EECS 598 VLSI for Wireless Communication and Machine Learning

DFG, Retiming, Pipelining

Prof. Hun-Seok Kim

hunseok@umich.edu

Slides from Prof. Zhengya Zhang's EECS 598 F13

Algorithm Transform

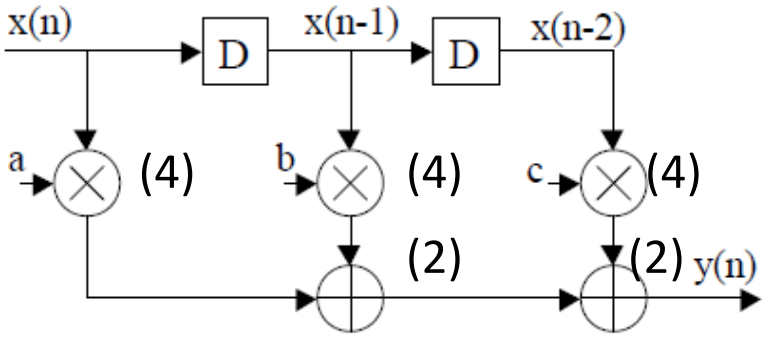
- Algorithm: a set of computational steps needed to achieve a specific functionality
- Architecture: physical manifestation of a computational engine, on which one or more algorithms can be mapped
  - Examples: microprocessor architecture, application-specific integrated circuit (ASIC) architecture
- Algorithm transform: the process of modifying algorithms in order to make them VLSI architecture friendly, and for mapping algorithms to practical VLSI architectures
  - Based on the data-flow diagram (DFG) representation of the algorithm

Representing DSP Algorithms

- Example: 3-tap FIR filter $y(n) = ax(n) + bx(n-1) + cx(n-2)$
  - non-terminating, i.e., $n = 1 \ldots \infty$
- Definitions:
  - **Iteration**: execution of all the computations in the algorithm once
  - Iteration period: the time required for one iteration
  - **Sampling rate** (or **throughput**): number of samples processed per second
  - **Critical path**: the longest path (computation time) between inputs and outputs (in combinational circuits), or the longest path between any two sequential delay elements (in sequential circuits)
  - Minimum **clock period**: determined by the critical path
  - **Latency**: the difference between the time an output is generated and the time at which its corresponding input was received by the system
    - Combinational circuit: latency is in terms of absolute time units
    - Sequential circuit: latency is in terms of number of clock cycles

Block Diagram

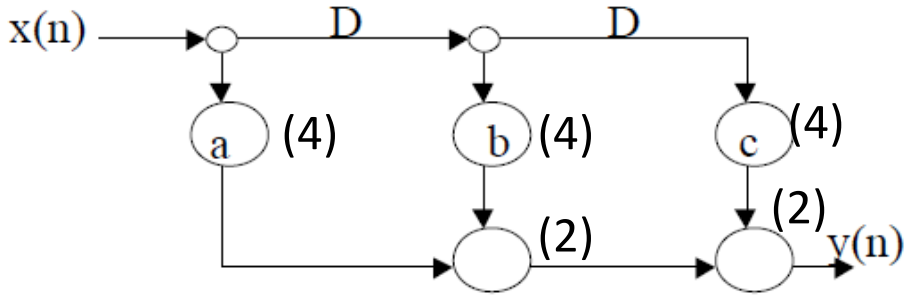
- Example: 3-tap FIR filter $y(n) = ax(n) + bx(n-1) + cx(n-2)$
  - non-terminating, i.e., $n = 1 \ldots \infty$



  - Consists of functional blocks connected with directed edges
  - Represents data flow from its input to output
  - Clock period is lower bounded by the critical path computation time

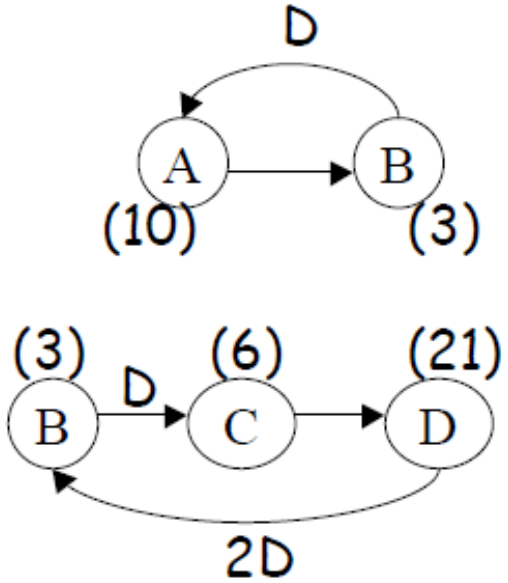
Data-Flow Graph (DFG)

- Example: 3-tap FIR filter $y(n) = ax(n) + bx(n-1) + cx(n-2)$
  - non-terminating, i.e., $n = 1 \ldots \infty$



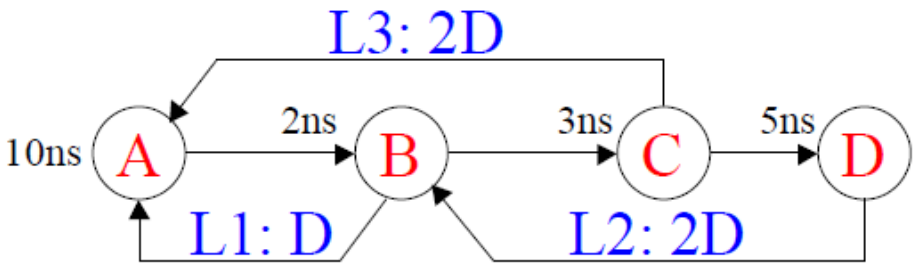
  - Nodes represent computations. Each has a **delay** in absolute time unit.
  - Directed edges represent data paths. Each has a **weight** in clock cycles (with period D).
  - Any node can fire when all inputs are available, and many nodes can fire concurrently
  - DFG is used to derive concurrent implementations onto parallel hardware

Paths and Loops



- Path: $p = B \rightarrow C \rightarrow D$: sequence of nodes connected by arcs
- Path weight: $w(p) = \sum_{e_i} w(e_i)$, path delay: $d(p) = \sum_{U_i} d(U_i)$
- Loop: a path that begins and ends at the same node
- Critical path: the path in the acyclic DFG with the longest delay (critical path delay)
- Critical path delay is the achievable lower-bound on the clock period needed to support a direct-mapped error-free execution of the DFG

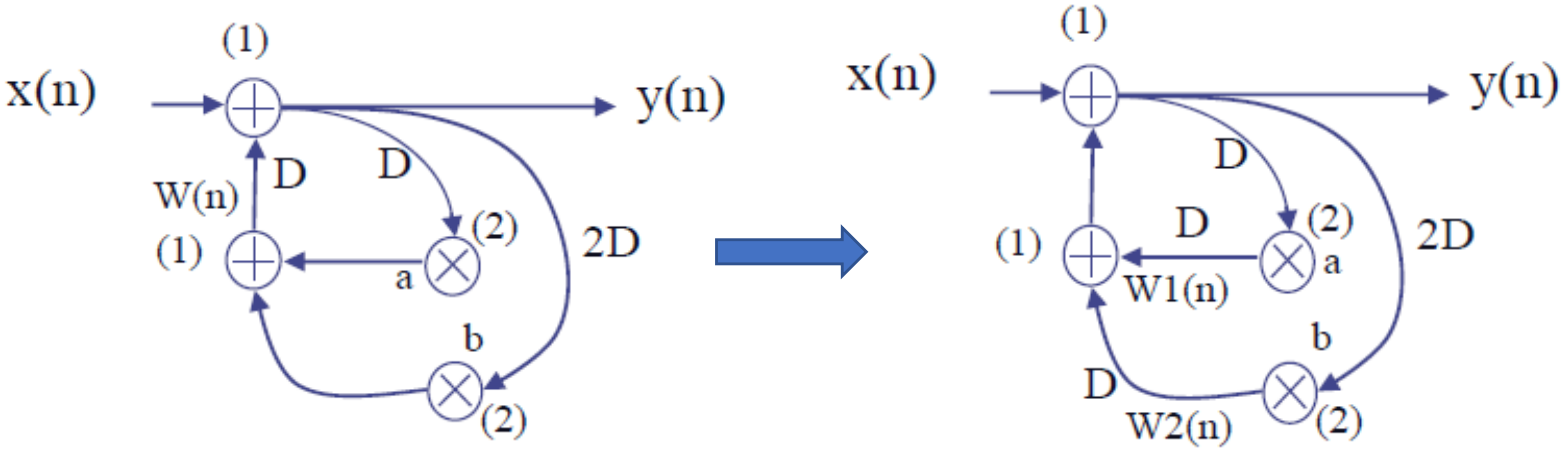
Iteration-Period Bound



- Three loops: L1, L2, L3
- Loop bound: $T_{LB}(l) = \frac{d(l)}{w(l)}$
- Critical loop: loop with the maximum loop bound
- Iteration-period bound (T_{IPB}): the loop bound of the critical loop
- Iteration-period bound is the achievable lower bound on clock period needed to support any error-free execution of the algorithm represented by the DFG

Retiming

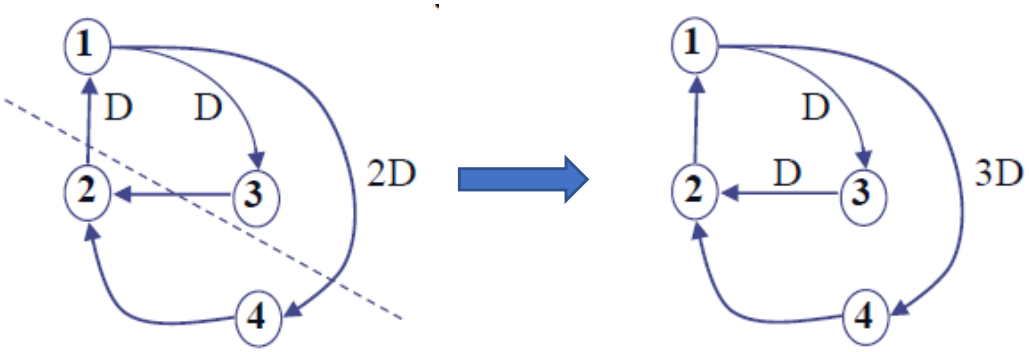
- Retiming: changing locations of delay elements without affecting the input/output characteristics
  - Reduce critical path and clock period (without changing iteration-period bound)
  - Reduce number of registers
  - Reduce power consumption by reducing switching: e.g., place registers at the inputs of nodes with large capacitances



Cutset and Delay Transfer

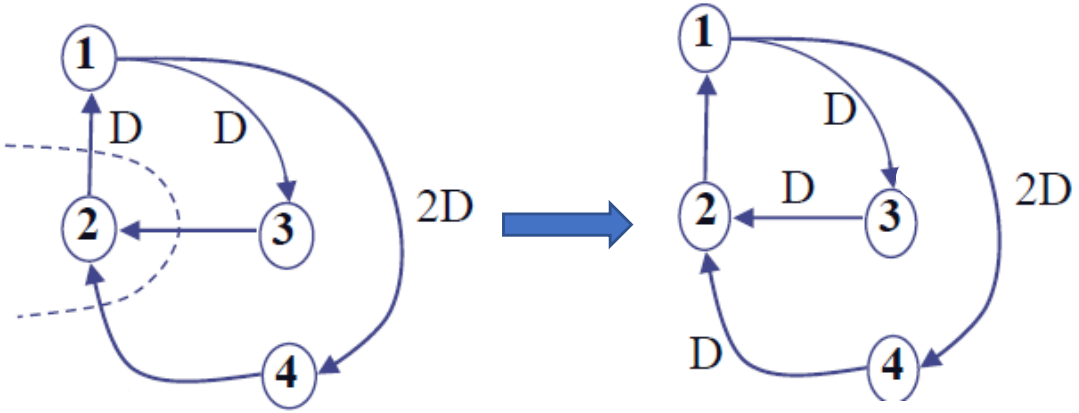
- Cutset
  - Cut: separate the vertices of a graph into two disjoint subsets: G1, G2
  - Cutset: the set of edges whose end points are in different subsets

- Delay transfer
  - Only affect edges in the cutset
  - Add *k* delays to each edge from G1 to G2
  - Remove *k* delays to each edge from G2 to G1



Special Case

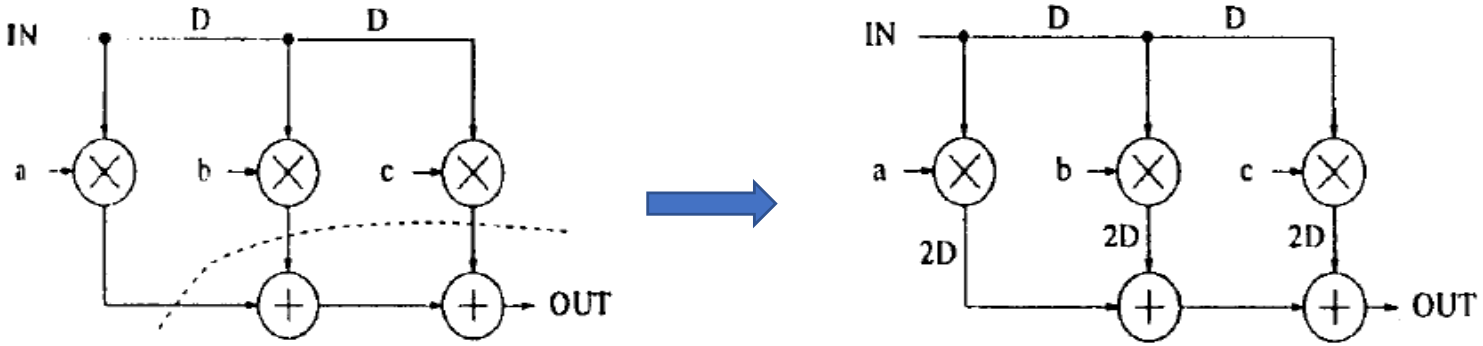
- Single node subgraph



- Remove 1 delay from each outgoing edge and add 1 delay to each incoming edge

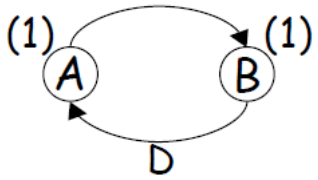
Special Case

- Pipelining

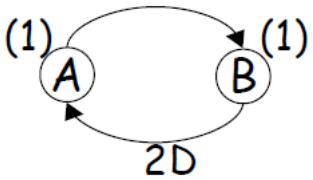


- Edges only going one direction

Delay Scaling



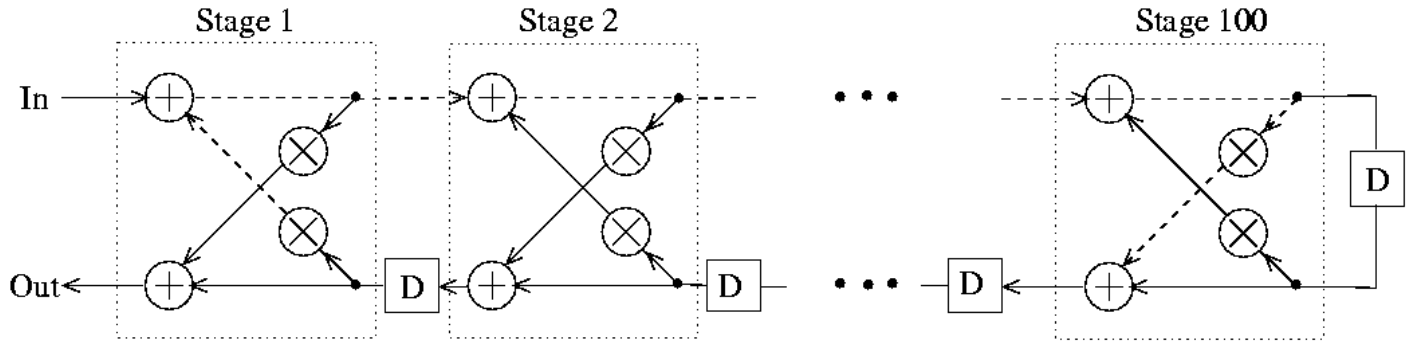
- Delay scaling: replaces each delay element D by αD, and interleave the input stream by α – 1 zeros/null operations
  - Example: replace each D by 2D, input sample every alternate cycle
  - Null operation used for odd clock cycle
  - Hardware utilized only 50% of the time



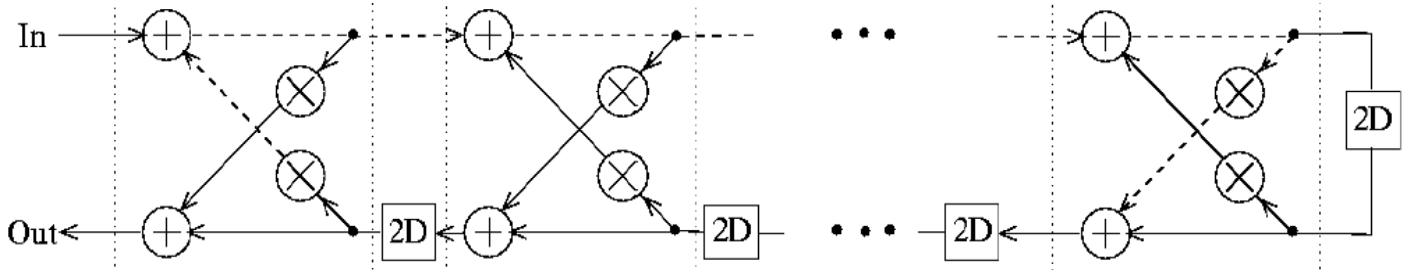
Clock	
0	A0→B0
1	
2	A1→B1
3	
4	A2→B2

Delay Scaling

- 100-stage lattice filter

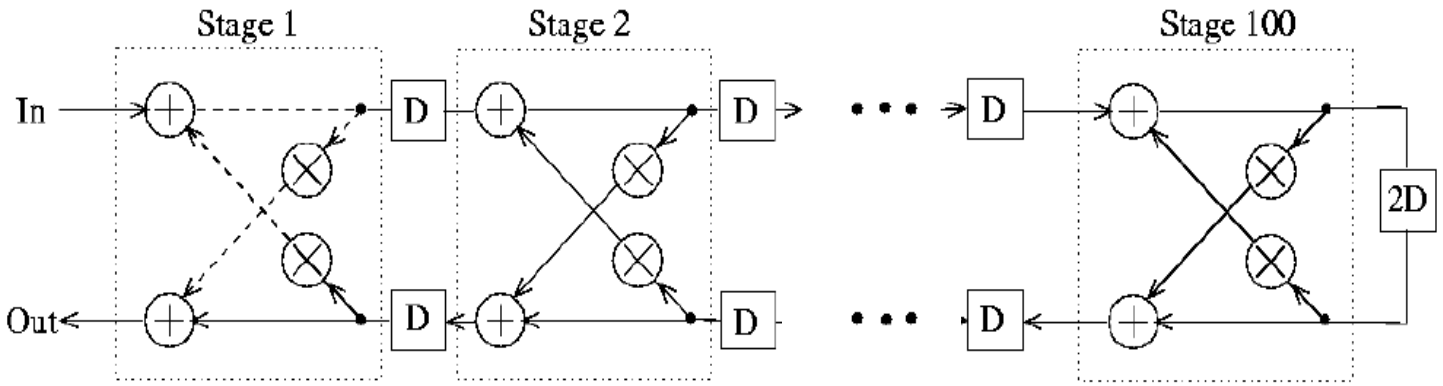


- Delay scaling



Delay Transfer

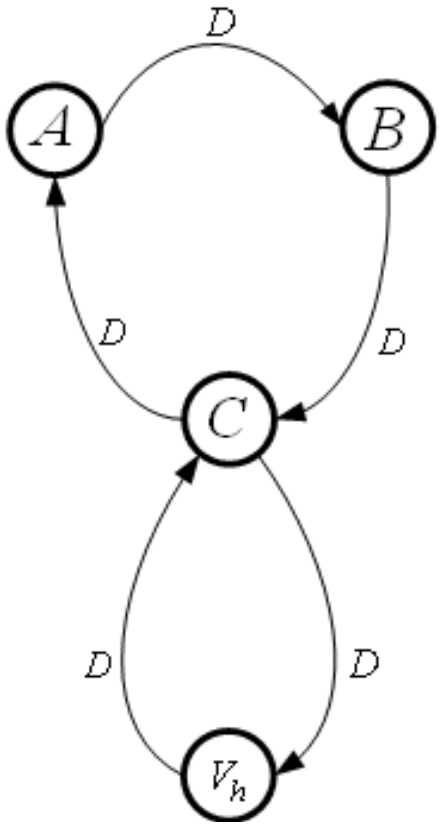
- Delay transfer



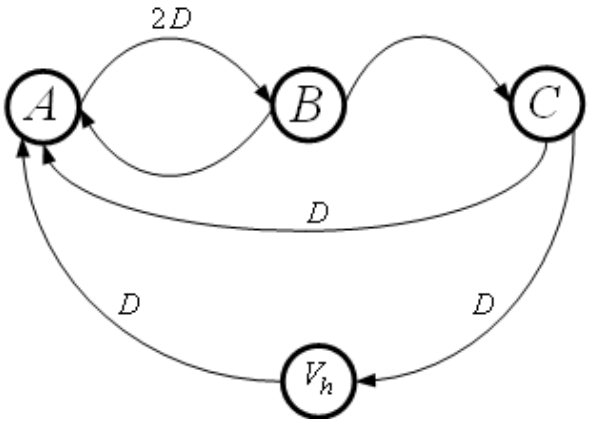
- New critical path delay:
- New sample period:

Systolic Retiming

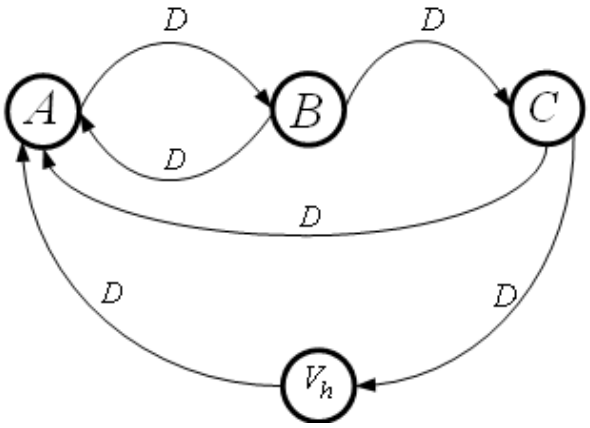
- Retiming attempts to place delays on all arcs
  - Reduces the clock period, T_{cp}
- Systolic DFG: a DFG with no delay-free arcs
  - strives to place delays on all arcs
  - $T_{CP} = max_{\forall U}[d(U)]$
  - Systolic DFG has the smallest T_{CP} of all topologically similar DFGs



Systolic Retiming Example



Original DFG

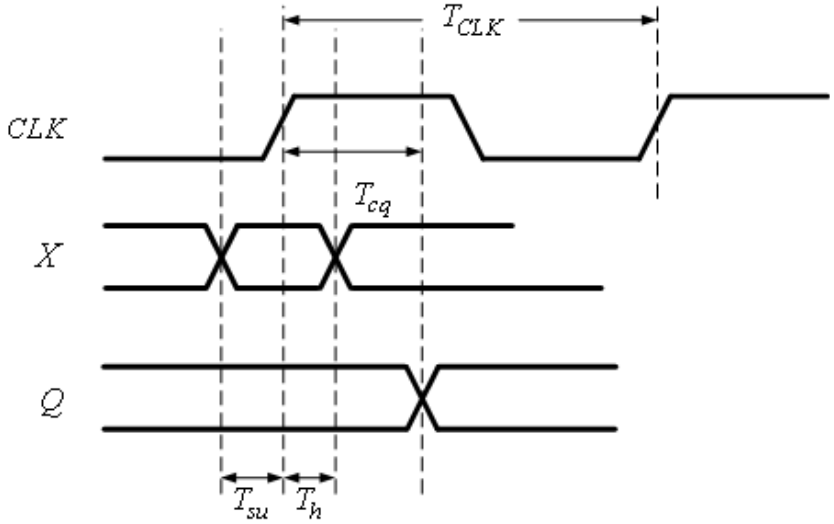


Systolic DFG

Pipelining

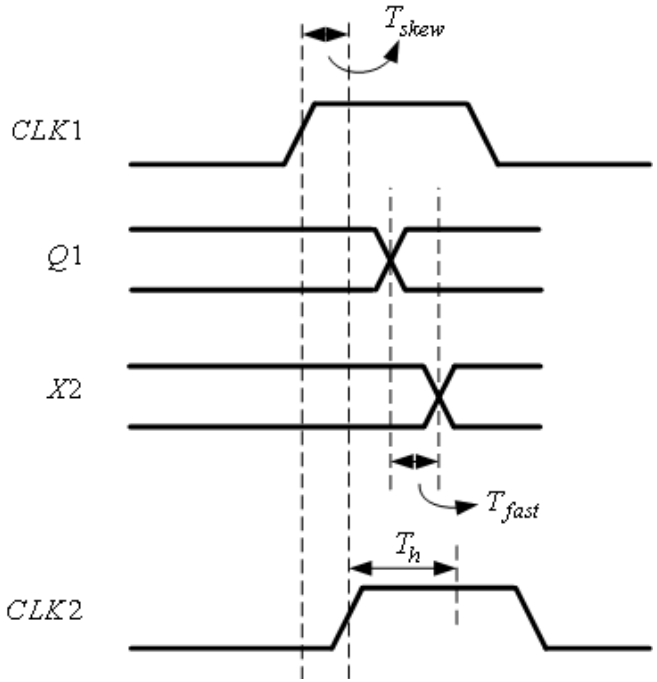
- Pipelining reduces the iteration period bound T_{IPB}.
- Pipelining followed by retiming reduces the iteration period T_{IP}.
- Difference between pipelining and retiming
  - Retiming moves delays around without impacting latency
  - Pipelining introduces delay and increases latency

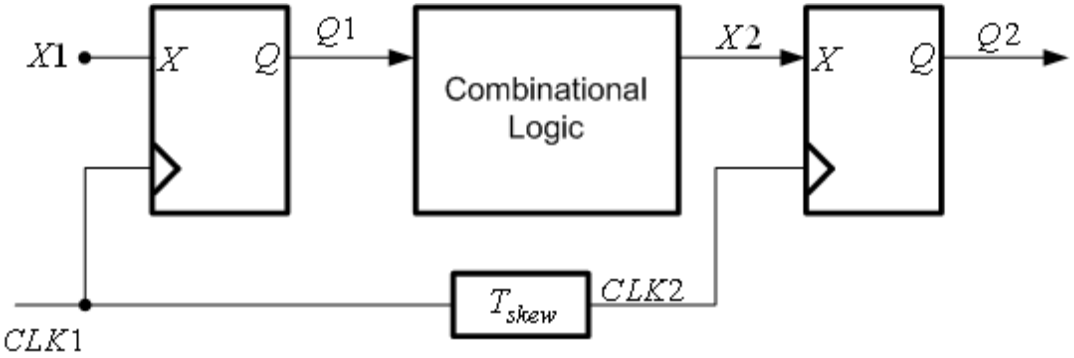
D Flip-Flops



- T_{su}: setup time
- T_h: hold time
- T_{cq}: clock-to-Q time
- T_{CLK}: clock period

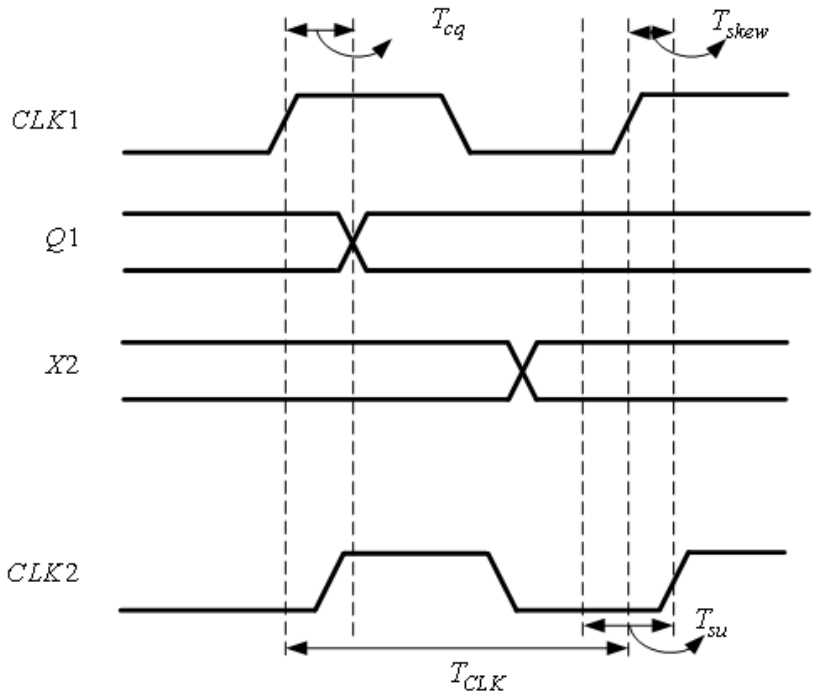
Fast Path

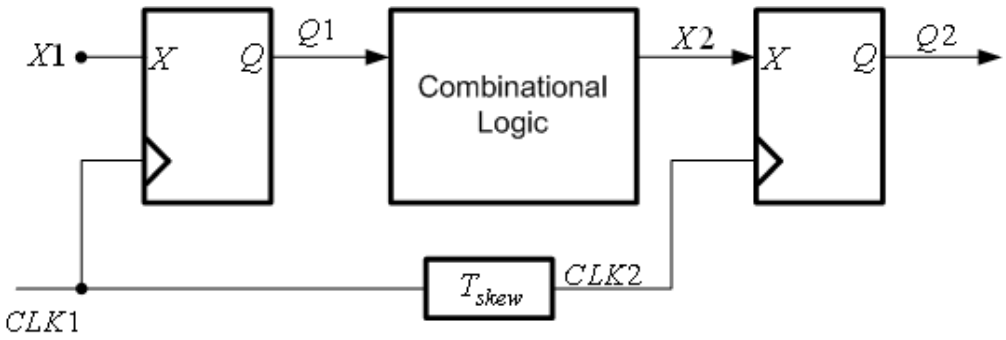




- $T_{cq} + T_{fast} > T_{skew} + T_h$
- Ensure fastest path is sufficiently slow
  - Output to be registered on the rising edge of the NEXT clock cycle

Slow Path





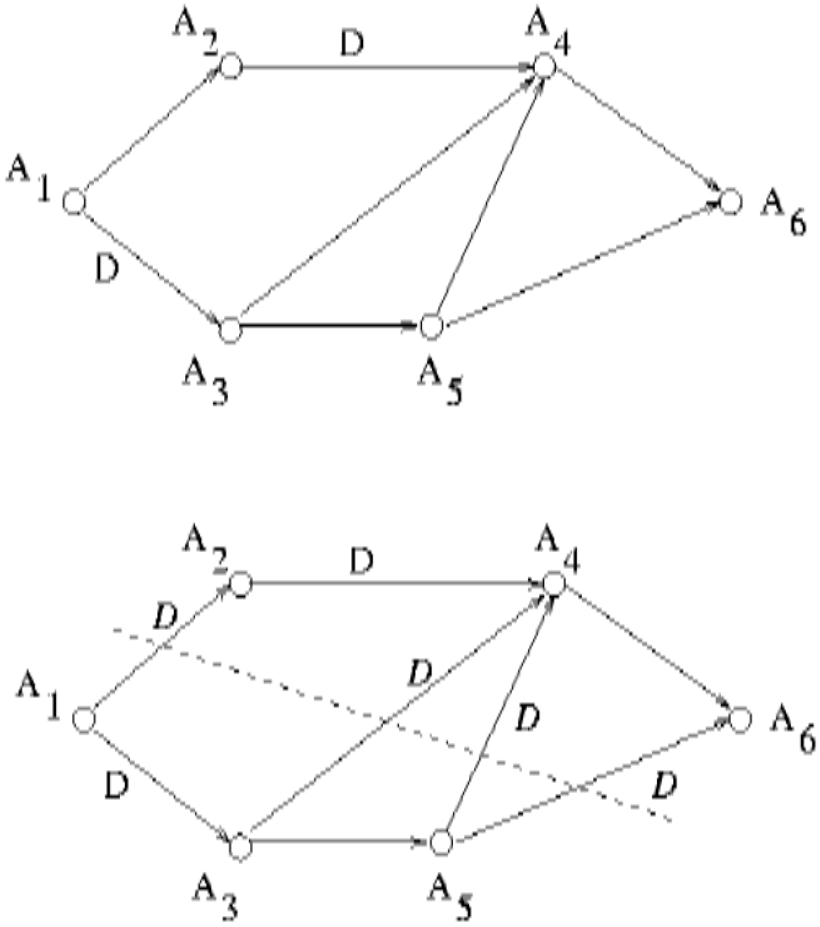
- $T_{cq} + T_{slow} + T_{su} < T_{skew} + T_{CLK}$
- The slowest path determines the clock period

Fast and Slow Path Constraints

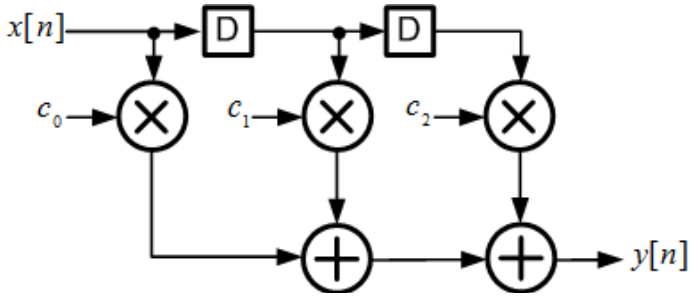
- Both fast and slow path constraints need to be met
- Slow path (setup) violations
  - Corrected by increasing T_{CLK}
  - Reduces throughput
- Fast path (hold) violations cannot be corrected by adjusting T_{CLK}
- Clock skews
  - Positive clock skew
    - good for setup but bad for hold time
  - Negative clock skew
    - good for hold time but bad for setup

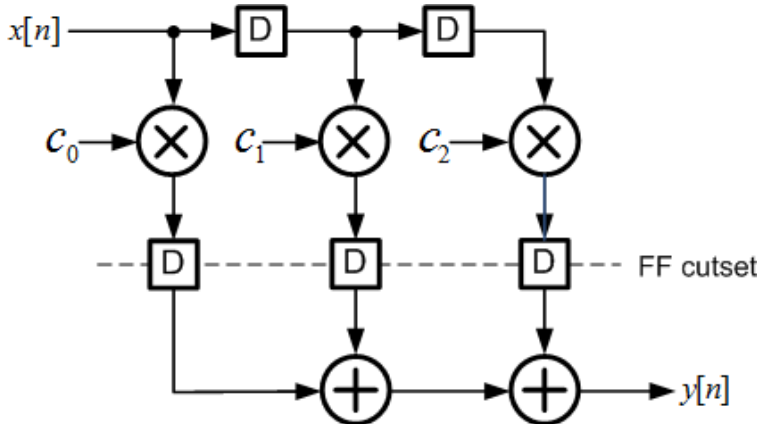
Pipelining Feed-Forward DFG

- Cut: separate the vertices of a graph into two disjoint subsets
- Cutset: the set of edges whose end points are in different subsets
- Feed-forward cutset: data move in the forward direction on all the edges of the cutset
- Place registers across any feed-forward cutset of the graph
- Watch out for fast path constraint violation



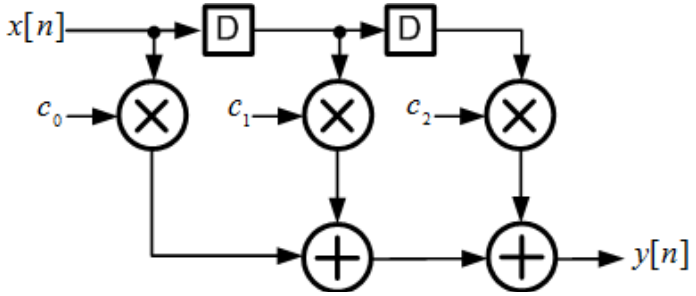
Pipelining an FIR Filter

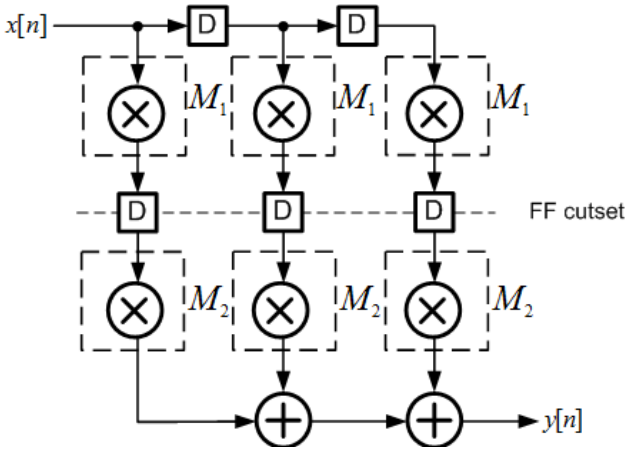




- Assume d(mult) = 3ns, d(add) = 1ns
- $T_{cp,\ serial}$ = d(mult) + 2d(adder) = 5ns
- $T_{cp,\ pipe}$ = max(d(mult), 2d(adder)) = 3ns

Uniform Pipelining





- Pipeline with equal delay stages
- Split multiplier: $d(\text{mult}_1) = d(\text{mult}_2) + 2d(\text{adder}) = 2.5\text{ns}$
- Speed-up = 2X, but practical speed up < 2X due to T_s, T_{cq}, T_{skew}.

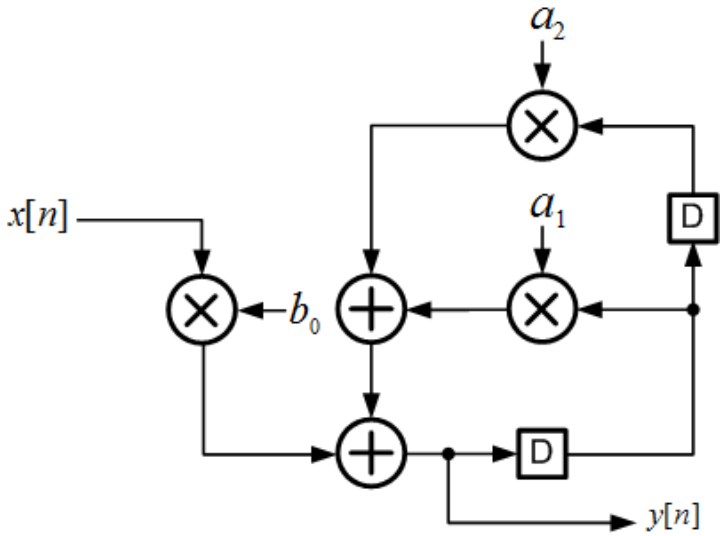
Pipelining as a Low Power Technique

- Delay and dynamic power move in opposite directions as a function of V_{dd}
  - $t_p \sim \frac{CV_{dd}}{k(V_{dd}-V_t)^{\alpha}}$ $\quad P = CV_{dd}^{\ 2} f$
  - For α = 2, $V_{dd} >> V_t$
  - *M*-level pipeline increases speed by *M* times faster
  - Only need to operate at a supply V_{dd}/M to meet the same throughput
  - Power savings by M^2

Practical Limitations

- $\alpha < 2$
- As *M* increases
  - Sub-linear increase in speed due to pipeline timing overhead
  - Power overhead of pipeline registers
  - Exponential increase in delay as V_{dd} approaches V_t

Pipelining Linear Recursions

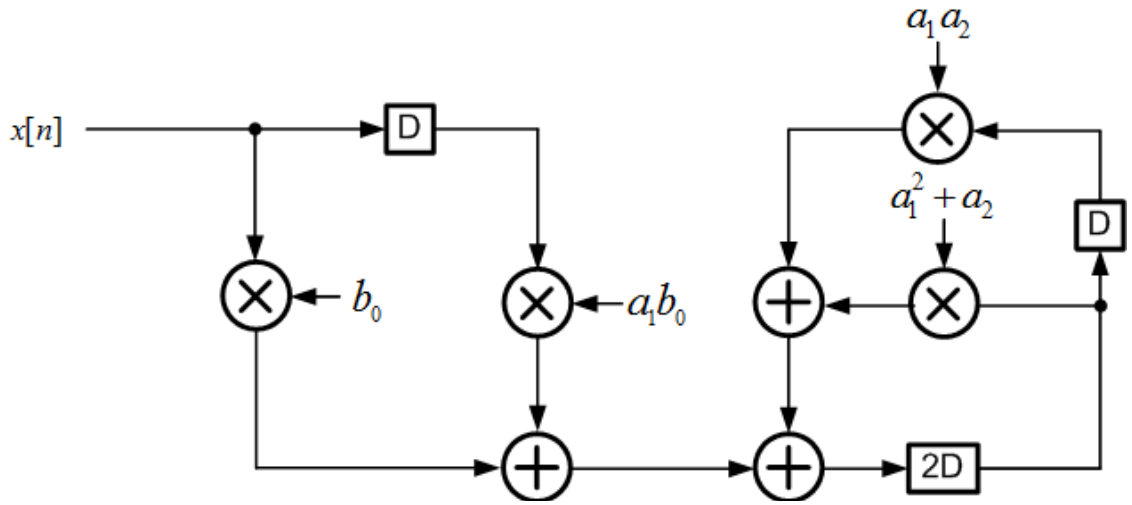


An infinite impulse response (IIR) filter

$$y[n] = b_0 x[n] + a_1 y[n-1] + a_2 y[n-2]$$

- T_{IPB} = d(mult) + 2d(adder)
- Retiming to reduce critical path?
- To reduce T_{IPB}, need to introduce delays in loops

Clustered Look-Ahead (CLA) Pipelining

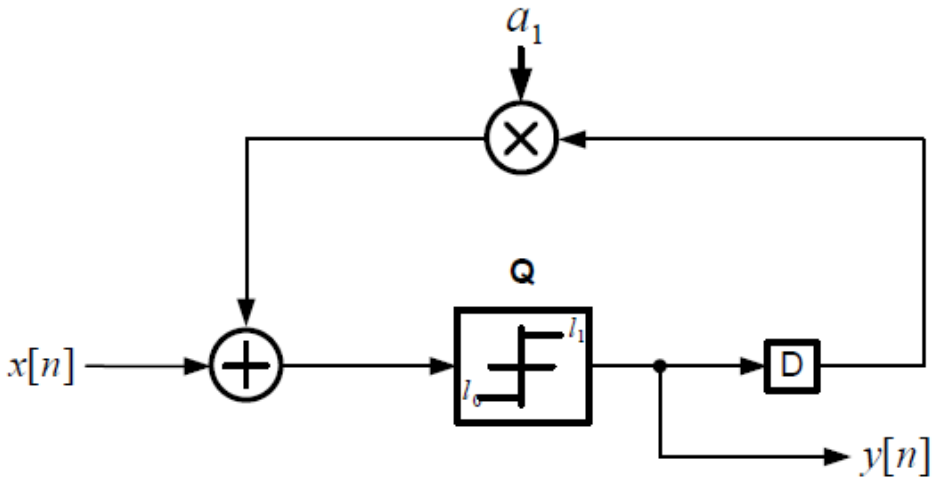


- Look ahead one step, replace $y[n-1]$ term

$$y[n] = b_0 x[n] + a_1 y[n-1] + a_2 y[n-2]$$
$$= b_0 x[n] + a_1 (b_0 x[n-1] + a_1 y[n-2] + a_2 y[n-3]) + a_2 y[n-2]$$
$$= b_0 x[n] + a_1 b_0 x[n-1] + (a_1^2 + a_2) y[n-2] + a_1 a_2 y[n-3]$$

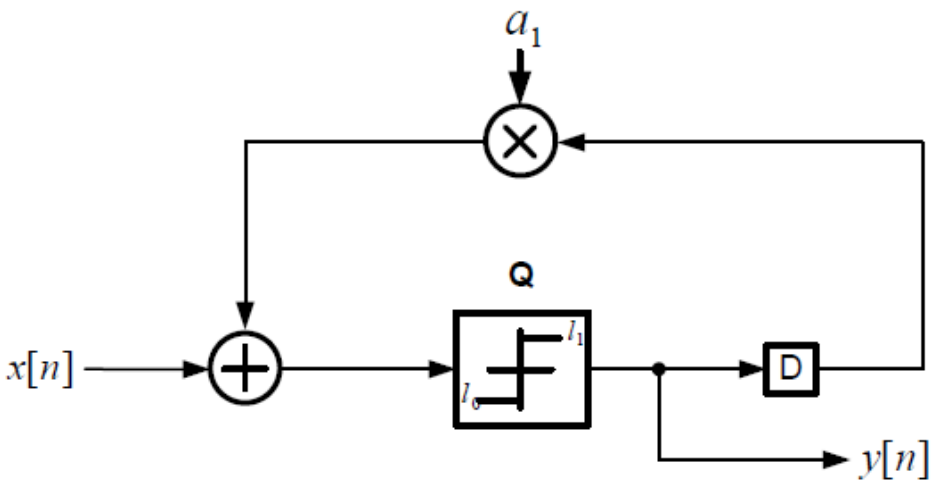
- T_{IPB} = 1/2 (d(mult) + 2d(adder))
- Need extra 1 mult, 1 adder

Pipelining Non-Linear Recursions



- Examples of non-linear recursions
  - Decision feedback equalizer (DFE)
  - Differential pulse code modulation (DPCM)
  - Sigma-delta modulator

Pipelining Non-Linear Recursions



$$y[n] = Q[a_1 y[n-1] + x[n]]$$

- $T_{cp} = T_{mult} + T_{add} + T_{quant}$
- Look-ahead cannot be applied directly

Transform to Index Domain

- Operate in the index domain

- Define $y_0[n]$
  - $y_0[n] = 1$ if $y[n] = l_1$
  - $y_0[n] = 0$ if $y[n] = l_0$

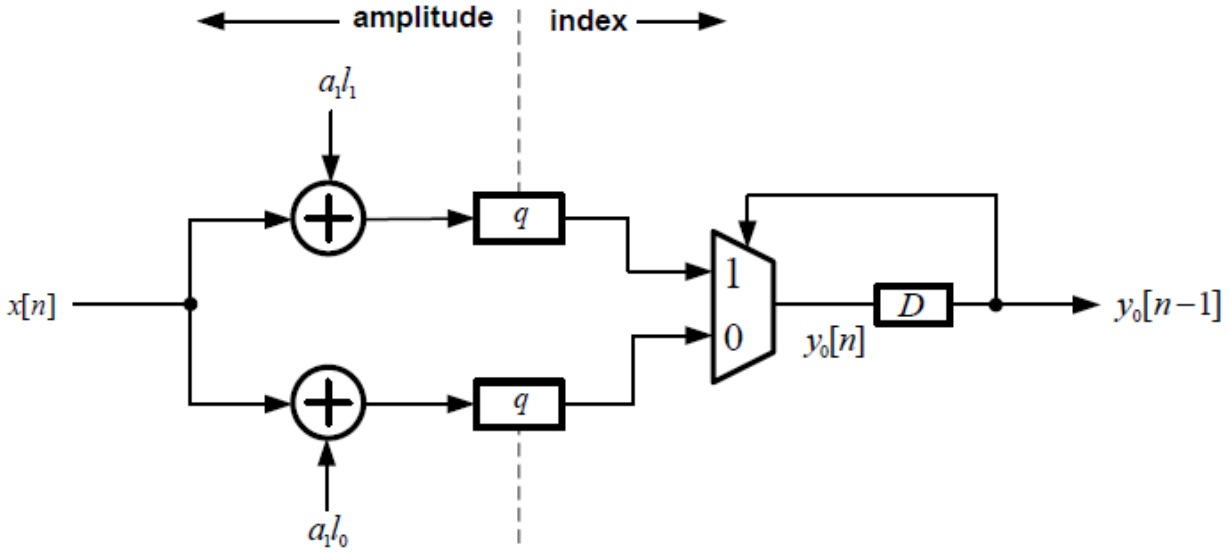
- Define $q[x]$
  - $q[x] = 1$ if $Q[x] = l_1$
  - $q[x] = 0$ if $Q[x] = l_0$

- Then

$$y[n] = Q\left[a_1 y[n-1] + x[n]\right]$$

$$y_0[n] = y_0[n-1]q[a_1 l_1 + x[n]] + \overline{y_0}[n-1]q[a_1 l_0 + x[n]]$$

Transform to Index Domain



$$y_0[n] = y_0[n-1]q[a_1 l_1 + x[n]] + \overline{y_0}[n-1]q[a_1 l_0 + x[n]]$$

- $T_{CP} = T_{mux}$