

EDA (CS286.5b)

Day 18
Retiming

Today

- Retiming
 - cycle time (clock period)
 - C-slow
 - initial states
 - register minimization

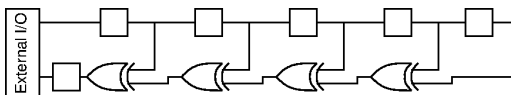
Problem

- **Given:** clocked circuit
- **Goal:** minimize clock period without changing (observable) behavior
- *i.e.* minimize maximum delay between any pair of registers
- **Freedom:** move placement of internal registers

Other Goals

- Minimize number of registers in circuit
- Achieve target cycle time
- Minimize number of registers while achieving target cycle time
- ...start talking about minimizing cycle...

Simple Example

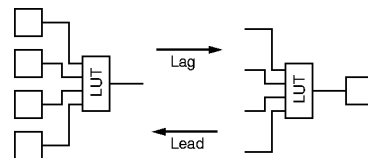


Path Length (L) = 4

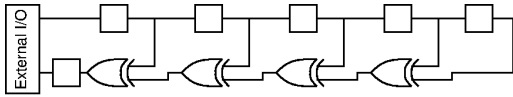
Can we do better?

Legal Register Moves

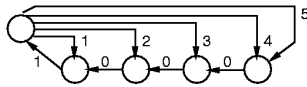
- Retiming Lag/Lead



Canonical Graph Representation



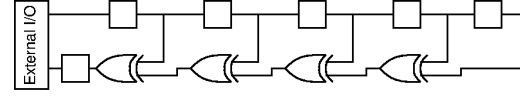
Observable I/O



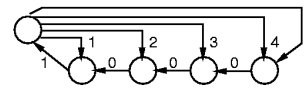
Separate arch for each path

Weight edges by number of registers
(weight nodes by delay through node)

Critical Path Length



Observable I/O

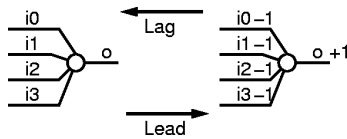


Critical Path: Length of longest path of zero weight nodes

Compute in $O(E)$ time by leveling network:

Topological sort, push path lengths forward until find register.

Retiming Lag/Lead



Retiming: Assign a lag to every vertex

$$\text{weight}(e') = \text{weight}(e) + \text{lag}(\text{head}(e)) - \text{lag}(\text{tail}(e))$$

Valid Retiming

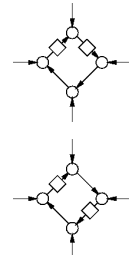
- Retiming is valid as long as:
 - $\forall e$ in graph
 - $\text{weight}(e') = \text{weight}(e) + \text{lag}(\text{head}(e)) - \text{lag}(\text{tail}(e)) \geq 0$
- Assuming original circuit was a valid synchronous circuit, this guarantees:
 - non-negative register weights on all edges
 - no travel backward in time :-)
 - all cycles have strictly positive register counts
 - propagation delay on each vertex is non-negative (assumed 1 for today)

Retiming Task

- Move registers \equiv assign lags to nodes
 - lags define all locally legal moves
- Preserving non-negative edge weights
 - (previous slide)
 - guarantees collection of lags remains consistent globally

Retiming Transformation

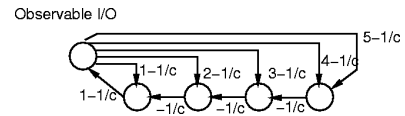
- N.B. -- unchanged by retiming
 - number of registers around a cycle
 - delay along a cycle
- Cycle of length P must have
 - at least P/c registers on it
 - to be retimeable to cycle c



Optimal Retiming

- There is a retiming of
 - graph G
 - w/ clock cycle c
 - iff $G-1/c$ has no cycles with negative edge weights
- $G-\alpha \equiv$ subtract α from each edge weight

$G-1/c$



Intuition

- Must have at most c delay between every pair of registers
- So, count $1/c$ th charge against register for every delay without out
 - (G provides credit of 1 register every time one passed)

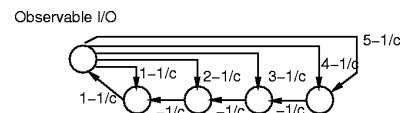
Compute Retiming

- $\text{Lag}(v) =$ shortest path to I/O in $G-1/c$
- Compute shortest paths in $O(|V||E|)$
 - Bellman-Ford
 - also use to detect negative weight cycles when c too small

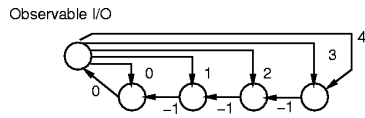
Bellman Ford

- For $k \leftarrow 0$ to N
 - $u_i \leftarrow -\infty$ (except $u_i=0$ for IO)
- For $k \leftarrow 0$ to N
 - for $e_{i,j} \in E$
 - $u_i \leftarrow \min(u_i, u_j + w(e_{i,j}))$
- for $e_{i,j} \in E$
 - if $u_i > u_j + w(e_{i,j})$
 - cycles detected

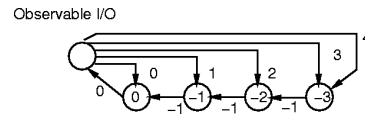
Apply to Example



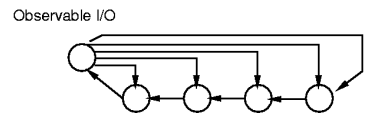
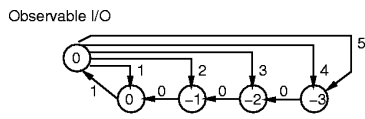
Apply: Find Lags



Apply: Lags

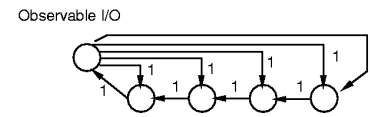
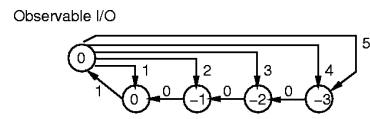


Apply: Move Registers

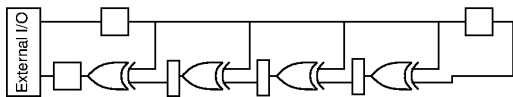
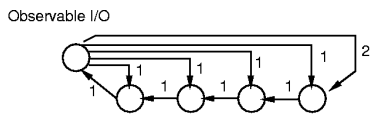


$$\text{weight}(e') = \text{weight}(e) + \text{lag}(\text{head}(e)) - \text{lag}(\text{tail}(e))$$

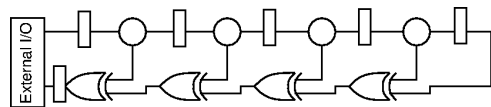
Apply: Retimed

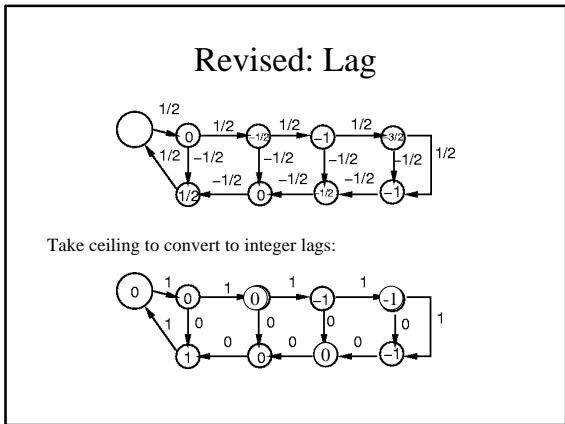
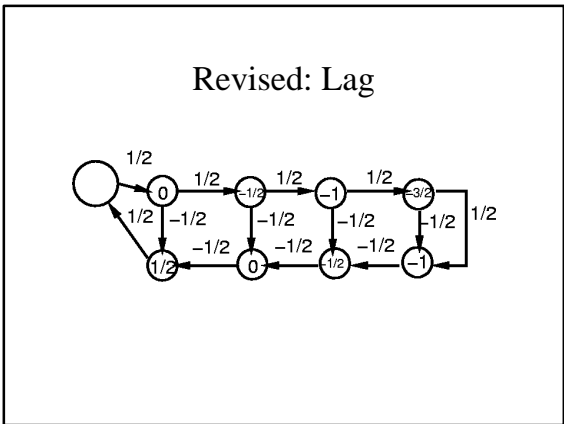
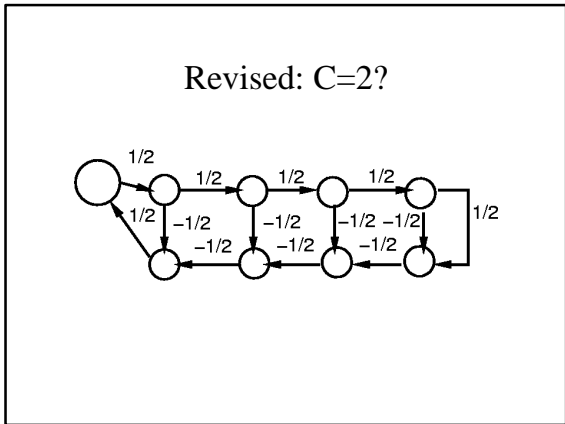
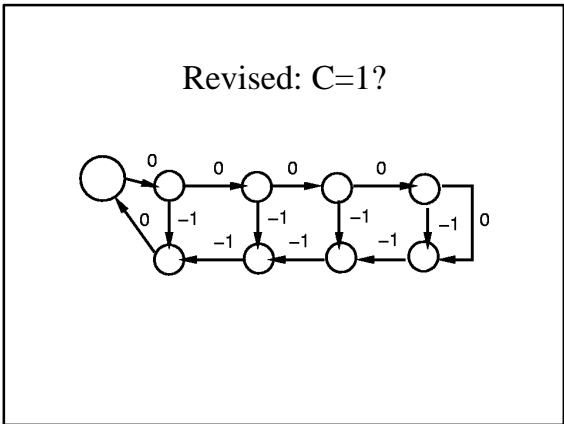
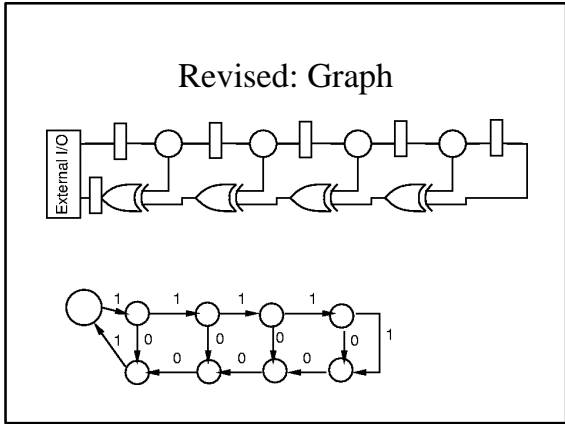
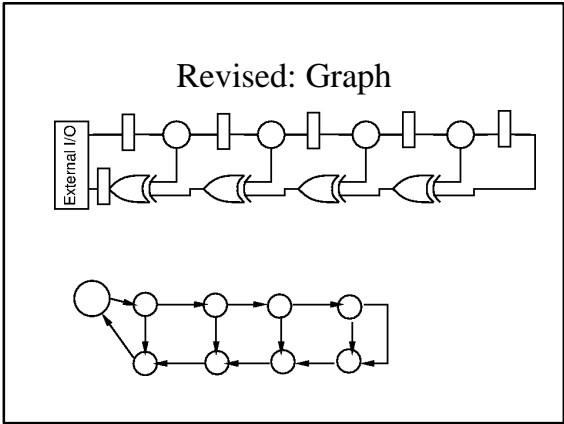


Apply: Retimed Design

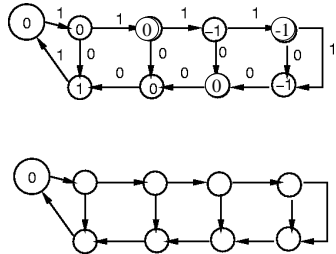


Revise Example (fanout delay)

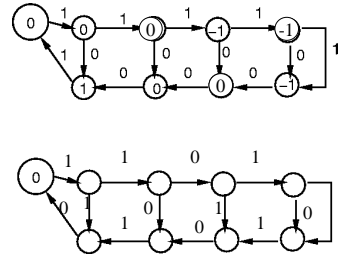




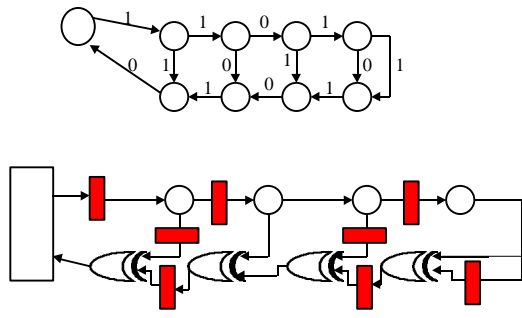
Revised: Apply Lag



Revised: Apply Lag



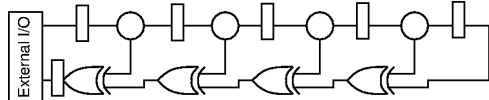
Revised: Retimed



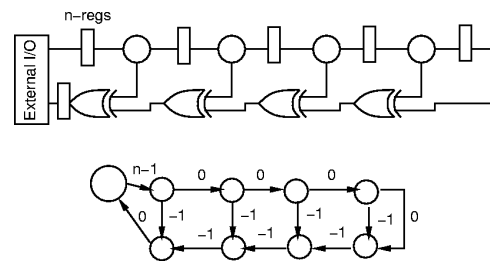
Pipelining

- Can use this retiming to pipeline
- Assume have enough (infinite supply) of registers at edge of circuit
- Retime them into circuit

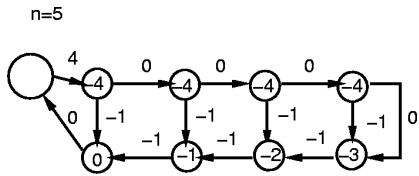
$C > 1 \implies$ Pipeline



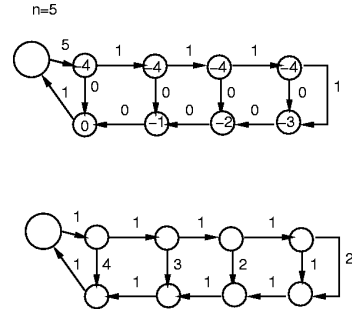
Add Registers



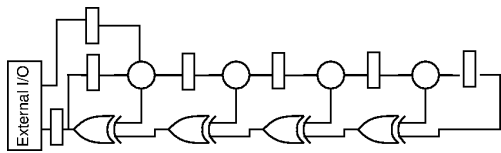
Pipeline Retiming: Lag



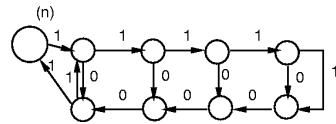
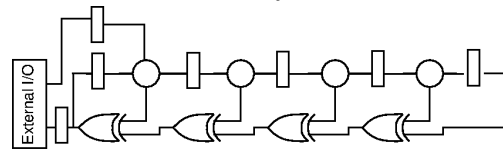
Pipelined Retimed



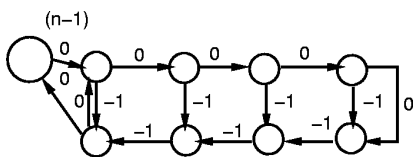
Real Cycle



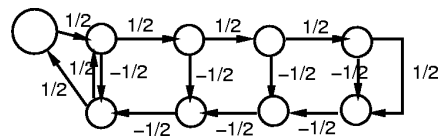
Real Cycle

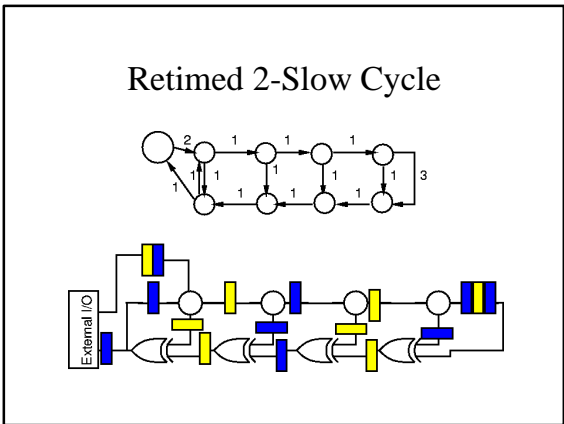
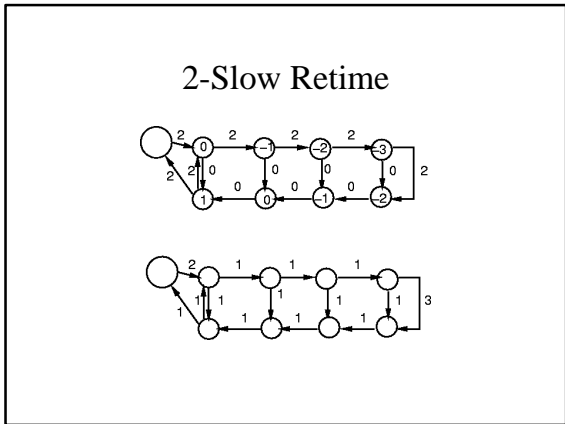
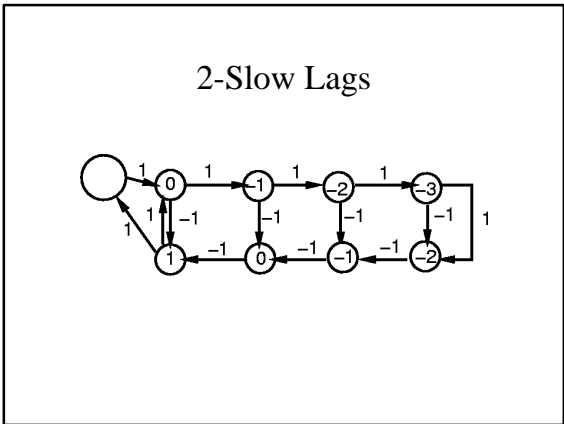
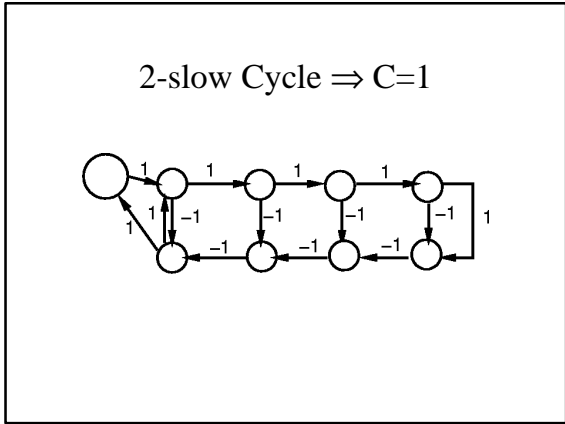
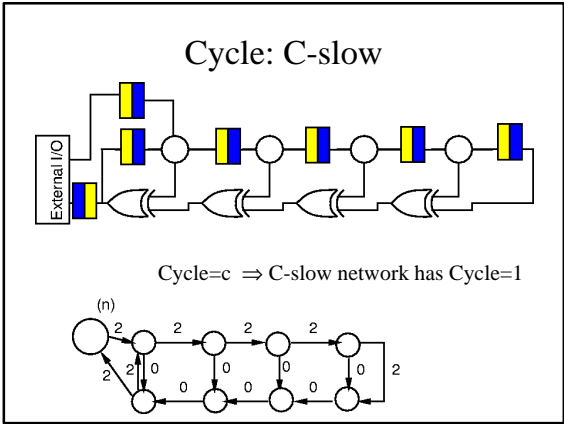


Cycle C=1?



Cycle C=2?





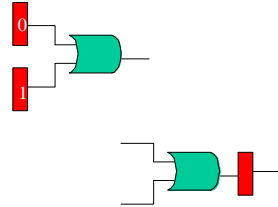
- ### C-Slow applicable?
- Available parallelism
 - solve C identical, independent problems
 - e.g. process packets (blocks) separately
 - e.g. independent regions in images
 - Commutative operators
 - e.g. max example

Note

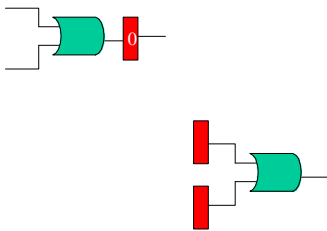
- Algorithm/examples shown
 - for special case of unit-delay nodes
- For general delay, still polynomial, but a bit more complicated
 - (but still polynomial)

Initial State

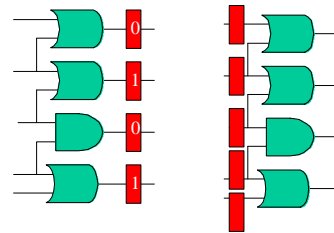
- What about initial state?



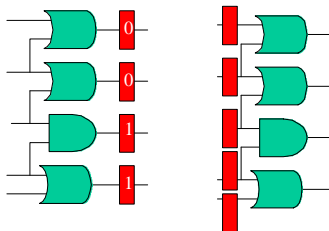
Initial State



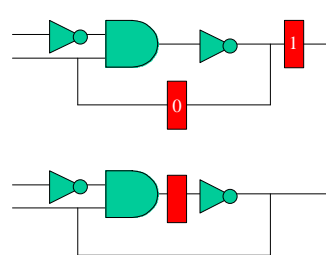
Initial State



Initial State



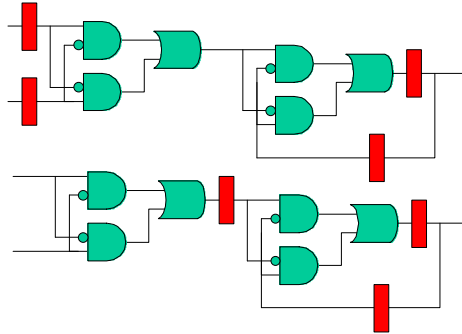
Initial State



Initial State

- Cannot always get exactly the same initial state behavior on the retimed circuit
 - without additional care in the retiming transformation
 - sometimes have to modify structure of retiming to preserve initial behavior
- Only a problem for startup transient
 - if circuit you're willing to clock to get into initial state, not a limitation

Minimize Registers



Minimize Registers

- Number of registers: $\sum w(e)$
- After retime: $\sum w(e) + \sum (FI(v) - FO(v)) \text{lag}(v)$
- delta only in lags
- So want to minimize: $\sum (FI(v) - FO(v)) \text{lag}(v)$
 - subject to earlier constraints
 - non-negative register weights, delays
 - positive cycle counts

Minimize Registers

- Can be formulated as flow problem
- Can add cycle time constraints to flow problem
- Time: $O(|V||E| \log(|V|) \log(|V|^2/|E|))$

Summary

- Can move registers to minimize cycle time
- Formulate as a lag assignment to every node
- Optimally solve cycle time in $O(|V||E|)$ time
- Also
 - Compute multithreaded computations
 - Minimize registers
- Watch out for initial values

Today's Big Ideas

- Exploit freedom
- Formulate transformations (lag assignment)
- Express legality constraints
- Technique:
 - graph algorithms
 - network flow