CS294-7: Reconfigurable Computing

Day 22: Thursday, April 10

Retiming

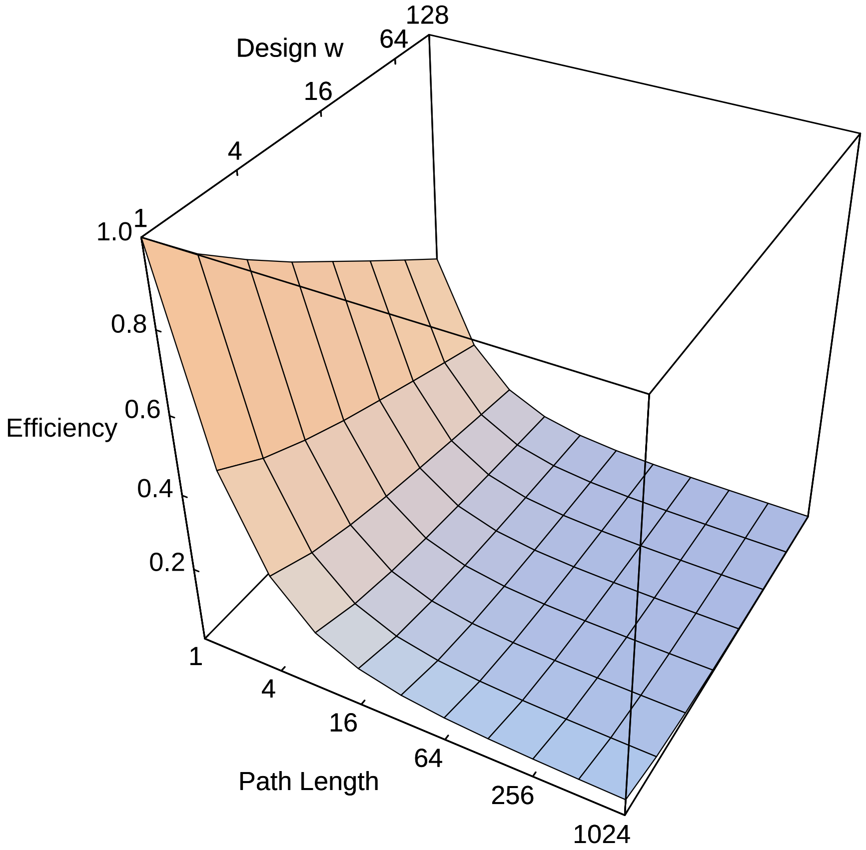
- Admin
- Fine-grained Retiming

Admin

- NEXT Thursday (4/17) $\rightarrow$ FCCM (No class)

Retiming

FPGAs → spatial computing
→ efficient utilization when cycle time is minimized



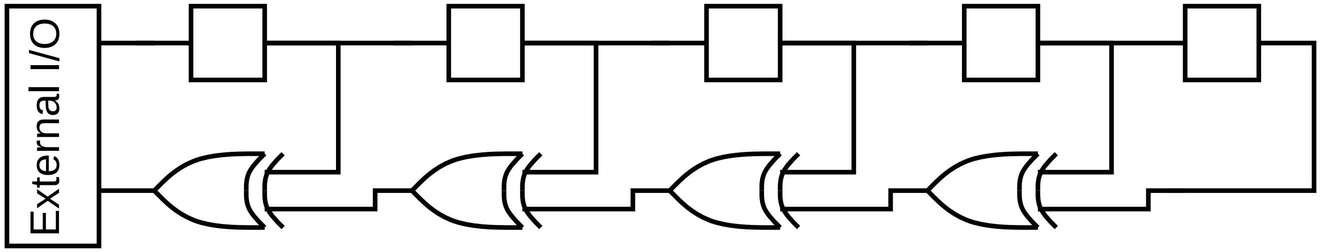
Retiming

We want to maximize efficiency:

- Minimize Path Length (L)
- Maximize reuse rate

...while minimizing resources used for retiming (minimize register usage)

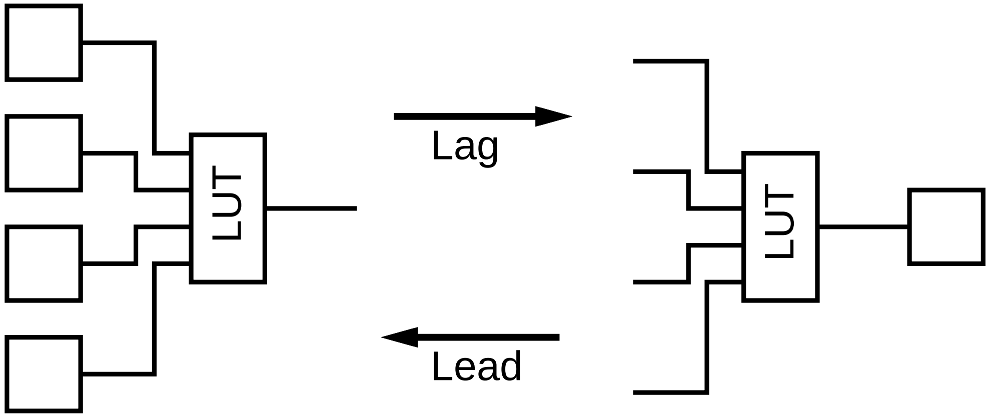
Example



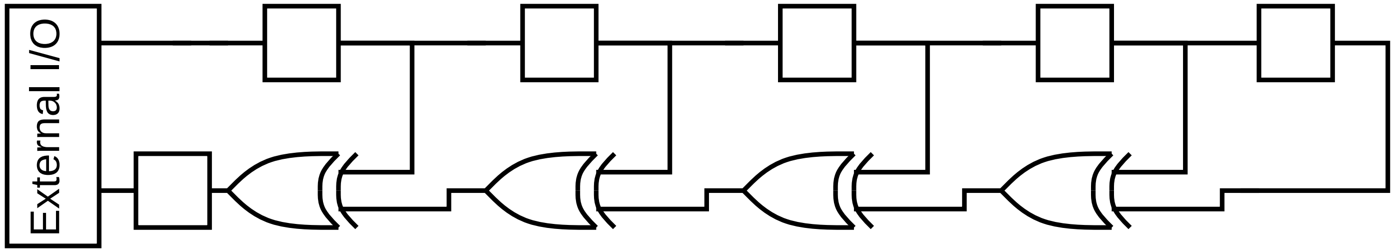
$$L = 4$$

Can we do better?

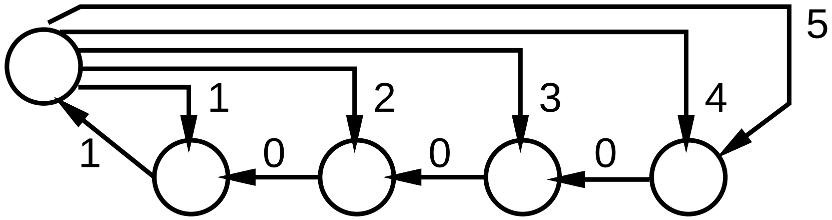
Retiming Lag/Lead



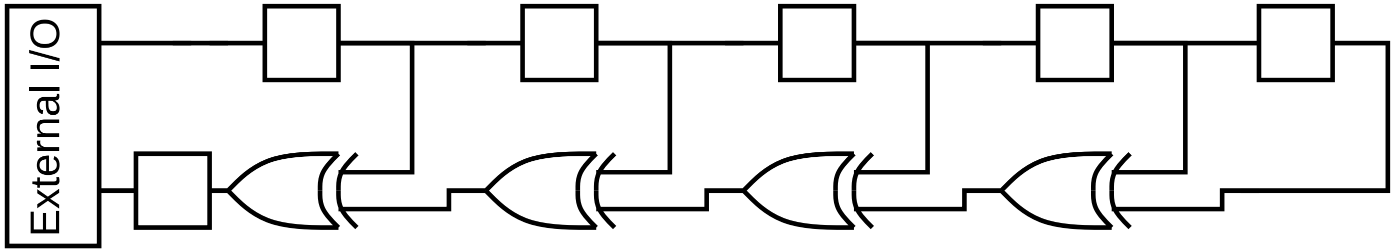
Canonical Graph Representation

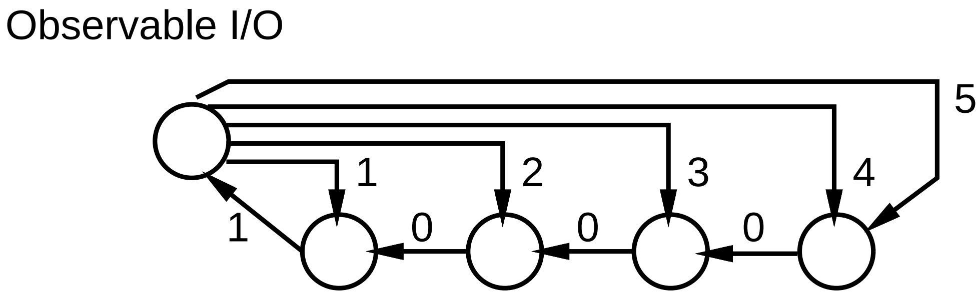


Observable I/O



Critical Path Length

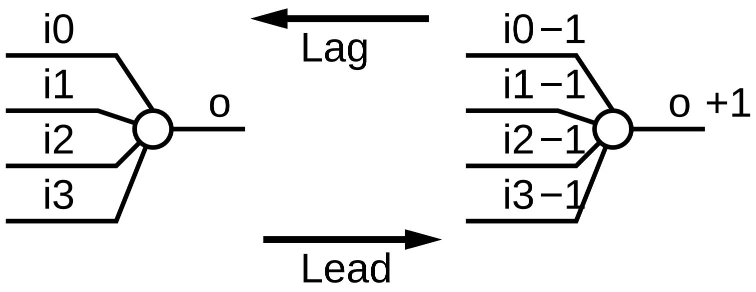




Critical Path: Length of longest path of zero weight nodes

(Compute in $O|E|$) by levelizing; topological sort, push path length forward)

Retiming Lag/Lead



Retiming: Assigning a lag to every vertex

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

Valid Retiming

Retiming is valid as long as:

$\forall e \in$ graph:

$$weight(e') = weight(e) + lag(head(e)) - lag(tail(e)) \geq 0$$

...assuming original circuit was a valid synchronous circuit

- nonnegative register weights on all edges
- all cycles have strictly positive register counts
- propagation delay on each vertex is nonnegative (assumed one here)

Optimal Retiming

There is a retiming of a graph G with clock cycle c *iff* $G - \frac{1}{c}$ has no cycles with negative edge weights.

Note:

- Number of registers around a cycle unchanged by retiming
- Delay along a cycle is unchanged by retiming

Cycle of length P must have, at least, $\frac{P}{c}$ registers on it to have a c-delay retming.

Optimal Retiming

Retiming:

lag(v) = shortest path to i/o in $G - \frac{1}{c}$

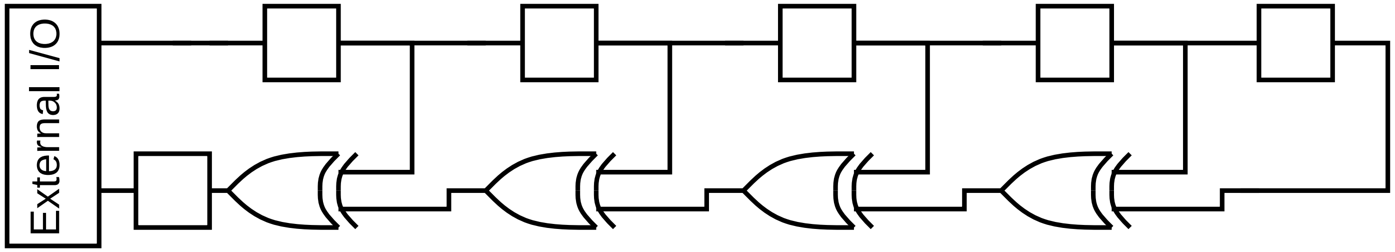
Compute in $O(|V| \cdot |E|)$ with Bellman-Ford shortests path algorithm.
(also use to detect cycles if c too small)

Bellman Ford

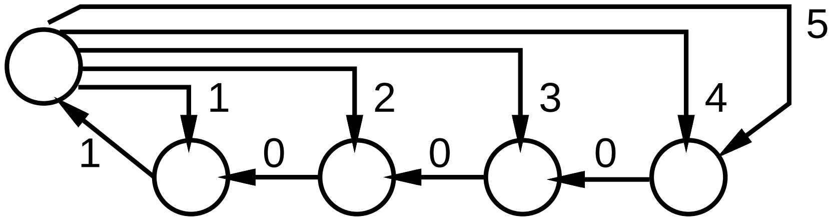
$$u_i = \min_{p: u_i \to v_h} w(p)$$

1. **for** $i \leftarrow 0$ **to** n **do** $u_i \leftarrow \infty$ **(**$u_i = v_h \leftarrow 0$**)**
2. **for** $k \leftarrow 0$ **to** n **do**
3.   **for** $e_{ij} \in E$ **do**
4. $u_i \leftarrow \min(u_i, u_j + w(e_{ij}))$
5. **for** $e_{ij} \in E$ **do**
6.   **if** $u_i > u_j + w(e_{ij})$ **cycles detected**

Revisit Example

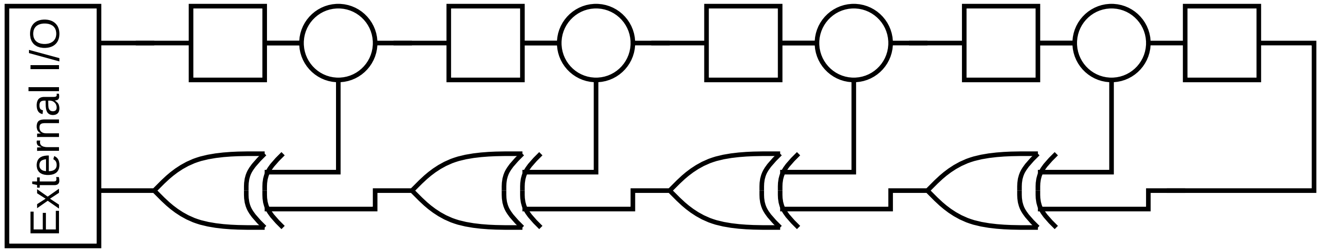


Observable I/O



Modified Example

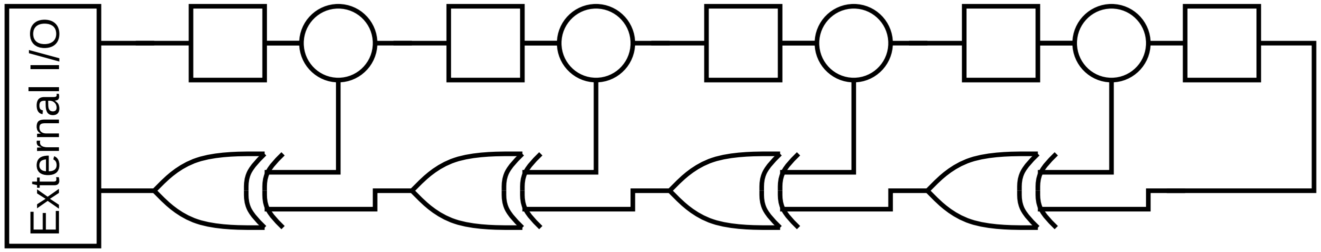
Add fanout delay:



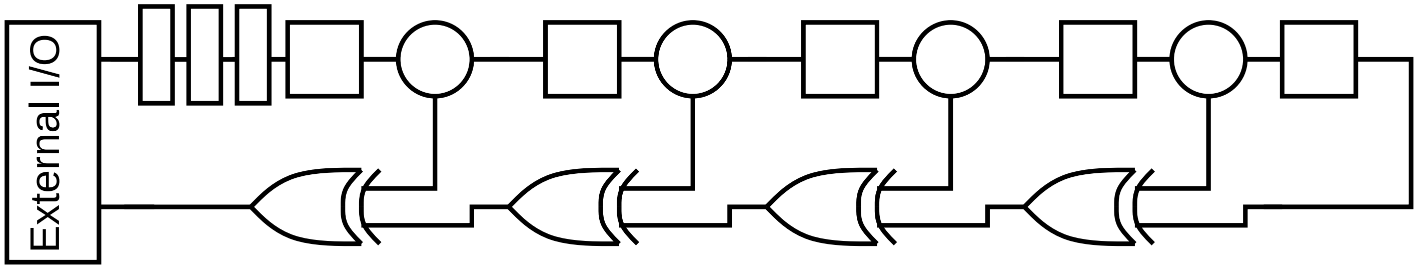
What next?

What can we do when $c > 1$?

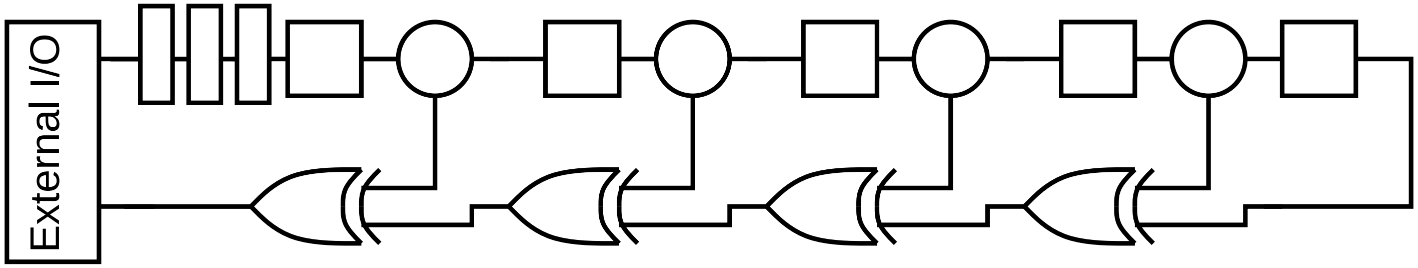
Pipelining



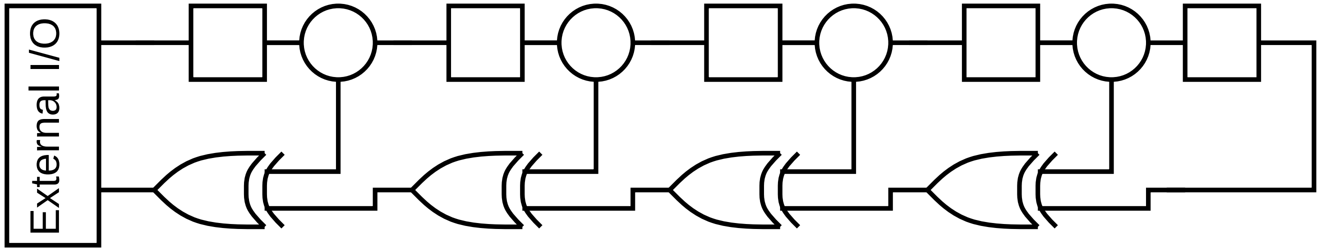
add pipeline stages



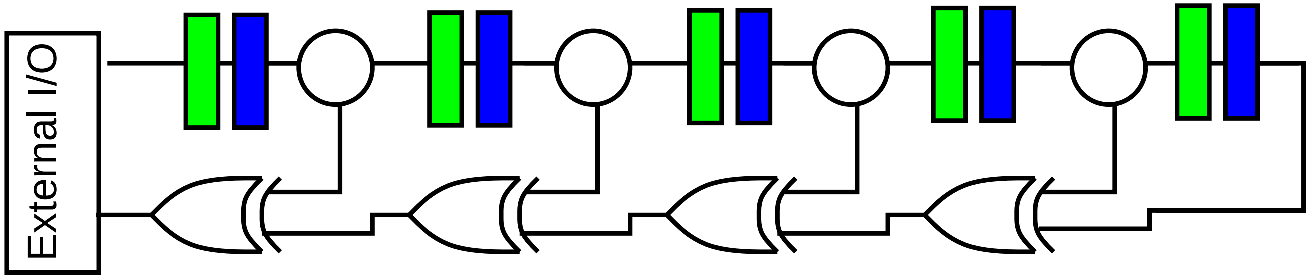
Pipelining



c-slow Execution

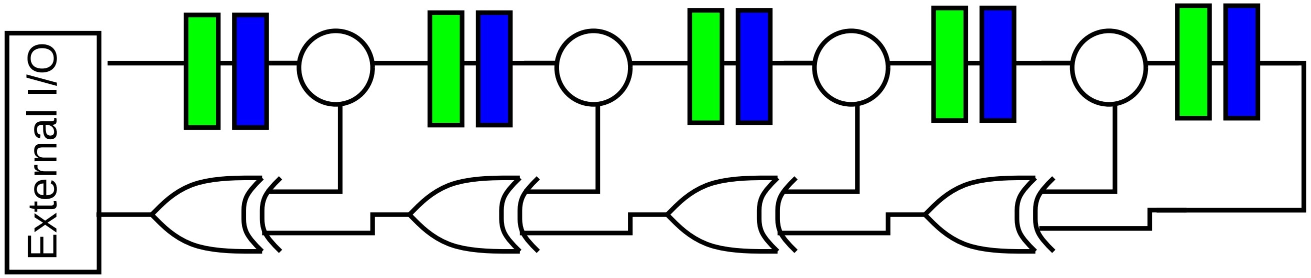


2-slow:



c-slow Execution

2-slow:



Multicontext Evaluation

Summary

- Can retime circuits to minimize cycle in polynomial time ($< O(n^3)$)
- Can increase efficiency for $c > 1$
  - pipelining
  - c-slow
  - multicontext