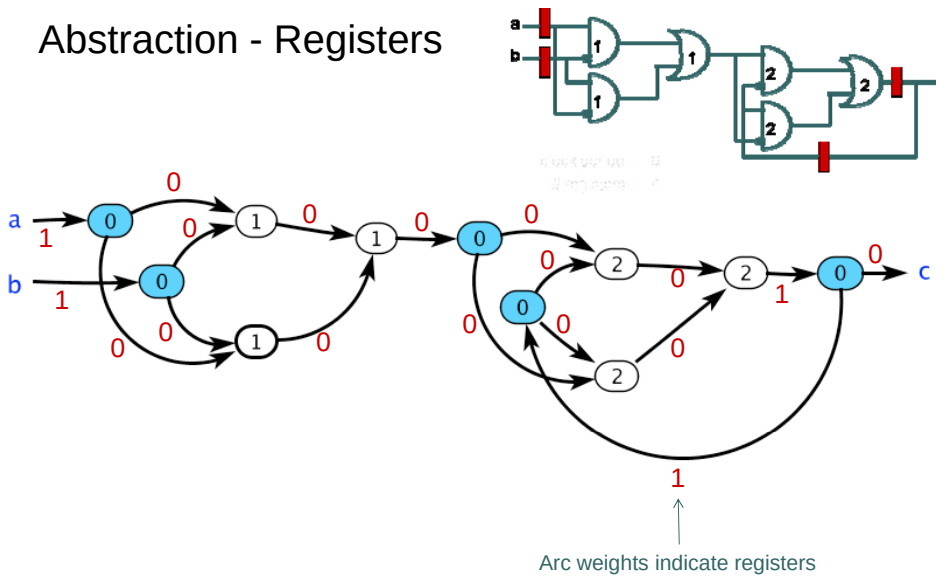
Abstraction: Circuit → Graph



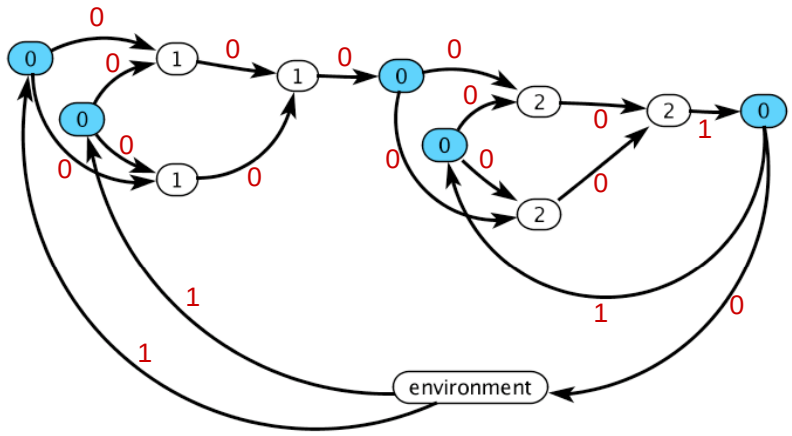
Abstraction: Registers?



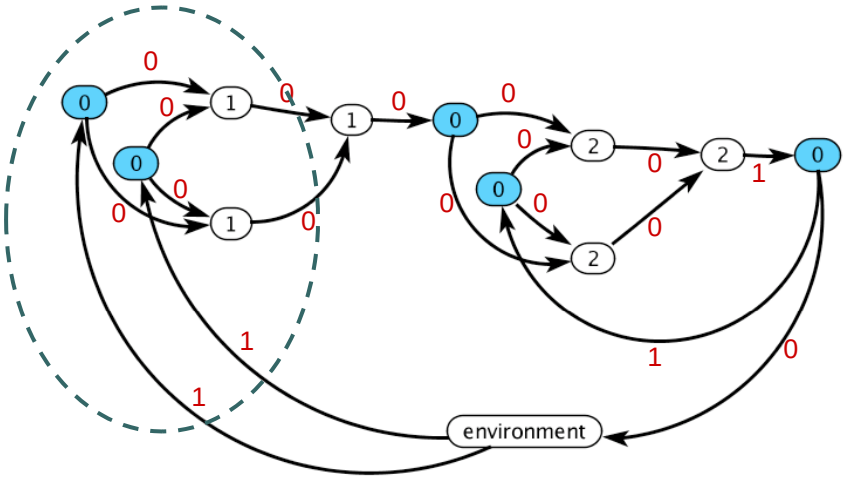
Abstraction - Registers



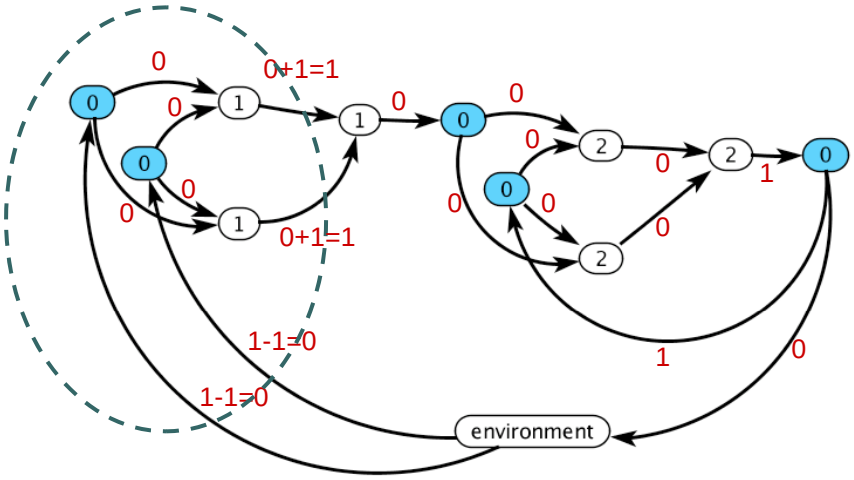
Abstraction - Environment



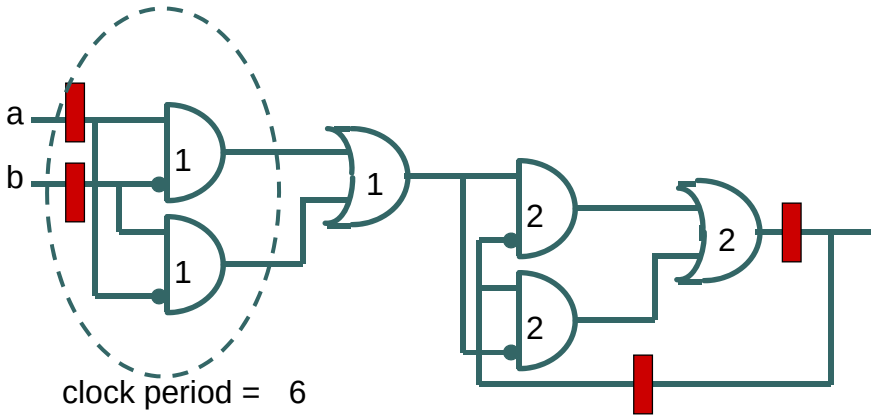
Cutset – Divides the Graph in Two



Retiming: Add a register on all arcs crossing the cutset in one direction, and subtract a register from all arcs crossing the cutset in the other direction.



Recall: Retiming Tradeoffs

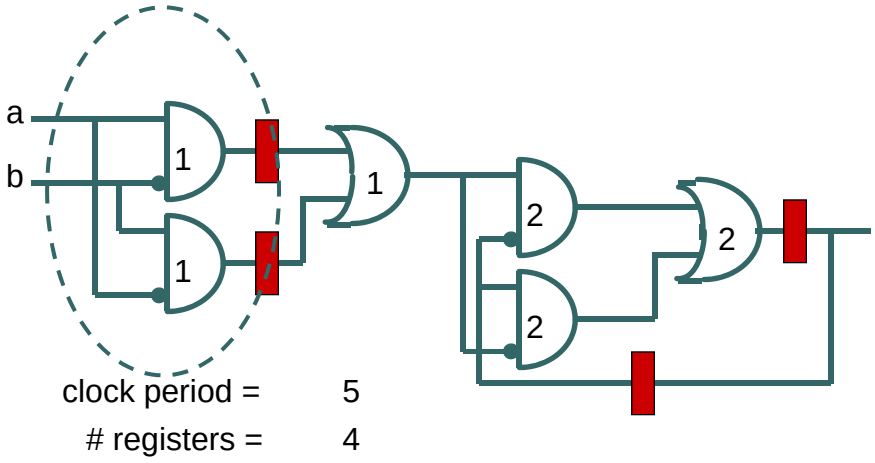


clock period = 6

registers = 4

[Shenoy, 1997]

Recall: Retiming Tradeoffs

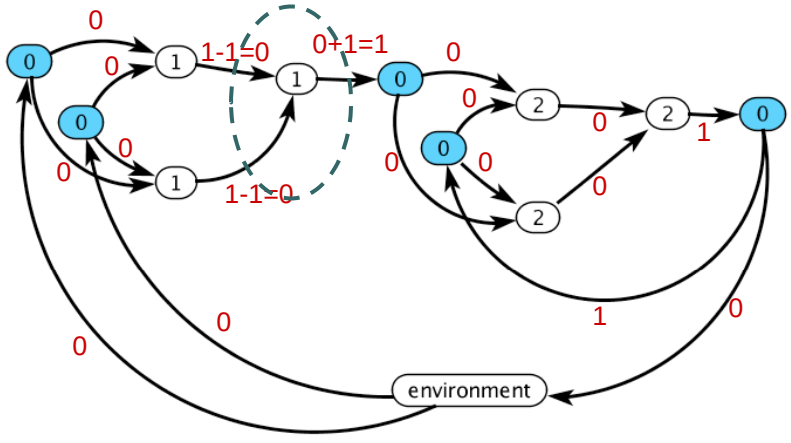


clock period = 5

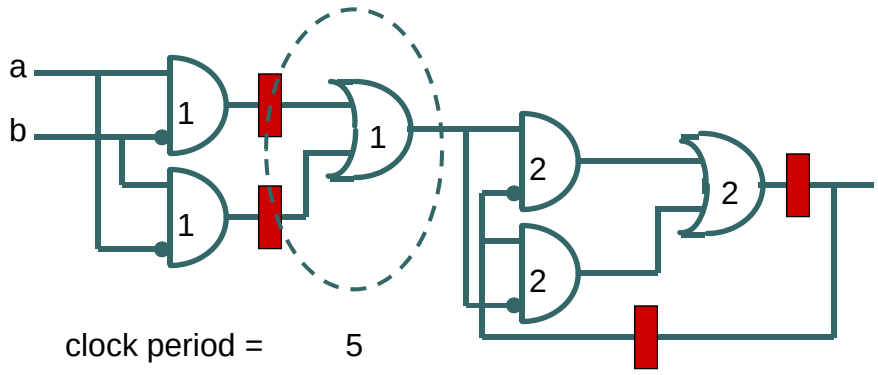
registers = 4

[Shenoy, 1997]

Simplest cutset surrounds one node



Recall: Retiming Tradeoffs

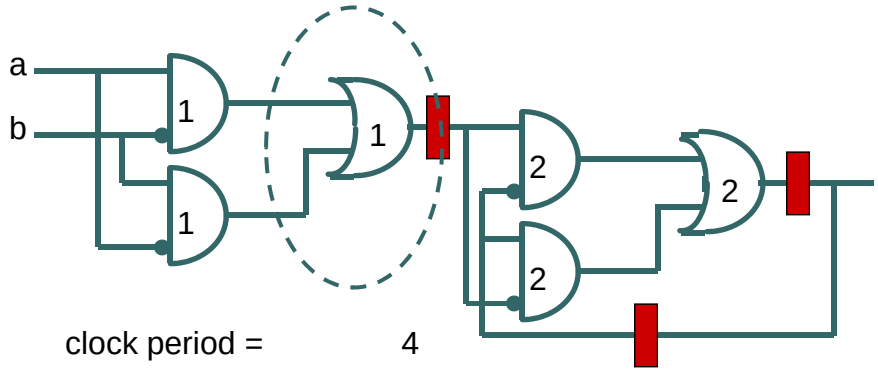


clock period = 5

registers = 4

[Shenoy, 1997]

Recall: Retiming Tradeoffs



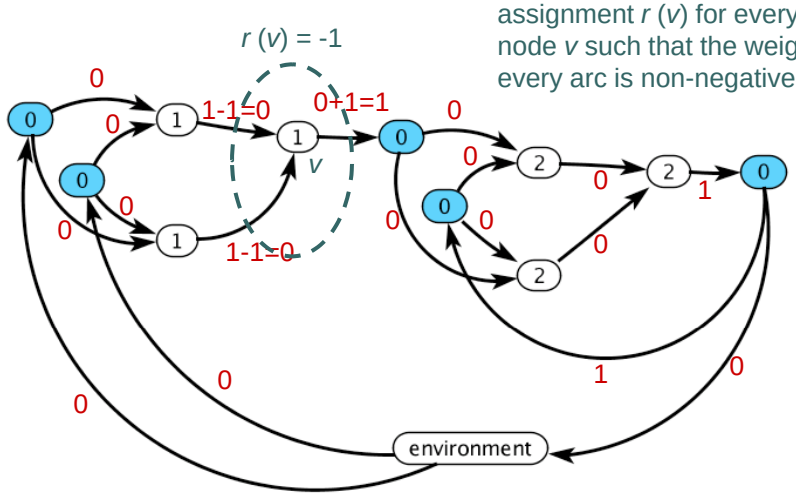
clock period = 4

registers = 3

[Shenoy, 1997]

For each node *v*, define *r* (*v*) = # of registers moved from the outputs to the inputs.

A *retiming* is now an assignment *r* (*v*) for every node *v* such that the weight of every arc is non-negative.



Rest of the story

Retiming operations = graph transformations

Each graph has a vector of costs: (clock period, # registers, …)

Formulate and solve optimization problem.

Papers (on bspace):

1. Leiserson, C. E. and J. B. Saxe (1983). "Optimizing synchronous systems." *Journal of VLSI and Computer Systems*: pp. 41-67.

2. Leiserson, C. E. and J. B. Saxe (1991). "Retiming synchronous circuitry." *Algorithmica* 6(1): pp. 5-35.

3. Shenoy, N. (1997). "Retiming: Theory and practice." *Integration, the VLSI Journal* 22: pp. 1-21.