

CS137: Electronic Design Automation

Day 7: February 3, 2002
Retiming



CALTECH CS137 Winter2004 -- DeHon

Today

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

CALTECH CS137 Winter2004 -- DeHon

Task

- Move registers to:
 - Preserve semantics
 - Minimize path length between registers
 - (make path length 1 for maximum throughput or reuse)
 - Maximize reuse rate
 - ...while minimizing number of registers required

CALTECH CS137 Winter2004 -- DeHon

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

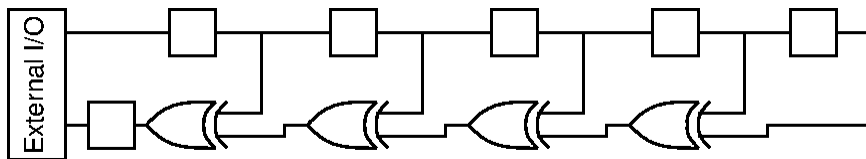
CALTECH CS137 Winter2004 -- DeHon

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...

CALTECH CS137 Winter2004 -- DeHon

Simple Example



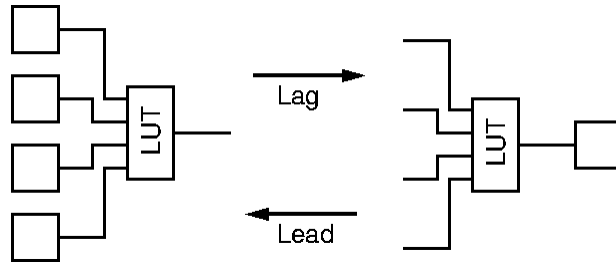
Path Length (L) = 4

Can we do better?

CALTECH CS137 Winter2004 -- DeHon

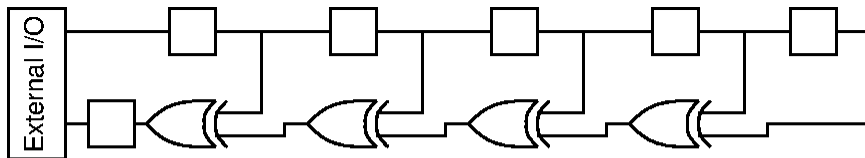
Legal Register Moves

- Retiming Lag/Lead

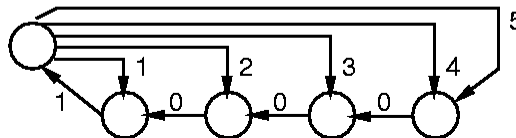


CALTECH CS137 Winter2004 -- DeHon

Canonical Graph Representation



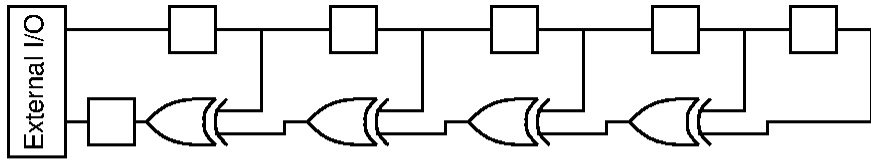
Observable I/O



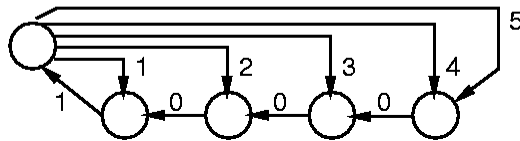
Separate arc for each path
Weight edges by number of registers
(weight nodes by delay through node)

CALTECH CS137 Winter2004 -- DeHon

Critical Path Length



Observable I/O



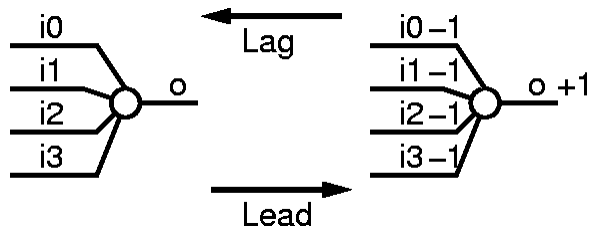
Critical Path: Length of longest path of zero weight nodes

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

Topological sort, push path lengths forward until find register.

CALTECH CS137 Winter2004 -- DeHon

Retiming Lag/Lead



Retiming: Assign a lag to every vertex

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

CALTECH CS137 Winter2004 -- DeHon

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)

CALTECH CS137 Winter2004 -- DeHon

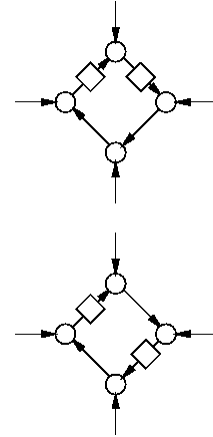
Retiming Task

- 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

CALTECH CS137 Winter2004 -- DeHon

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



CALTECH CS137 Winter2004 -- DeHon

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

CALTECH CS137 Winter2004 -- DeHon

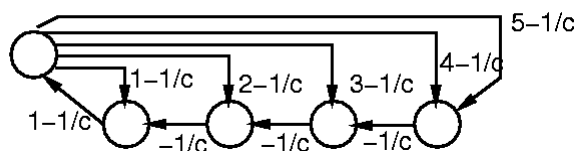
1/c Intuition

- 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 $\text{regs} = P/c$ constraints

CALTECH CS137 Winter2004 -- DeHon

$G - 1/c$

Observable I/O



CALTECH CS137 Winter2004 -- DeHon

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

CALTECH CS137 Winter2004 -- DeHon

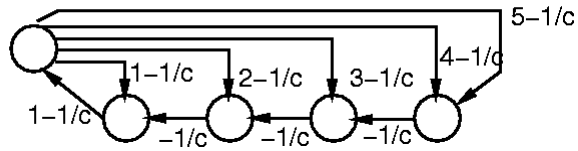
Bellman Ford

- For $l \leftarrow 0$ 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 $e_{i,j} \in E$ *//still update \rightarrow negative cycle*
 - if $u_i > u_j + w(e_{i,j})$
 - cycles detected

CALTECH CS137 Winter2004 -- DeHon

Apply to Example

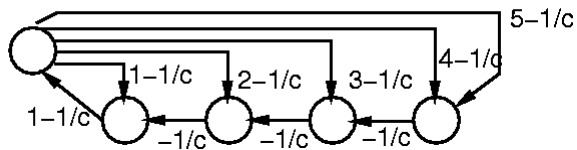
Observable I/O



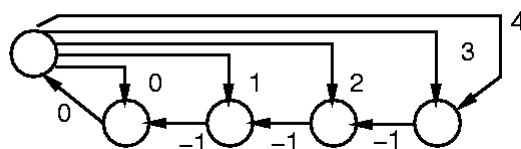
CALTECH CS137 Winter2004 -- DeHon

Try $c=1$

Observable I/O



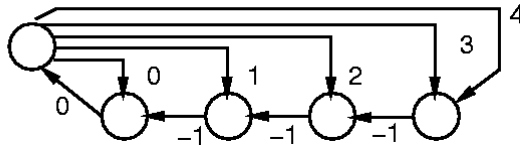
Observable I/O



CALTECH CS137 Winter2004 -- DeHon

Apply: Find Lags

Observable I/O

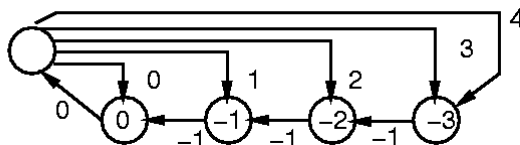


Negative weight cycles?
Shortest paths?

CALTECH CS137 Winter2004 -- DeHon

Apply: Lags

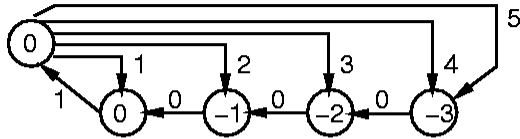
Observable I/O



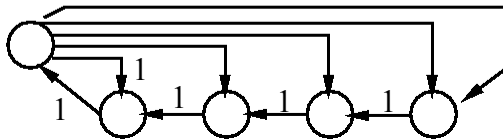
CALTECH CS137 Winter2004 -- DeHon

Apply: Move Registers

Observable I/O



Observable I/O

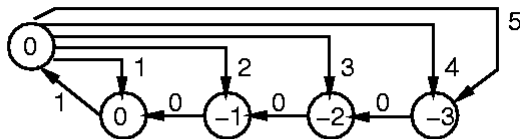


$$\text{weight}(e') = \text{weight}(e) + \text{lag}(\text{head}(e)) - \text{lag}(\text{tail}(e))$$

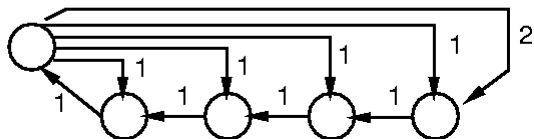
CALTECH CS137 Winter2004 -- DeHon

Apply: Retimed

Observable I/O



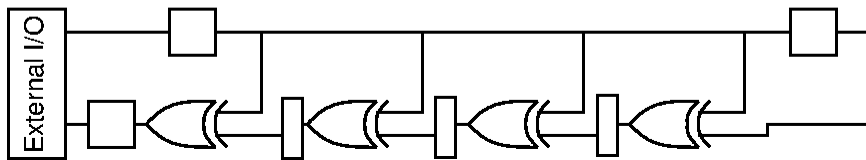
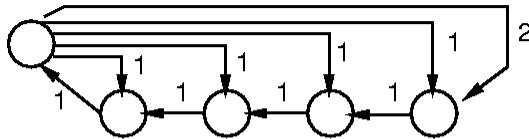
Observable I/O



CALTECH CS137 Winter2004 -- DeHon

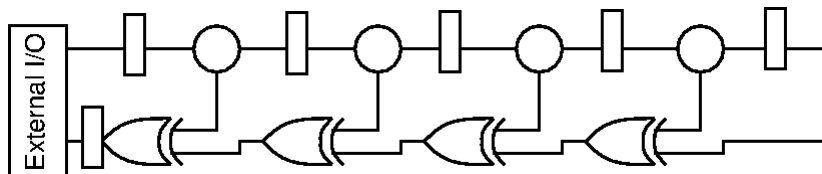
Apply: Retimed Design

Observable I/O



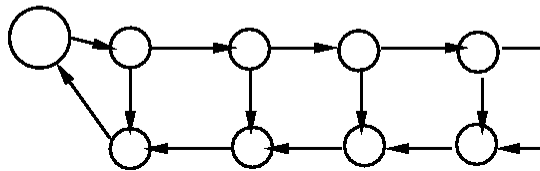
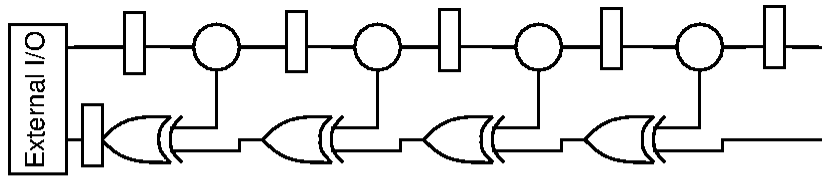
CALTECH CS137 Winter2004 -- DeHon

Revise Example (fanout delay)



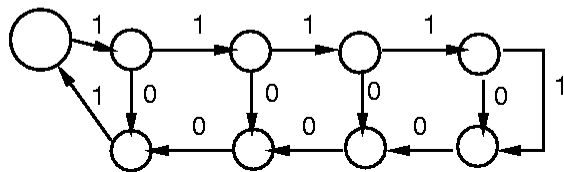
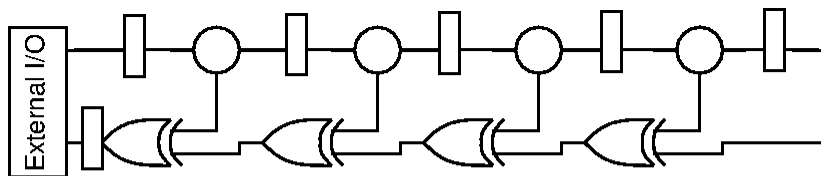
CALTECH CS137 Winter2004 -- DeHon

Revised: Graph



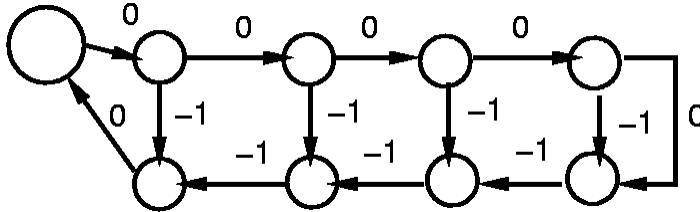
CALTECH CS137 Winter2004 -- DeHon

Revised: Graph



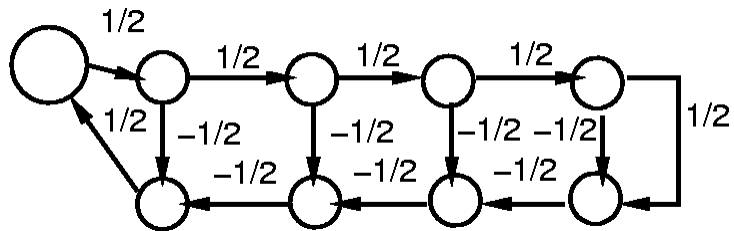
CALTECH CS137 Winter2004 -- DeHon

Revised: $C=1?$



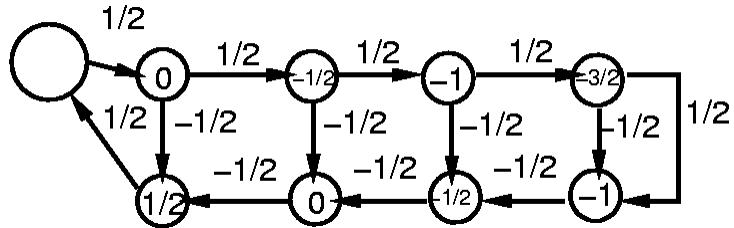
CALTECH CS137 Winter2004 -- DeHon

Revised: $C=2?$



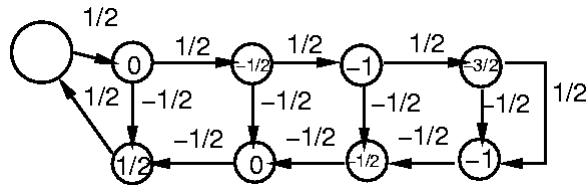
CALTECH CS137 Winter2004 -- DeHon

Revised: Lag

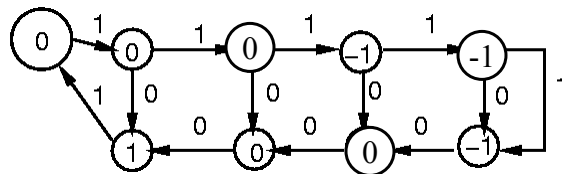


CALTECH CS137 Winter2004 -- DeHon

Revised: Lag

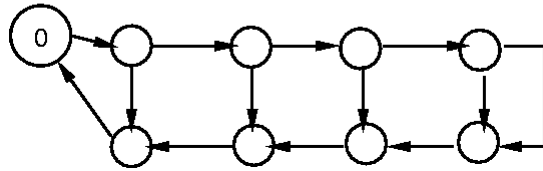
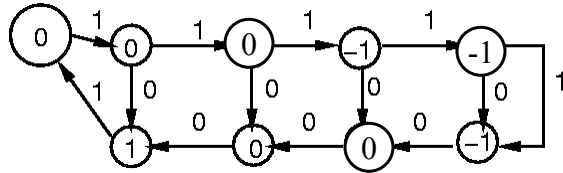


Take ceiling to convert to integer lags:



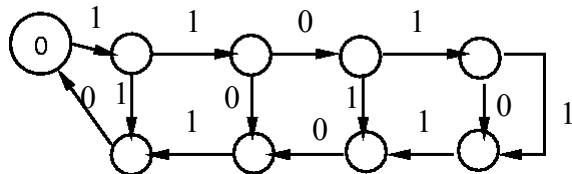
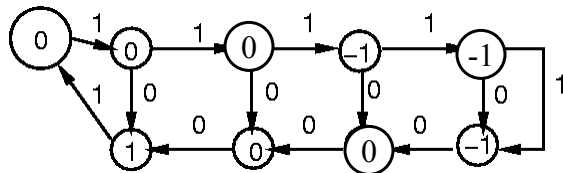
CALTECH CS137 Winter2004 -- DeHon

Revised: Apply Lag



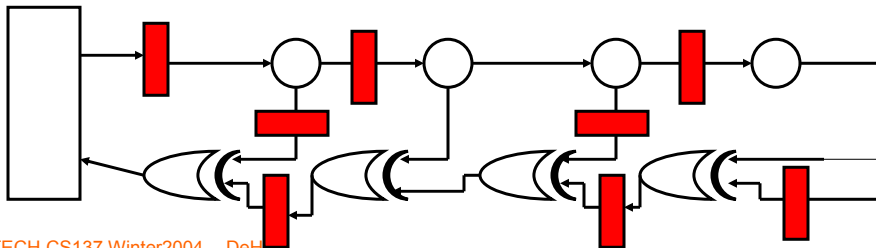
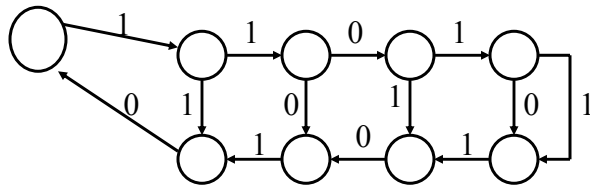
CALTECH CS137 Winter2004 -- DeHon

Revised: Apply Lag



CALTECH CS137 Winter2004 -- DeHon

Revised: Retimed



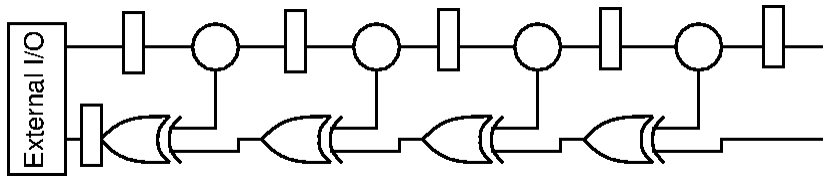
CALTECH CS137 Winter2004 -- DeHon

Pipelining

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

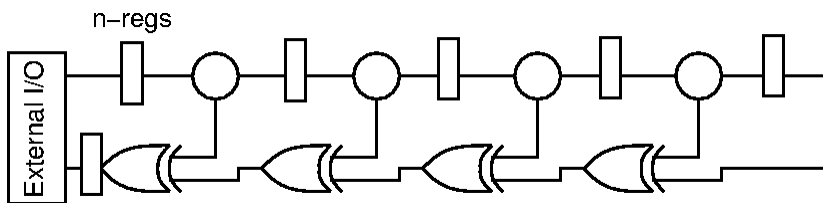
CALTECH CS137 Winter2004 -- DeHon

C > 1 ==> Pipeline

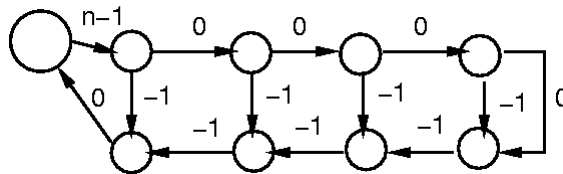


CALTECH CS137 Winter2004 -- DeHon

Add Registers



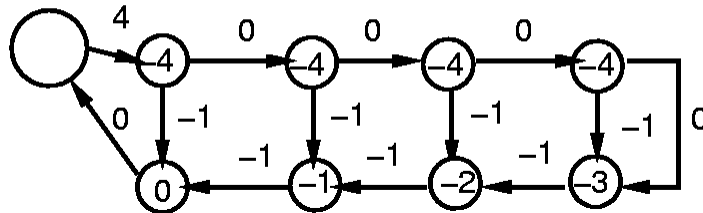
Note: jump to G-1 graph \rightarrow future...show G first



CALTECH CS137 Winter2004 -- DeHon

Pipeline Retiming: Lag

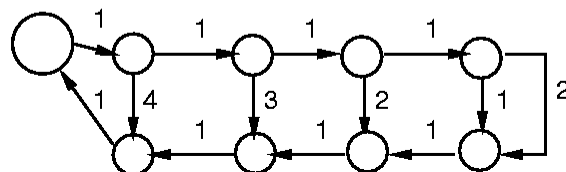
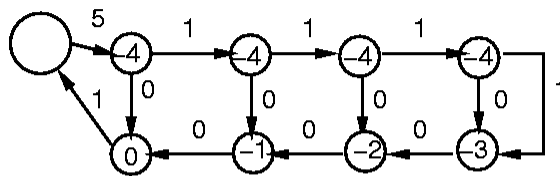
n=5



CALTECH CS137 Winter2004 -- DeHon

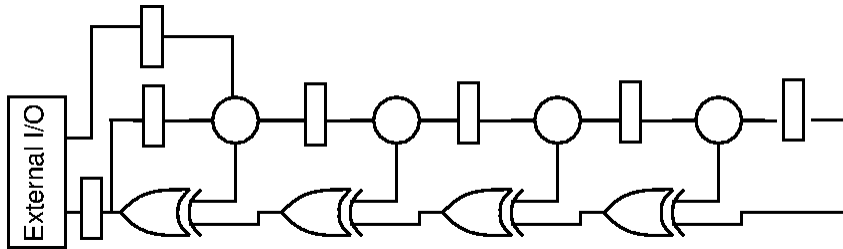
Pipelined Retimed

n=5



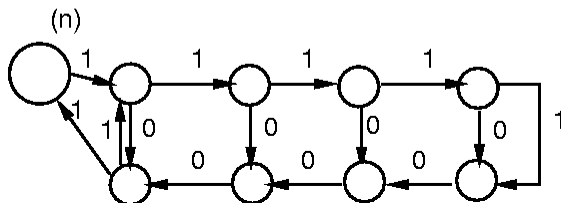
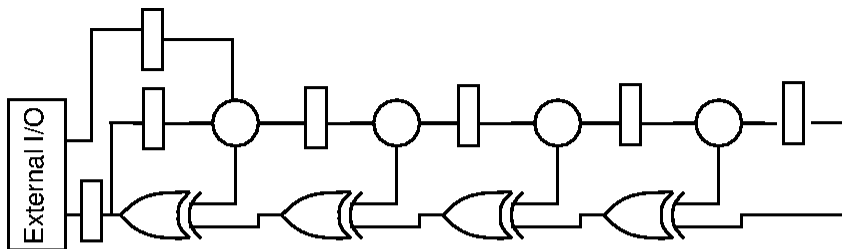
CALTECH CS137 Winter2004 -- DeHon

Real Cycle



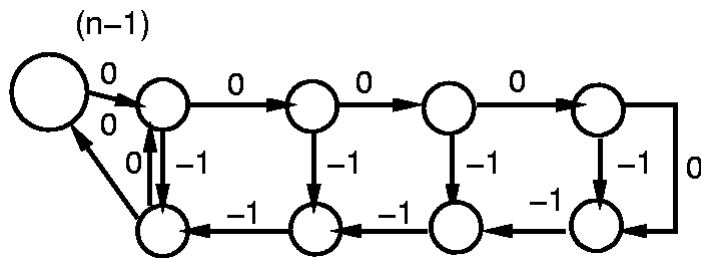
CALTECH CS137 Winter2004 -- DeHon

Real Cycle



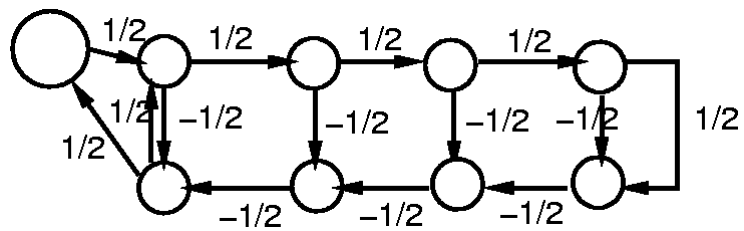
CALTE

Cycle $C=1$?



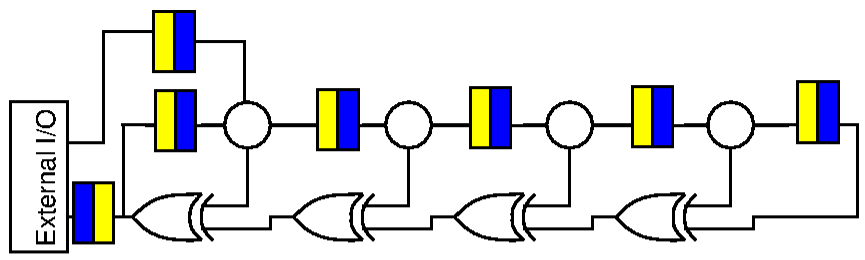
CALTECH CS137 Winter2004 -- DeHon

Cycle $C=2$?

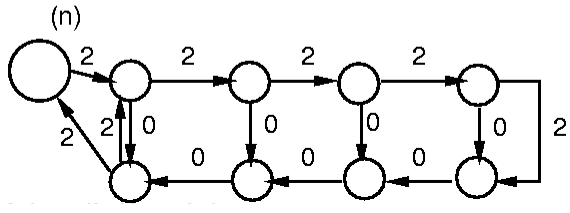


CALTECH CS137 Winter2004 -- DeHon

Cycle: C-slow

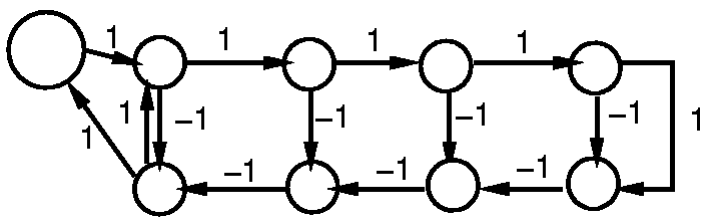


Cycle=c \Rightarrow C-slow network has Cycle=1

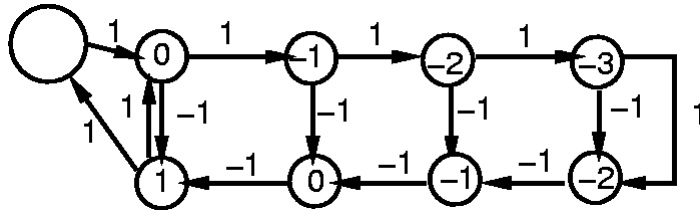


CALT

2-slow Cycle \Rightarrow C=1

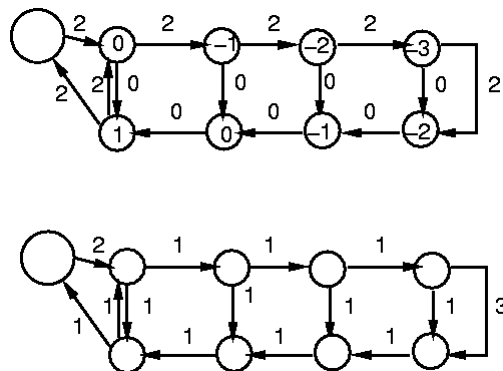


2-Slow Lags



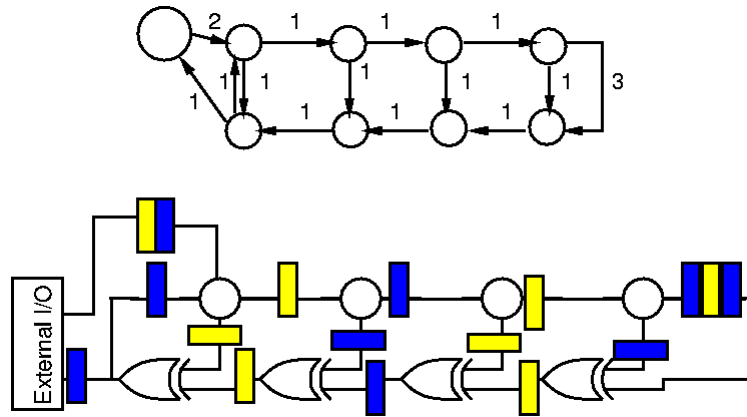
CALTECH CS137 Winter2004 -- DeHon

2-Slow Retime



CALTECH CS137 Winter2004 -- DeHon

Retimed 2-Slow Cycle



CALTECH CS137 Winter2004 -- DeHon

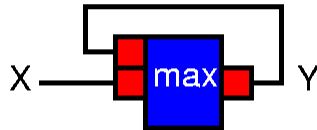
C-Slow applicable?

- 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

CALTECH CS137 Winter2004 -- DeHon

Max Example

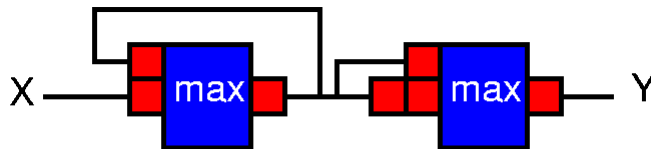
2-Slow design:



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

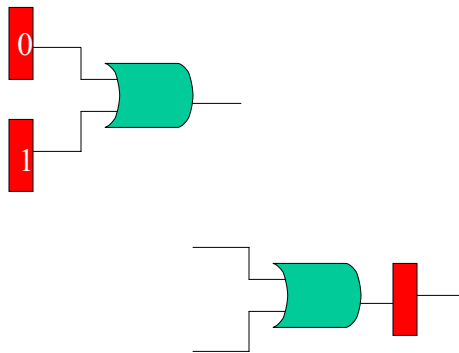
Note

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

CALTECH CS137 Winter2004 -- DeHon

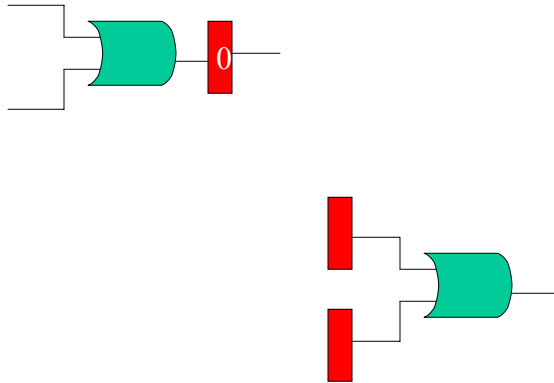
Initial State

- What about initial state?



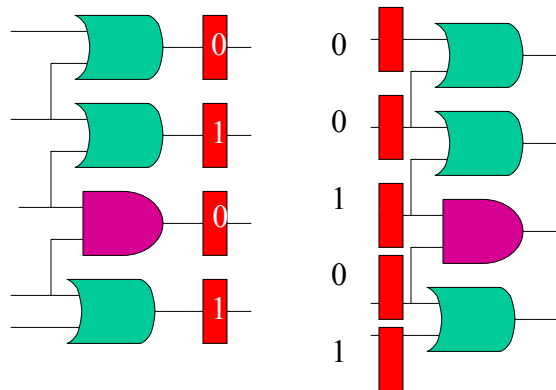
CALTECH CS137 Winter2004 -- DeHon

Initial State



CALTECH CS137 Winter2004 -- DeHon

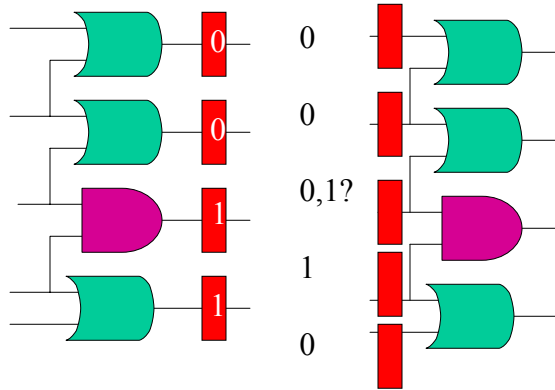
Initial State



In general, constraints \rightarrow satisfiable?

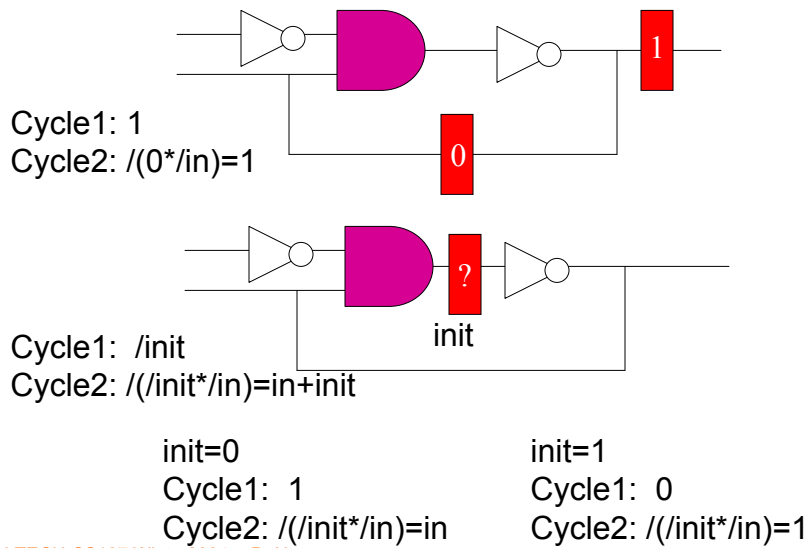
CALTECH CS137 Winter2004 -- DeHon

Initial State



CALTECH CS137 Winter2004 -- DeHon

Initial State



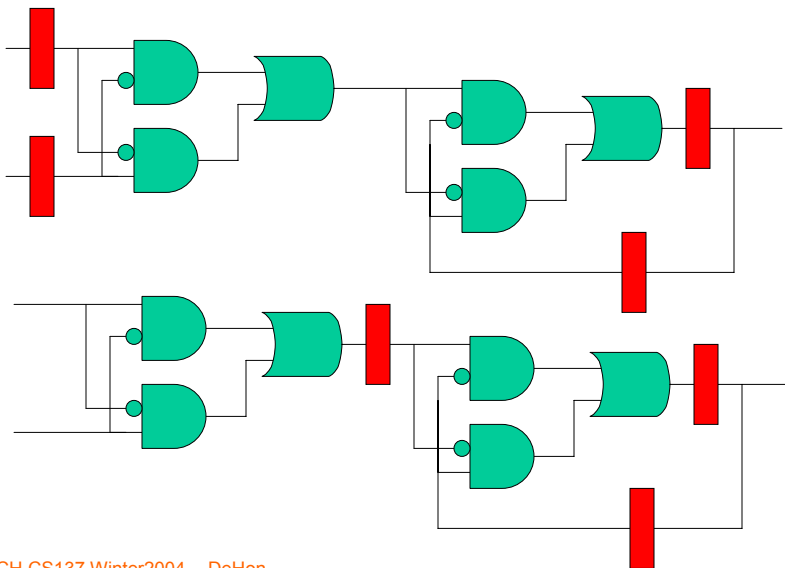
CALTECH CS137 Winter2004 -- DeHon

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

CALTECH CS137 Winter2004 -- DeHon

Minimize Registers



CALTECH CS137 Winter2004 -- DeHon

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

CALTECH CS137 Winter2004 -- DeHon

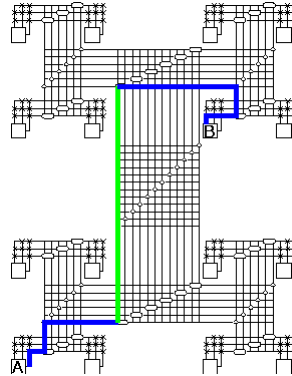
Minimize Registers

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

CALTECH CS137 Winter2004 -- DeHon

HSRA Retiming

- HSRA
 - adds mandatory pipelining to interconnect
- One additional twist
 - long, pipelined interconnect
 - \Rightarrow need more than one register on paths



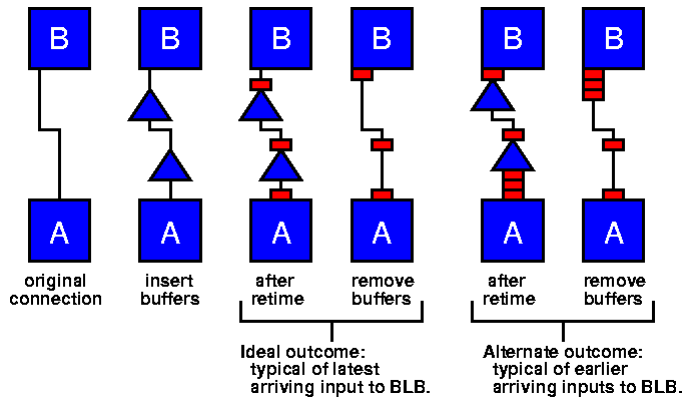
CALTECH CS137 Winter2004 -- DeHon

Accommodating HSRA Interconnect Delays

- Add buffers to LUT \rightarrow LUT path to match interconnect register requirements
- Retime to $C=1$ as before
- Buffer chains force enough registers to cover interconnect delays

CALTECH CS137 Winter2004 -- DeHon

Accommodating HSRA Interconnect Delays



CALTECH CS137 Winter2004 -- DeHon

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
- Can accommodate mandatory delays

CALTECH CS137 Winter2004 -- DeHon

Today's Big Ideas

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