DSP Design – Lecture 5

Retiming

Dr. Fredrik Edman

fredrik.edman@eit.lth.se

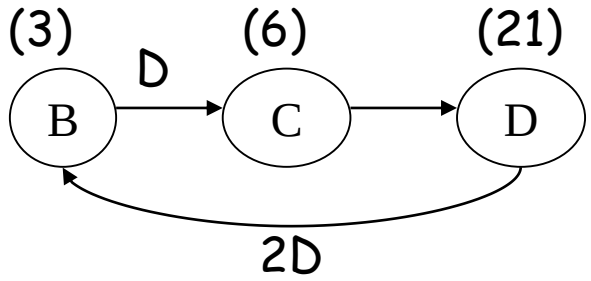
Repetition

- **Critical path** - the combinational path with maximum total execution time
- **Loop (=cycle)** - a path beginning and ending at same node
- **Loop bound for loop**

$$\frac{T_j}{W_j}$$

T_j loop computation time

W_j number of delays in loop



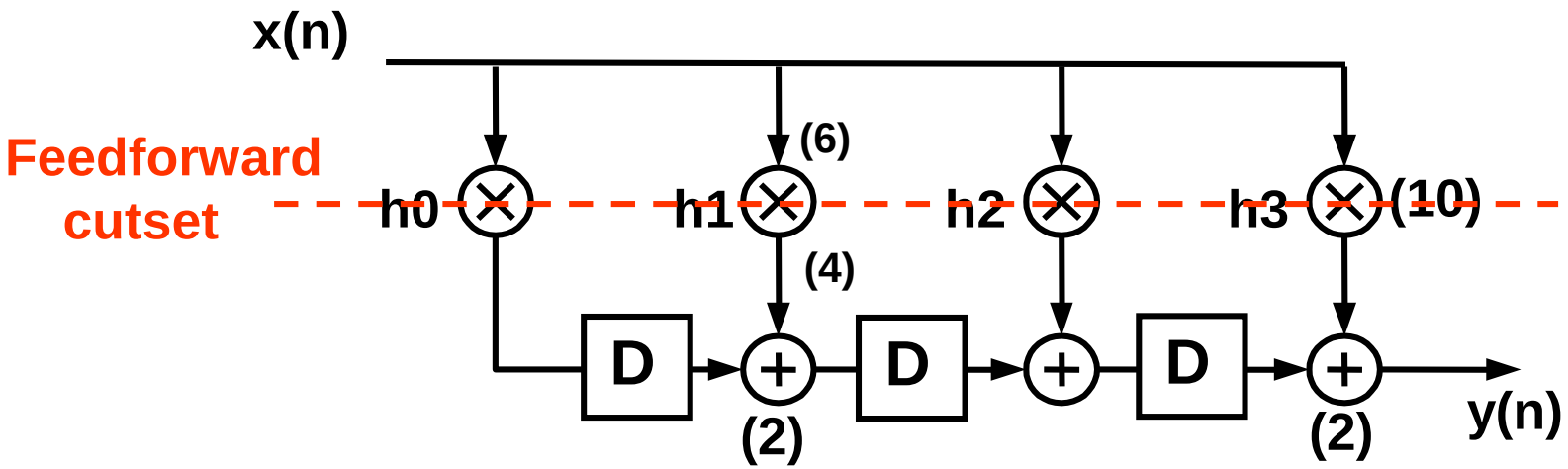
- **Iteration Bound** - maximum of all loop bounds

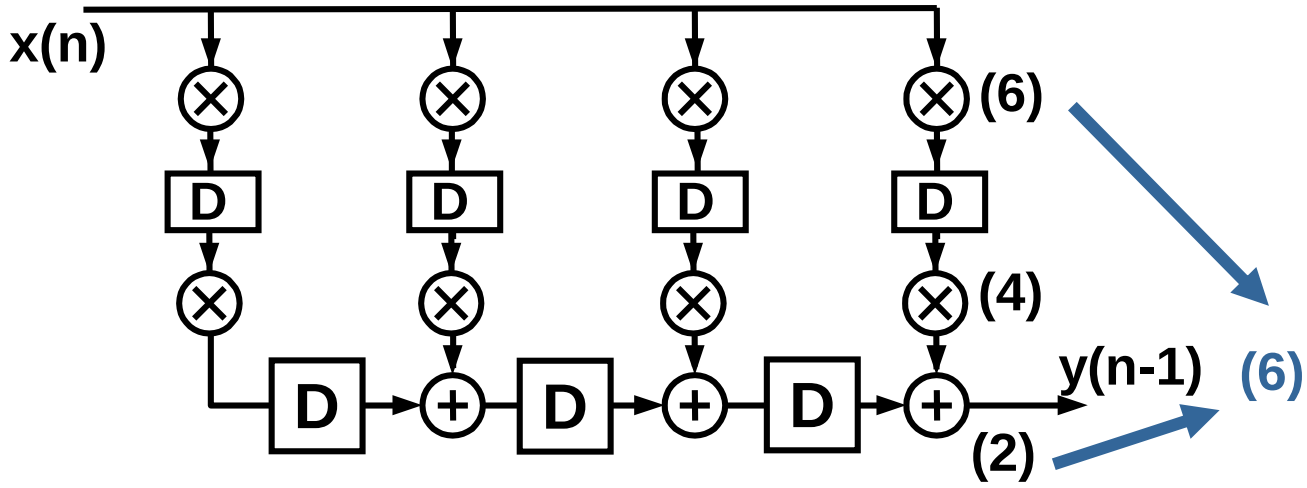
$$T_\infty = \max_{l \in L} \left\{ \frac{t_l}{w_l} \right\}$$

It is the lower bound on execution time for DFG (assuming only pipelining, retiming, unfolding)

Fine-Grain pielining

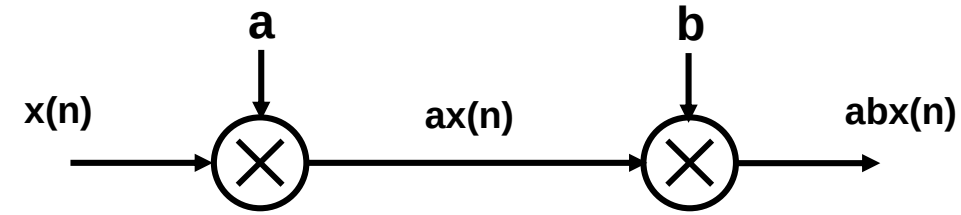
Repetition

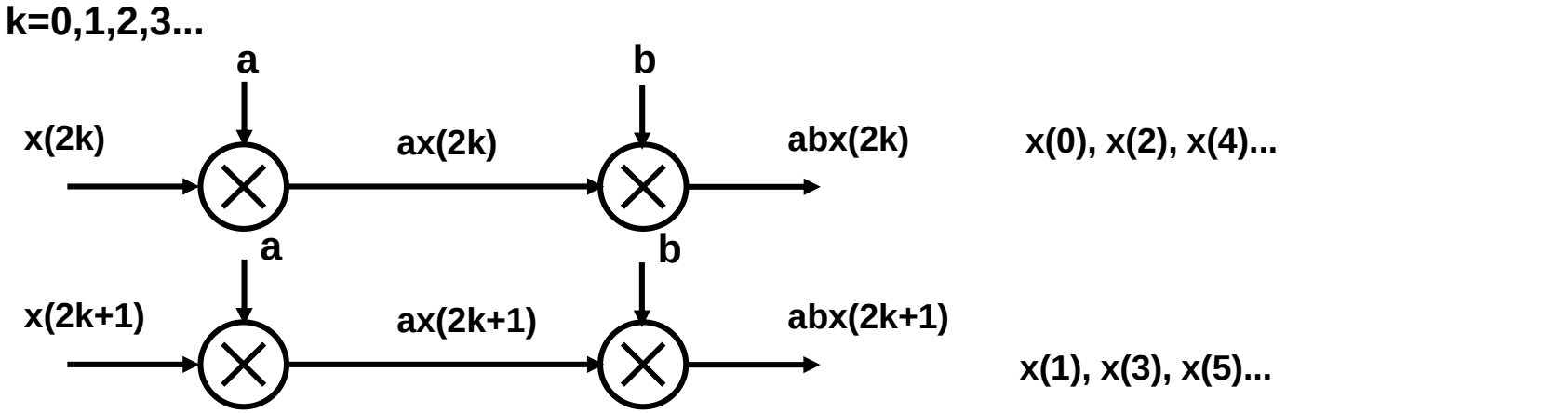




Parallel Processing

Repetition





Two samples are processed in parallel => double throughput or lower power consumption due to reduced V_{DD}

When block size is 2, 1 delay element = 2 sampling delays

Power Consumption

Repetition

– The power consumption in original architecture

$$P_{seq} = f\, C_L\, V_{DD}^{\,2}$$

– The supply voltage can be reduced to βV_{DD} , $(0<\beta<1)$.
Hence, the power consumption of the pipelined filter is:

$$P_{pipe} = f\, C_L \left(\beta V_{DD}\right)^2 = \beta^2 P_{seq}$$

$$P_{para} = LC_L \frac{f}{L} (\beta V_{DD})^2 = \beta^2 \cdot P_{seq}$$

The Effect of Pipelining and Parallelization

Repetition

We have seen that the power can be significantly reduced in a system using pipelining and parallelization.

$$P = f\,C\,V^2$$

Pipelining only

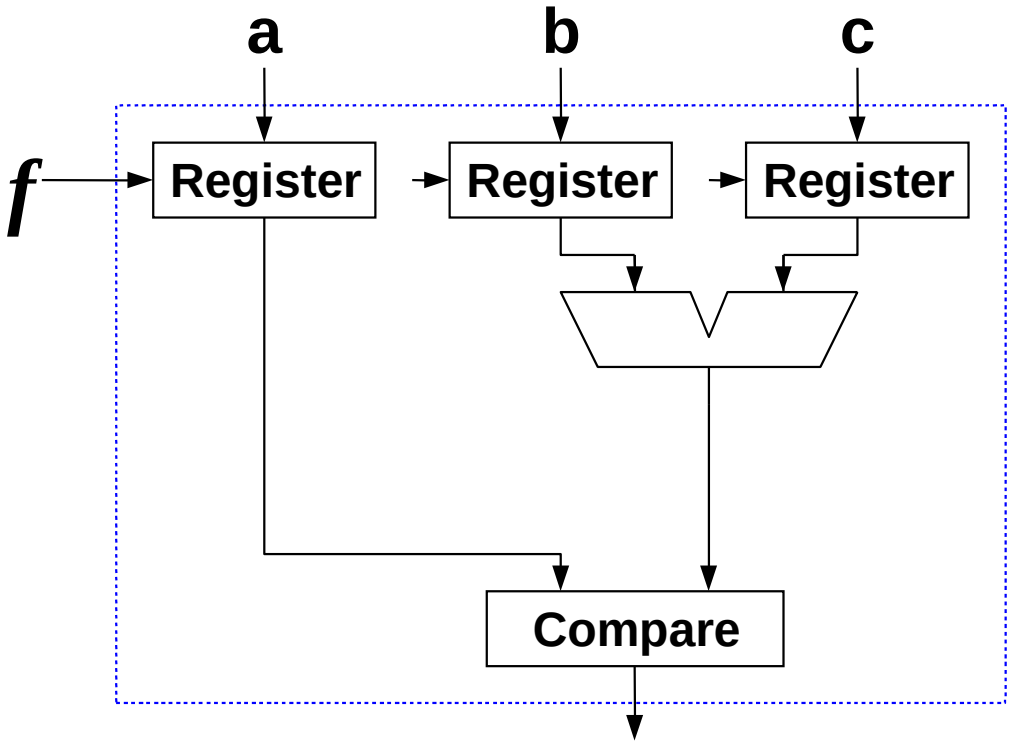
$$P_{pipe} = f \times 1.1C \times (0.58V)^2 = 0.37P$$

Parallelization only

$$P_{par} = 0.5f \times 2.15C \times (0.58V)^2 = 0.36P$$

Pipe- and Parallelization

$$P_{par,pipe} = 0.5f \times 2.35C \times (0.4V)^2 = 0.19P$$



Retiming

Chapter 4.

What is retiming?

Retiming is a transformation technique used to change the location of delay elements in a circuit without affecting the characteristics of the circuit.

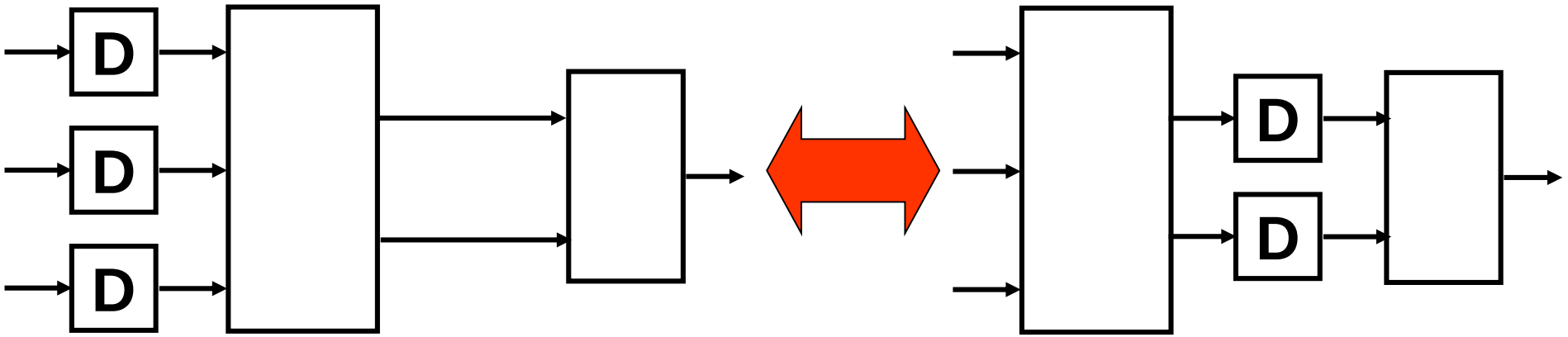
Retiming is the technique of moving the structural location of latches or registers in a digital circuit to improve its performance, area, and/or power characteristics in such a way that preserves its functional behaviour at its outputs.

Can be used for:

- *Reducing the clock period*
- *Reducing the number of registers*
- *Reducing the power consumption*
- *Logic synthesis (not in this course)*

The lower bound on the clock period of the circuit can be achieved by retiming the circuit.

Retiming is about moving delays!



Delays can be moved from ALL inputs to ALL outputs

Reduce Critical Path

- **faster**
- **reduced power consumption**

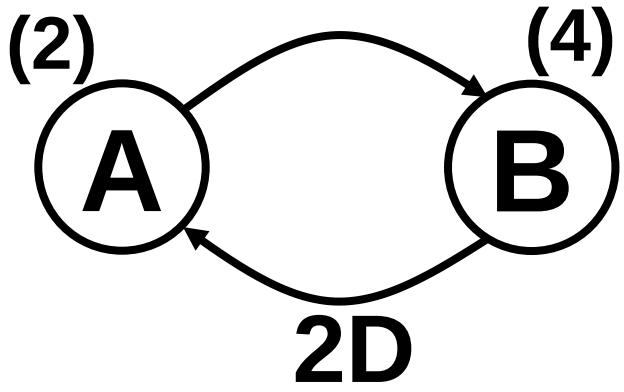
Reduced number of Register

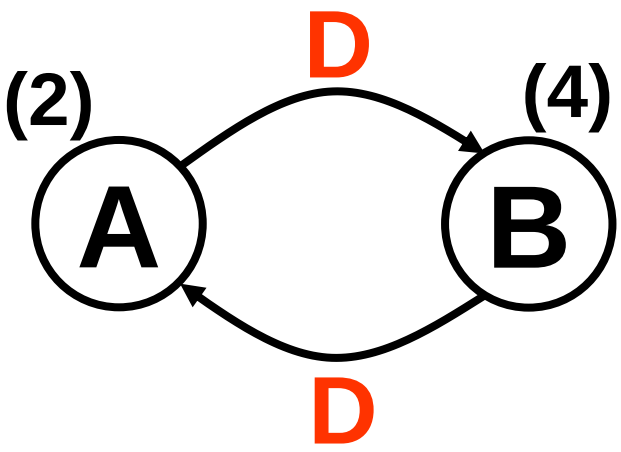
Retiming

Loop bound = $\frac{T_j}{W_j}$ — T_j: loop computation time; W_j: number of delays in the loop

Retiming does not change

- **delay in loop**
- **the iteration bound** $T_\infty = \max_{l \in L}\left\{\frac{t_l}{w_l}\right\}$





Critical path = 6

Loop bound = 6/2 = 3

Critical path = 4

Loop bound = 6/2 = 3

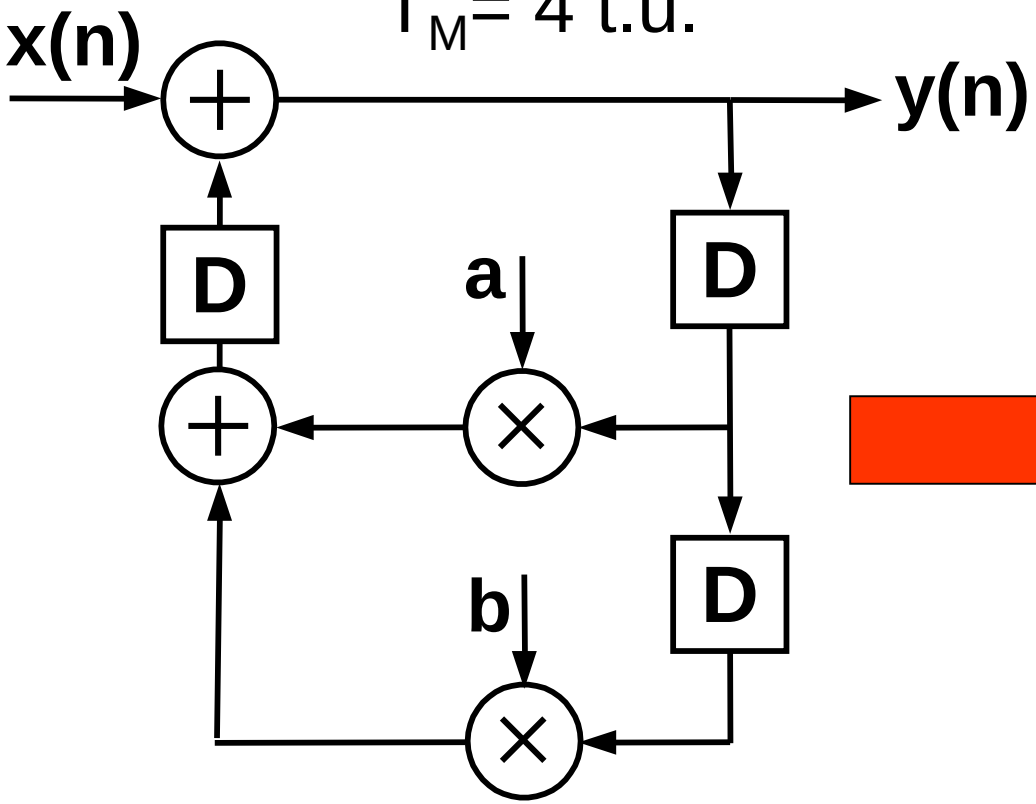
...but it changes the critical path!

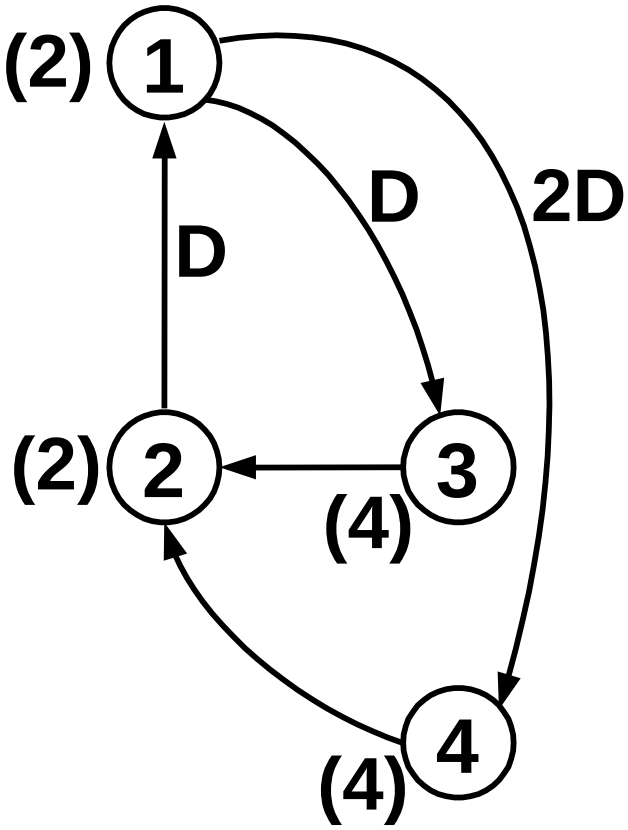
Overview Retiming

- **Cutset retiming and pipelining** (visual retiming)
- **Systolic transformation**
- **Retiming Formulation**
- **Retiming for clock period minimization**

Ex. Cutset Retiming

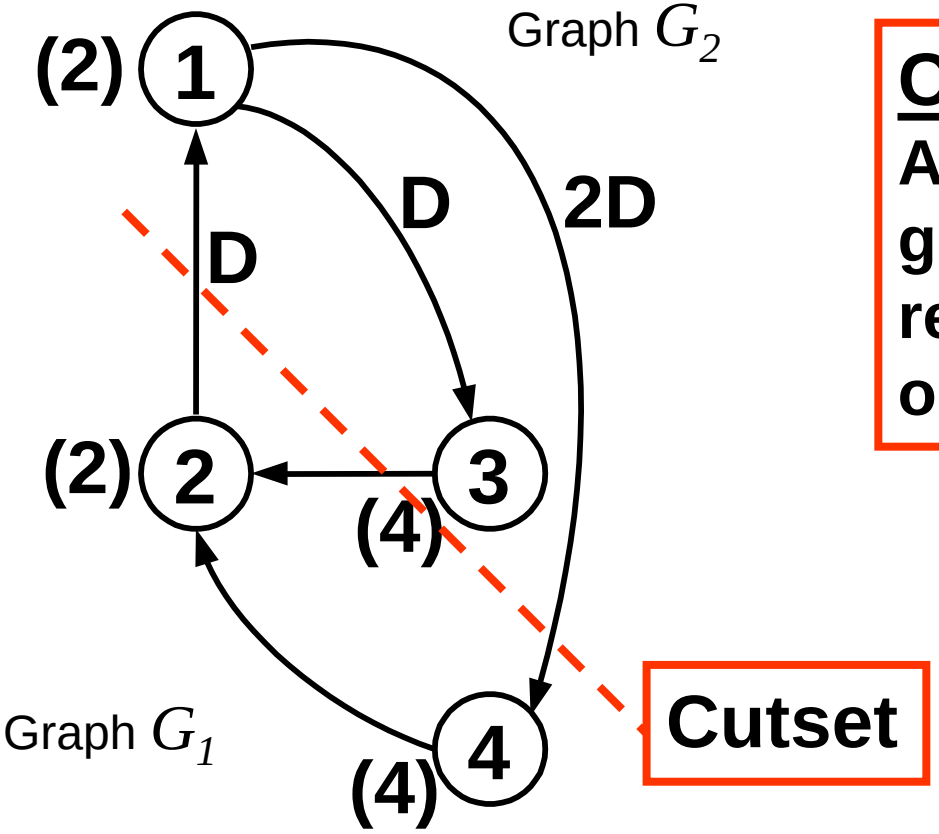
T_A= 2 t.u.
T_M= 4 t.u.





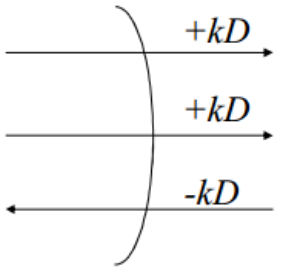
Ex. Cutset Retiming

Cutset: A set of edges that if removed, or cut, results in two disjoint graphs.



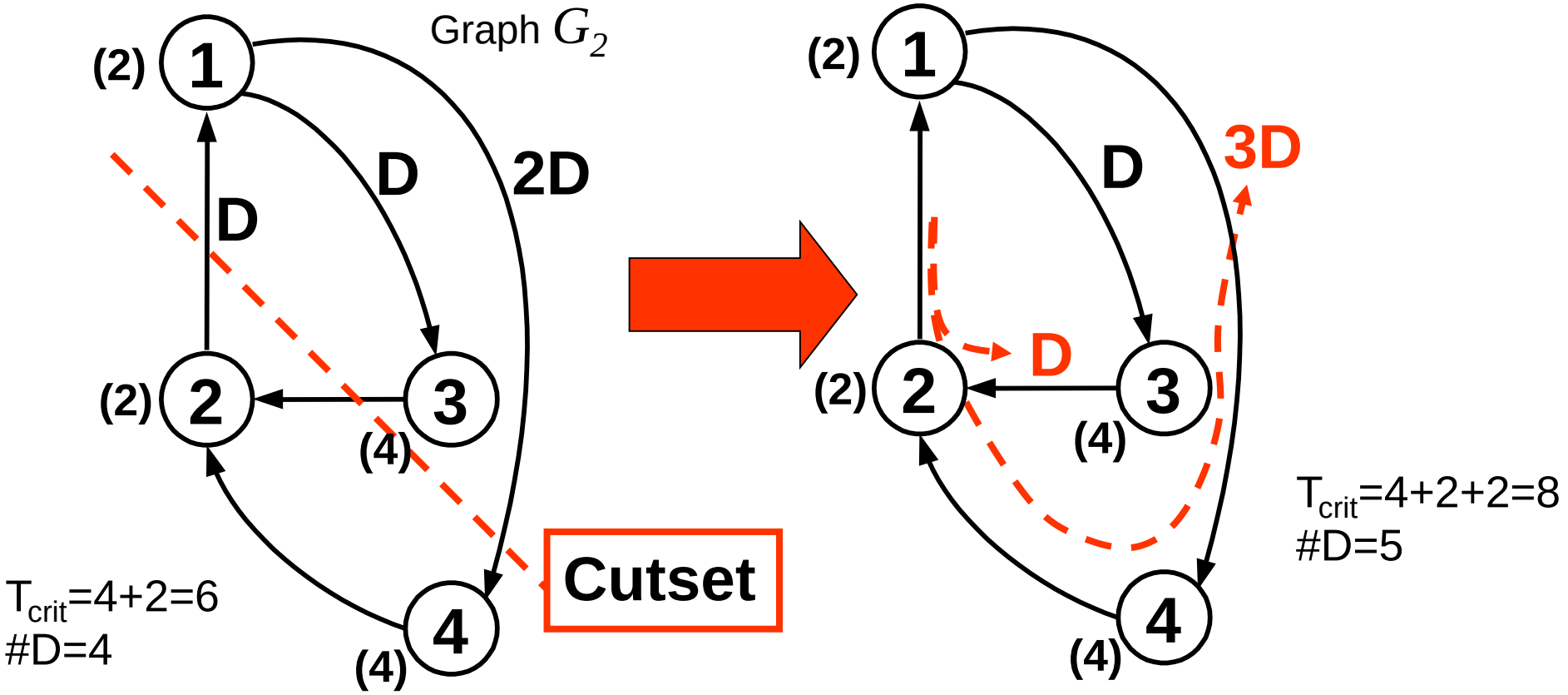
Cutset Retiming

Add *k* delays to edges going one way and remove *k* delays from ones going the other.



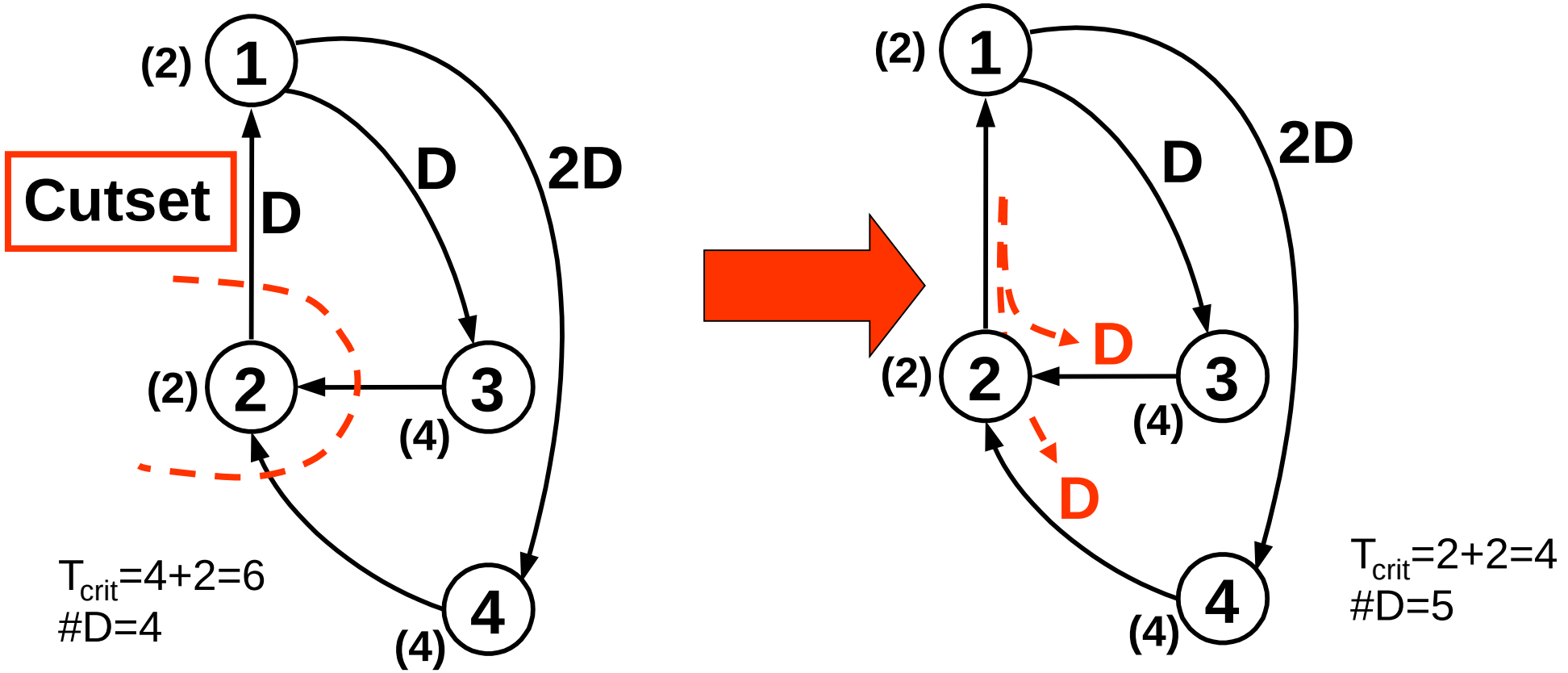
Ex. Cutset Retiming

Cutset Retiming: Add delays to edges going one way and remove from edges going the other.



Ex. Node Cutset Retiming

Node Retiming: Cutset around one node.

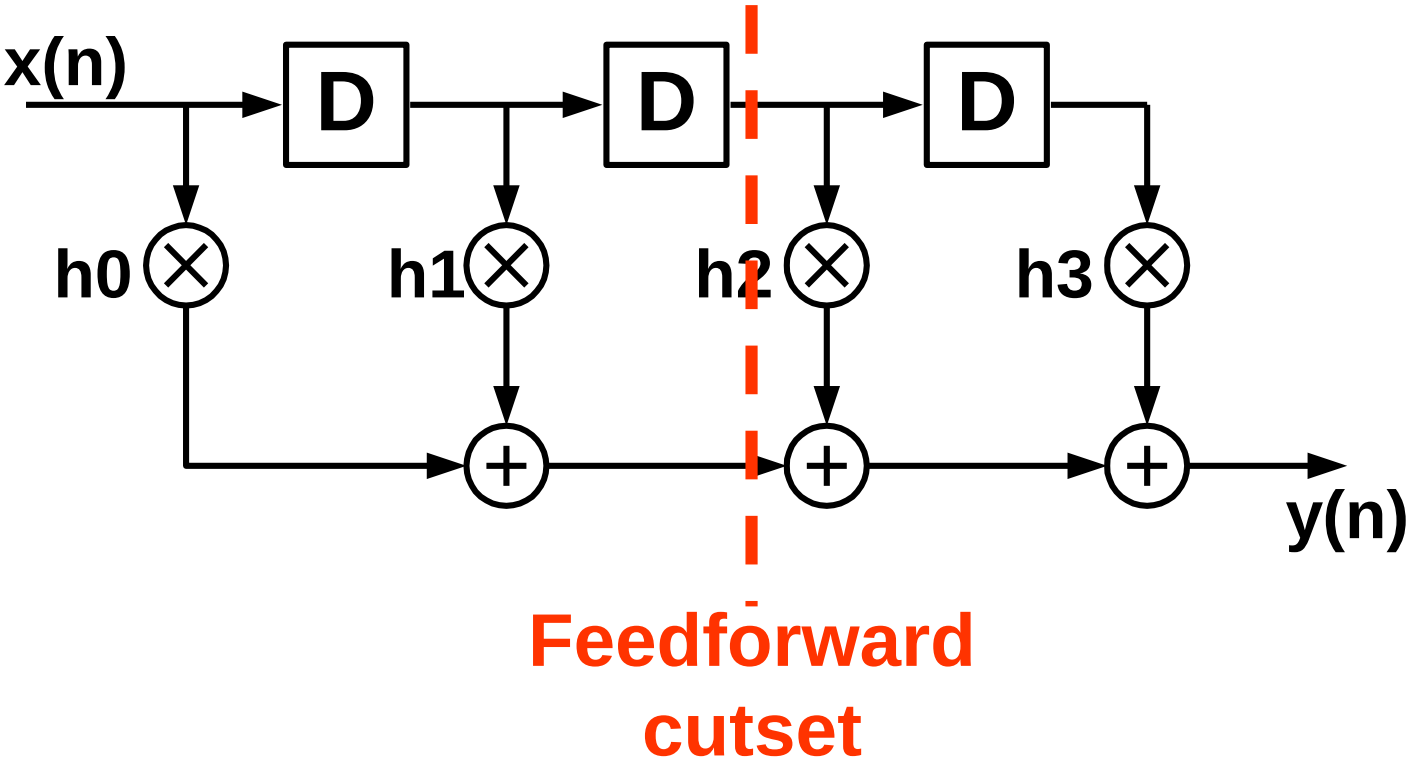


Pipelining

- Pipelining is a special case of cutset retiming where there are no edges in the cutset from G_2 to G_1, i.e., pipelining applies to graphs without loops.
- Retiming is a generalization of pipelining
- Pipelining is equivalent to introducing delays at the input followed by retiming

Pipelining – Feedforward Cutset

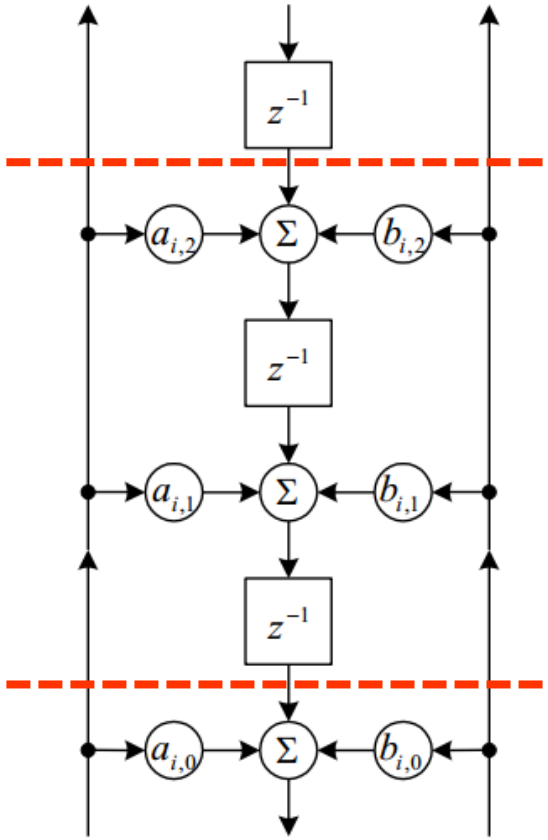
Pipelining is a special case of cutset retiming: Placing delays at feedforward cutsets.



Overview Retiming

- Cutset retiming and pipelining
- Systolic transformation
- Retiming Formulation
- Retiming for clock period minimization

Systolic Transformation



Eliminating global broadcasting

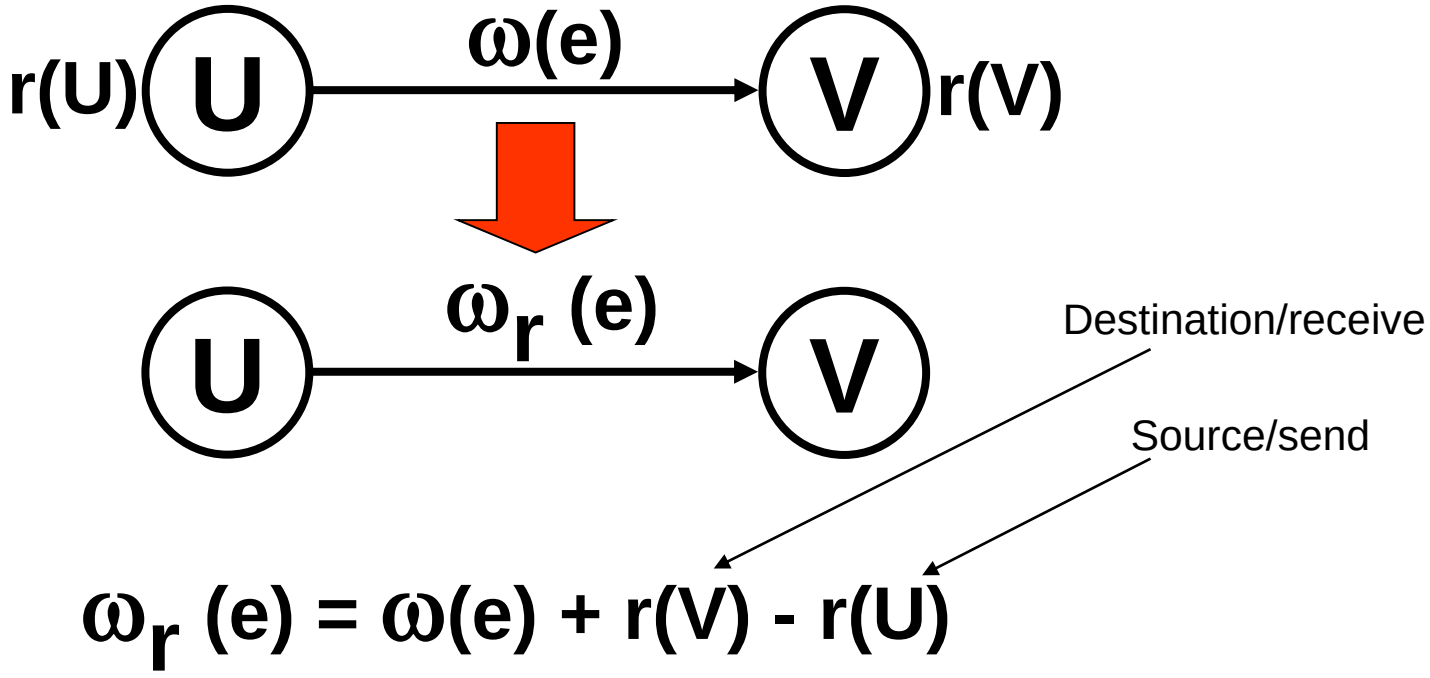
Overview Retiming

- Cutset retiming and pipelining
- Systolic transformation
- Retiming Formulation
- Retiming for clock period minimization

Retiming Formulation

ω(e) = weight of edge e = # of delays

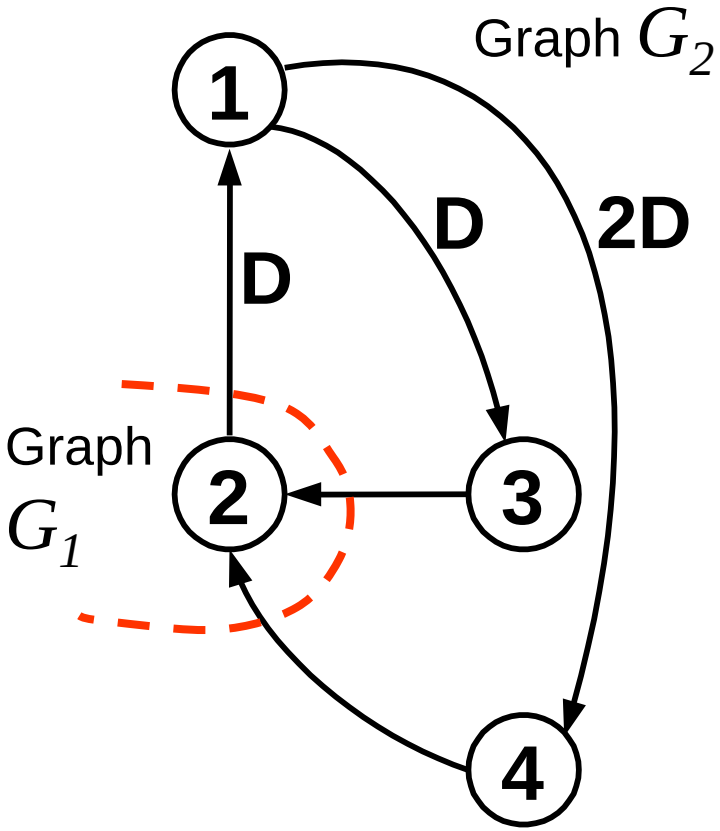
r(x) = retiming values



$$\omega_r(e) = \omega(e) + r(V) - r(U)$$

Valid retiming if all $\omega_r(e) >= 0$ for all edges!

Retiming Formulation for a Node

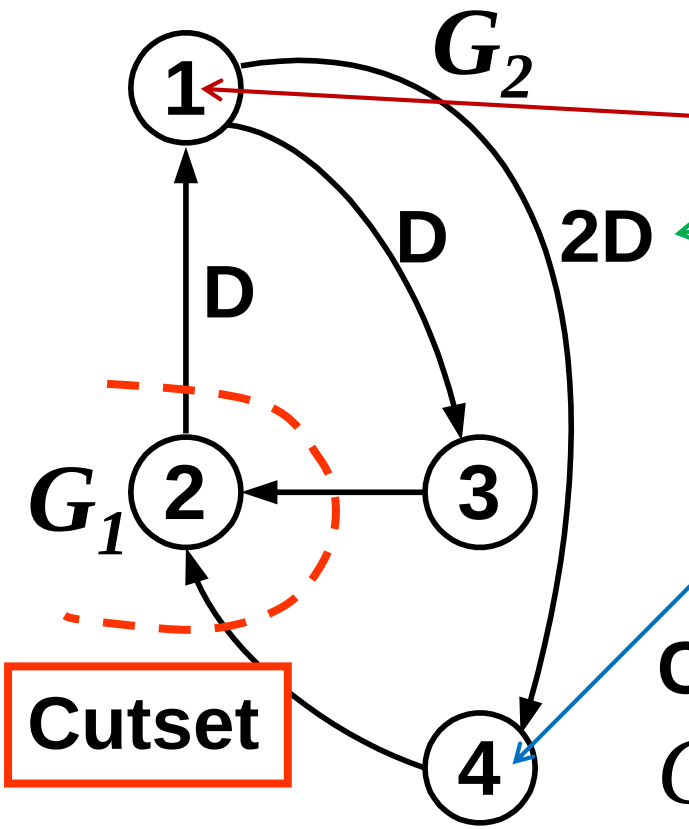


Each node in G_1 has retiming value j and each node in G_2 has retiming value $j+k$

Any value j results in the same retimed graph.

Valid retiming only if all $\omega_r(e) >= 0$

Node Retiming with Formulation



Original weights (ω(e))

$$1 \rightarrow 3 = 1$$

$$1 \rightarrow 4 = 2$$

$$2 \rightarrow 1 = 1$$

$$3 \rightarrow 2 = 0$$

$$4 \rightarrow 2 = 0$$

Choose any retiming values

$G_2 = 0$ ⇨ r(1)=r(3)=r(4)=0

$G_1 = 1$ ⇨ r(2)=1

Node Retiming with Formulation

Original weights

$$1 \rightarrow 3 = 1$$

$$1 \rightarrow 4 = 2$$

$$2 \rightarrow 1 = 1$$

$$3 \rightarrow 2 = 0$$

$$4 \rightarrow 2 = 0$$

$$\omega_r(e) = \omega(e) + r(V) - r(U)$$

Receive Send

$$1 \rightarrow 3 = 1 + 0 - 0 = 1$$

$$1 \rightarrow 4 = 2 + 0 - 0 = 2$$

$$2 \rightarrow 1 = 1 + 0 - 1 = 0$$

$$3 \rightarrow 2 = 0 + 1 - 0 = 1$$

$$4 \rightarrow 2 = 0 + 1 - 0 = 1$$

Retiming values

r(1)=0, r(2)=1, r(3)=0 and r(4)=0

Node Retiming with Formulation (3)

Retimed weights

$$1 \rightarrow 3 = 1$$

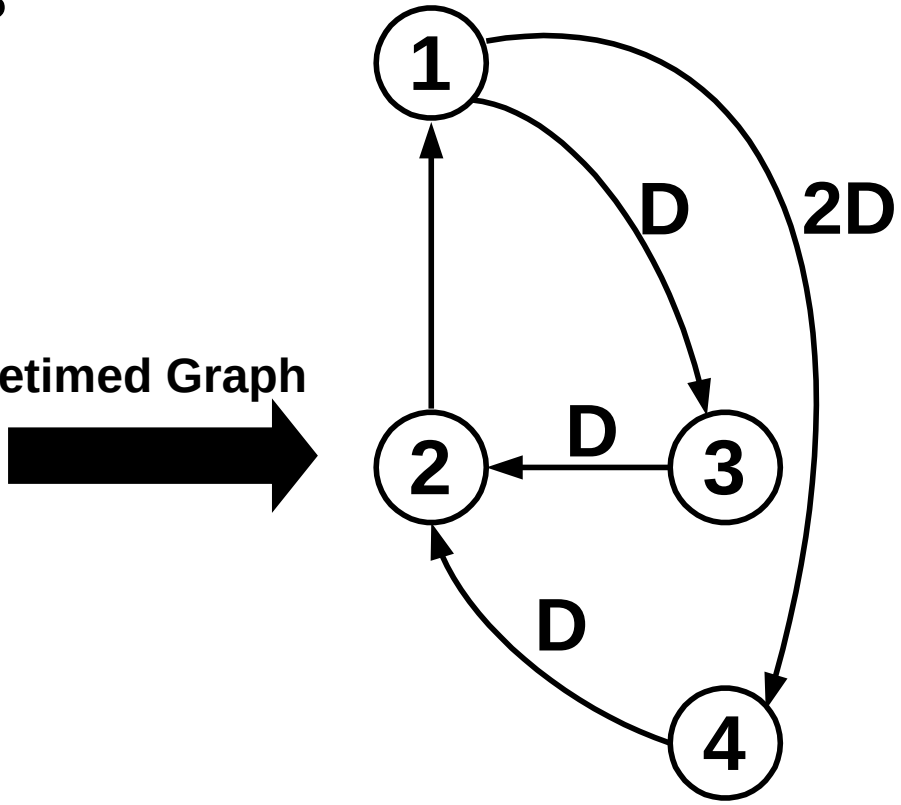
$$1 \rightarrow 4 = 2$$

$$2 \rightarrow 1 = 0$$

$$3 \rightarrow 2 = 1$$

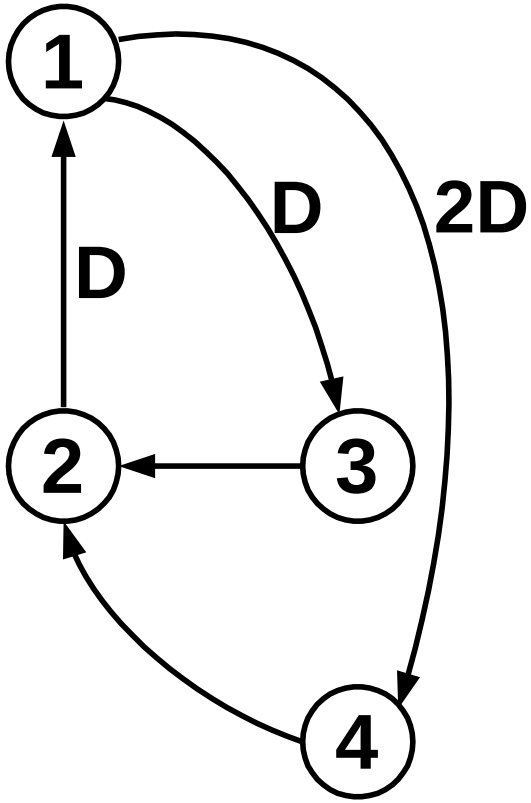
$$4 \rightarrow 2 = 1$$

Retimed Graph

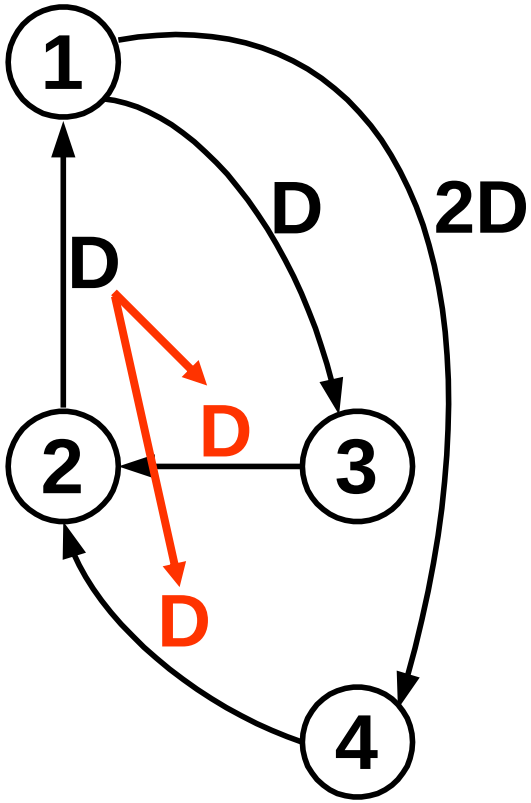


Algorithm for clock period minimization in 4.4.2

Node Retiming with Formulation (3)



What happened is that delays have been moved “around”



Overview Retiming

- Cutset retiming and pipelining
- Systolic transformation
- Retiming Formulation
- Retiming for clock period minimization

Retiming for Minimizing Clock Period

- Note that retiming will NOT alter the iteration bound T_∞.
- Iteration bound is the theoretical minimum clock period to execute the algorithm.
- Let edge *e* connect node *u* to node *v*. If the node computing time t(u) + t(v) > T_∞, then clock period T > T_∞. For such an edge, we require that $w_r(e) \geq 1$
- In other words, for any possible critical path in the DFG that is larger than T_∞, we require $w_r(e) \geq 1$.

Formula: Retiming for Minimizing Clock Period

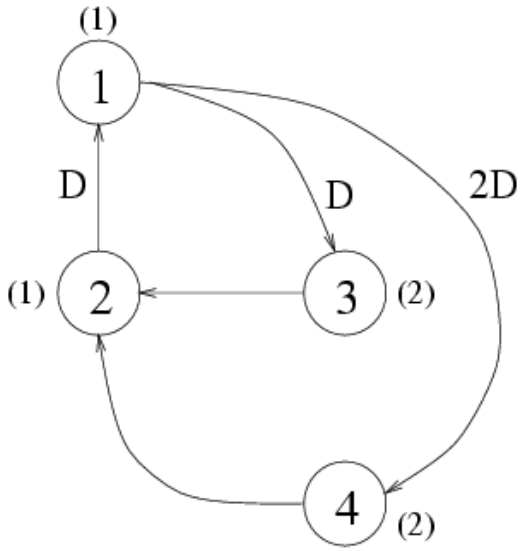
(1) Check that $t(u) + t(v) \geq T_\infty$ then
$w_r(e) \geq 0$ *if* $t(u)+t(v)=T_\infty$
$w_r(e) \geq 1$ *if* $t(u)+t(v)>T_\infty$

(2) Solve $w_r(e_{uv}) = w(e) + r(v) - r(u)$

(3) Check that the retime values are valid!

Ex. Retiming for Minimizing Clock Period

Check that $t(u) + t(v) \geq T_\infty$ then
$w_r(e) \geq 0$ if $t(u)+t(v)=T_\infty$
$w_r(e) \geq 1$ if $t(u)+t(v)>T_\infty$



$T_\infty = 2$

$w_r(e_{21}) \geq 0$, since $t(2)+t(1) = 2 = T_\infty$
$w_r(e_{13}) \geq 1$, since $t(1)+t(3) = 3 > T_\infty$
$w_r(e_{14}) \geq 1$, since $t(1)+t(4) = 3 > T_\infty$
$w_r(e_{32}) \geq 1$, since $t(3)+t(2) = 3 > T_\infty$
$w_r(e_{42}) \geq 1$, since $t(4)+t(2) = 3 > T_\infty$

Use eq. $w_r(e_{uv}) = w(e) + r(v) - r(u)$ then

$w(e_{21}) + r(1) - r(2) = 1 + r(1) - r(2) \geq 0$
$w(e_{13}) + r(3) - r(1) = 1 + r(3) - r(1) \geq 1$
$w(e_{14}) + r(4) - r(1) = 2 + r(4) - r(1) \geq 1$
$w(e_{32}) + r(2) - r(3) = 0 + r(2) - r(3) \geq 1$
$w(e_{42}) + r(2) - r(4) = 0 + r(2) - r(4) \geq 1$

Ex. Retiming for Minimizing Clock Period

- Since the retimed graph G_r remain the same if all node retiming values are added by the same constant. We thus can set r(1) = 0.

- The inequalities then become:

  $1 - r(2) \geq 0$ or $r(2) \leq 1$

  $1 + r(3) \geq 1$ or $r(3) \geq 0$

  $2 + r(4) \geq 1$ or $r(4) \geq -1$

  $r(2) - r(3) \geq 1$ or $r(3) \leq r(2) - 1$

  $r(2) - r(4) \geq 1$ or $r(2) \geq r(4) + 1$

However for larger systems…

- Since

  $1 \geq r(2) \geq r(3) + 1 \geq 0 + 1 = +1$

  one must have $r(2) = +1$.

- This implies $r(3) \leq 0$.

  But we also have $r(3) \geq 0$.

  Hence $r(3)=0$.

- These leave $-1 \leq r(4) \leq 0$.

- Hence the two sets of solutions are:

  $r(0) = r(3) = 0$, $r(2) = +1$, and $r(4) = 0$ or (-1).

Solving a Systems of Inequalities

Given a systems of inequalities:

$$r(i) - r(j) \leq k;\ 1 \leq i,j \leq N$$

Construct a constraint graph:

1. Map each r(i) to node i. Add a node N+1.
2. For each inequality
   $$r(i) - r(j) \leq k,$$
   draw an edge e_{ji}
   such that $w(e_{ji}) = k$.
3. Draw N edges $e_{N+1,i} = 0$.

The system of inequalities has a solution if and only if the constraint graph contains **no negative cycles**

If a solution exists, one solution is where r_i is the minimum length path from the node N+1 to the node *i*.

Shortest path algorithms: (Applendix A)

- Bellman-Ford algorithm
- Floyd-Warshall algorithm

See chapter 4.4.2

Drawbacks with Retiming

- The state encoding of the circuit may be destroyed, making testing and verification more difficult.
- Some retimed circuits may require complicated initialization logic to have the circuit start in a special initial state.
- Retiming changes the circuit's topology which have consequences in other logical and physical synthesis steps that make design closure more difficult.

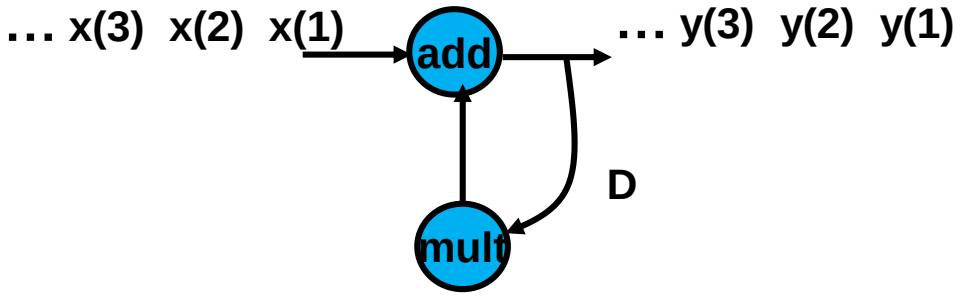
Time Scaling using Slow Down and Retiming

- A slow down process is often used in combination with the cutset retiming process

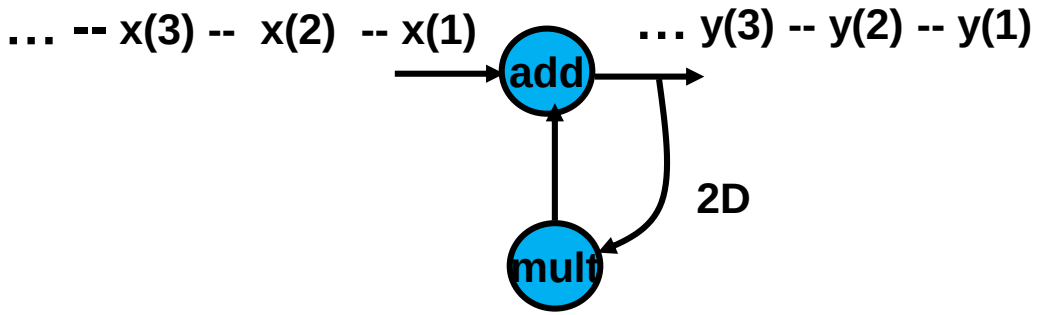
- Replace each delay in a DFG with N delays
- N-slow DFG N-1
- Null operations (or 0 samples) must be interleaved to preserve the functionality

Time Scaling (Slow Down)

- Transform each delay element (register) D to *N*D Reduce the sample frequency by *N*-fold to slow down the computation *N* times.



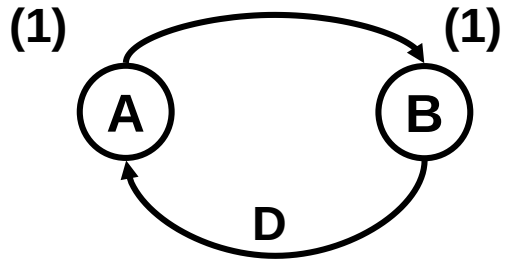
- In slow down the clock cycle time remains unchanged. Only the sampling time is increased.



This provides opportunity for retiming, and interleaving.

Slow Down by k

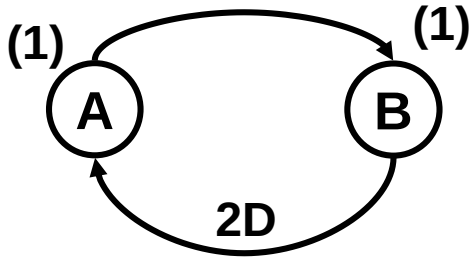
Replace each D by kD



Clock	
0	A0 → B0
1	A1 → B1
2	A2 → B2

T_{clk}= 2t.u.
T_{iter}= 2t.u.

After 2-slow transformation



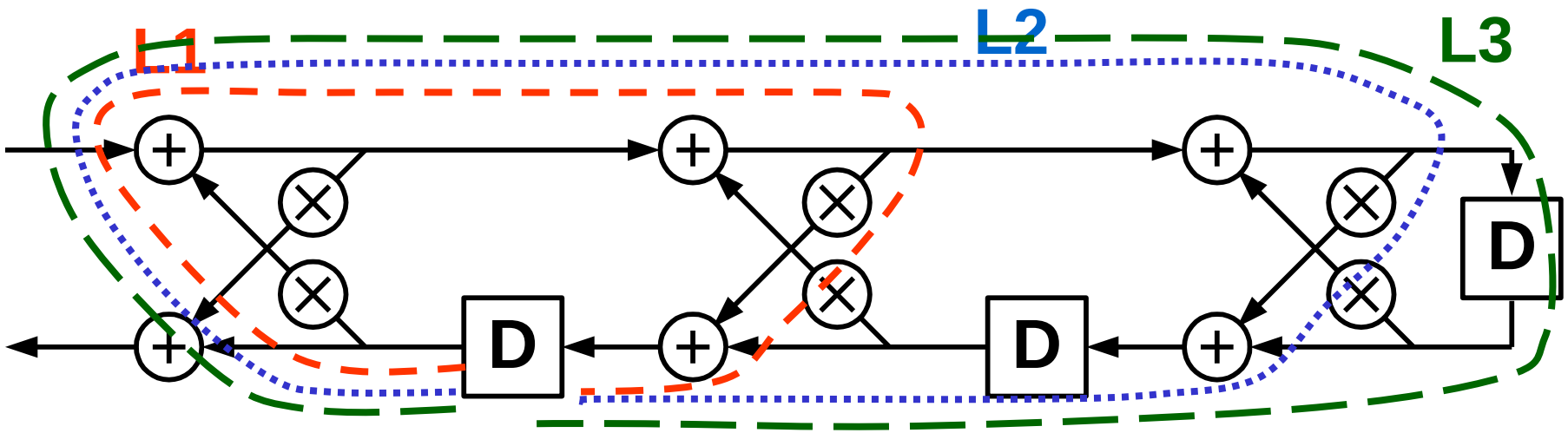
Clock	
0	A0→B0
1	
2	A1→B1
3	
4	A2→B2

T_{clk}= 2t.u.
T_{iter}=2×2t.u.
=4t.u.

- Input new samples every alternate cycles.
- null operations account for odd clock cycles.
- Hardware utilized only 50% time

Retiming Example: 3-stage Lattice Filter

Loop Bounds



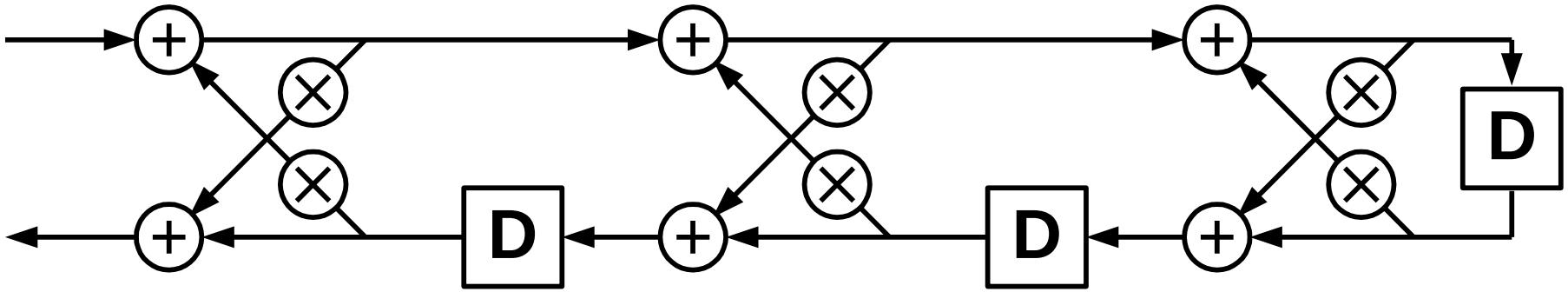
$$T_{L1} = \frac{3T_{Add} + 2T_{Mult}}{1}$$

$$T_{L3} = \frac{5T_{Add} + T_{Mult}}{3}$$

$$T_{L2} = \frac{5T_{Add} + 2T_{Mult}}{2}$$

Example: 3-stage Lattice Filter

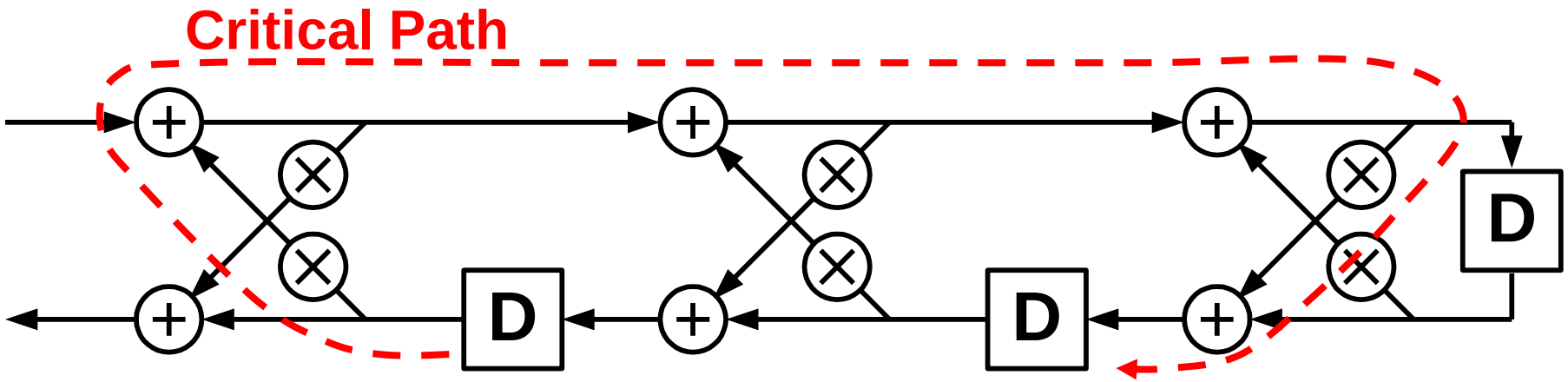
Iteration Bound



$$T_\infty = \max\{T_{L1}, T_{L2}, T_{L3}, \ldots\} = \frac{3T_{Add} + 2T_{Mult}}{1}$$

Example: 3-stage Lattice Filter

Critical Path

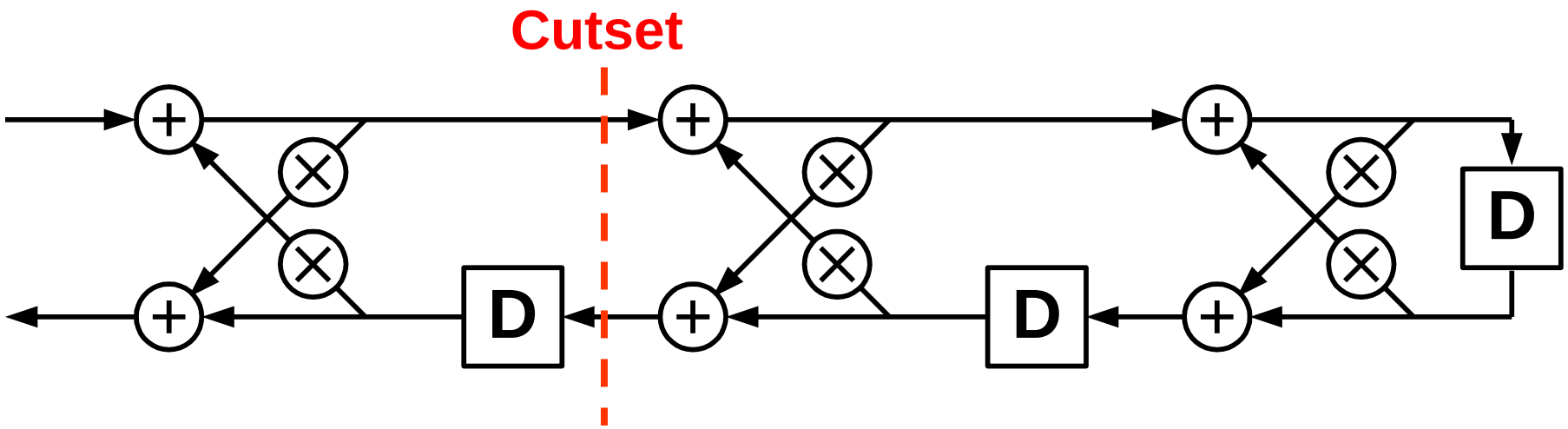


Critical Path = (N+1) Adders +2 Mult

N = Number of Stages

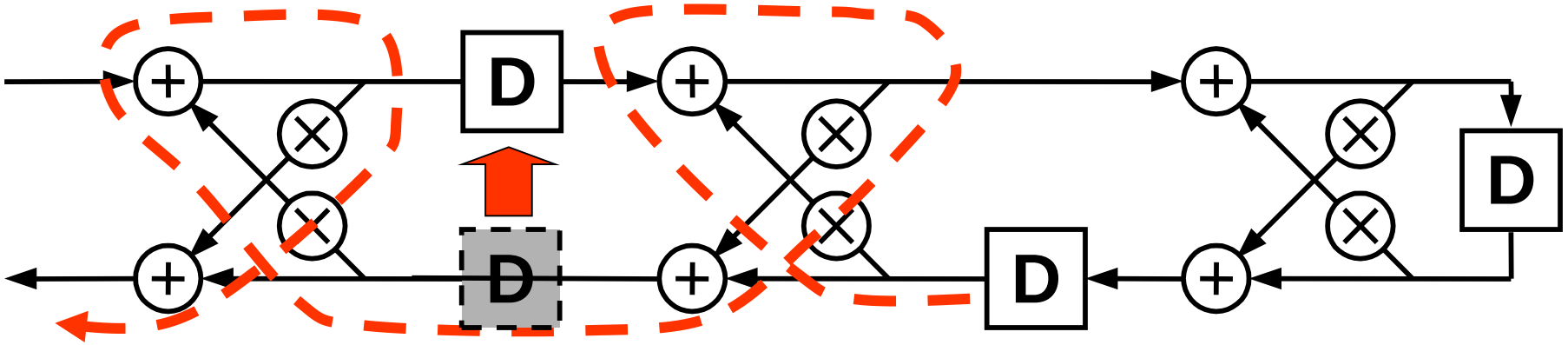
Example: 3-stage Lattice Filter

Cutset Retiming



Example: 3-stage Lattice Filter

Cutset Retiming



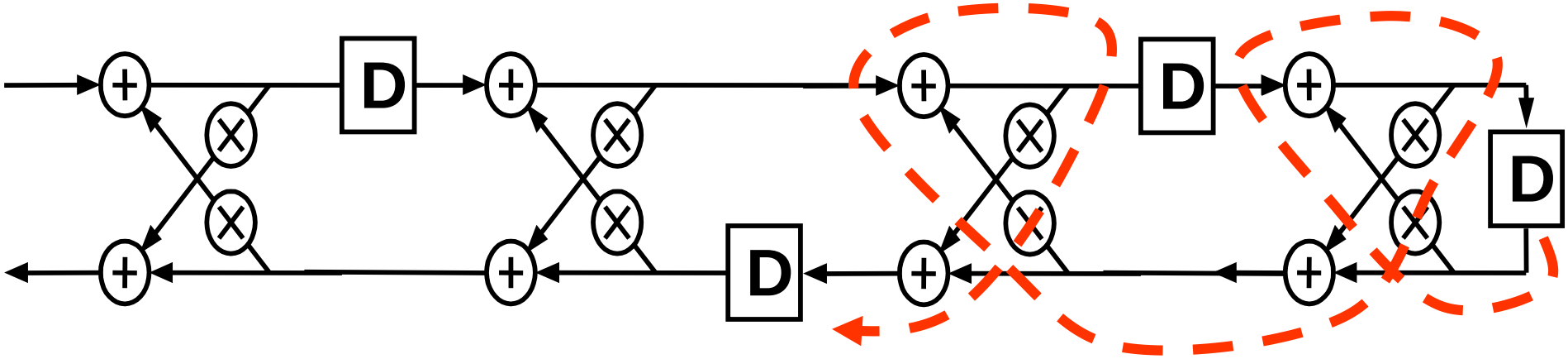
Critical Path = 4 Adders + 4 Mult
Independent of N = nr. of stages in filter

compared to

Original Critical Path = (N+1) Adders +2 Mult

Worse for Low N. Trade-off depends on T_M and T_A

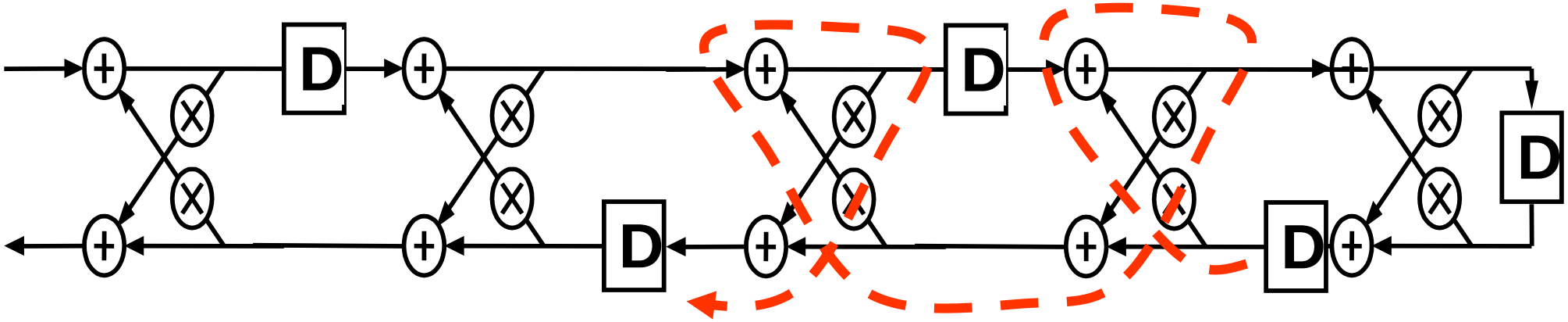
Compare: Cutset Retiming of a 4-stage Lattice Filter



Move every second delay

Critical Path = 4Adders + 4Mult

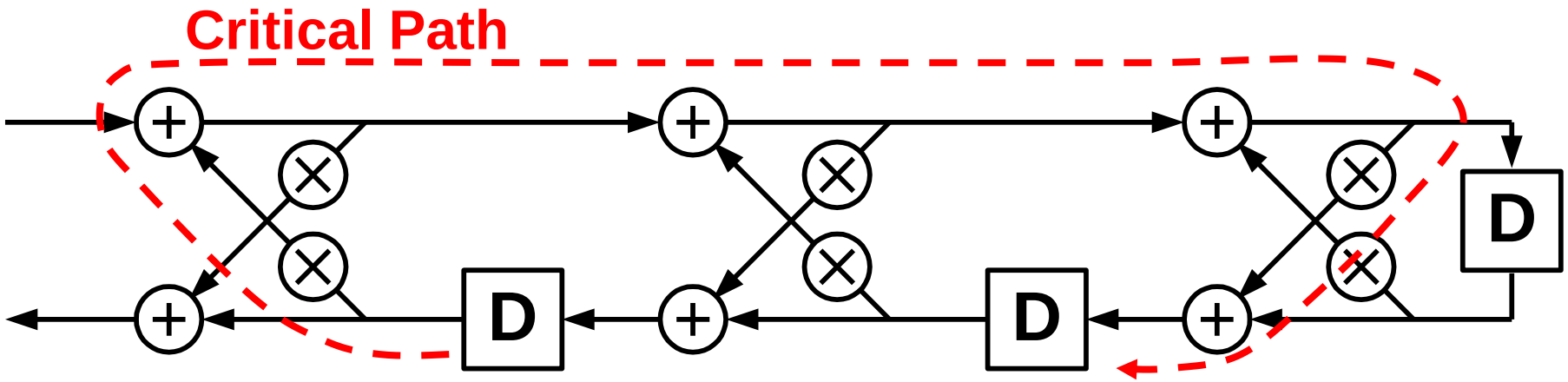
Compare: Cutset Retiming of a 5-stage Lattice Filter



Critical Path = 4Adders +4Mult

Example: 3-stage Lattice Filter

Cutset Retiming with Slowdown

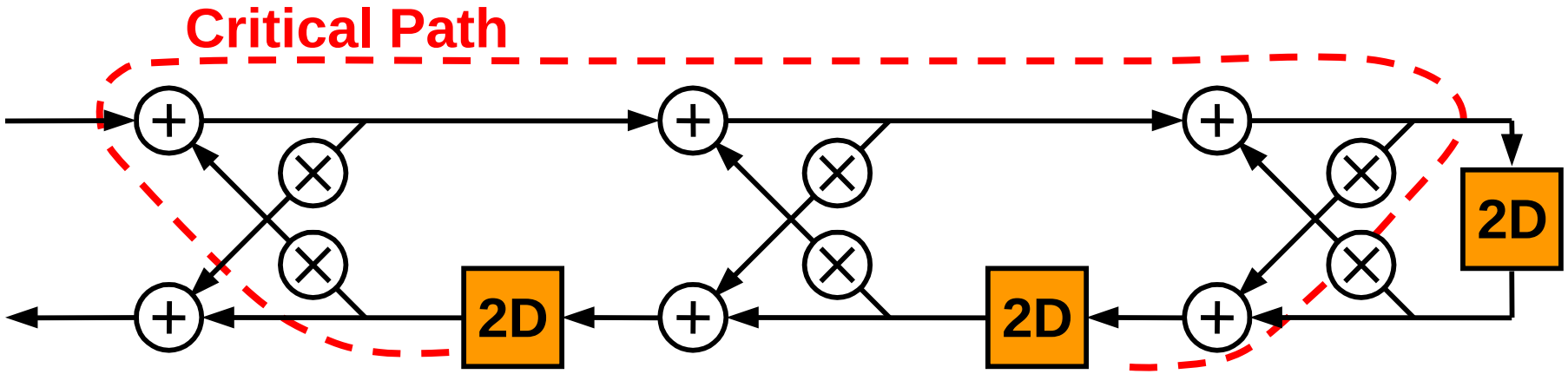


Critical Path = (N+1) Adders +2 Mult

N = Number of Stages

Example: 3-stage Lattice Filter

Slowdown by factor 2



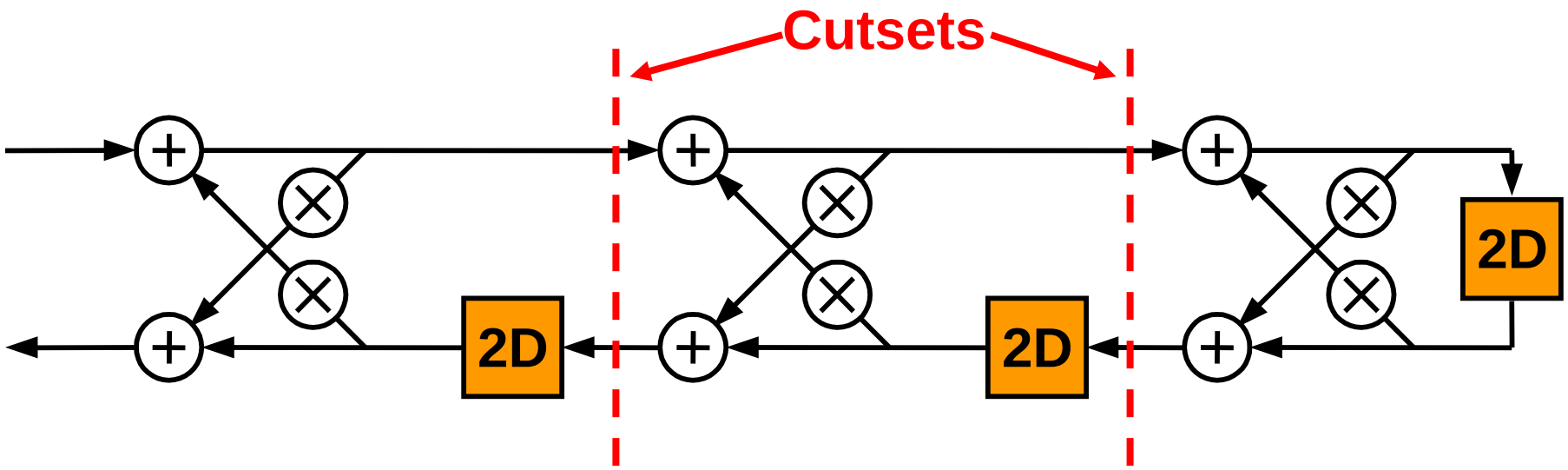
Critical Path has not changed. Same $f_{clk.}$

Insertion of 0 samples to preserve behavior!

x_0 x_0 x_1 x_1 x_2 x_2 x_3 x_3 . . . or
x_0 0 x_1 0 x_2 0 x_3 0 . . .

Example: 3-stage Lattice Filter

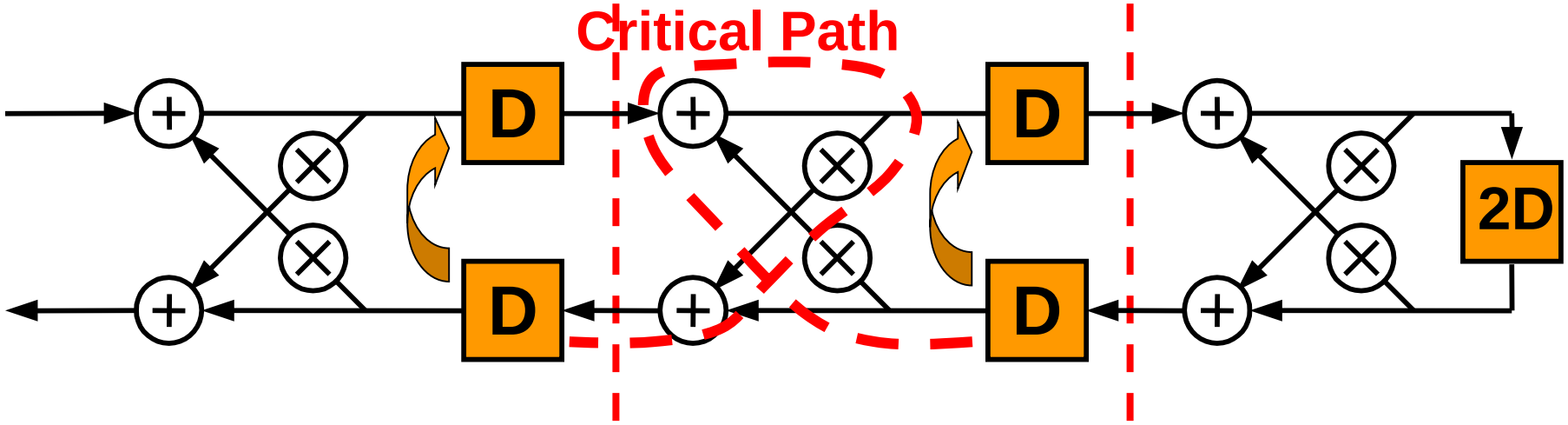
Slowdown by factor 2



Add delays on Edges in one direction and remove in the other

Example: 3-stage Lattice Filter

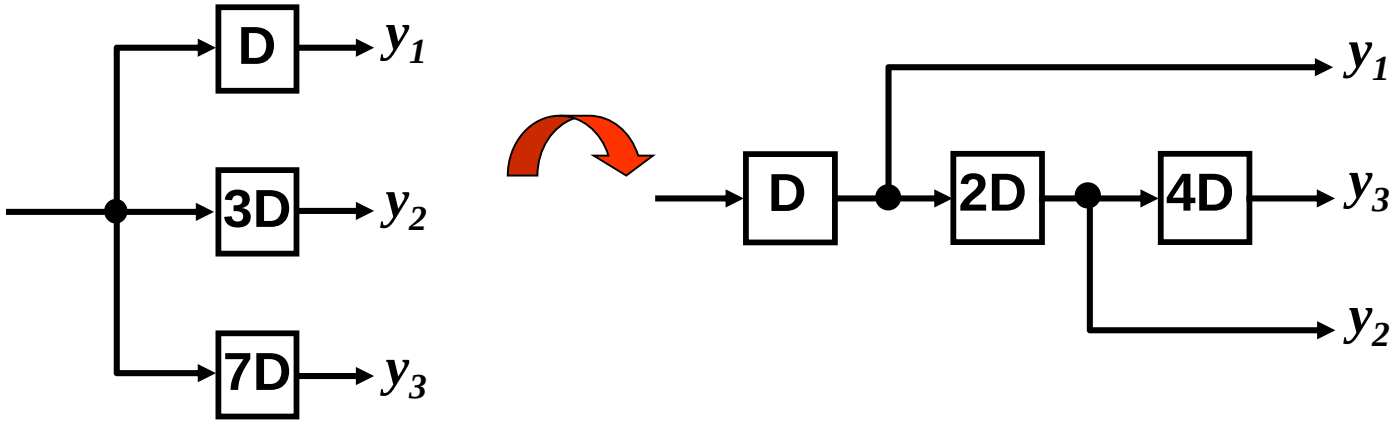
Slowdown by factor 2



Critical Path = 2 Adders +2 Mult

But requires twice the number of clock cycles

Register Minimization



- **Register Sharing**

  When a node has multiple fan-out with different number of delays, the registers can be shared so that only the branch with max. # of delays will be needed.

- **Register reduction through node delay transfer from multiple input edges to output edges (e.g. r(v) > 0)**
- **Should be done only when clock cycle constraint (if any) is not violated.**

Algorithm for register minimization in 4.4.3

End of Retiming