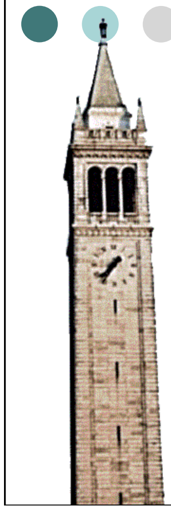


Fundamental Algorithms for System Modeling, Analysis, and Optimization

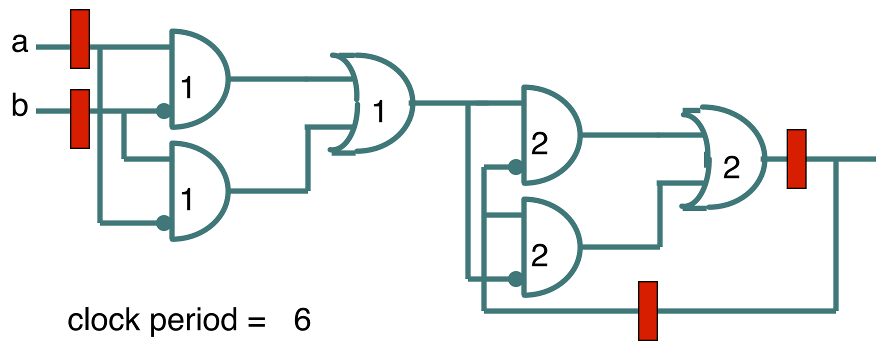
Edward A. Lee, Jaijeet Roychowdhury, Sanjit A. Seshia
UC Berkeley
EECS 144/244
Fall 2011

With thanks to R. K. Brayton, K. Keutzer, N. Shenoy, and A. Kuehlmann

Copyright © 2010-11,
E. A. Lee, J. Roydhowdhury, S. A. Seshia, All rights reserved

Lecture N: Retiming

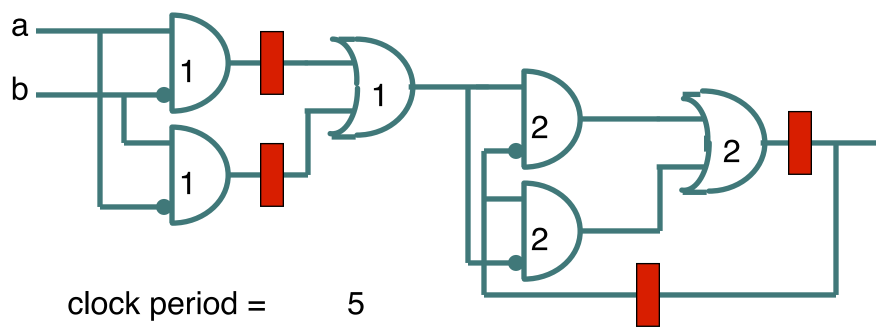
Retiming Tradeoffs



clock period = 6
# registers = 4

[Shenoy, 1997]

Retiming Tradeoffs

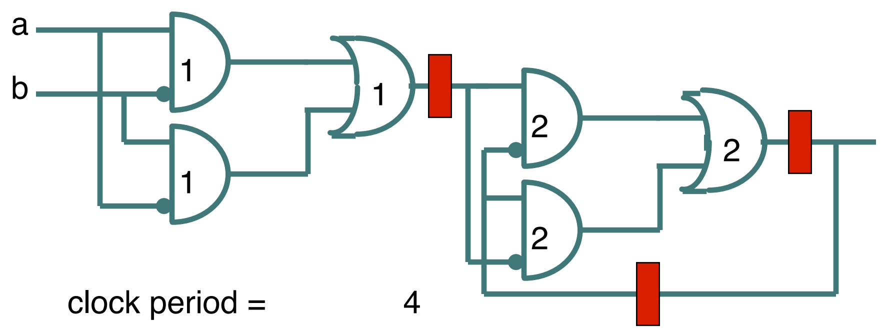


clock period = 5

# registers = 4

[Shenoy, 1997]

Retiming Tradeoffs

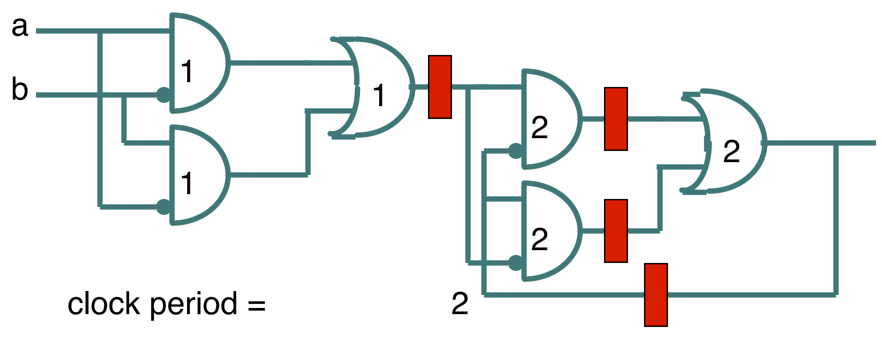


clock period = 4

registers = 3

[Shenoy, 1997]

Retiming Tradeoffs



clock period = 2

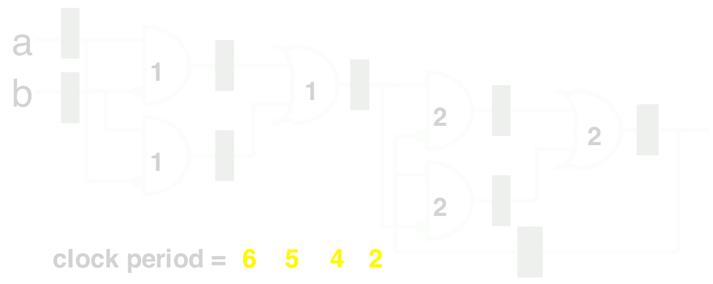
# registers = 4

[Shenoy, 1997]

Goals of Retiming

Possible goals:

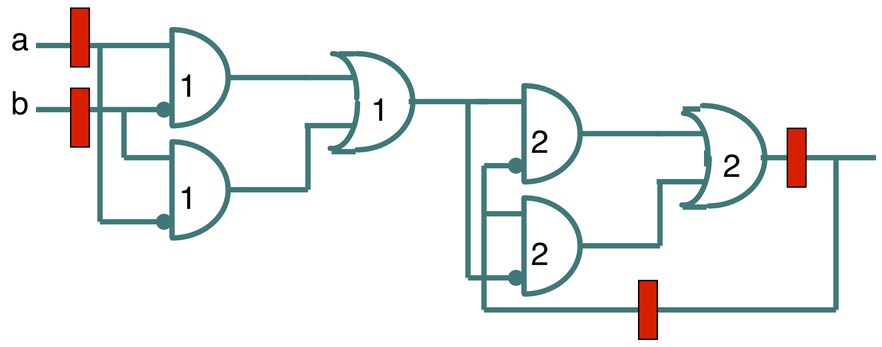
- Minimize clock period (min-period retiming)
- Minimize number of registers (min-area retiming)
- Minimize number of registers for a target clock period (constrained min-area retiming)



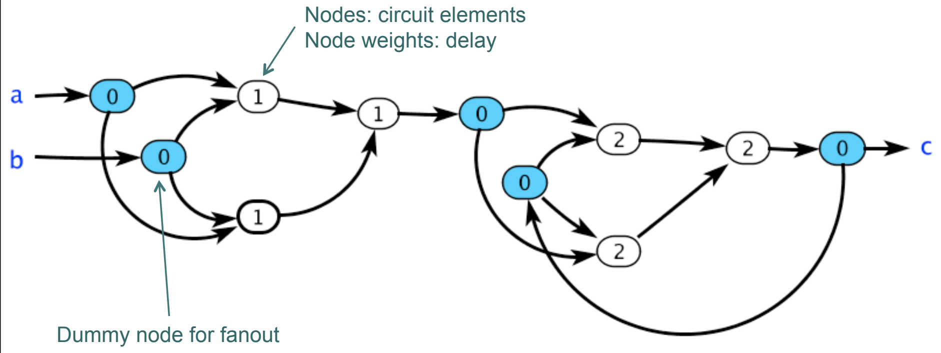
clock period =	6	5	4	2
# registers =	4	4	3	4

[Shenoy, 1997]

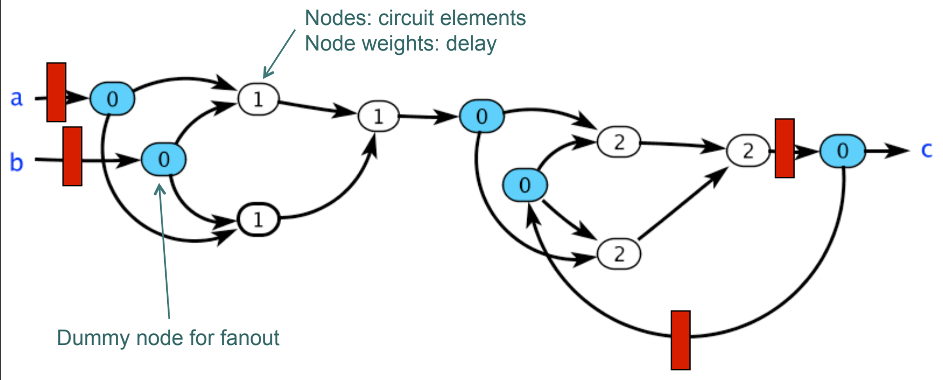
Abstraction



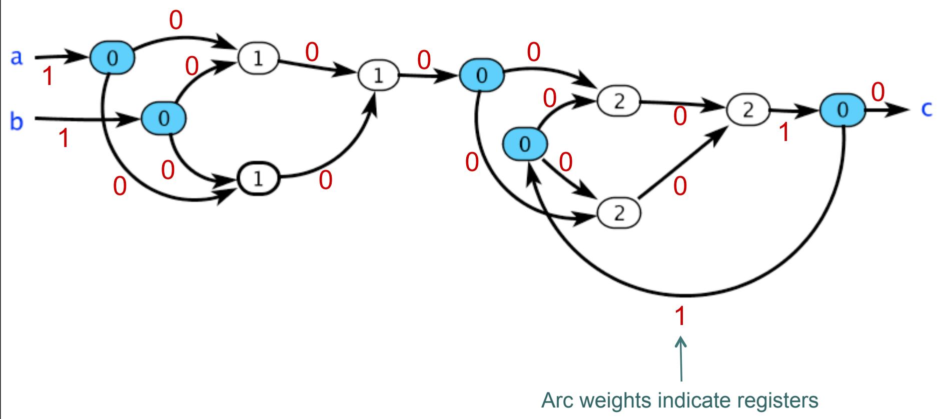
Abstraction - Graph



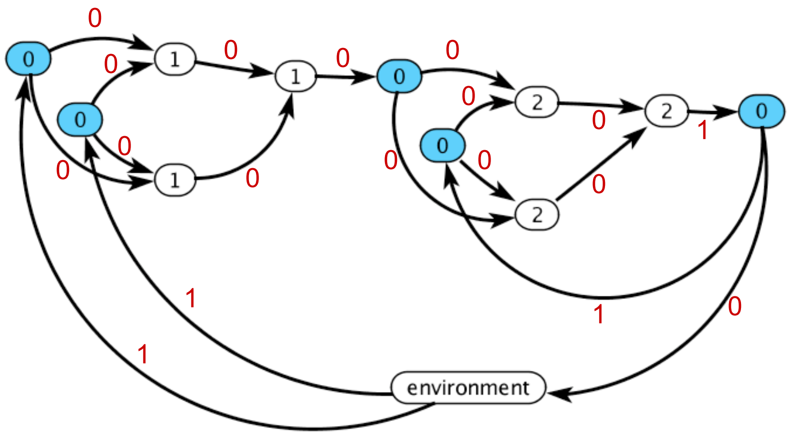
Abstraction - Registers



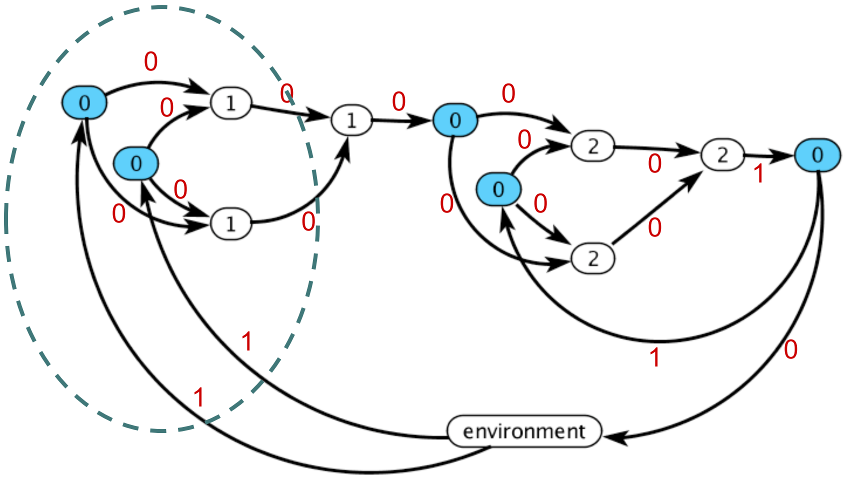
Abstraction - Registers



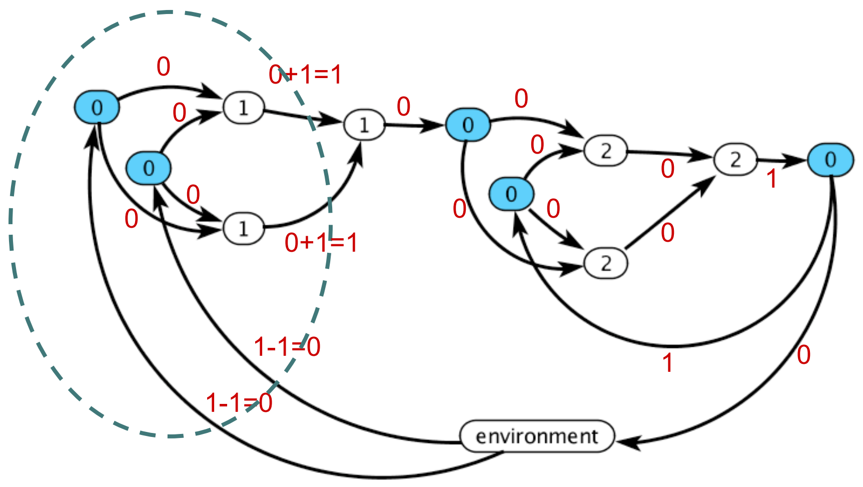
Abstraction - Environment



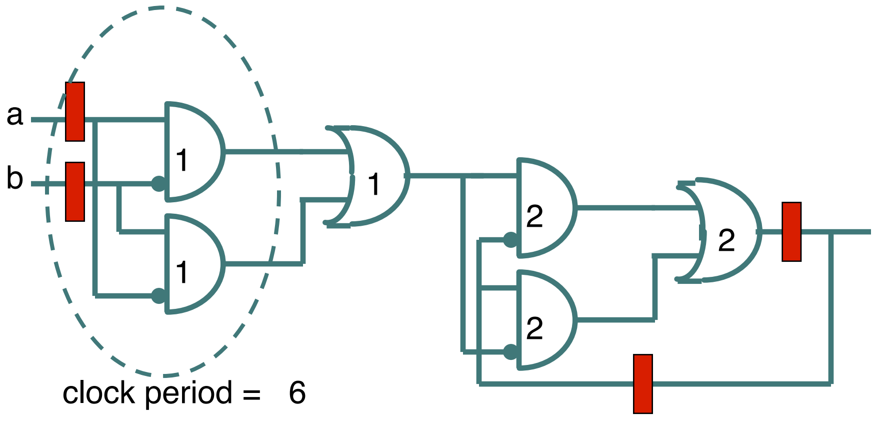
Cutset – Divides the Graph in Two



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

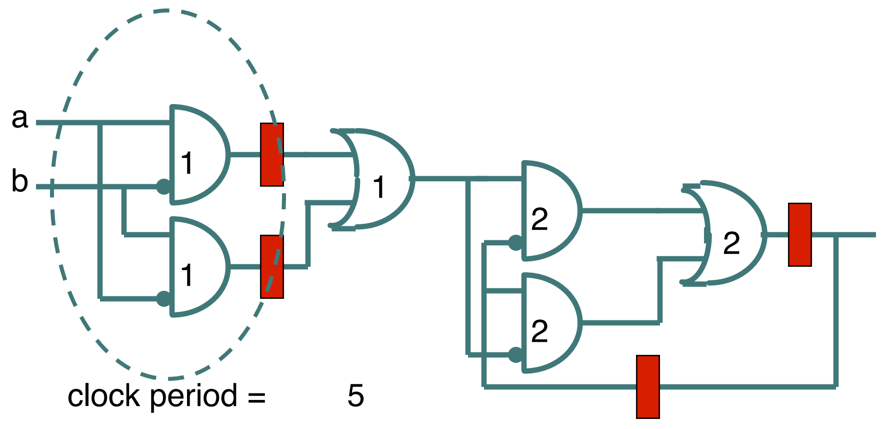


clock period = 6

# registers = 4

[Shenoy, 1997]

Recall: Retiming Tradeoffs

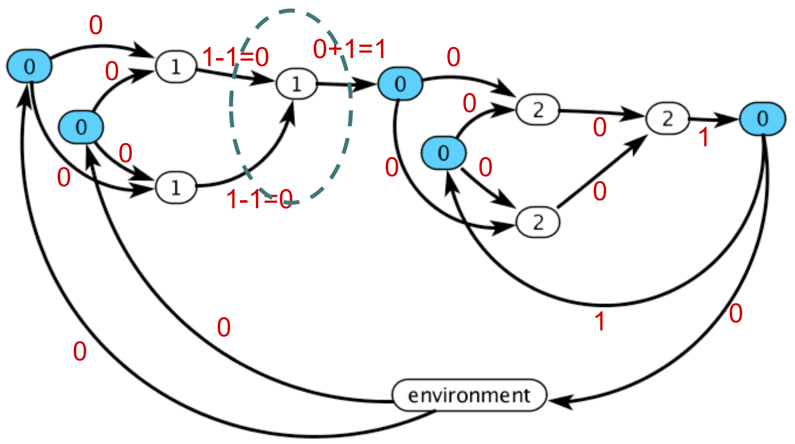


clock period = 5

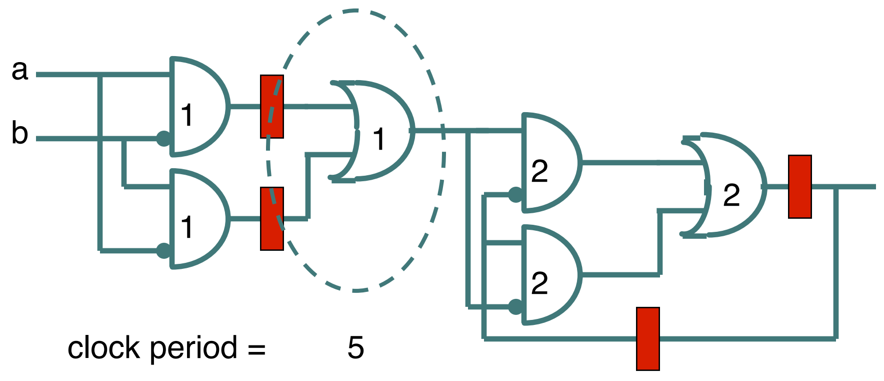
registers = 4

[Shenoy, 1997]

Simplest cutset surrounds one node



Recall: Retiming Tradeoffs

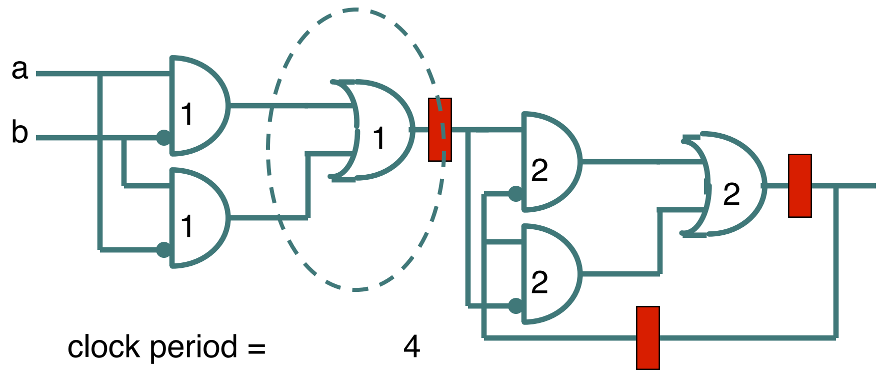


clock period = 5

# registers = 4

[Shenoy, 1997]

Recall: Retiming Tradeoffs



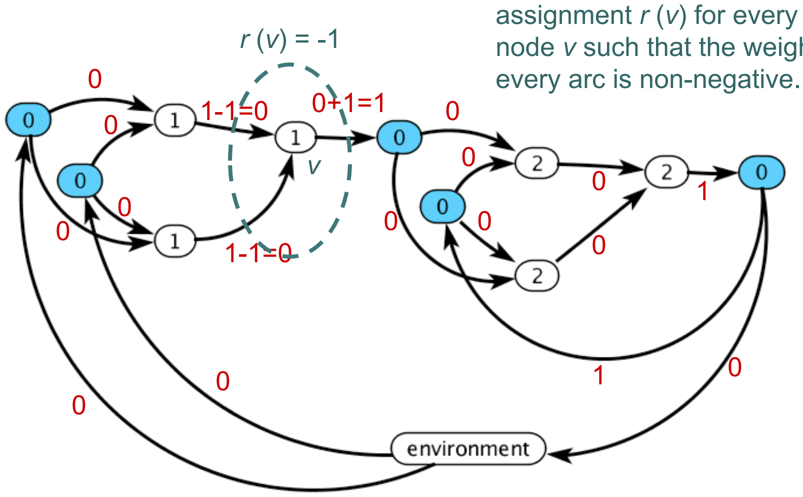
clock period = 4

# 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.



Problem Setup

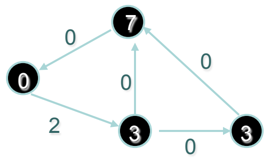
- For a path p: $v_0\, v_1\, v_2 \ldots v_k$, $e_i = (v_i , v_{i+1})$

$$d(p) = \sum_{i=0}^{k} d(v_i) \quad \text{(includes endpoints)}$$

$$w(p) = \sum_{i=0}^{k-1} w(e_i)$$

- Clock cycle

$$c = \max_{p:w(p)=0} \{d(p)\}$$



For example on right: c = ?

Problem Setup

- For a path p: $v_0\, v_1\, v_2\, \ldots\, v_k,\ e_i = (v_i, v_{i+1})$

$$d(p) = \sum_{i=0}^{k} d(v_i) \quad \text{(includes endpoints)}$$

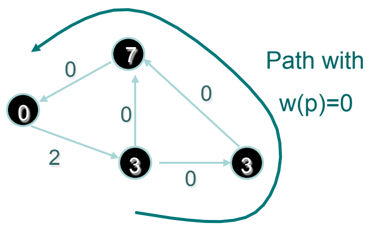
$$w(p) = \sum_{i=0}^{k-1} w(e_i)$$

- Clock cycle

$$c = \max_{p:w(p)=0} \{d(p)\}$$

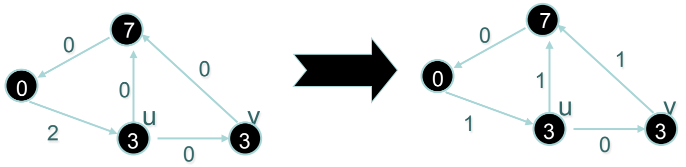
(delay of longest path without registers)

For example on right: c = 13



Basic Operation

- Thus in the example, r(u) = -1, r(v) = -1 results in



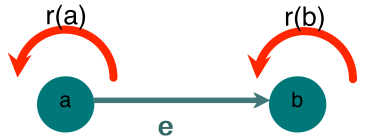
- For a path p: s→t, $W_r(p) = w(p) + r(t) - r(s)$
- Retiming
  - r: V→Z, an integer vertex labeling
  - $w_r(e) = w(e) + r(v) - r(u)$ for edge e = (u,v)
  - A retiming r is *legal* if $w_r(e) \geq 0,\ \forall\, e \in E$

Retiming - Assumptions

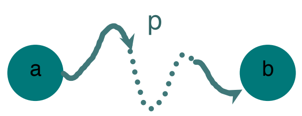
- Each loop in circuit contains at least one register
- Circuit uses single clock and edge-triggered elements (identical skew)
- Gate delay is constant (and non-negative)
- Registers are ideal (set-up, drive independent of load)
- Any power-up state of the design can be safely handled by the environment (initial state assumption)

Retiming - Formulation

- Come up with r(v) values: Assign integers to each vertex so that objective is met
- Valid retiming constraints



$\mathbf{w_r(e) = w(e)} + r(b) - r(a) \geq 0$



$w_r(p) = w(p) + r(b) - r(a)$

Retiming for Minimum Clock Cycle

- Problem Statement: (Minimum cycle time)
- Given G(V, E, d, w), find a Legal retiming r so that

$$c = \max_{p:W_r(p)=0} \{d(p)\} \qquad \text{(A)}$$

is minimized

- 2 important matrices
  - Register weight matrix

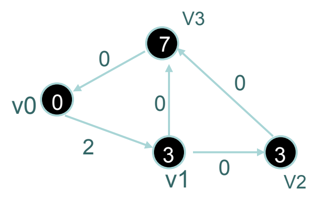
$$W(u,v) = \min\{w(p) : u \xrightarrow{p} v\}$$

  - Delay matrix

$$D(u,v) = \max\{d(p) : u \xrightarrow{p} v, w(p) = W(u,v)\}$$

$$D(u,v) > c \Rightarrow W(u,v) \ge 1 \qquad \text{(B)}$$

Retiming for Minimum Clock Period



W – register path weight matrix, min # of registers on all paths between u and v

D – path delay matrix, max delay among all paths between u and v with W(u,v) (minimum) registers

W

	V0	V1	V2	V3
V0	0	2	2	2
V1	0	0	0	0
V2	0	2	0	0
V3	0	2	2	0

D

	V0	V1	V2	V3
V0	0	3	6	13
V1	13	3	6	13
V2	10	13	3	10
V3	7	10	13	7

Note that the D matrix indicates that the least possible clock period is 7, and a period of 13 will obviously work, so the minimum clock period is between 7 and 13, inclusive. Binary search of these possibilities will work.

Conditions for Retiming

- Suppose we need to check if a retiming exists for a clock cycle α
- Legal retiming: $w_r(e) \geq 0$ for all e. Hence

  $w_r(e) = w(e) + r(v) - r(u) \geq 0$ or

  $r(u) - r(v) \leq w(e)$
- For all paths p: $u \rightarrow v$ such that $d(p) \geq \alpha$, we require $w_r(p) \geq 1$
  - Thus

$$
\begin{aligned}
1 \leq w_r(p) &= \sum_{i=0}^{k-1} w_r(e_i) \\
&= \sum_{i=0}^{k-1} [w(e_i) + r(v_{i+1}) - r(v_i)] \\
&= w(p) + r(v_k) - r(v_0) \\
&= w(p) + r(v) - r(u)
\end{aligned}
$$

Or take the least w(p) (tightest constraint) $r(u)-r(v) \leq W(u,v)-1$

i.e. there are many paths *p*, choose the *p* that gives tightest constraint

Note: we just need to apply it to (u, v) such that $D(u,v) > \alpha$

Solving the Constraints

- All constraints in "difference of 2 variables" form
- How to solve?

Consider our example for α = 7

W

$$
\begin{array}{c|cccc}
 & V0 & V1 & V2 & V3 \\
\hline
V0 & 0 & 2 & 2 & 2 \\
V1 & 0 & 0 & 0 & 0 \\
V2 & 0 & 2 & 0 & 0 \\
V3 & 0 & 2 & 2 & 0
\end{array}
$$

D

$$
\begin{array}{cccc|c}
V0 & V1 & V2 & V3 & \\
\hline
0 & 3 & 6 & 13 & V0 \\
13 & 3 & 6 & 13 & V1 \\
10 & 13 & 3 & 10 & V2 \\
7 & 10 & 13 & 7 & V3
\end{array}
$$

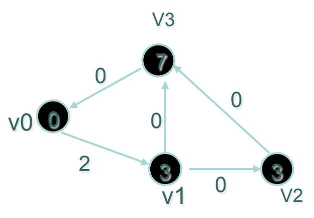
Legal: $r(u)-r(v) \leq w(e)$

$$
\begin{aligned}
r(v_0) - r(v_1) &\leq 2 \\
r(v_1) - r(v_2) &\leq 0 \\
r(v_1) - r(v_3) &\leq 0 \\
r(v_2) - r(v_3) &\leq 0 \\
r(v_3) - r(v_0) &\leq 0
\end{aligned}
$$

D>7:
$r(u)-r(v) \leq W(u,v)-1$

$$
\begin{aligned}
r(v_0) - r(v_3) &\leq 1 \\
r(v_1) - r(v_0) &\leq -1 \\
r(v_1) - r(v_3) &\leq -1 \\
r(v_2) - r(v_0) &\leq -1 \\
r(v_2) - r(v_1) &\leq 1 \\
r(v_2) - r(v_3) &\leq -1 \\
r(v_3) - r(v_1) &\leq 1 \\
r(v_3) - r(v_2) &\leq 1
\end{aligned}
$$

Notice that these constraints are unaffected by adding or subtracting any constant to/ from all $r(v_i)$. Why?



Solving the Constraints: Construct a Constraint Graph

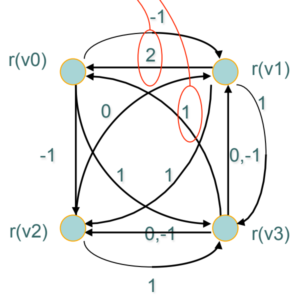
Constraint graph has one arc for each difference-of-two-variables inequality.

Legal: r(u)-r(v)≤w(e)

$r(v_0) - r(v_1) \le 2$
$r(v_1) - r(v_2) \le 0$
$r(v_1) - r(v_3) \le 0$
$r(v_2) - r(v_3) \le 0$
$r(v_3) - r(v_0) \le 0$

D>7:
r(u)-r(v)≤W(u,v)-1

$r(v_0) - r(v_3) \le 1$
$r(v_1) - r(v_0) \le -1$
$r(v_1) - r(v_3) \le -1$
$r(v_2) - r(v_0) \le -1$
$r(v_2) - r(v_1) \le 1$
$r(v_2) - r(v_3) \le -1$
$r(v_3) - r(v_1) \le 1$
$r(v_3) - r(v_2) \le 1$



Solving the Constraints: Add a dummy start node to the constraint graph

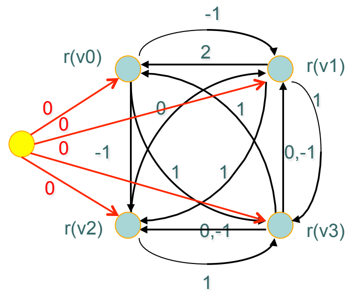
A solution to the constraints, if it exists: *r* (*v*$_i$) is the minimum path weight from the dummy node to node *r* (*v*$_i$).
No solution exists if there are cycles with negative weight. Why?

Legal: r(u)-r(v)≤w(e)

$r(v_0) - r(v_1) \le 2$
$r(v_1) - r(v_2) \le 0$
$r(v_1) - r(v_3) \le 0$
$r(v_2) - r(v_3) \le 0$
$r(v_3) - r(v_0) \le 0$

D>7:
r(u)-r(v)≤W(u,v)-1

$r(v_0) - r(v_3) \le 1$
$r(v_1) - r(v_0) \le -1$
$r(v_1) - r(v_3) \le -1$
$r(v_2) - r(v_0) \le -1$
$r(v_2) - r(v_1) \le 1$
$r(v_2) - r(v_3) \le -1$
$r(v_3) - r(v_1) \le 1$
$r(v_3) - r(v_2) \le 1$



Solving the Constraints: Use the Bellman-Ford Algorithm: O(|V|^3)

Cannot use Dijkstra's algorithm, which works to find minimum path weights only if the weights all have the same sign.

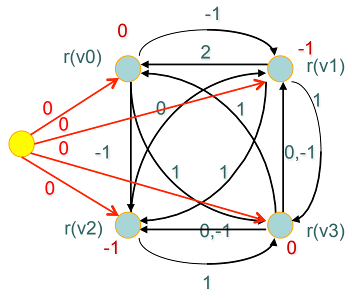
Bellman-Ford can detect where there are cycles with negative weight.

Legal: r(u)-r(v)≤w(e)

$$
\begin{aligned}
r(v_0)-r(v_1) &\le 2\\
r(v_1)-r(v_2) &\le 0\\
r(v_1)-r(v_3) &\le 0\\
r(v_2)-r(v_3) &\le 0\\
r(v_3)-r(v_0) &\le 0
\end{aligned}
$$

D>7:
r(u)-r(v)≤W(u,v)-1

$$
\begin{aligned}
r(v_0)-r(v_3) &\le 1\\
r(v_1)-r(v_0) &\le -1\\
r(v_1)-r(v_3) &\le -1\\
r(v_2)-r(v_0) &\le -1\\
r(v_2)-r(v_1) &\le 1\\
r(v_2)-r(v_3) &\le -1\\
r(v_3)-r(v_1) &\le 1\\
r(v_3)-r(v_2) &\le 1
\end{aligned}
$$

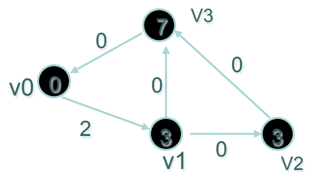


Notice that this algorithm will only yield non-positive values of $r(v_i)$. Why is this OK?

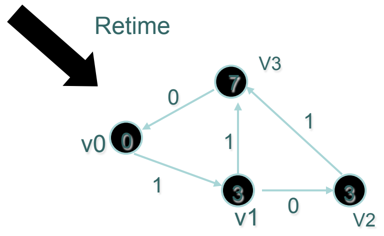
A solution is $r(v_0) = r(v_3) = 0$, $r(v_1) = r(v_2) = -1$

Retiming Solution

To find the minimum cycle time, do a binary search among the entries of the D matrix 0(|V|^{3}log|V|)



Clock period = 3+3+7=13



Clock period= 7

W

$$
\begin{array}{c|cccc}
 & V0 & V1 & V2 & V3\\
\hline
V0 & 0 & 2 & 2 & 2\\
V1 & 0 & 0 & 0 & 0\\
V2 & 0 & 2 & 0 & 0\\
V3 & 0 & 2 & 2 & 0
\end{array}
$$

D

$$
\begin{array}{cccc|c}
V0 & V1 & V2 & V3 & \\
\hline
0 & 3 & 6 & 13 & V0\\
13 & 3 & 6 & 13 & V1\\
10 & 13 & 3 & 10 & V2\\
7 & 10 & 13 & 7 & V3
\end{array}
$$

Drawbacks of this Algorithm

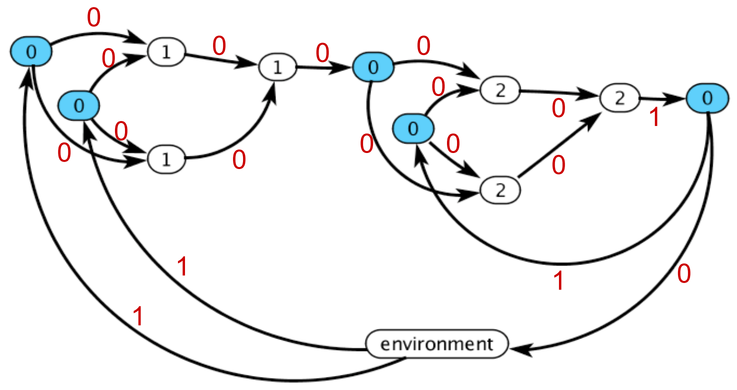
- Requires W/D matrix computation
- O(|V|^2) clock period constraints, most of which are redundant
- Average case is worst case

Fortunately, there's another algorithm we can use:

- "relaxation-based" algorithm called FEAS
- See: [Shenoy, 1997]

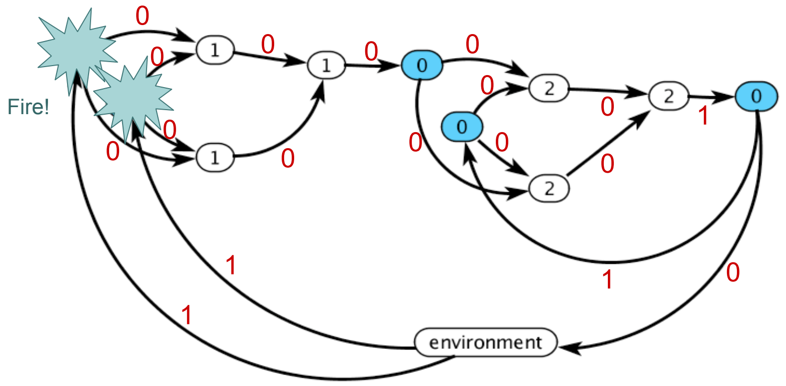
Applications beyond circuits

In a *dataflow* graph, nodes represent *actors*, which *fire* when input *tokens* are available. Firing performs a computation that takes time. Weights represent *initial* tokens. Retiming can be interpreted as a *preamble* to a periodic schedule, and may have the goal of maximizing parallelism so that the dataflow graph executes fast on a multicore machine.



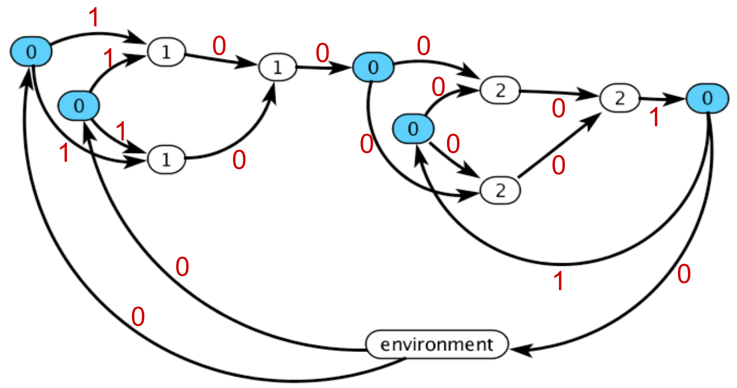
Applications beyond circuits

In a *dataflow* graph, nodes represent *actors*, which *fire* when input *tokens* are available. Firing performs a computation that takes time. Weights represent *initial* tokens. Retiming can be interpreted as a *preamble* to a periodic schedule, and may have the goal of maximizing parallelism so that the dataflow graph executes fast on a multicore machine.



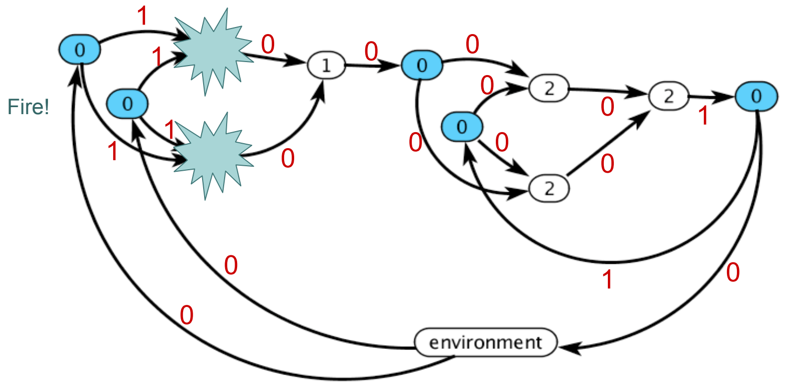
Applications beyond circuits

In a *dataflow* graph, nodes represent *actors*, which *fire* when input *tokens* are available. Firing performs a computation that takes time. Weights represent *initial* tokens. Retiming can be interpreted as a *preamble* to a periodic schedule, and may have the goal of maximizing parallelism so that the dataflow graph executes fast on a multicore machine.



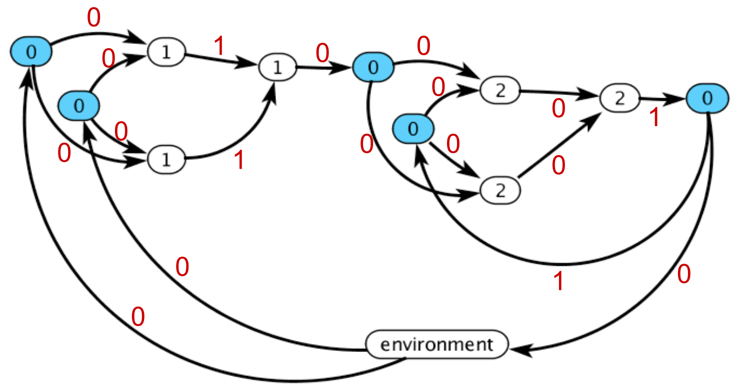
Applications beyond circuits

In a *dataflow* graph, nodes represent *actors*, which *fire* when input *tokens* are available. Firing performs a computation that takes time. Weights represent *initial* tokens. Retiming can be interpreted as a *preamble* to a periodic schedule, and may have the goal of maximizing parallelism so that the dataflow graph executes fast on a multicore machine.



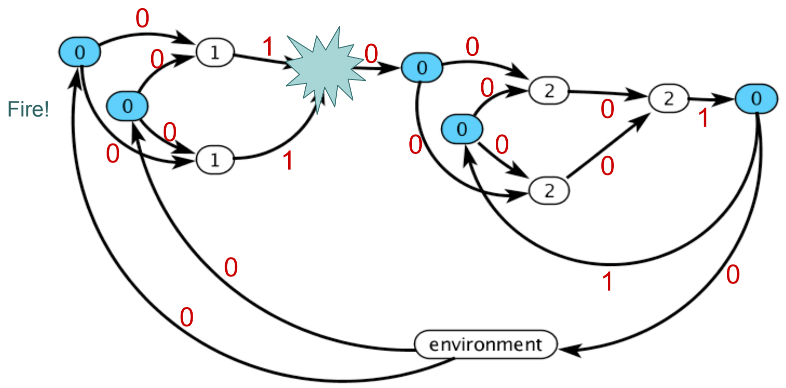
Applications beyond circuits

In a *dataflow* graph, nodes represent *actors*, which *fire* when input *tokens* are available. Firing performs a computation that takes time. Weights represent *initial* tokens. Retiming can be interpreted as a *preamble* to a periodic schedule, and may have the goal of maximizing parallelism so that the dataflow graph executes fast on a multicore machine.



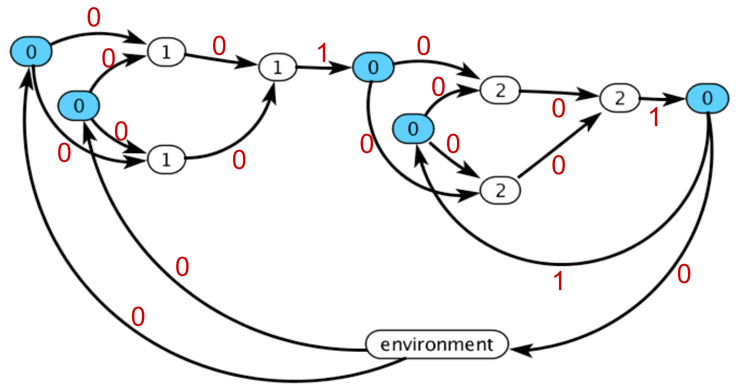
Applications beyond circuits

In a *dataflow* graph, nodes represent *actors*, which *fire* when input *tokens* are available. Firing performs a computation that takes time. Weights represent *initial* tokens. Retiming can be interpreted as a *preamble* to a periodic schedule, and may have the goal of maximizing parallelism so that the dataflow graph executes fast on a multicore machine.



After retiming

After retiming the graph, may be able to construct a better (faster) parallel schedule that will be executed periodically.



References

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.