ESE535: Electronic Design Automation

Day 8: February 13, 2008
Retiming

Today

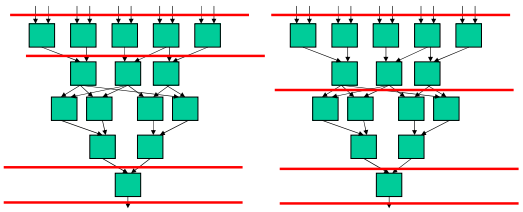
- Retiming
  - Cycle time (clock period)
  - C-slow
  - Initial states
  - Register minimization

Task

- Move registers to:
  - Preserve semantics
  - Minimize path length between registers
    - Maximize reuse rate
  - …while minimizing number of registers required

Example: Same Semantics

- Externally: no observable difference



Problem

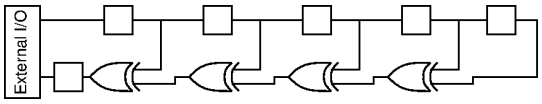
- **Given**: clocked circuit
- **Goal**: minimize clock period without changing (observable) behavior
- *I.e.* minimize maximum delay between any pair of registers
- **Freedom**: move placement of internal registers

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

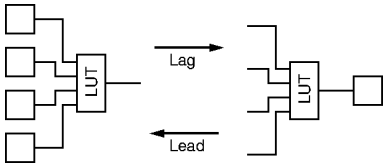


Path Length (L) = 4

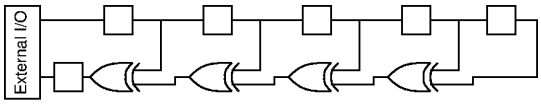
Can we do better?

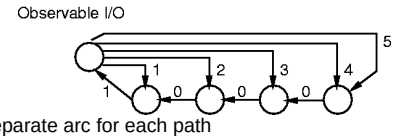
Legal Register Moves

- Retiming Lag/Lead



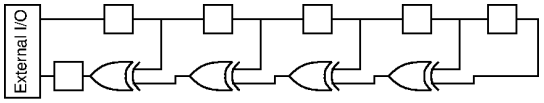
Canonical Graph Representation

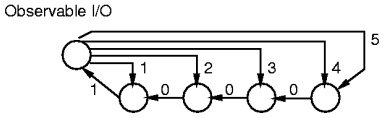




Separate arc for each path
Weight edges by number of registers
(weight nodes by delay through node)

Critical Path Length



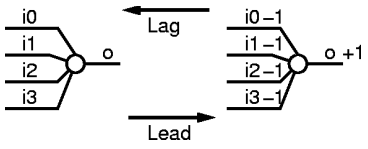


Critical Path: Length of longest path of zero weight nodes

Compute in O(|E|) time by levelizing network:

Topological sort, push path lengths forward until find register.

Retiming Lag/Lead



Retiming: Assign a lag to every vertex

weight(e') = weight(e) + lag(head(e))-lag(tail(e))

Valid Retiming

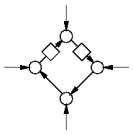
- Retiming is valid as long as:
  - ∀e in graph
    - weight(e') = weight(e) + lag(head(e))-lag(tail(e)) ≥ 0
- Assuming original circuit was a valid synchronous circuit, this guarantees:
  - non-negative register weights on all edges
    - no travel backward in time :-)
  - all cycles have strictly positive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

Retiming Task

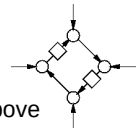
- Move registers ≡ assign lags to nodes
  - lags define all locally legal moves
- Preserving non-negative edge weights
  - (previous slide)
  - guarantees collection of lags remains consistent globally

Retiming Transformation

- Properties invariant to retiming
  1. number of registers around a cycle
  2. delay along a cycle



- Cycle of length *P* must have
  - at least *P/c* registers on it to be retimeable to cycle *c*
  - Can be computed from invariant above



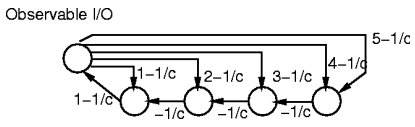
Optimal Retiming

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

1/c Intuition

- Want to place a register every c delay units
- Each register adds one
- Each delay subtracts 1/c
- As long as remains more positives than negatives around all cycles
  - can move registers to accommodate
  - Captures the regs=P/c constraints

G-1/*c*



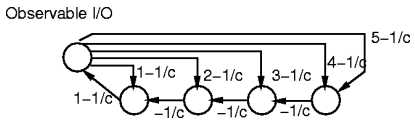
Compute Retiming

- 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

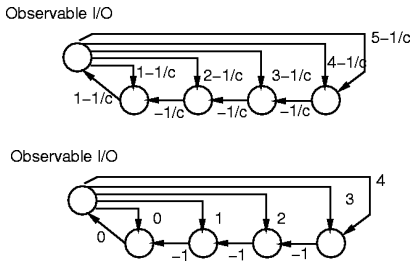
Bellman Ford

- For I←0 to N
  - $u_i \leftarrow \infty$ (except u_i=0 for IO)
- For k←0 to N
  - for $e_{i,j} \in E$
    - $u_i \leftarrow \min(u_i, u_j+w(e_{i,j}))$
- For $e_{i,j} \in E$ *//still update→negative cycle*
  - if $u_i > u_j+w(e_{i,j})$
    - cycles detected

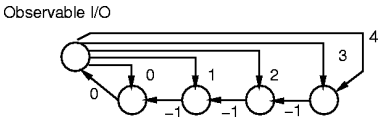
Apply to Example



Try c=1

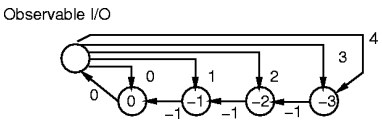


Apply: Find Lags

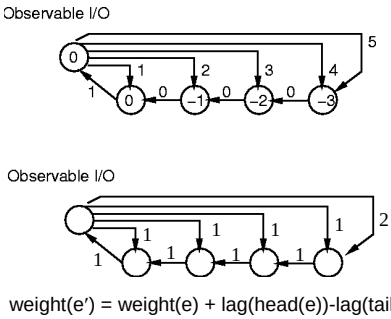


Negative weight cycles?
Shortest paths?

Apply: Lags

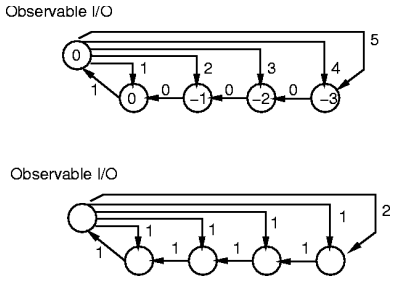


Apply: Move Registers

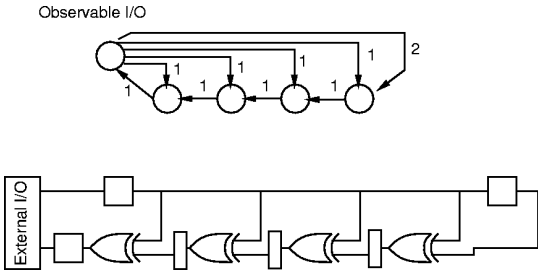


weight(e') = weight(e) + lag(head(e))-lag(tail(e))

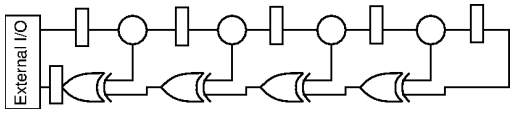
Apply: Retimed



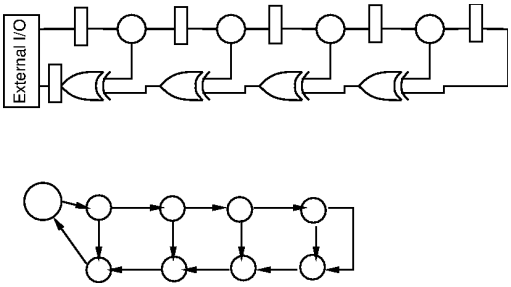
Apply: Retimed Design



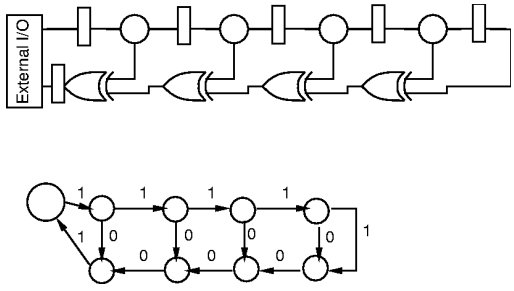
Revise Example (fanout delay)



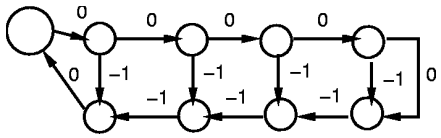
Revised: Graph



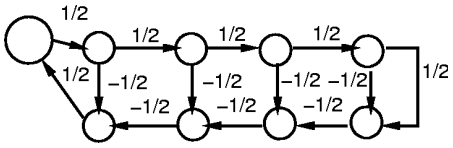
Revised: Graph



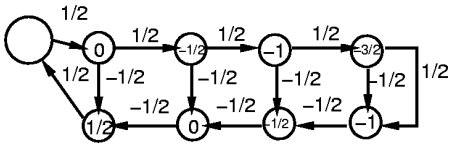
Revised: C=1?



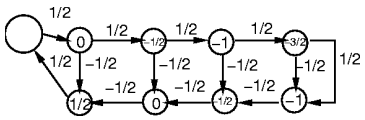
Revised: C=2?



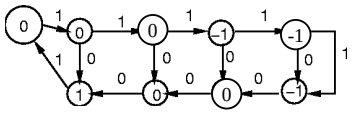
Revised: Lag



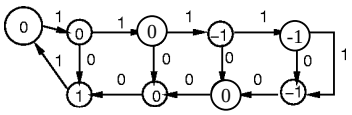
Revised: Lag

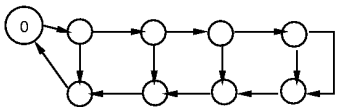


Take ceiling to convert to integer lags:

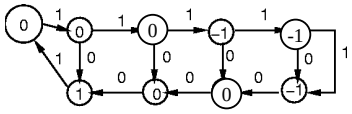


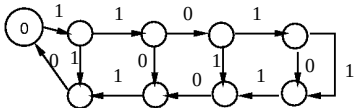
Revised: Apply Lag



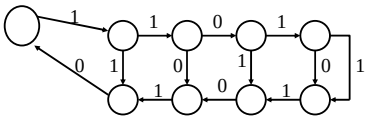


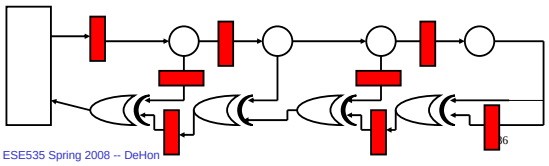
Revised: Apply Lag





Revised: Retimed

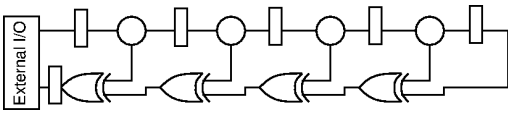




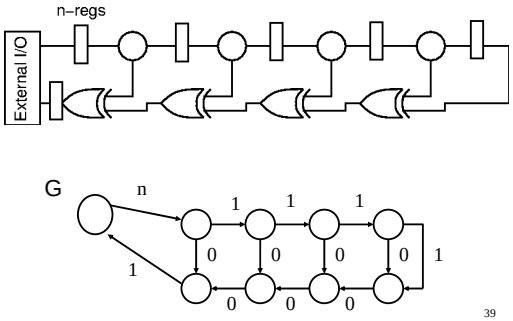
Pipelining

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

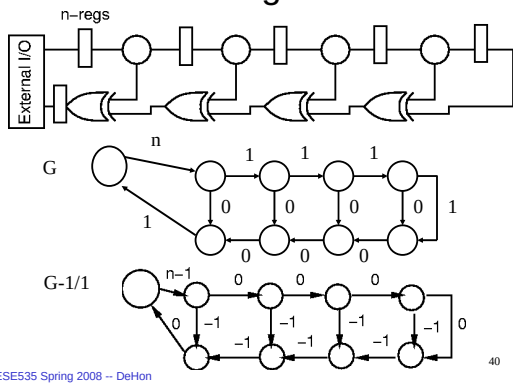
C>1 ==> Pipeline



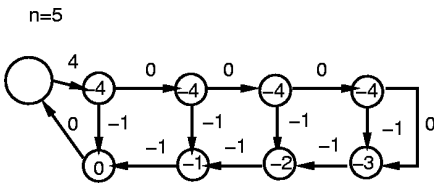
Add Registers



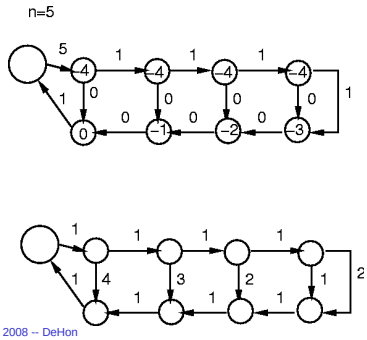
Add Registers



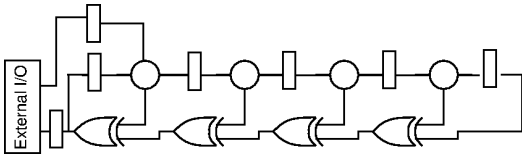
Pipeline Retiming: Lag



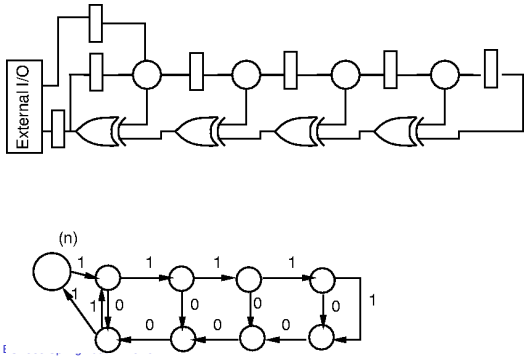
Pipelined Retimed



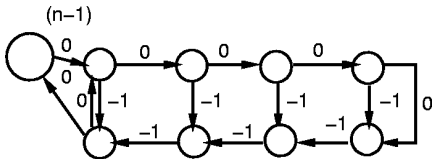
Real Cycle



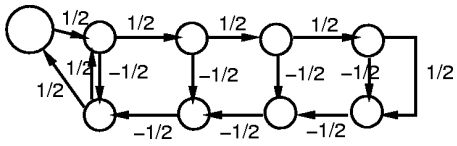
Real Cycle



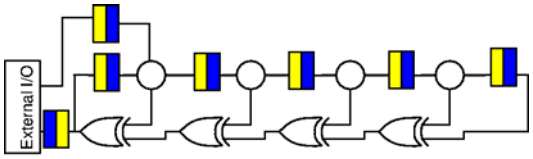
Cycle C=1?



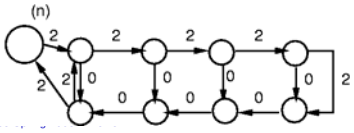
Cycle C=2?



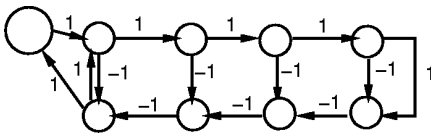
Cycle: C-slow



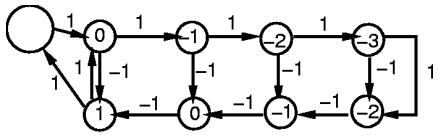
Cycle=c $\Rightarrow$ C-slow network has Cycle=1



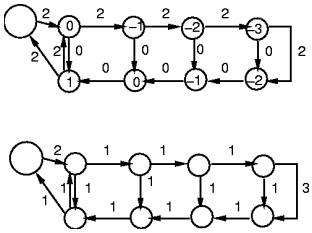
2-slow Cycle $\Rightarrow$ C=1



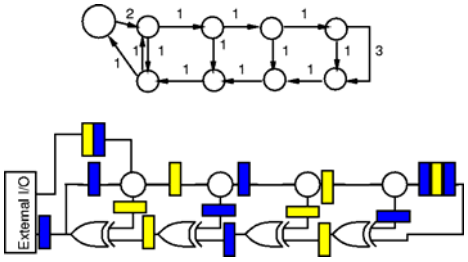
2-Slow Lags



2-Slow Retime



Retimed 2-Slow Cycle

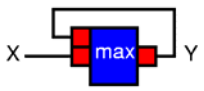


C-Slow applicable?

- 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

Max Example

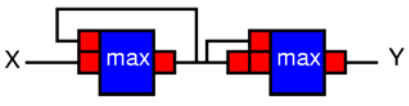
2–Slow design:



X2 X2 X1 X1 X0 X0 → Y2 ? Y1 ? Y0 ?

B2 A2 B1 A1 B0 A0 → YA2 YB1 YA1 YB0 YA0 ?

Max Example



Computes two interleaved streams: even max, odd max

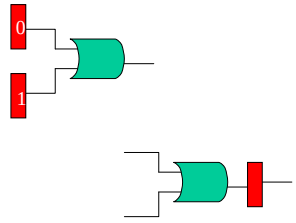
Computes final max of even and odd pairs

Note

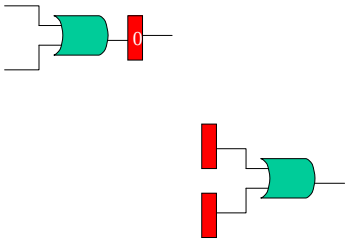
- Algorithm/examples shown
  - for special case of unit-delay nodes
- For general delay,
  - a bit more complicated
  - still polynomial

Initial State

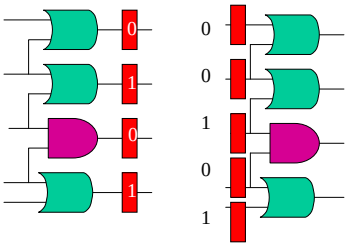
- What about initial state?



Initial State

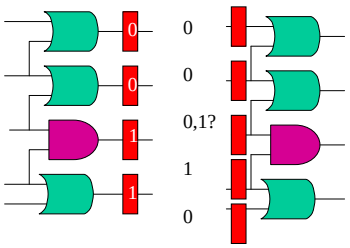


Initial State

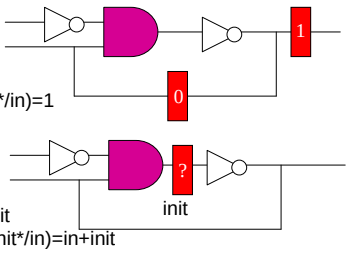


In general, constraints → satisfiable?

Initial State



Initial State



Cycle1: 1
Cycle2: /(0*/in)=1

Cycle1: /init
Cycle2: /(/init*/in)=in+init

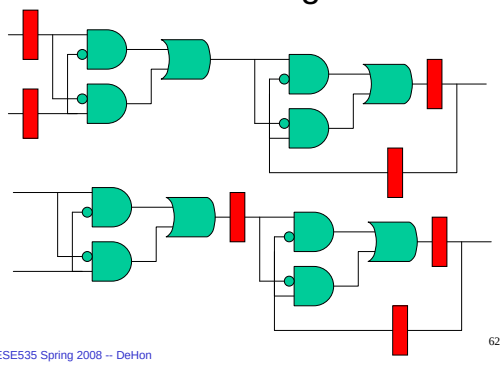
init=0
Cycle1: 1
Cycle2: /(/init*/in)=in

init=1
Cycle1: 0
Cycle2: /(/init*/in)=1

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 you're willing to clock to get into initial state, not a limitation

Minimize Registers



Minimize Registers

- Number of registers: Σ w(e)
- After retime: Σ w(e)+Σ (FI(v)-FO(v))lag(v)
- delta only in lags
- So want to minimize: Σ (FI(v)-FO(v))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

Admin

- Homework #2 due Monday
- Reading for Monday on web

Big Ideas

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