# VLSI-based Implementation of Digital Signal Processing Systems

Wonyong Sung

School of Electrical Engineering
**Seoul National University**

# Contents

# 1. Architectures for DSP Systems

- **Requirements for DSP system implementation**
  - Correct operation (algorithm level)
  - Speed (throughput)
  - Low chip area and power consumption (cost)
  - Fast and low-cost design, design flexibility (design upgrade, portability)
- **Custom VLSI (HW based) vs Programmable DSP (SW based)**
  - HW based for higher throughput
  - Large initial investment for VLSI
  - Low power and low chip cost for large volume VLSI

# Custom VLSI vs Programmable DSP

- **Large initial investment, but small cost for each chip – good for large volume**
  - Needs to be ~ 1M units or over in most cases
  - FPGA based designs are alternatives for small quantities
- **High-throughput architecture, highly optimized for each application**
  - Inflexible in most cases
- **Low-power consumption when compared to program based architecture**
- **CPU + peripheral + application specific HW -> SOC (System On Chip), platform based design**

# Issues in VLSI based system design

- **BW matching between the difference of the signal sampling clock frequency and the system clock frequency**
  - Reduce the number of hardware elements when the signal sampling frequency is small.
    - Full array architecture: high hardware cost, high throughput
    - Time multiplexed architecture
    - Bit serial architecture: low hardware cost, low throughput

# Characteristics of DSP algorithms

- **Arithmetic intensive**
  - ex) FIR filter order of 60, 10 MHz
    multiplication, addition 600 Mop/sec.
- **In most cases, f system >> f sampling**
  - fsystem (system clock frequency): mostly 10MHz ~ 1GHz
  - fsampling (sampling clock frequency):
    - Speech and Audio: 8KHz ~ 100KHz
    - Video: 10MHz ~ 100MHz

# Algorithm v.s. System

- **Algorithm**
  - Operating at a rate of sampling clock freq.
  - Sample delay: $z^{-1}$ (= 1/ fsignal)
  - Arithmetic: multiplication, addition operations
  - Corresponds to the clock freq to ADC or DAC
- **System (Hardware)**
  - Clock for digital systems
  - Arithmetic: multipliers or adders
  - Delay: D-FF (=1/fsystem)

fsignal = fsystem: full array architecture
fsignal > fsystem: hyper parallel architecture
fsignal < fsystem: time-multiplexed architecture

# Bandwidth matching

- **Compensates for the difference of system and signal frequencies to better utilize the resources.**
  - For example, 10 tap FIR filtering and 4$^{th}$ order IIR filtering (2*5 multiply operations) with 10MHz sampling clock frequency:
    - Needed multiplications: 10*10M + 2*5*10M = 200M/sec
  - Assuming that system clock frequency is 100MHz, only two multipliers are needed.
  - 20 multiply operations with 10MHz
    –BW matching-> 2 multipliers with 100MHz

Note: Another important factor determining the architecture: algorithm complexity. When the algorithm is too complex, it would be very difficult to implement using a custom VLSI architecture.

# Methods of BW matching

- **fsignal ≥ fsystem**
  - Algorithm transformation to increase the fsystem (pipelining)
  - Parallel (or block) processing to obtain multiple output samples per each system clock.
- **fsignal < fsystem**
  - Use one HW unit for multiple operations (time-multiplexed architecture)
  - Use simple but slow arithmetic units (bit serial architecture)

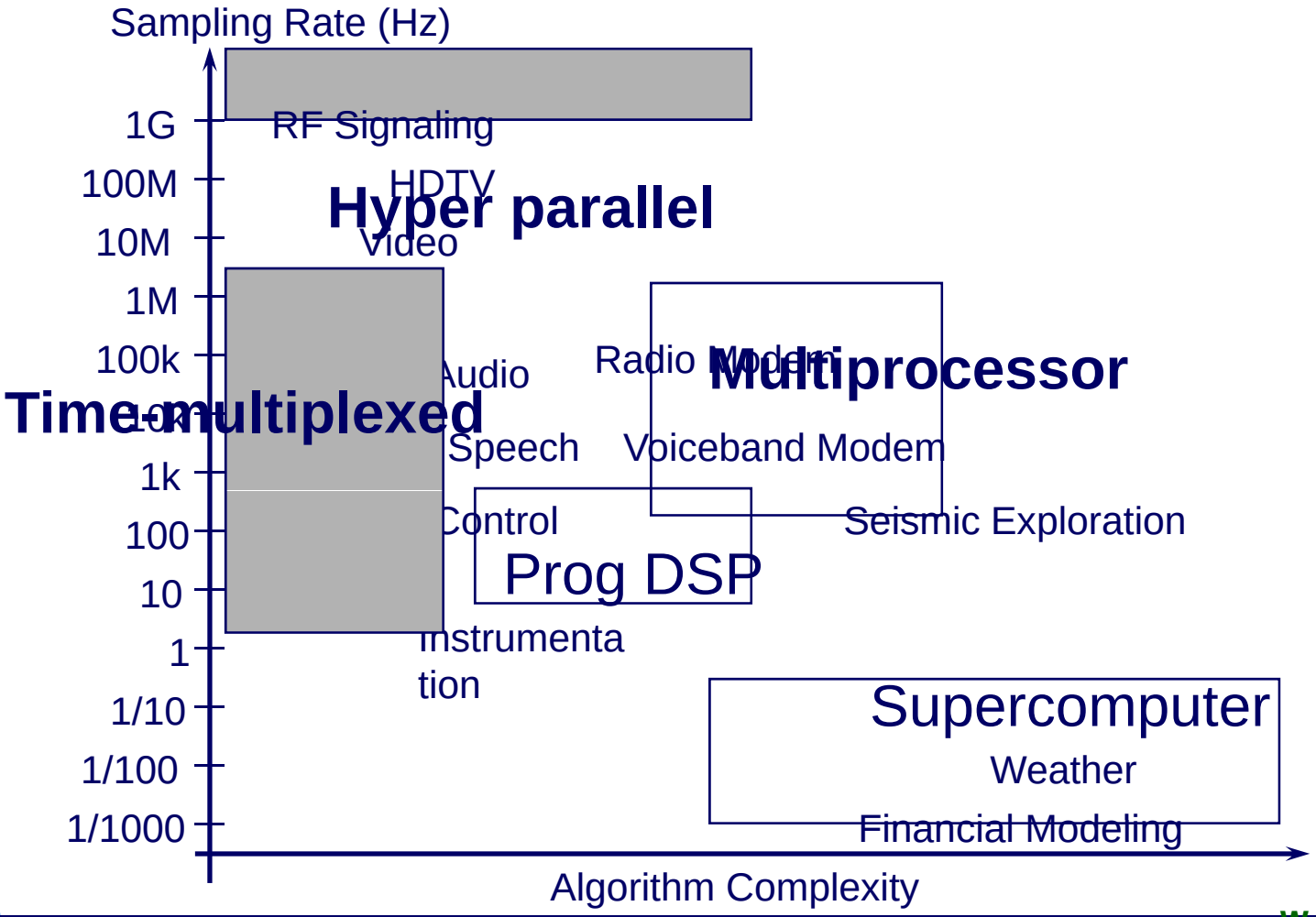
# Implementation architectures

- Fully serial: Use only one processor, program based implementation (ultimate time-multiplexed implementation)
- Time multiplexed: Utilize one hardware unit (multiplier, adder) several times during one sampling period. Hardware delay << sampling period
- Bit serial architecture: Utilize one slow hardware unit (bit serial arithmetic components) only one or a few times during one sampling point.
- Full array: fsignal = fsystem, use one arithmetic elements for each arithmetic operation
- Hyper-parallel: fsignal $\geq$ fsystem
  Needs to transform the algorithm, use multiple or pipelined elements for each operation

# Time-multiplexed operation

- **Bit-parallel arithmetic elements, operation time-muxed**
  - processes one word of signal at one clock, but sequentially conducts different operations
  - needs parallel multipliers, adders, and memory
  - needs fairy complex control and address generation
- **Bit-serial simple arithmetic elements, operation dedicated**
  - processes one bit of signal at one clock using a dedicated one-bit arithmetic or memory elements
  - almost hardwired (simple) control
  - efficient implementation, limit in throughput

# Applications and architectures



# Algorithm complexity v.s. architectures

- **For simple but repeating blocks**
  - Digital filter, FFT
  - hardwired control
  - efficient hardware and VLSI implementation
- **For complex algorithms**
  - Speech, audio, data modem
  - program controlled is advantageous
  - programmable DSP based implementation

# 2. High-speed Parallel Architecture

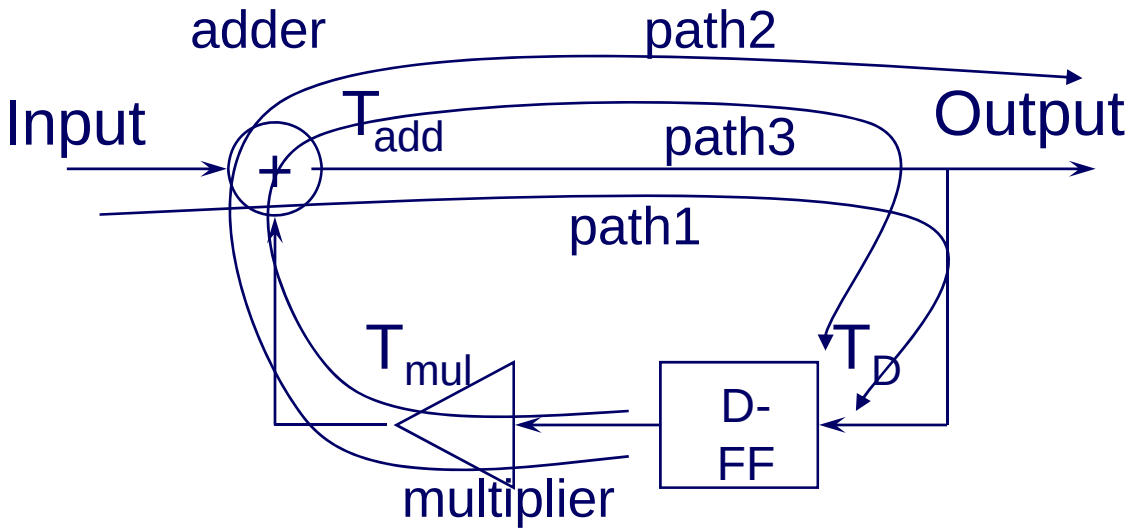
- **Full array implementation architecture**
  - Sampling rate = system clock rate
  - Mostly high-bandwidth RF
  - Signal flow graph – (**retiming**) -> Hardware schematic
    - Addition or multiplication operations -> adder, multiplier
    - $z^{-1}$ -> D-FF
  - Speed limitation: circuit delay from the output of D-FF (or input port) to input of the D-FF (the output). -> retiming equalizes the delays and minimizes the maximum of them
  - No scheduling for HW resource minimization needed

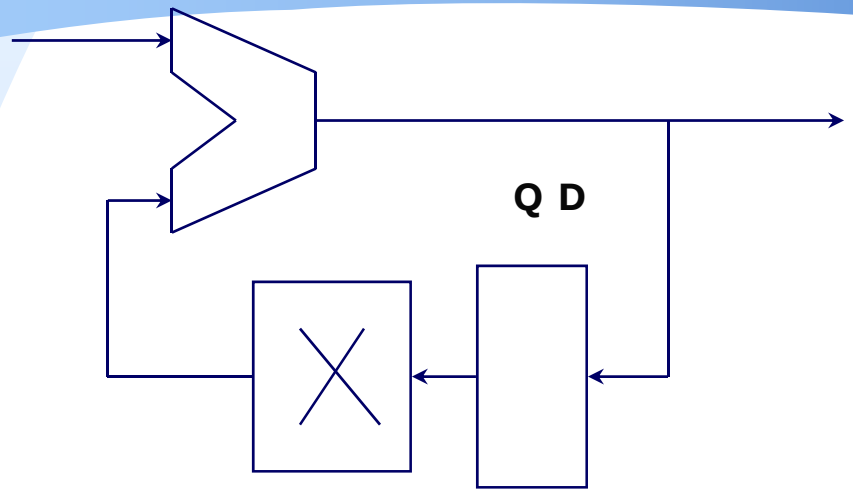
# Speed limitation?

- **Is it right?**
  - For a given signal flow graph, the throughput can be increased without any limit as long as enough HW resources are available.
  - If there is no loop (circle) inside, it is yes.
  - If not,…
  - In terms of resource hazard, this may be right.
  - But, what is another kind of hazard?

# Full array architecture and speed limitation

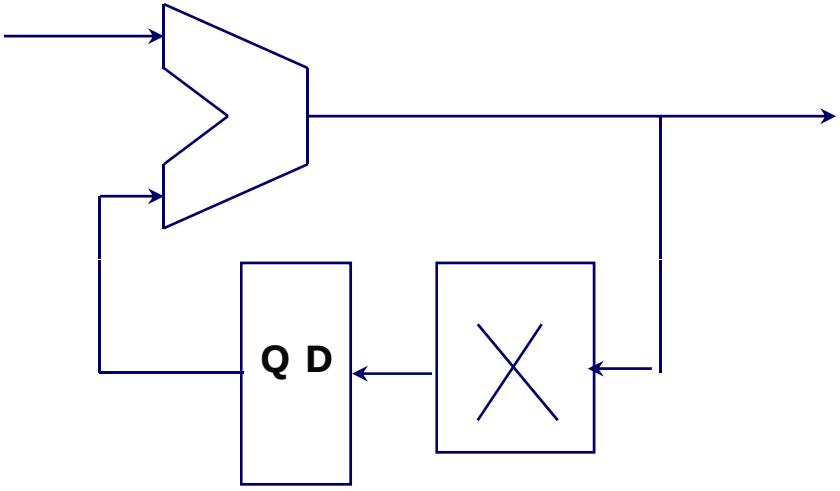
- **1st order IIR digital filter**
  - Path 1: $T_{add}$ (10 ns)
  - Path 2,3: $T_{add}$ + $T_{mul}$ (30 ns) = 40 ns <- critical path delay, this determines the maximum clock freq.





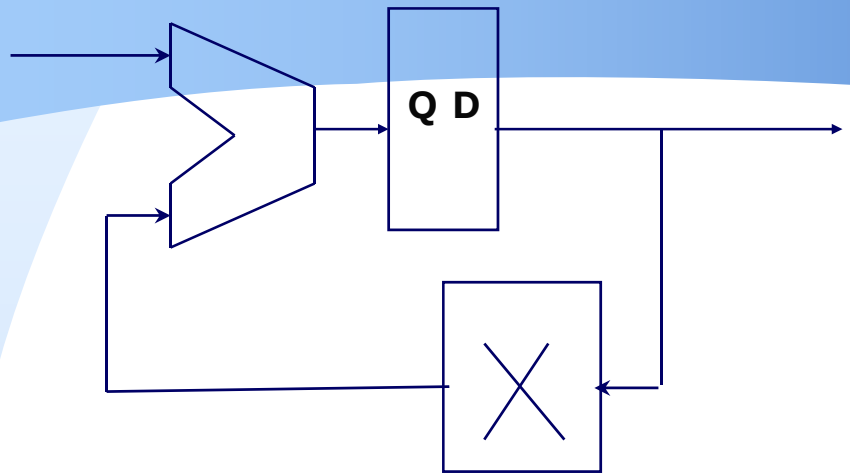

Implementation 1:
critical path delay: 40ns

$H[z] = 1/(1-az^{-1})$




Implementation 2:
critical path delay: 40ns

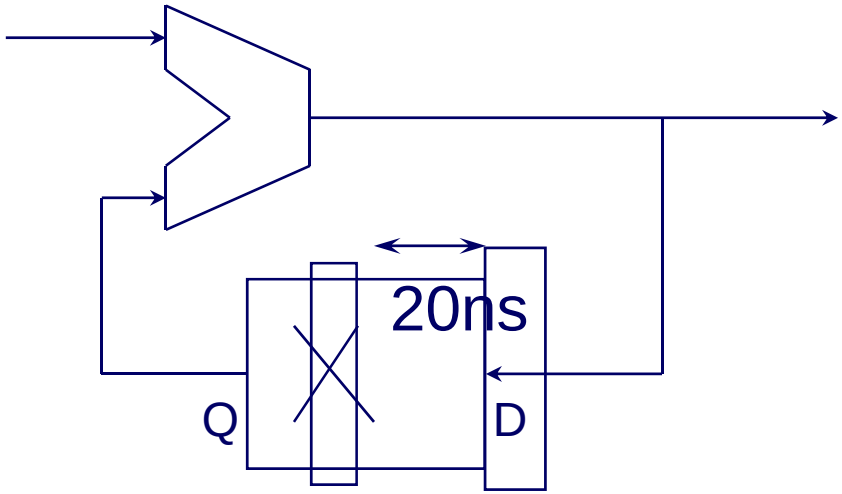
$H[z] = 1/(1-az^{-1})$




## Implementation 3:
## critical path delay: 40ns

$H[z] = z^{-1}/(1-az^{-1})$

Different transfer function but the same frequency response. Zero at the center does not affect the f.r.




Pipelining register

## Implementation 2:
## Critical path delay: 20ns

$H[z] = 1/(1-az^{-2})$ <- different filter!

# Max operating frequency of a signal flow graph and equivalence transform

- The number of delay in a loop (N), the total circuit delay ($T_a$) => Theoretical min iteration period (after retiming) (Iteration Period Bound): $T_a / N$
- For multiple loops, the largest delay determines the maximum clock frequency <- critical loop
- Retiming : move the location of delays to reduce the critical path delay (doing equivalence transform)

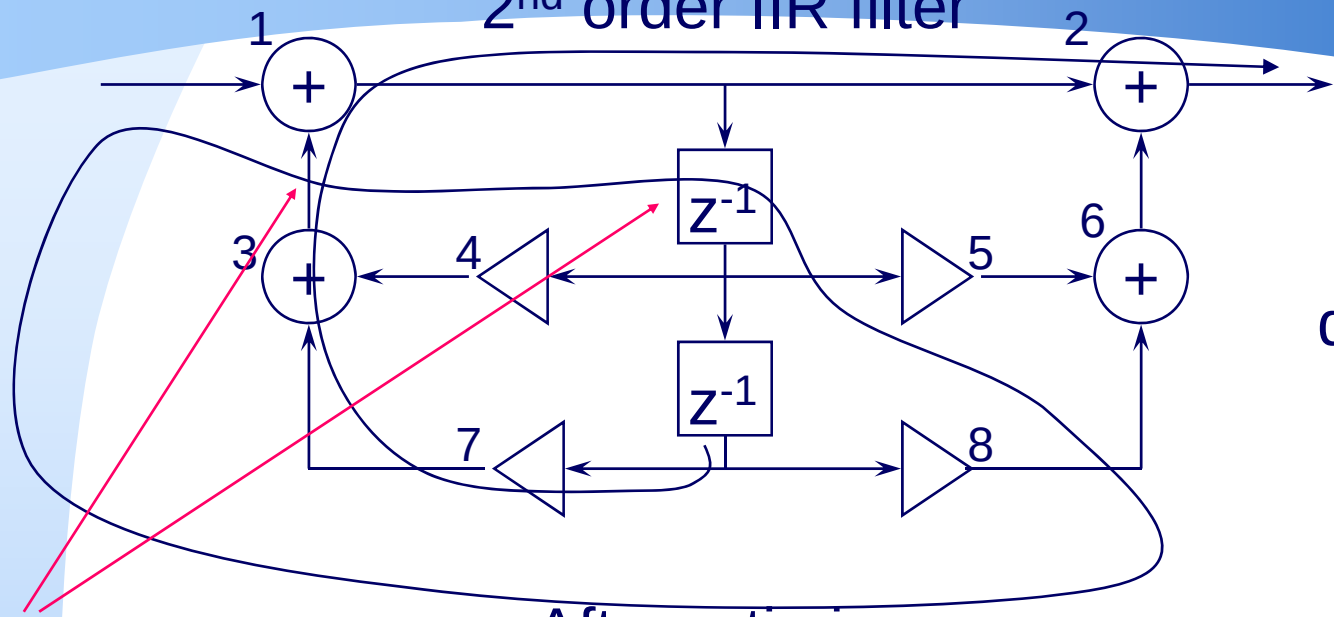
# Longest path matrix algorithm

- **Find out the total circuit delays from an output of a storage (D) to the input of another one. And find out the max (circuit delay / #of_delays).**
- **Start with constructing $L^{(1)}$ matrix**
  - $l^m(i,j)$ is the longest computation time of all paths from delay di to delay dj that path through m-1 delays.
  - From $L^{(1)}$ matrix, compute $L^{(2)}$ $L^{(3)}$ $L^{(4)}$
    - $L(m+1)i,j = \max(-1, l^{(1)}_{i,k,} + l^{(m)}_{k,j})$
  - $T = \max$ (for all i, m) $\{l^m(i,i)/m\}$, (diagonal elements)

# Equivalence transform

- Move the location of delays while not changing the transfer function and the finite wordlength effects of digital filters. -> this changes the critical path delay in many cases
- For a certain directed graph, make a closed loop so that it just cut the branches (not any node), and add d0 delays to branches that are going out and subtract d0 delays to branches that are going in. In this case, the total number of delays for any loop is not changing, and, as a result, the transfer function is not altered.
- If the above closed loop pass through a loop in a signal flow graph, the number of delays added equals to the one that are subtracted. -> The total number of delays for a loop is unchanged. -> The transfer function is preserved.
- For a feedforward path, the closed loop inserts delays, but the number to each path to the output is the same, as a result, the output just comes later as much as the number of delays added.

## 2nd order IIR filter

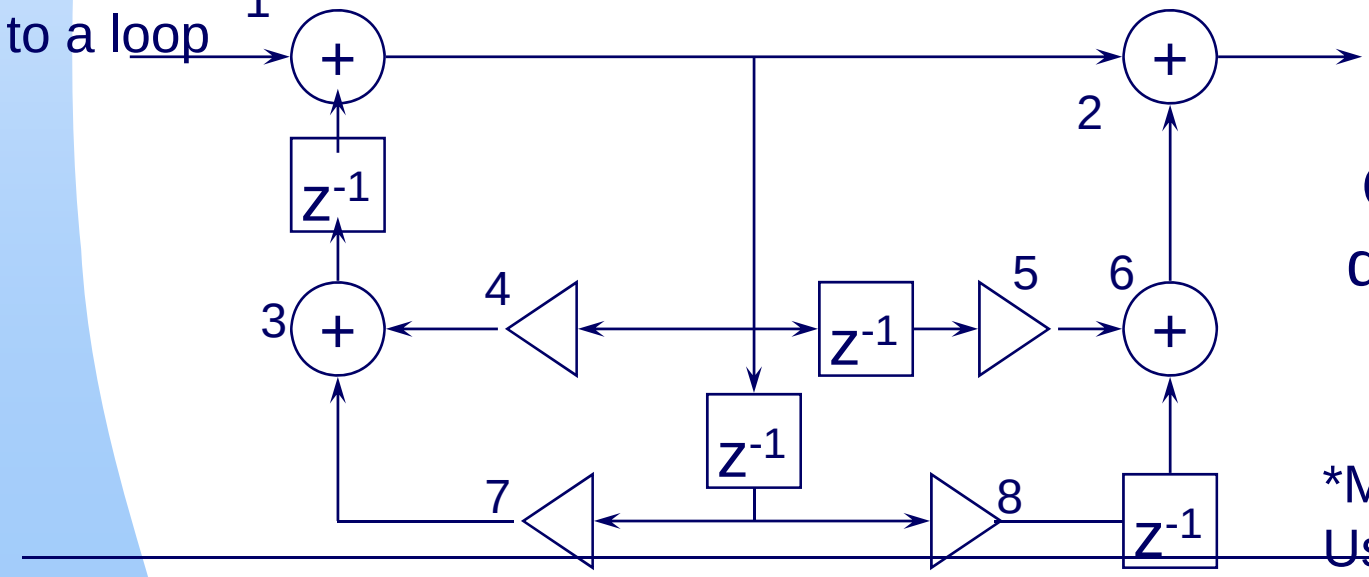



Canonical form: Minimum # of storages

Critical path delay : $T_{mul} + 3T_{add}$

## After retiming

Delay added or subtracted to a loop

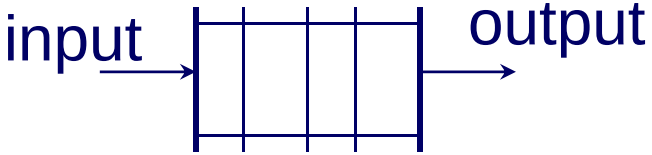


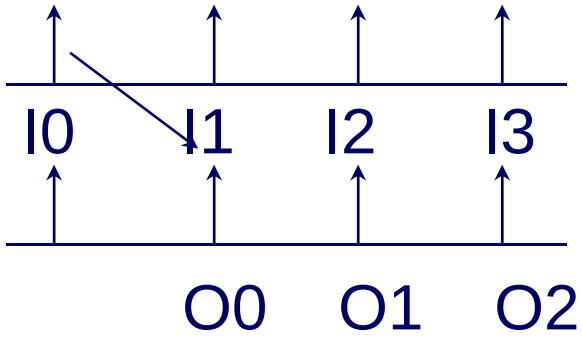

Critical path delay : $T_{mul} + 2T_{add}$

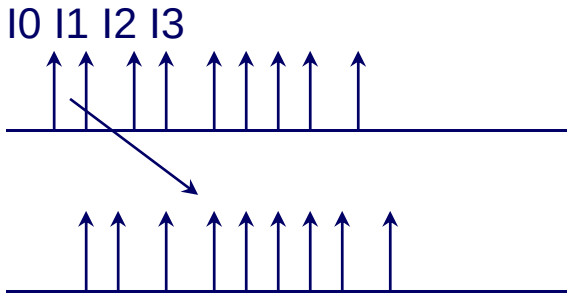
*More registers are Used! Not canonical

# Pipelining

- Increasing the speed by inserting delays inside
- Throughput : the rate of applying periodic input
- Latency : the delay from the input to the corresponding output

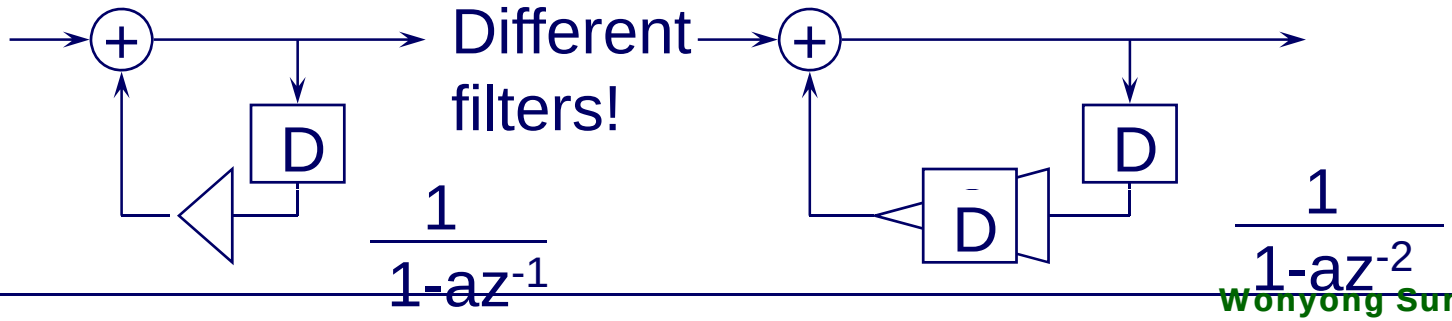



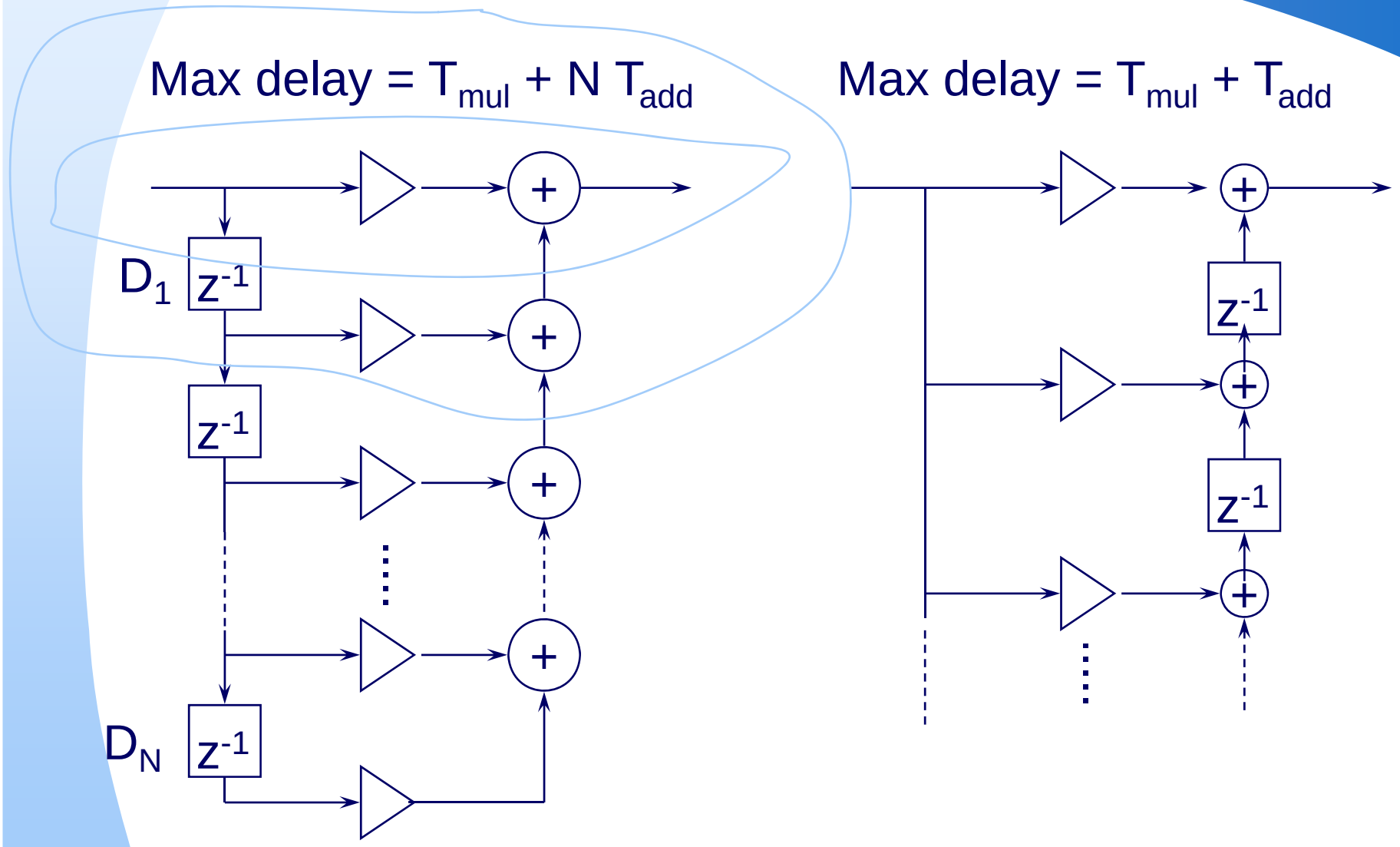



# Effects of pipelining

- Throughput is more important than the latency in real-time signal processing.
- The throughput is increased by pipelining but not the latency
- For feedback based system, the pipelining may change the transfer functions.
- Usually, pipelining in the feed-forward path is OK.
  (add delays and retiming)



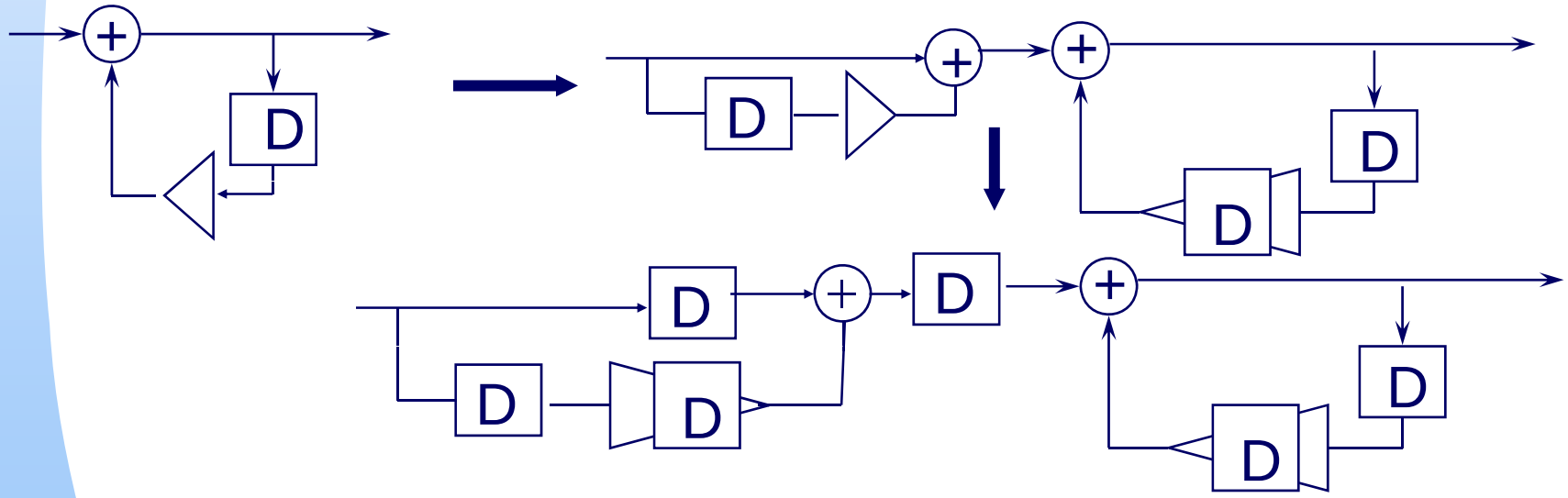
- Full array implementation example of an N-tap FIR filter



# Hyper parallel implementation(1)

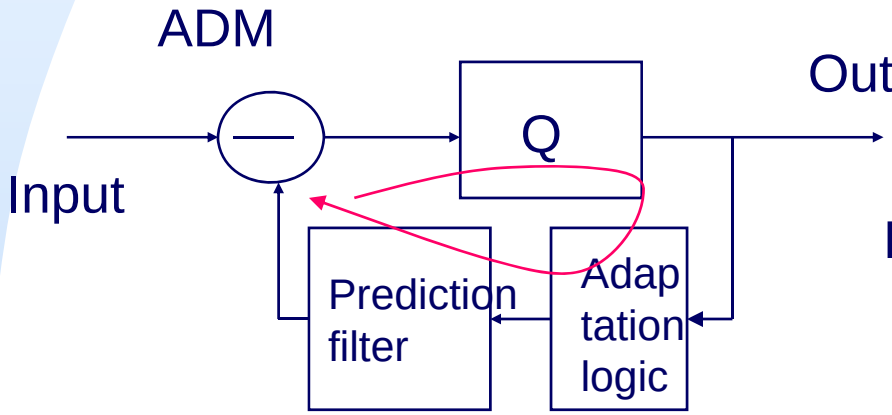
- **Is it possible to increase the speed (sampling clock freq.) beyond the iteration period bound?**
  - Yes. By applying the look-ahead transformation.

$$H(z) = 1/ (1-az^{-1}) = (1+az^{-1})/(1-a^2z^{-2})$$

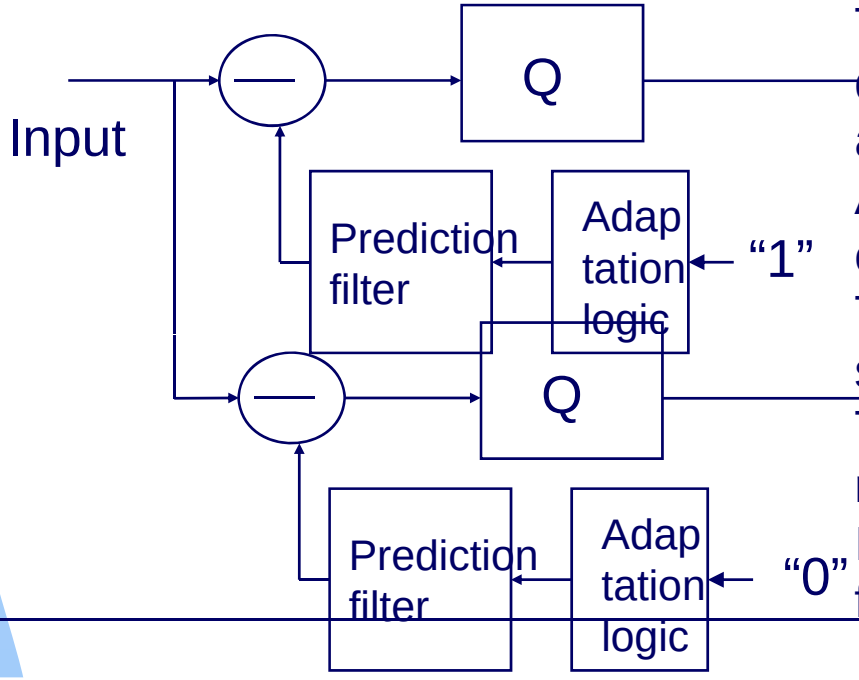



The total number of multiplication is increased.
Quantization effects are different.

# Hyperparallel implementation(2)




Determining the loop bound




The next prediction is computed before determining the current output by assuming both "1" and "0".
After the computation, the right one is chosen.
The idea is similar to that of the carry select adder.
This is only applicable when the number of output is small.
If the Q is two bits, we need to implement four different cases.

# CSD (Canonic Signed Digit) coefficients based FIR filter

- **Reduces the complexity of constant multiplications**
  - One multiplication is converted to a few (one to three, usually) addition/subtractions.
- **Represent the coefficients with +1/-1/0 and try to increase the number of zero's.**
  - 00111111 => 0100000(-1) : effective coefficients word-length is 2
- **May increases the passband and stop band ripples when the number of '1' is limited.**
- **Only applicable to full-array (not for time-multiplexed) implementations.**

# 3. Time-multiplexed Architecture (Folded Architecture)

- **System frequency > Sampling frequency**
  - Use a smaller number of arithmetic elements than that of the arithmetic operations
  - Ex:10MHz sampling frequency, 30 tap FIR filter
    - With the system clock of 50MHz, the minimum number of HW would be 6.
    - With the system clock of 100MHz, the minimum HW would be 3.
  - More HW resources are needed in many cases.
    - Dependency relation which forces some units underutilized
    - Unequal job allocation
    - Internal signal delay (interconnection delays)

# Design methods for time-muxed architecture

- **Scheduling based method**
  - Start from a data flow graph, consider the HW resource and time-bound.
- **Utilizing the iterative structure**
  - $12^{th}$ order IIR filter using $2^{nd}$ order section
  - Use 6-times time-multiplexing of one $2^{nd}$ order section
- **Program based method**
  - Flexible but needs program memory storage

# Scheduling algorithm

- **Unconstrained minimum-latency scheduling**
  - ASAP (As Soon As Possible)
  - ALAP (As Late As Possible)
- **Resource-constrained minimum-latency (or Latency-constrained minimum-resource) scheduling**
  - List scheduling
  - Force-directed scheduling
- **Example (Euler's method for solving differential equation)**

  $y'' + 3xy' + 3y = 0$

  initial value: $x(0)$, $y(0)$, $y'(0)$

  $y(a) = ?$

  stepsize = $dx$

  $x_{i+1} = x_i + dx$

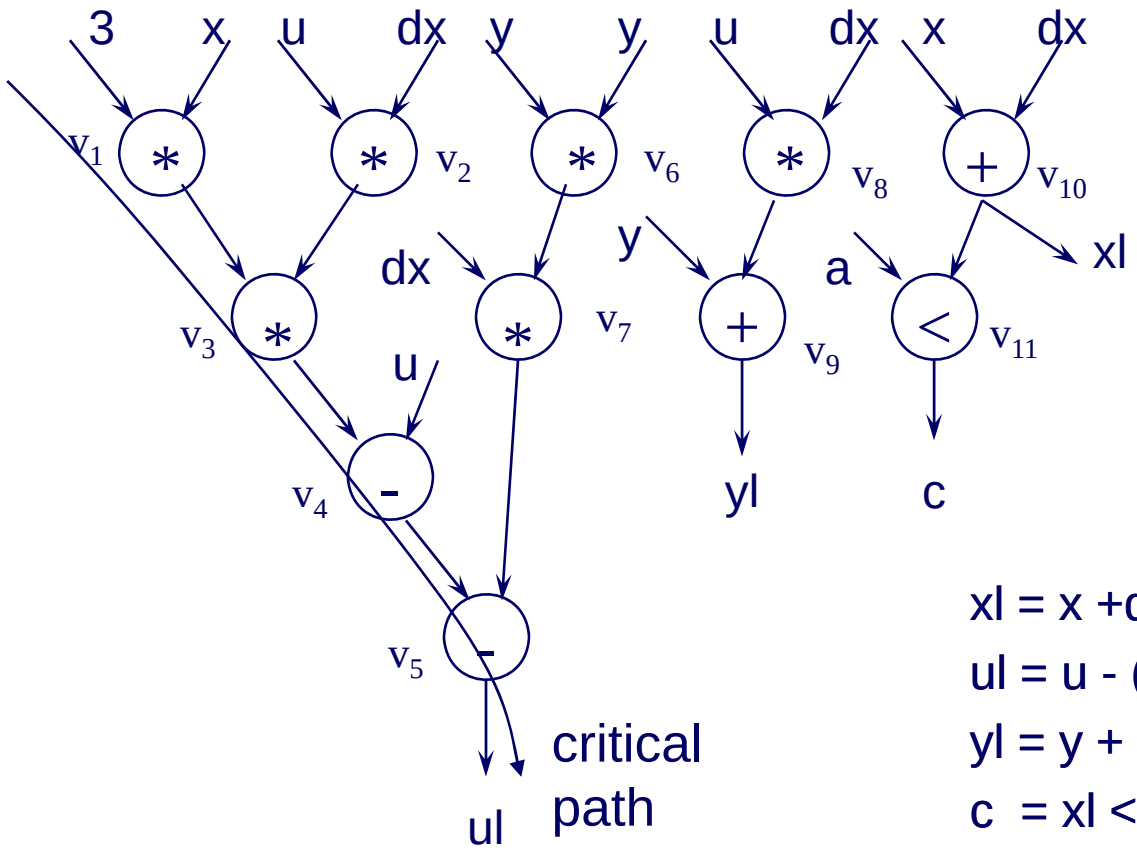
  $u = y'$

  $u' + 3xu + 3y = 0$

  $u_{i+1} = u_i + u_i'dx = u_i - 3x_iu_idx - 3y_idx$

  $y_{i+1} = y_i + y_i'dx = y_i + u_idx$

## ❖ Data flow graph for a differential equation



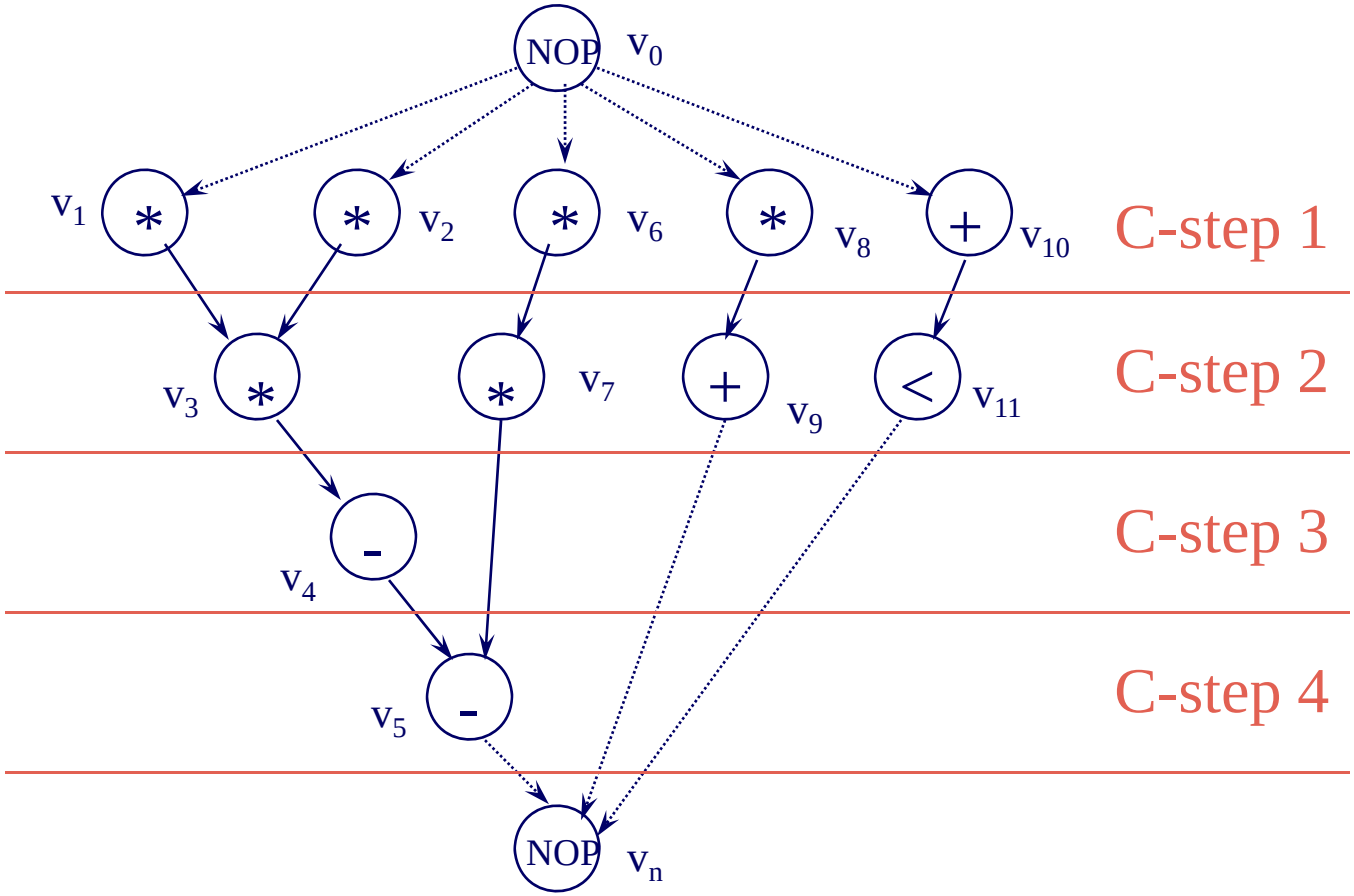

xl = x +dx;

ul = u - (3*x*u*dx) - (3*y*dx);

yl = y + (u*dx);

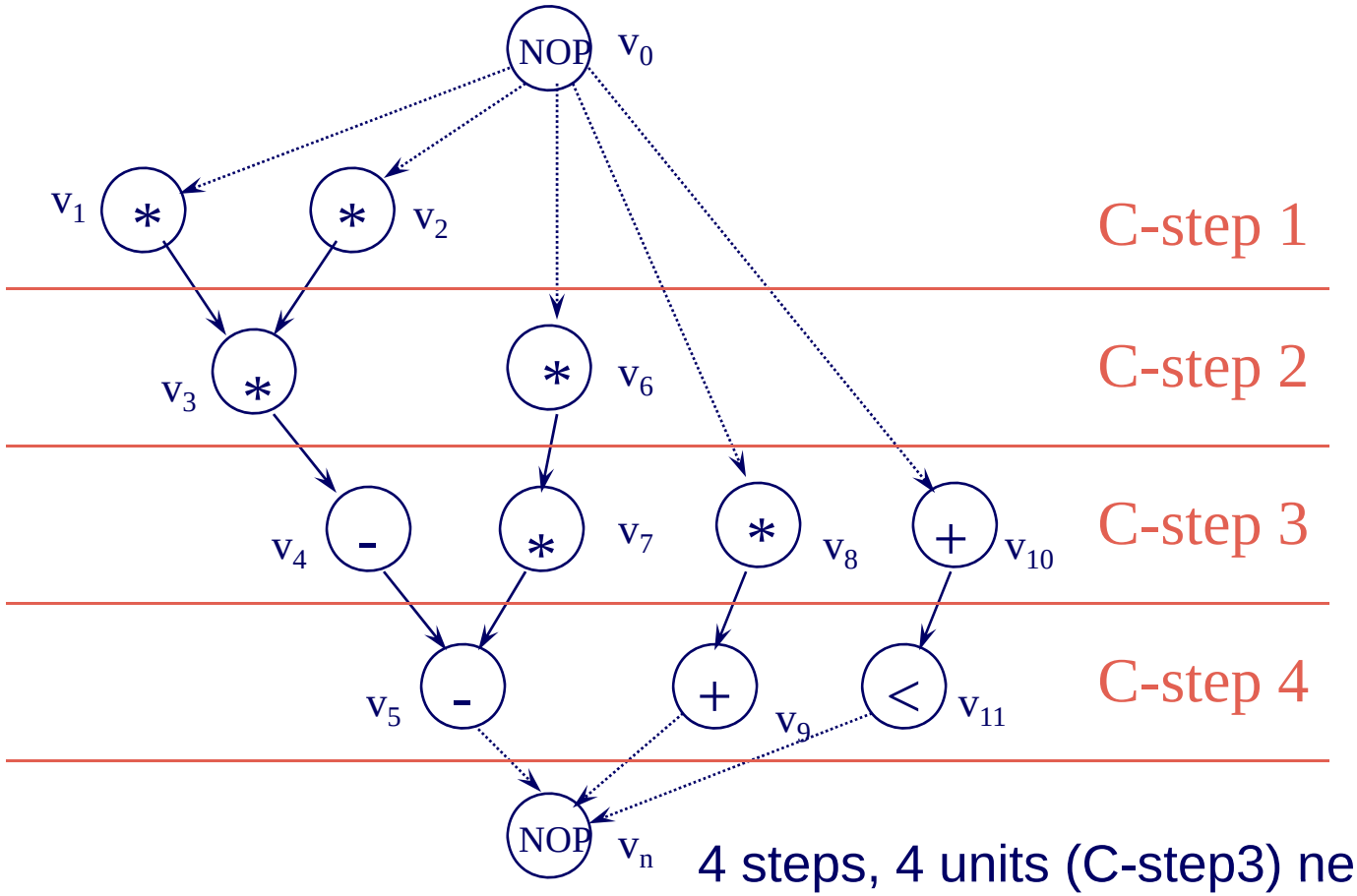
c = xl < a;

## ❖ ASAP (As Soon As Possible) scheduling – unconstrained minimum latency




4 steps, 5 units needed (C-step1 needs 5 units)

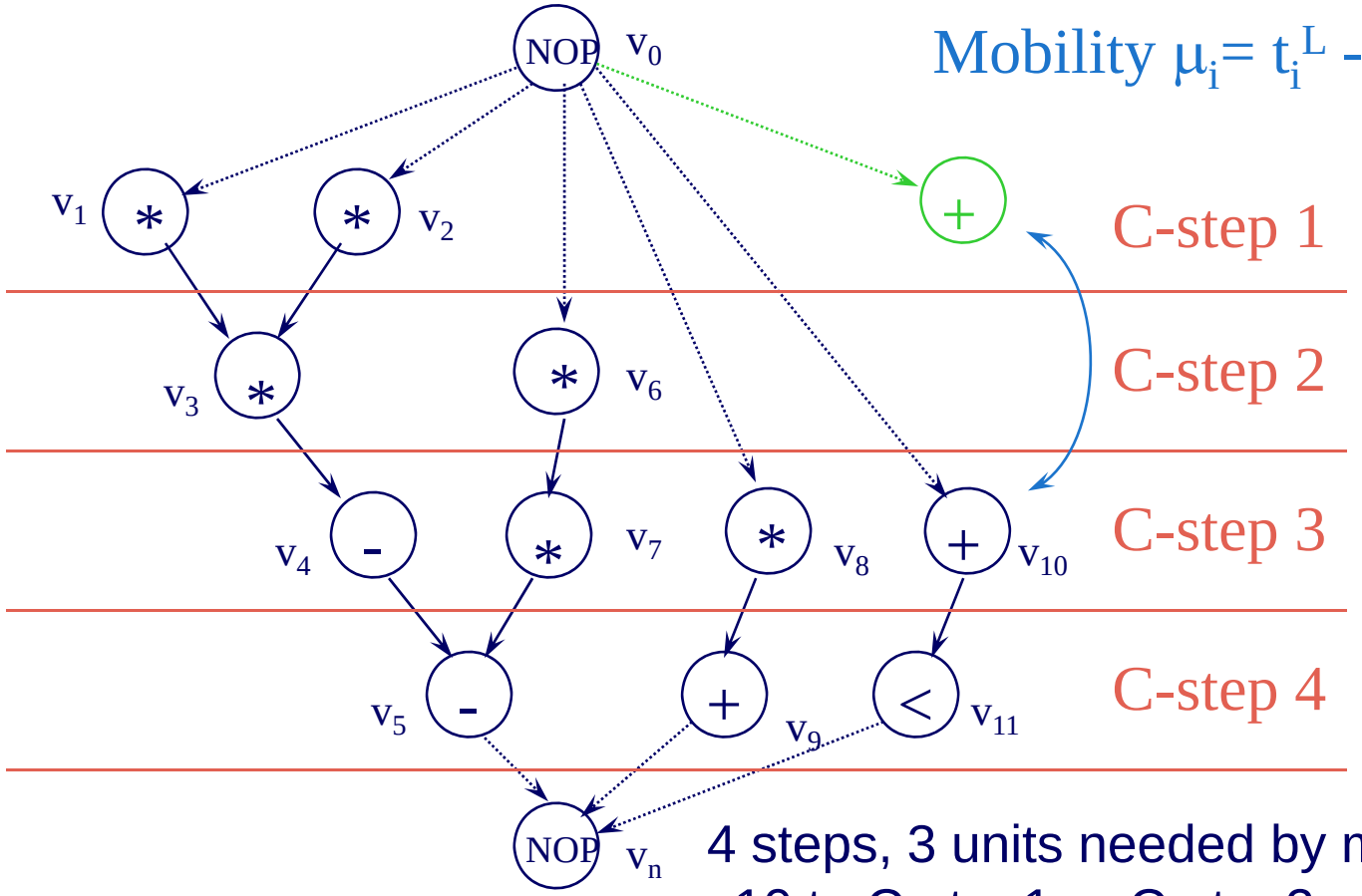
## ❖ ALAP (As Late As Possible) scheduling




4 steps, 4 units (C-step3) needed

# Mobility

- The difference of timing step between the ASAP and ALAP for an operation.
- It is allowed to move the corresponding operations within the mobility region. -> this allows a better resource utilization by moving an operation from a busy step to a free step.




4 steps, 3 units needed by moving v10 to C-step1 or C-step2.

## ❖ List scheduling (resource-constrained minimum-latency)

- example 1

  $a_1$ = 2 mult

  $a_2$ = 2 ALU

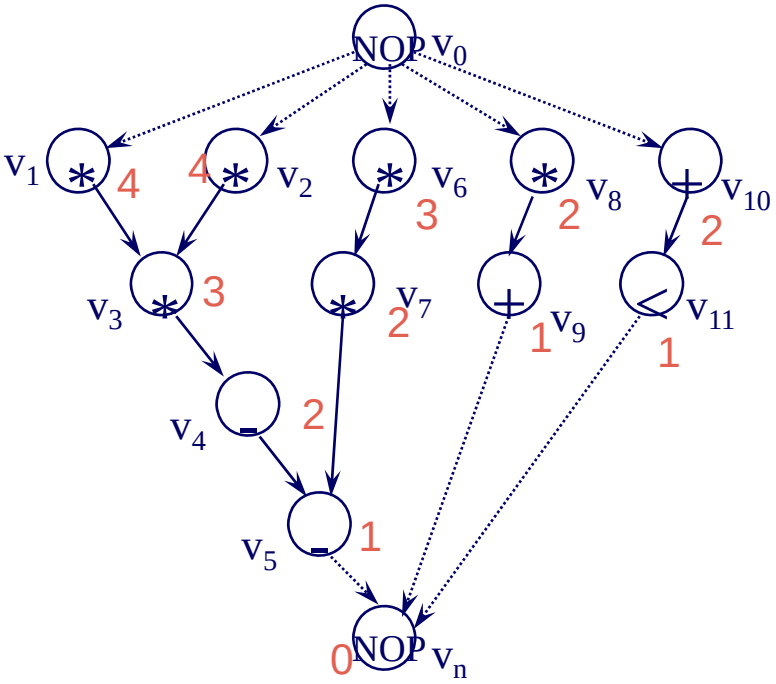
  $\{v_1, v_2\}$ $\{v_{10}\}$

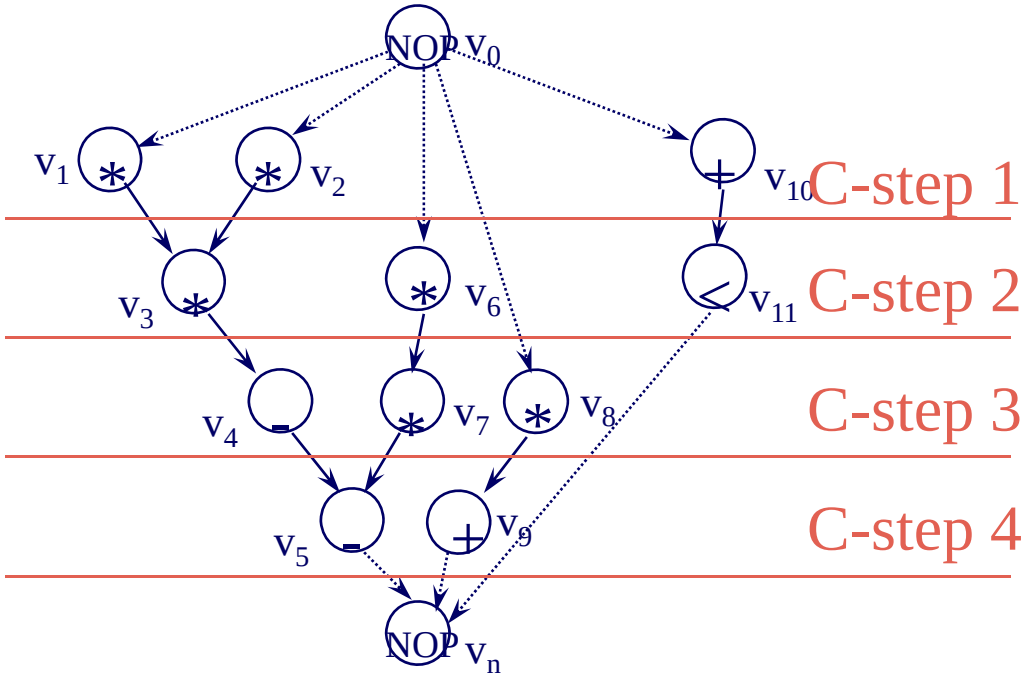
  $\{v_3, v_6\}$ $\{v_{11}\}$

  $\{v_7, v_8\}$ $\{v_4\}$

  $\{v_5, v_9\}$

Schedule the operations that are urgent (which are in the critical path, or longest way to go to the completion) first.



## ❖ List scheduling (latency-constrained minimum-resource)

- example

$a = [1, 1]^T$

$\{v_1, v_2\}$ ---> $a = [2, 1]^T$

$\{v_3, v_6\}$

$\{v_7, v_8\}$

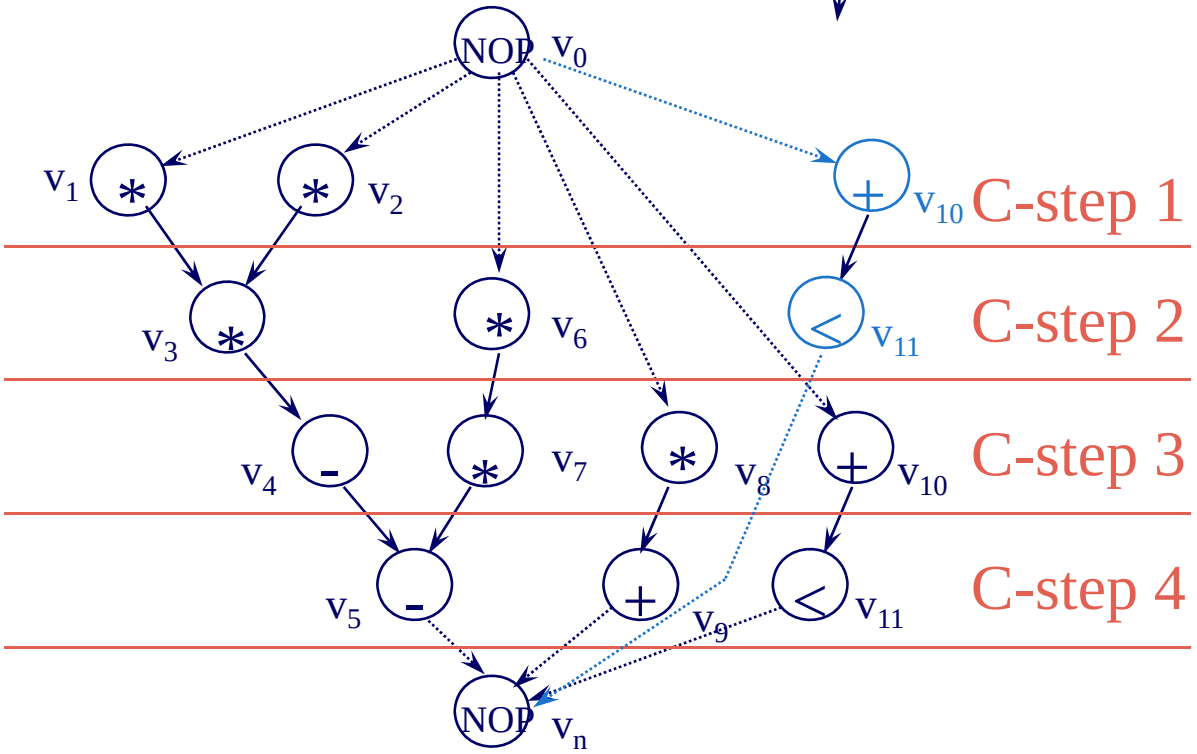
zero slack

$\{v_{10}\}$

$\{v_{11}\}$

$\{v_4\}$

$\{v_5, v_9\}$ ---> $a = [2, 2]^T$



# Resource sharing and binding

- **Scheduling before binding**
  - resource dominated circuits: operation scheduling
- **Binding before scheduling**
  - general circuit: mux and wire delay/area may not be ignored
- **General circuit**
  - scheduling affects binding

    ---> affects the use of mux, wire, and register

    ---> affects delay and area (non-linear function of binding $B$)

    ---> affects scheduling
  - use piecewise linear functions and solve scheduling and binding simultaneously with an ILP solver
  - iterate scheduling and binding
  - simulated annealing
  - genetic algorithm

# Scheduling and retiming

- **Retiming changes the signal flow graph (starting and ending points)**
- **Ex: 2nd order IIR filter : 4 mult, 4 add/output**
  - Time multiplexing ratio (system clock freq. / sampling freq.) = 4

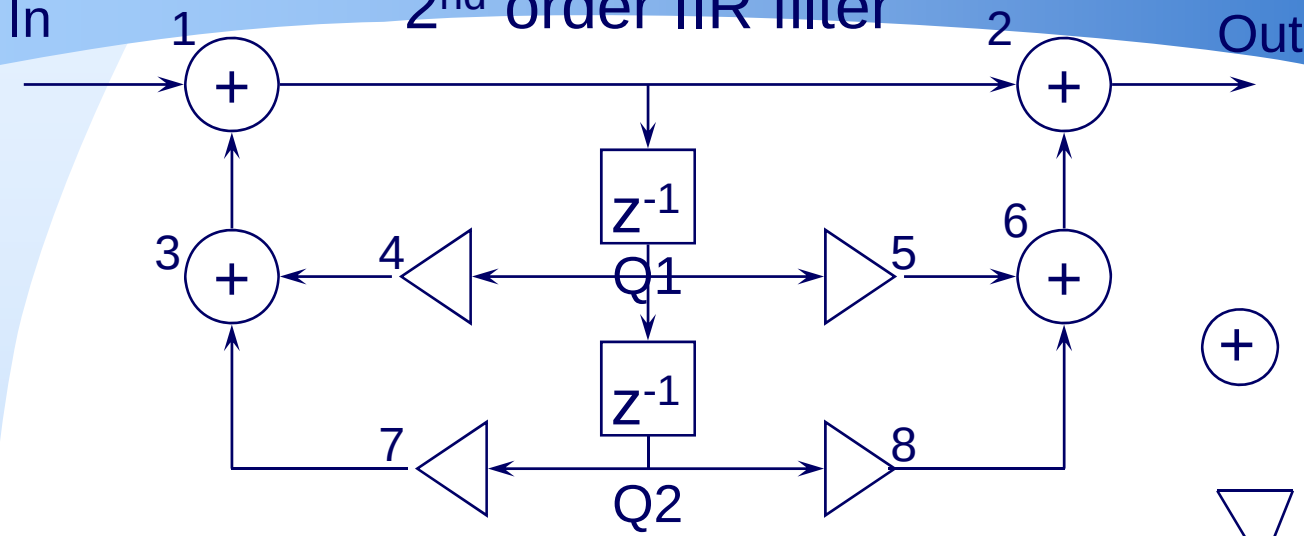
| CYCLE | MULTIPLIER | ADDER |
|---|---|---|
| 1 | 4, 7 | |
| 2 | 5, 8 | 3 |
| 3 | | 1, 6 |
| 4 | | 2 |

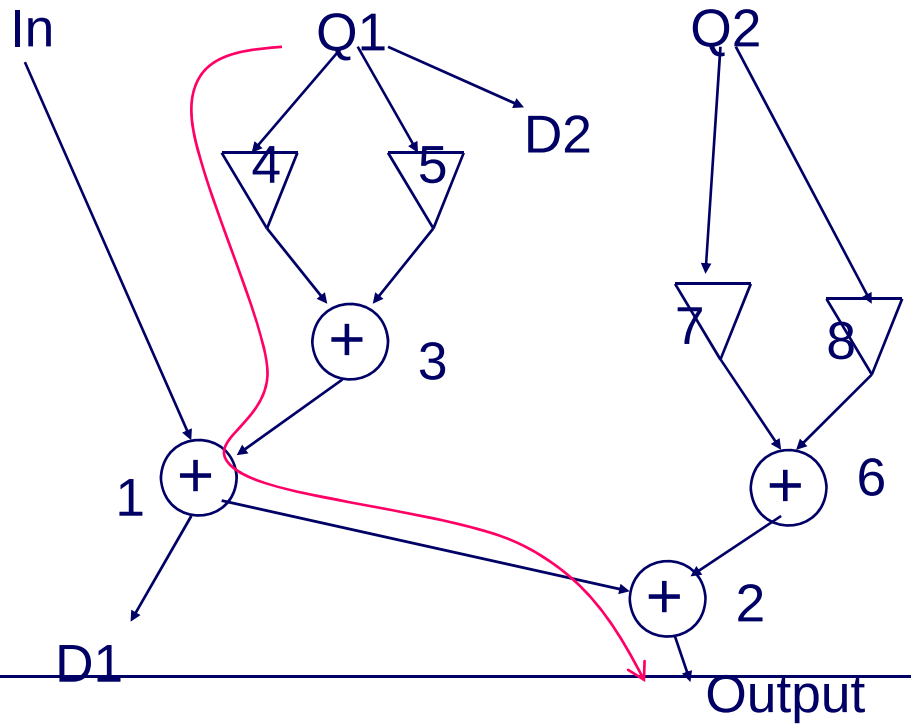
IIR filter scheduling before retiming (2 mult, 2 adder needed)

| CYCLE | MULTIPLIER | ADDER |
|---|---|---|
| 1 | 5 | 1 |
| 2 | 4 | 6 |
| 3 | 7 | 2 |
| 4 | 8 | 3 |

IIR filter scheduling after retiming (1 mult, 1 adder needed)

# 2nd order IIR filter

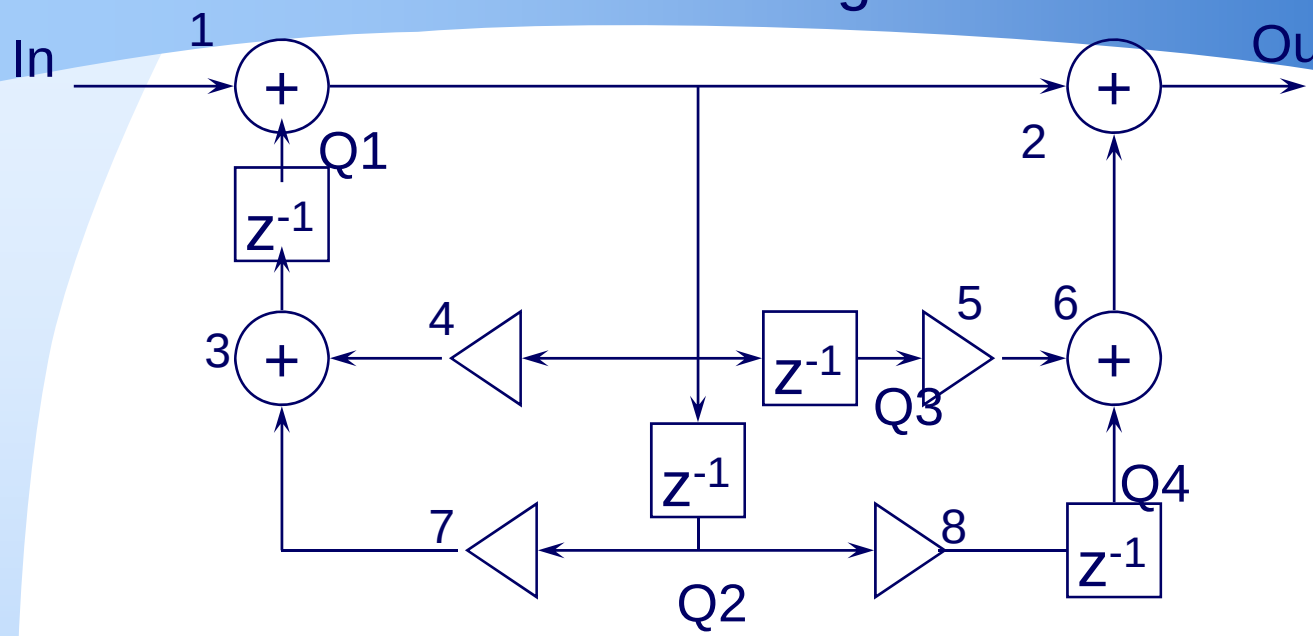


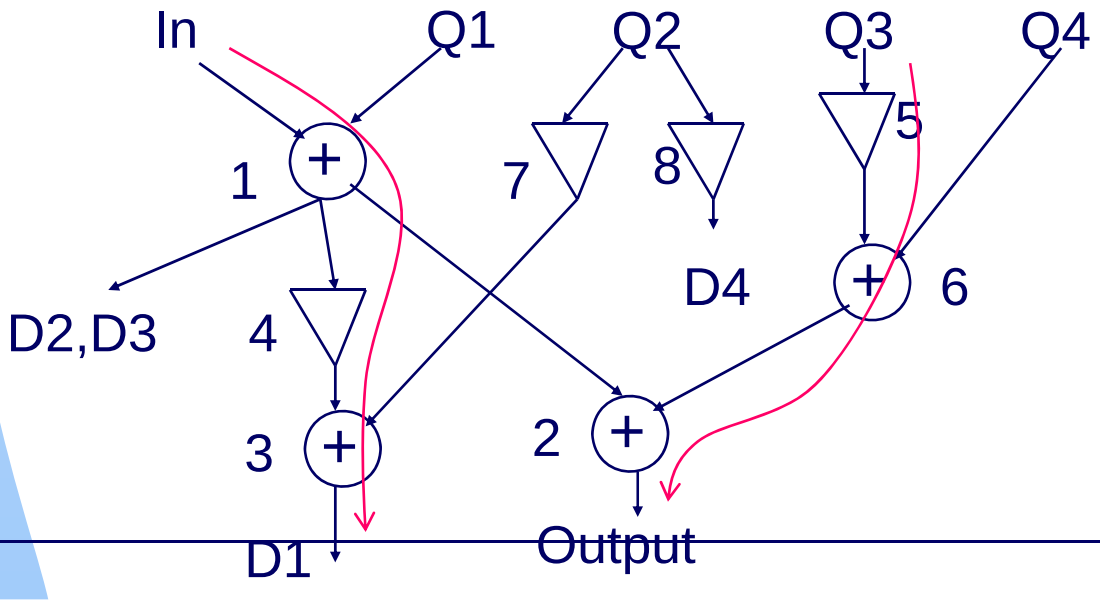




Tcritical

= Tmul+3Tadd

# After retiming

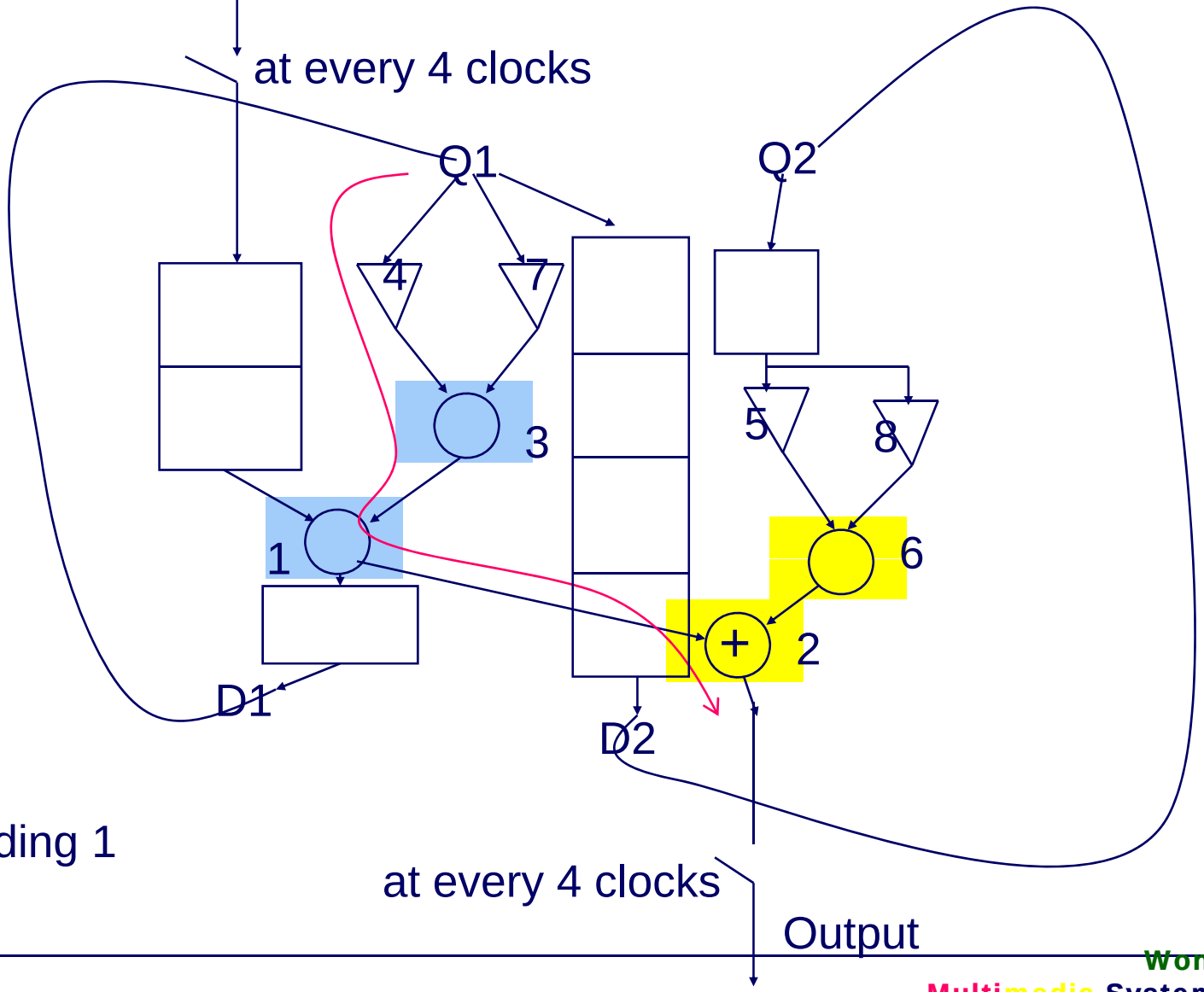


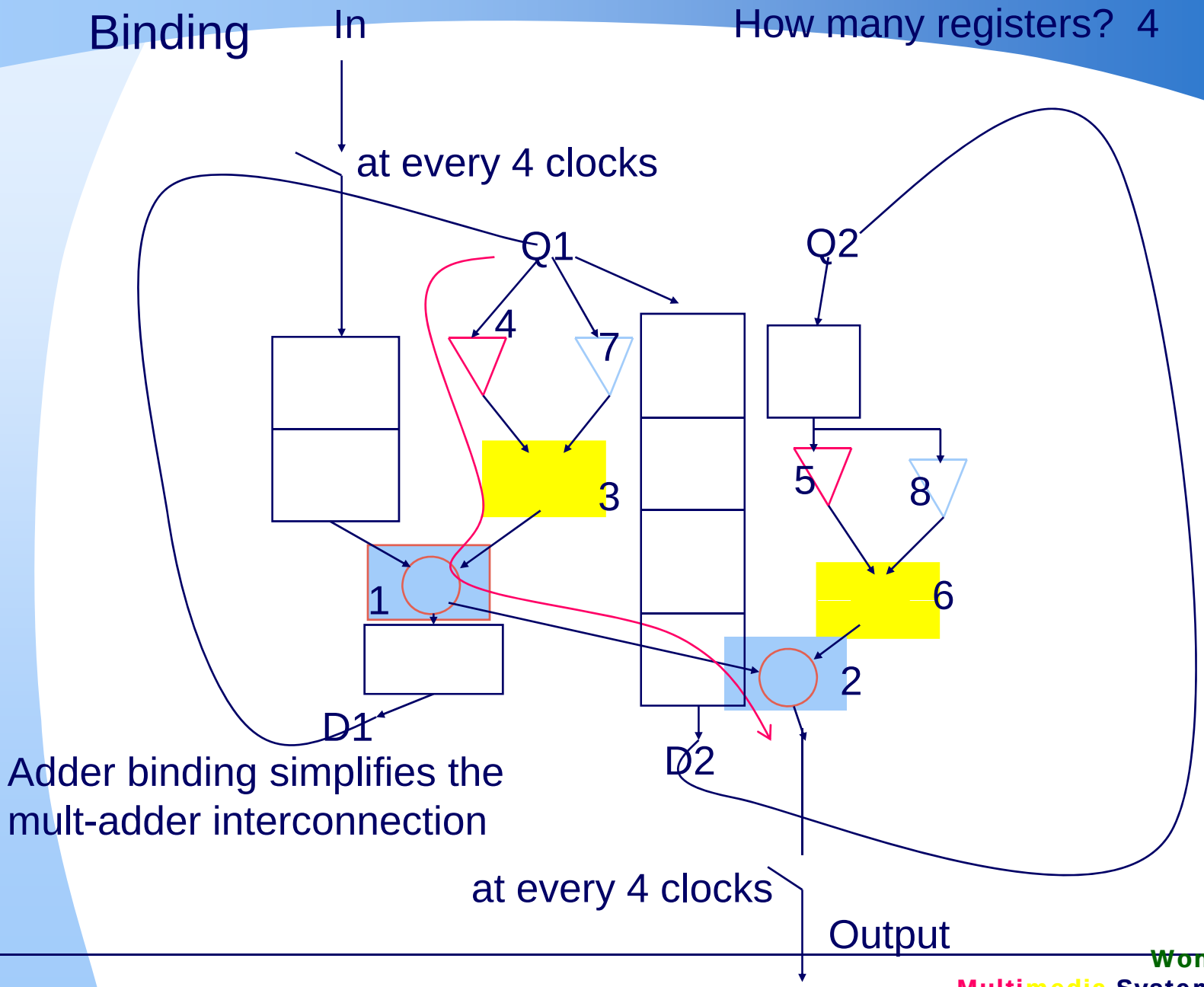
# Memory design

- **Why needed?**
  - To store the state variables shown in the flow graph (retimed version needs 4, while the original needs only 2)
  - To store the early finished results for synchronization. In the original flow graph, if it is scheduled in 4 clock cycles, “In” signal needs to wait 2 cycles to be added.
- **Memory architecture**
  - Addressable memory based: flexible but need more area. Maybe a bottleneck for high throughput (in this case, multi-ported, or multiple memory blocks are needed).
  - Distributed register based: inflexible, but good for high-throughput

How many registers? 4

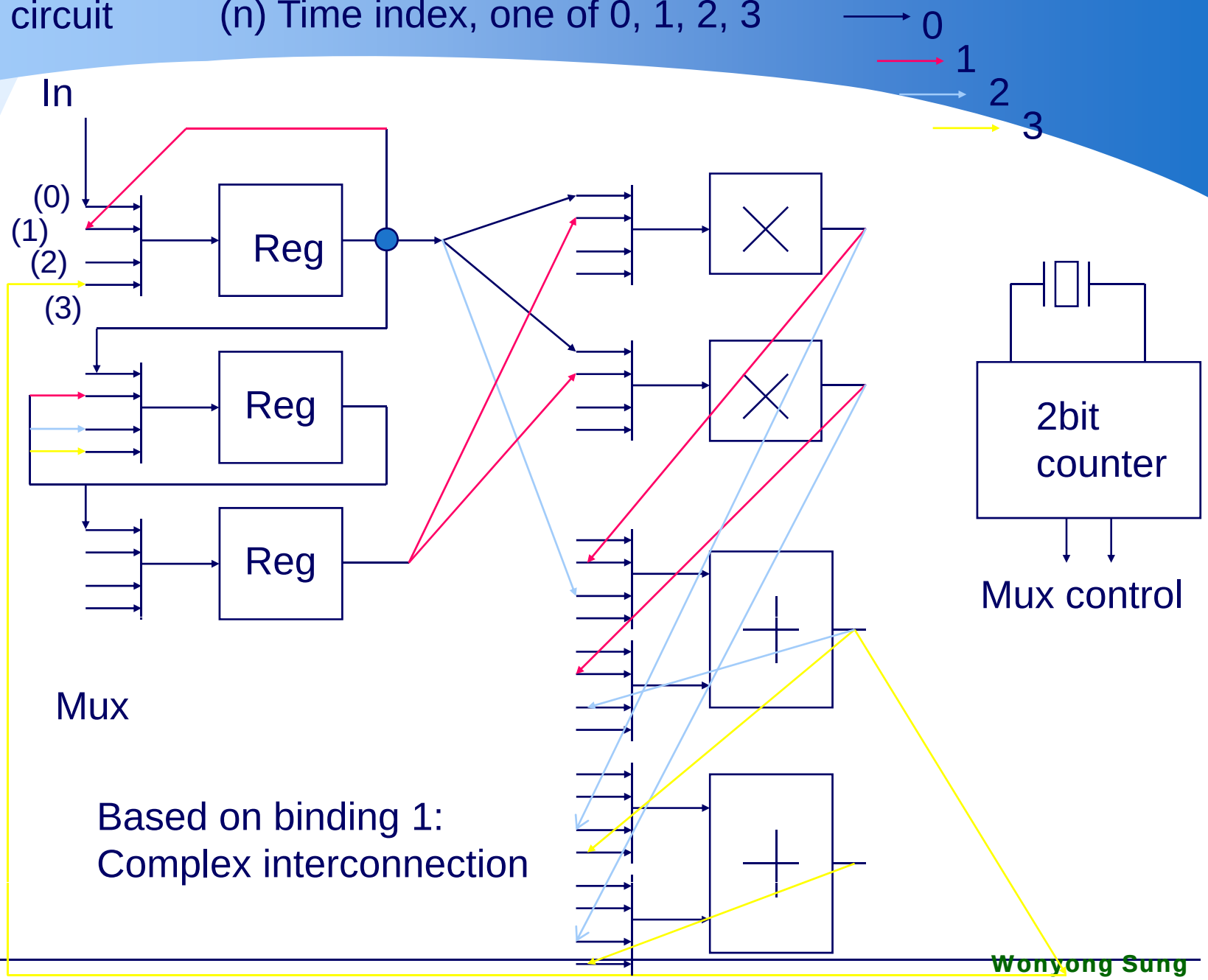


# Binding

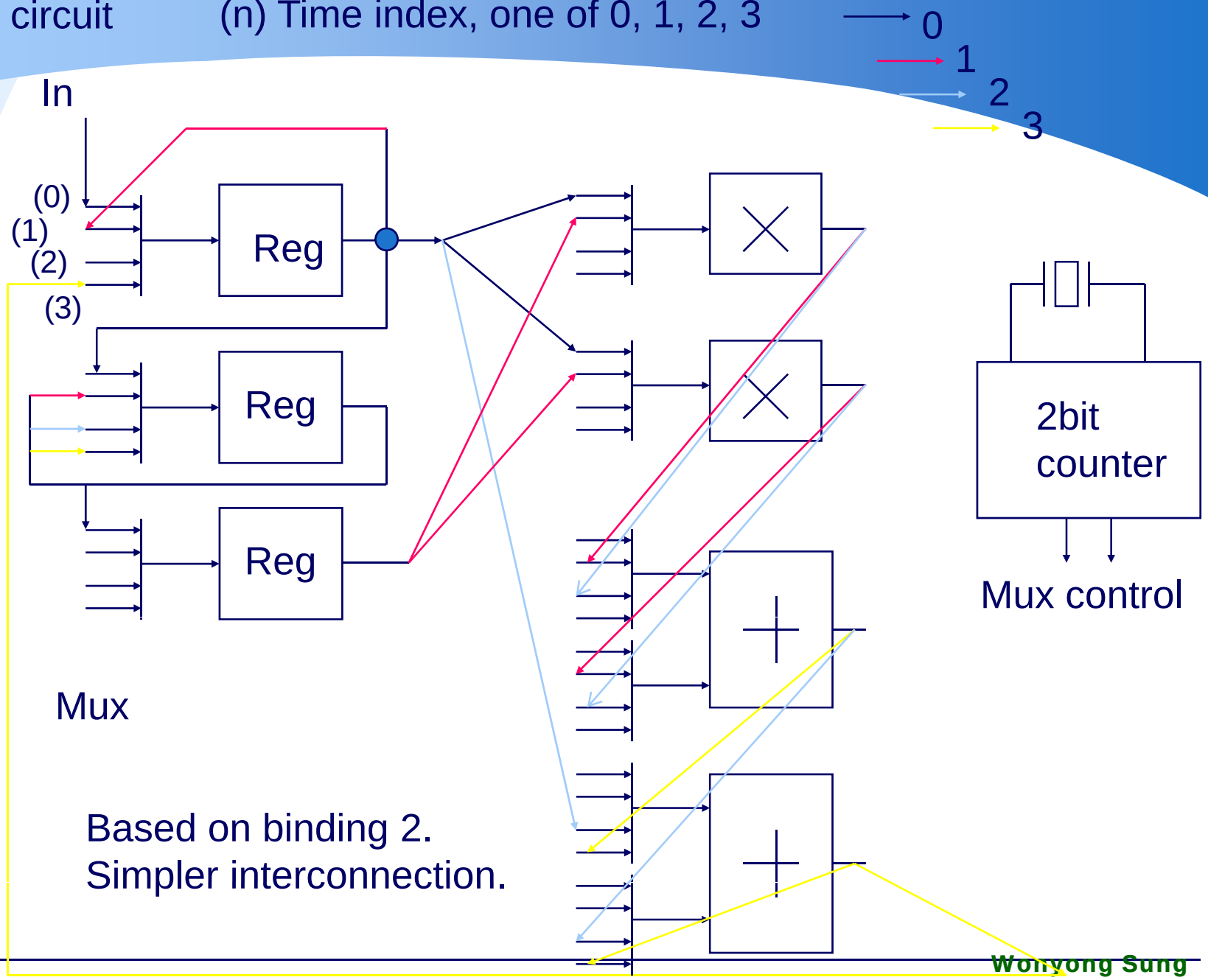



Adder binding simplifies the mult-adder interconnection

Whole circuit (n) Time index, one of 0, 1, 2, 3



Whole circuit (n) Time index, one of 0, 1, 2, 3



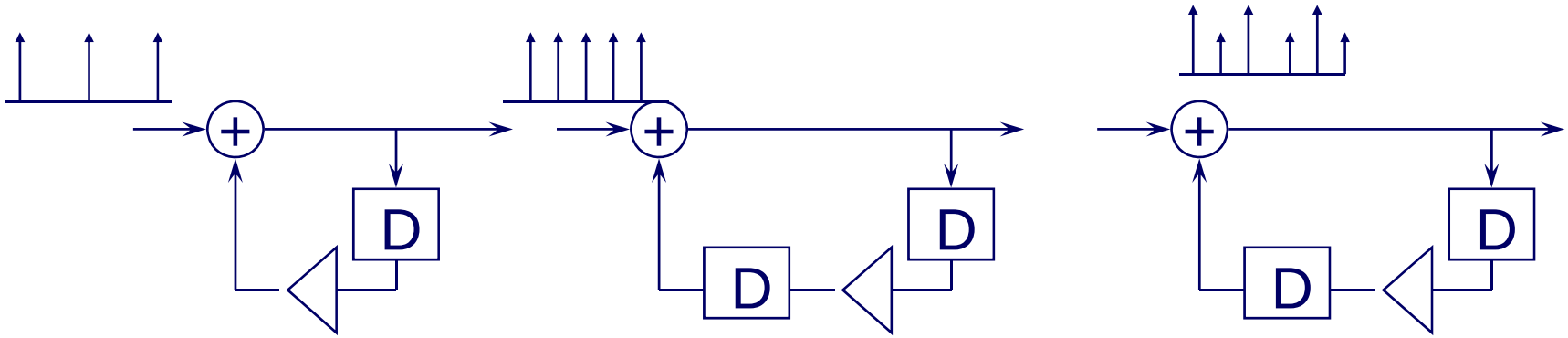

Based on binding 2.
Simpler interconnection.

# Interleaving and iterative structure based design

- **Interleaving**
  - Processing multiple input channel alternatively. So, it is a kind of time-multiplexing supporting the same function for both channels.
  - $z^{-1}$ corresponds to two (or interleaving factor) clock delays, which leads to shorter loop bound for a recursive loop.
  - Can increases the efficiency of the hardware, but do not increase the throughput for a certain channel.
- **Application: stereo processing with mono hardware. Multi-stage system implementation**

# Interleaving

- **Time multiplexing through interleaving**
  - Insert two (or three, …) registers for one z-1, and retiming for critical path minimization. And, apply two (or three, …) channels of input. In this case, one register delay corresponds to $z^{-1/2}$ and the original transfer function is not changed.

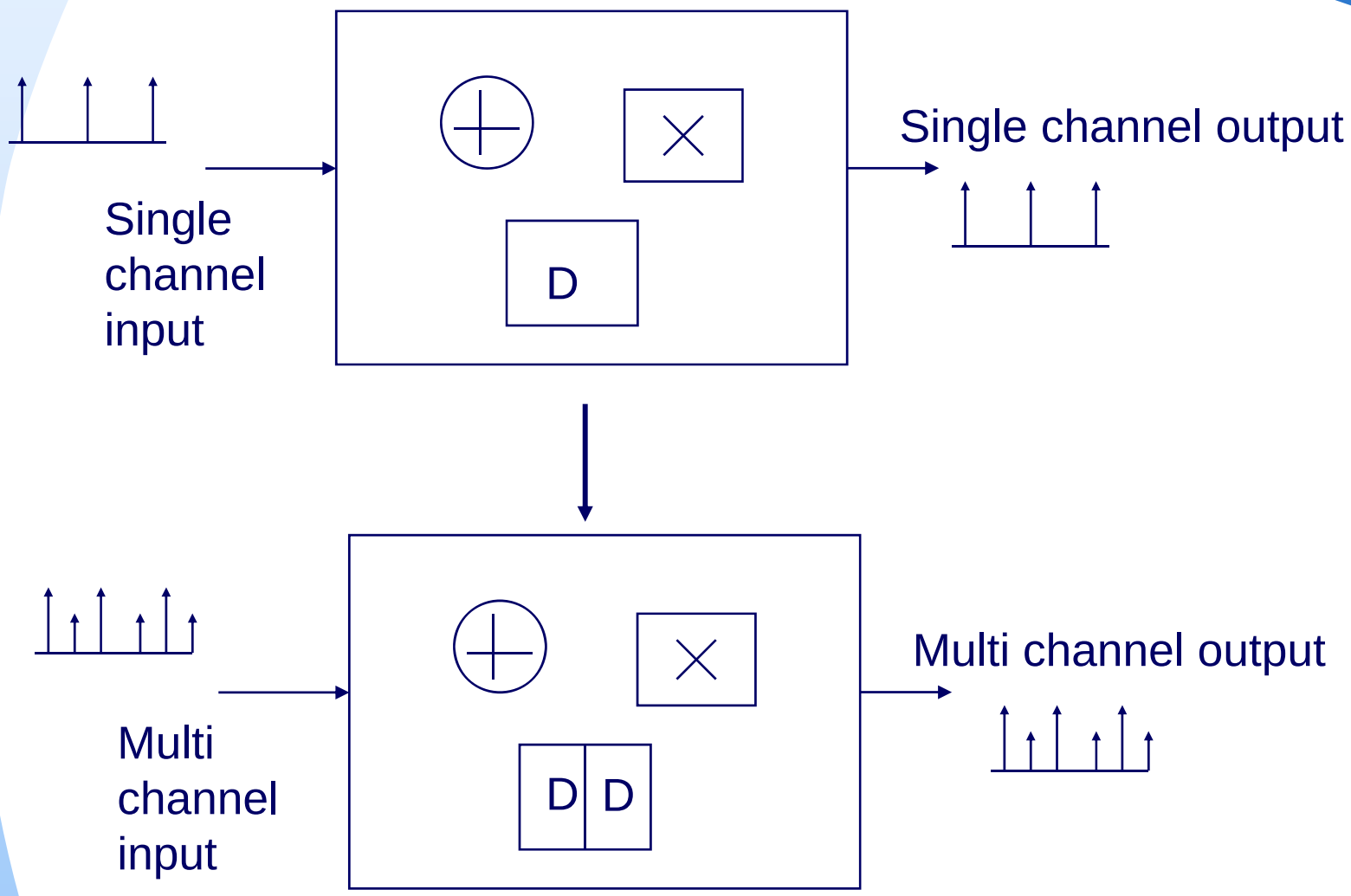



If D= $z^{-1}$ | If D= $z^{-1}$ | If D= $z^{-1/2}$

$H(z) = 1/(1-az^{-1})$ | $H(z) = 1/(1-az^{-2})$ | $H(z) = 1/(1-az^{-1})$

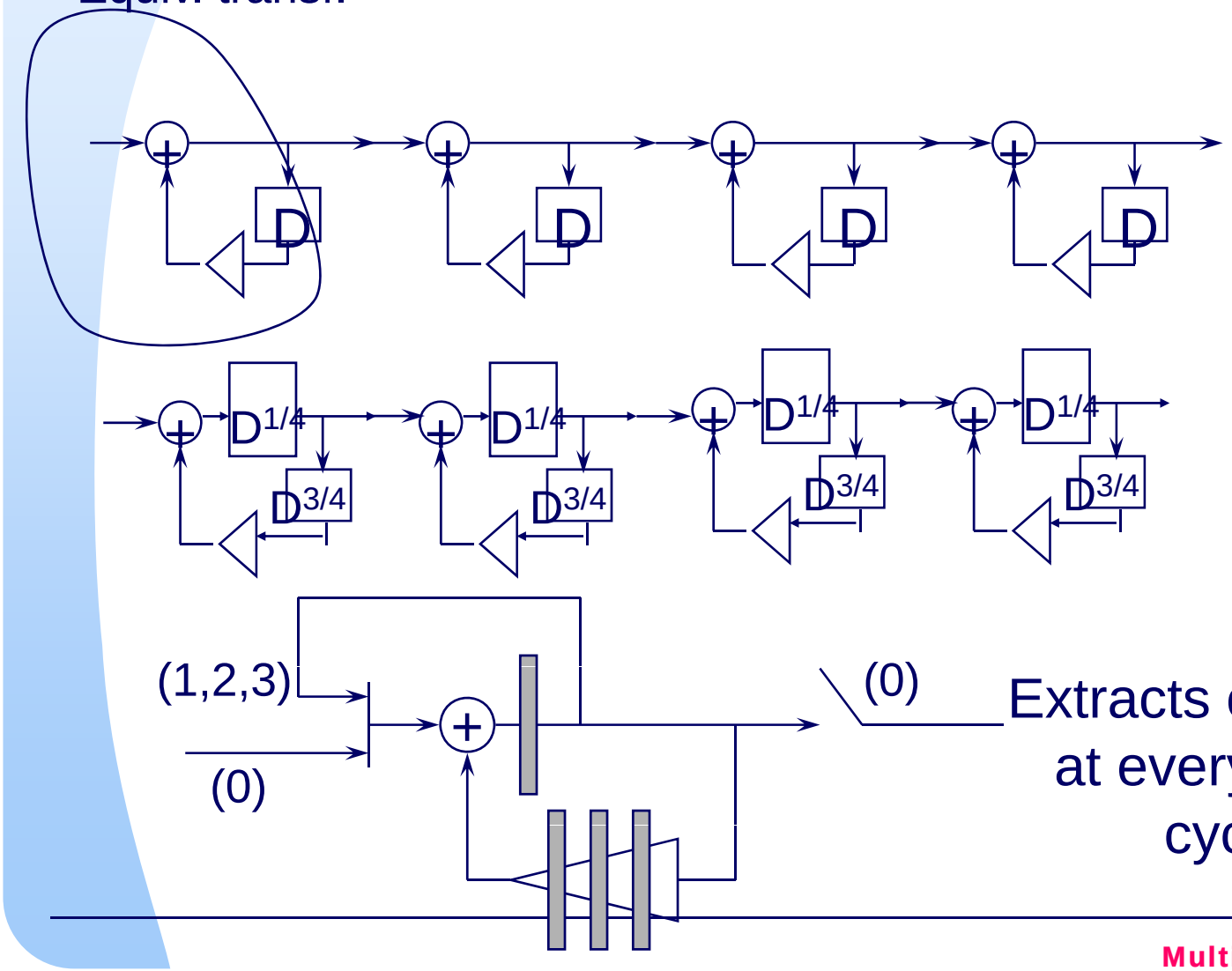
original | High throughput
But different filter

# Interleaving for multi-channel



# Interleaving for cascade or parallel systems

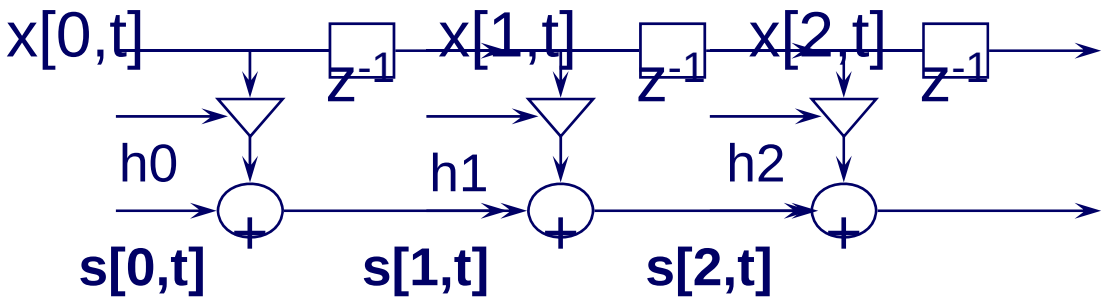
Equiv. transf.




Extracts one output at every 4 clock cycles.

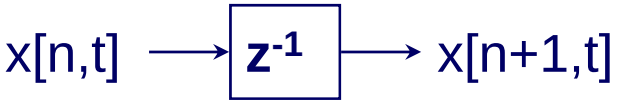
# Iterative structure based design of time-multiplexed architecture (generalized interleaving)

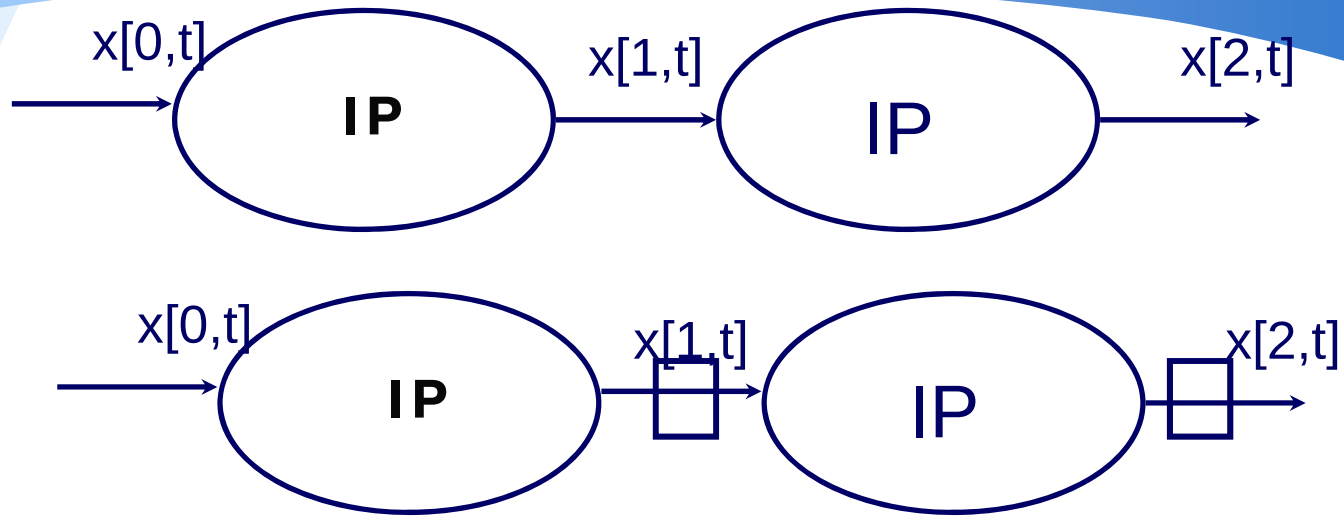
Digital filter: consists of iterative operations (stage, tap, or so on)



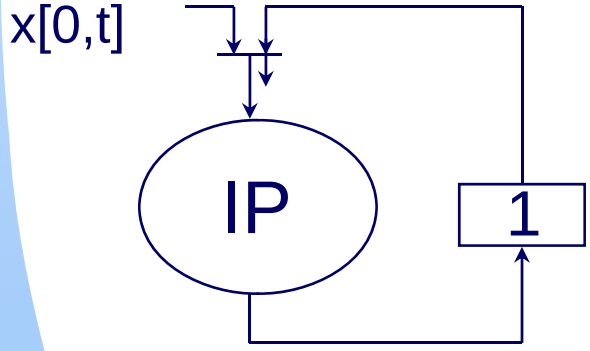


x[n,t]
n:stage index,
t:time index





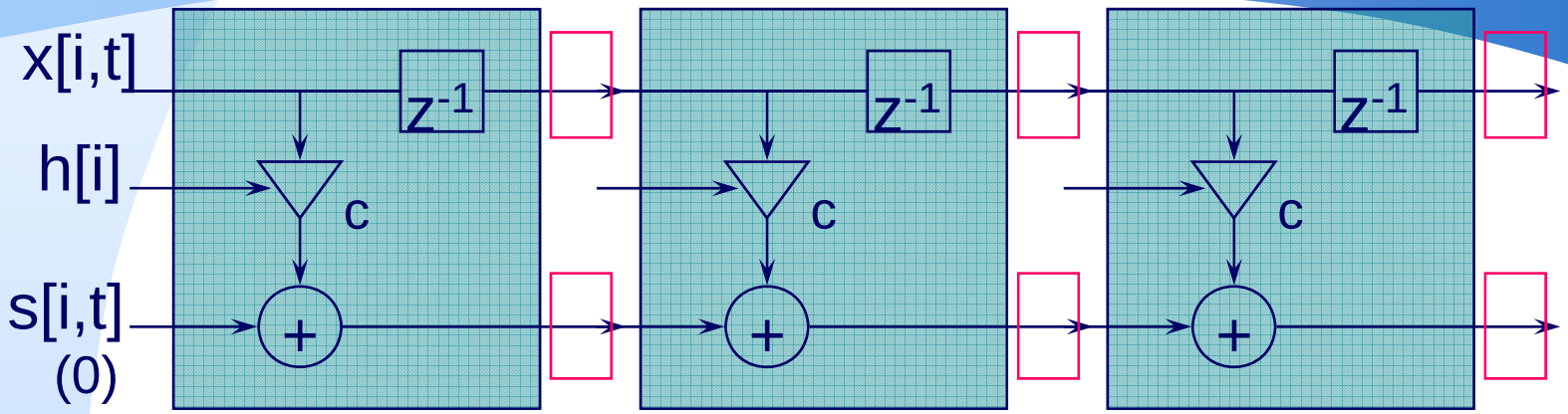

Forward computation: x[0,t] -> x[1,t] -> x[2,t]



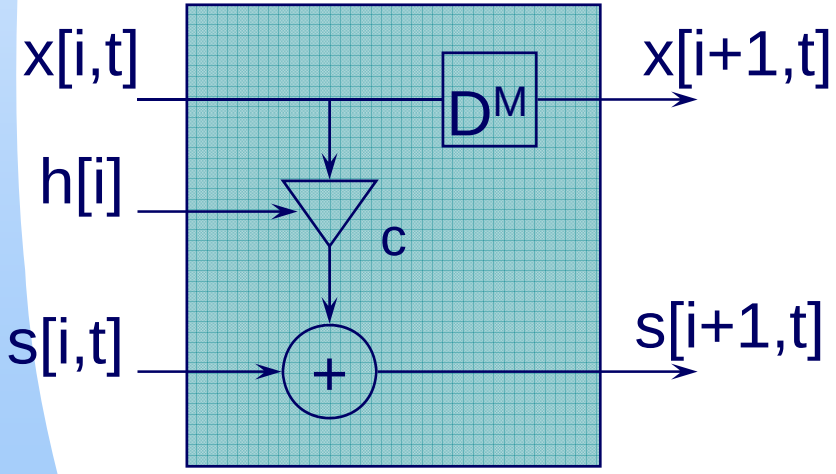

buffer: needed for HW operation

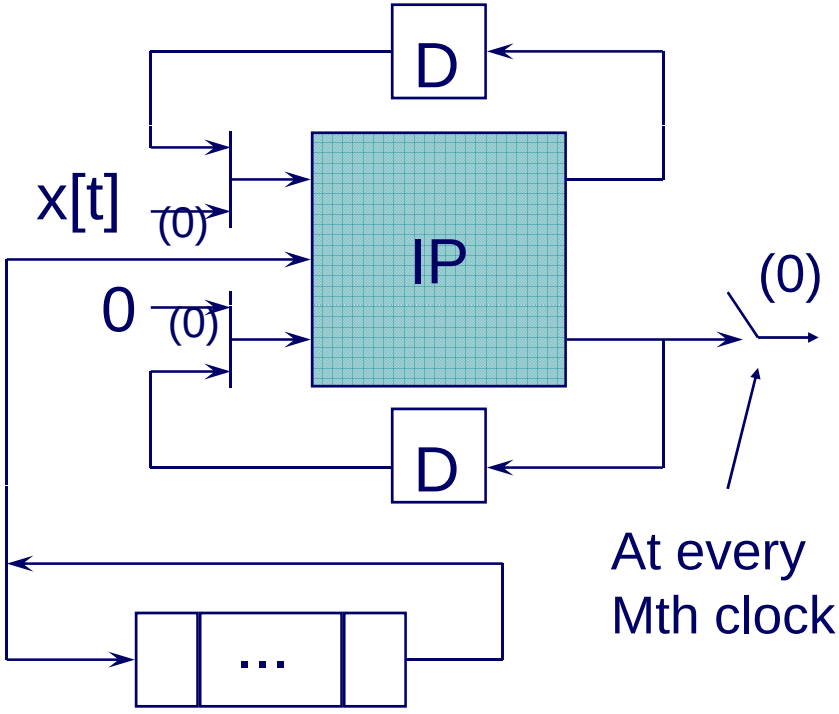
If the time-mux ratio is M,
$z^{-1}$ corresponds to M clock delays.

# FIR filter with time-mux ratio M




Iteration process



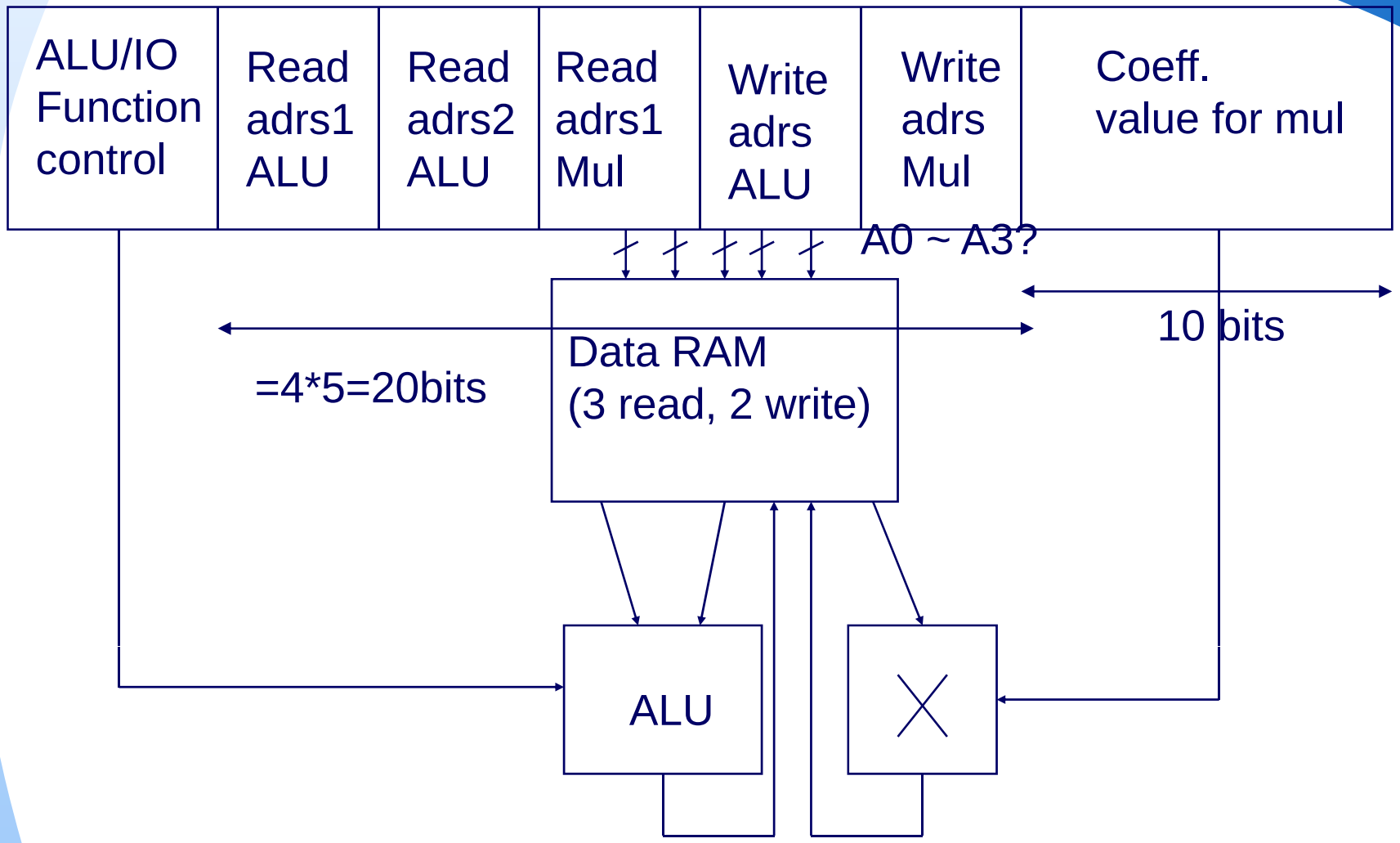



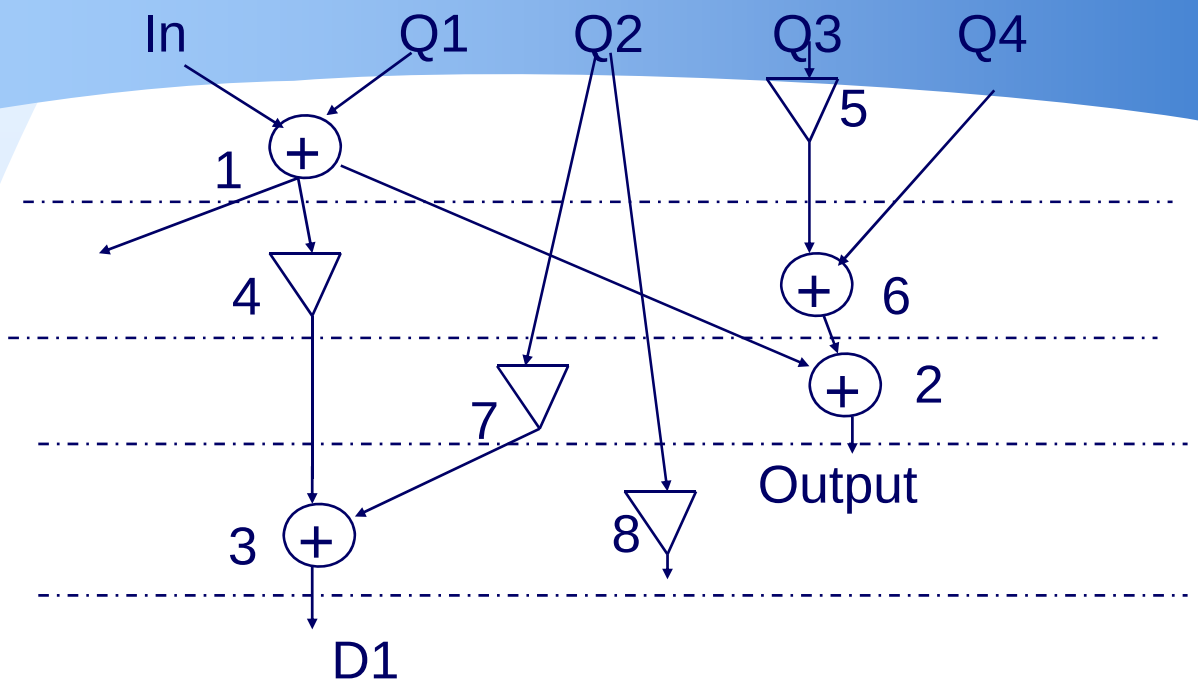
# Program based method

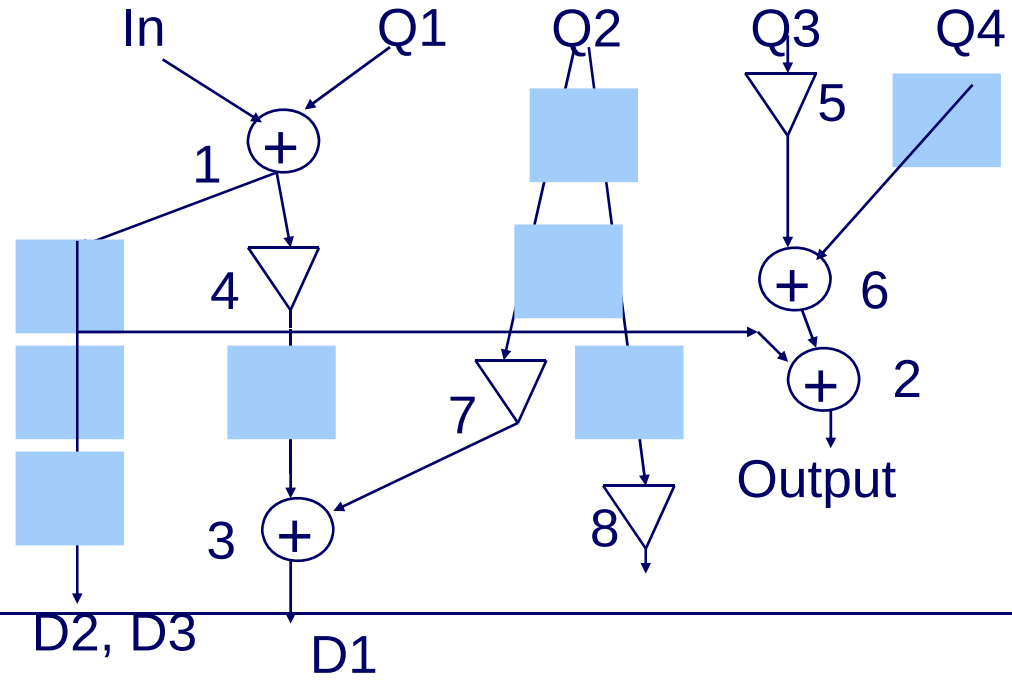
- Generate control signal using program ROM
- Consists of datapath, program ROM, data memory, and controller.
- Can optimize the data-path structure (the performance is better than the general purpose DSP's).
- CAD software => Cathedral II (microcoded multiprocessor architecture), Lisatek (application specific instruction set and data-path design)

# Mircro-program based implementation

Microprogram ROM: total 4 words, but very wide.






Determining the total number of registers

# Overall design procedure for program based architecture

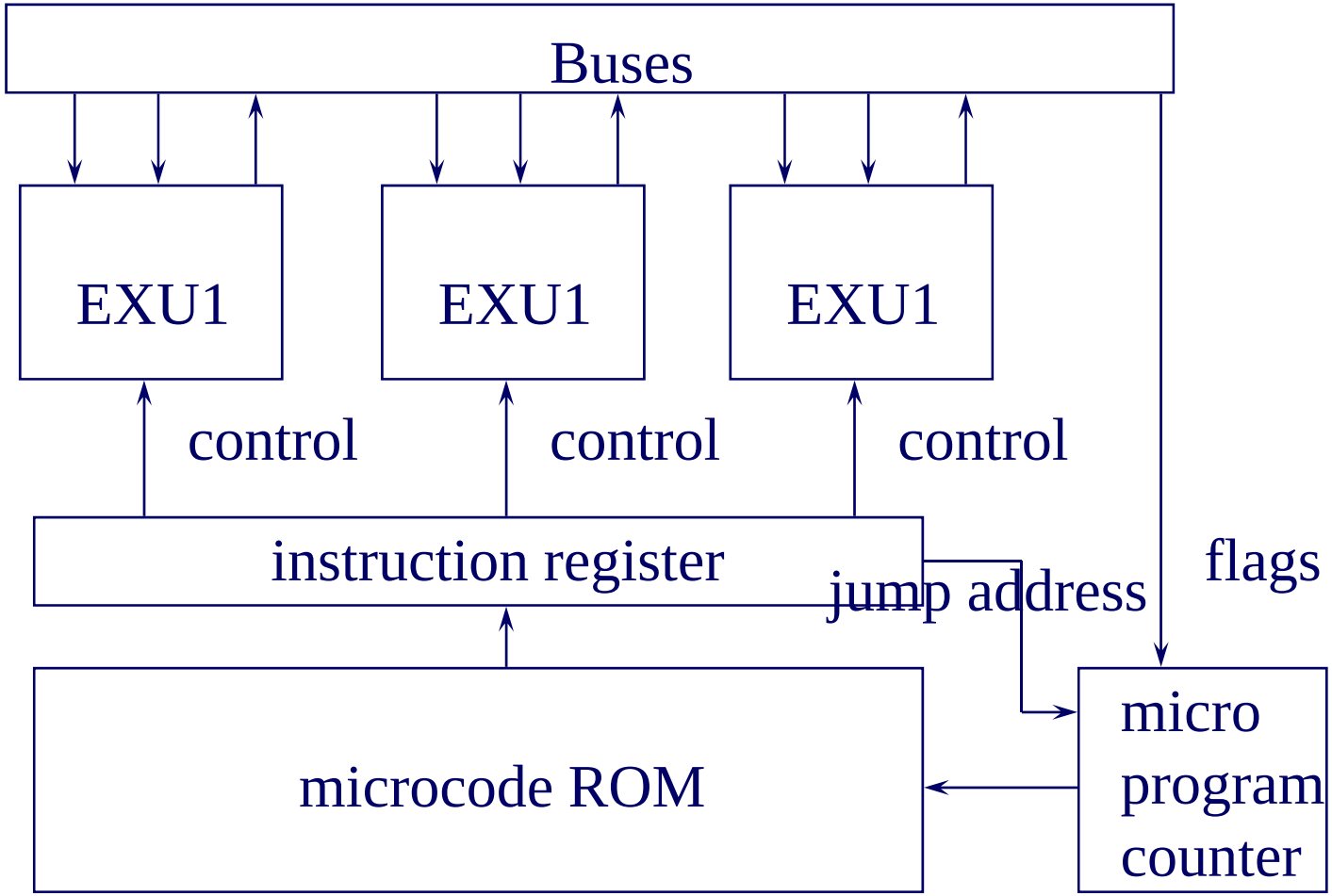
- Data-path structure design by scheduling and binding
- Memory system design
- Interconnection of the components or develop microprogram
- Control signal generation

# Cathedral (Mistral) II

**A silicon compiler for complex decision making applications in the KHz - 1Mhz range**

- micorcoded architecture
- multiple parameterizable execution units
- behavioral specification in Silage

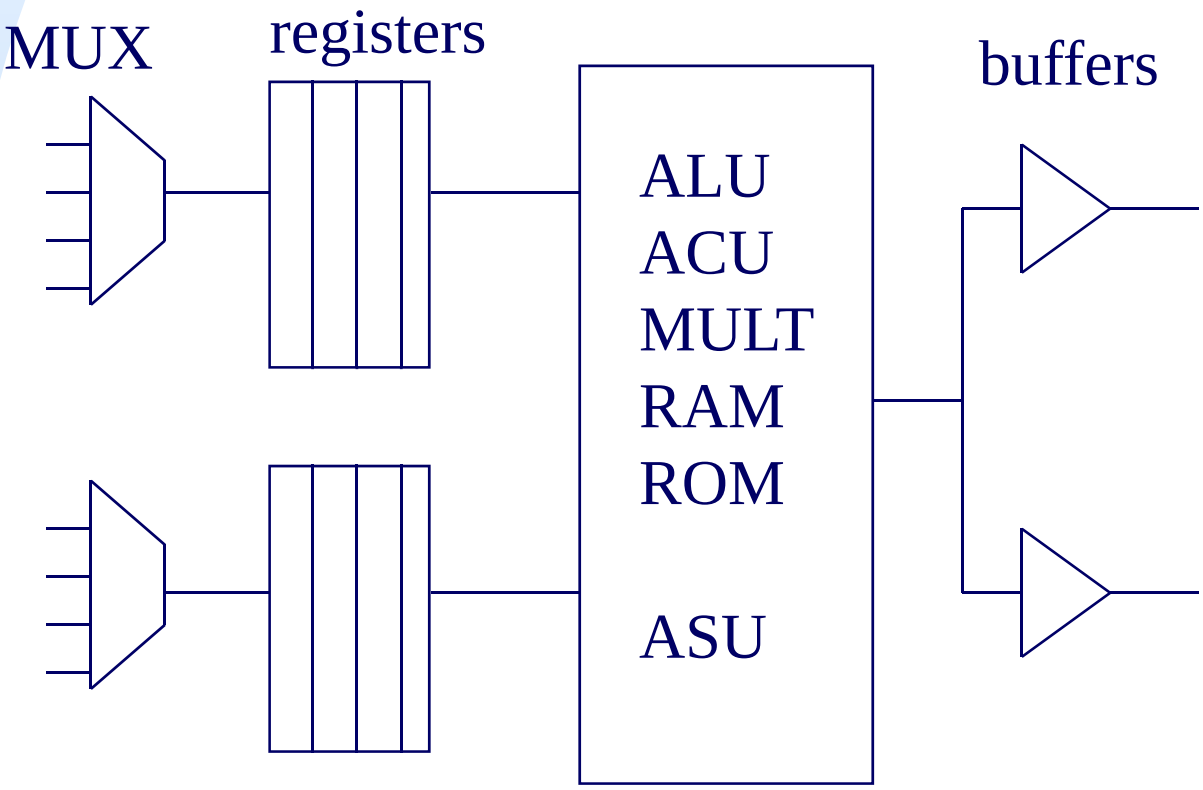
# Microcoded processor architecture



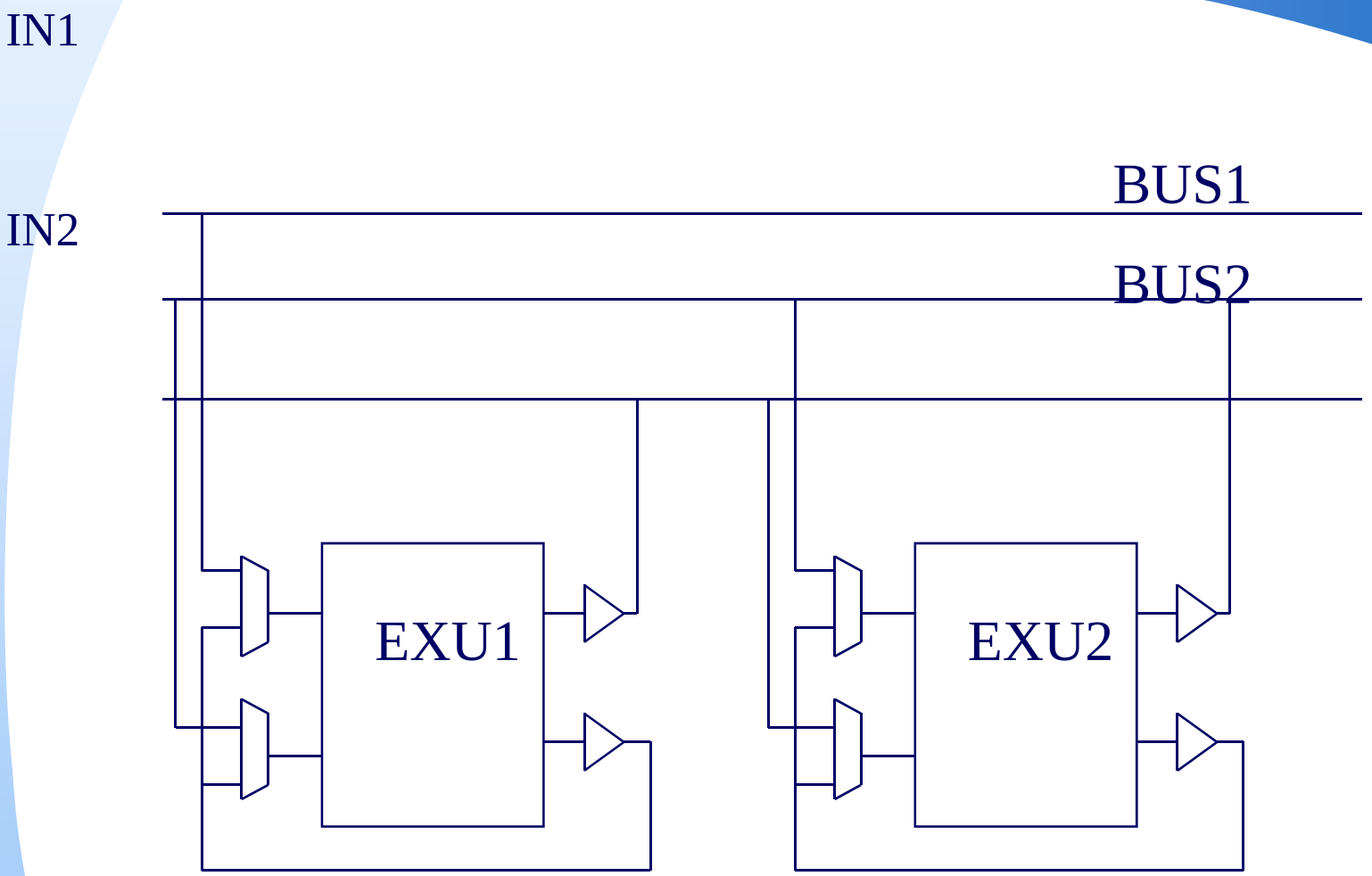
# EXU overview

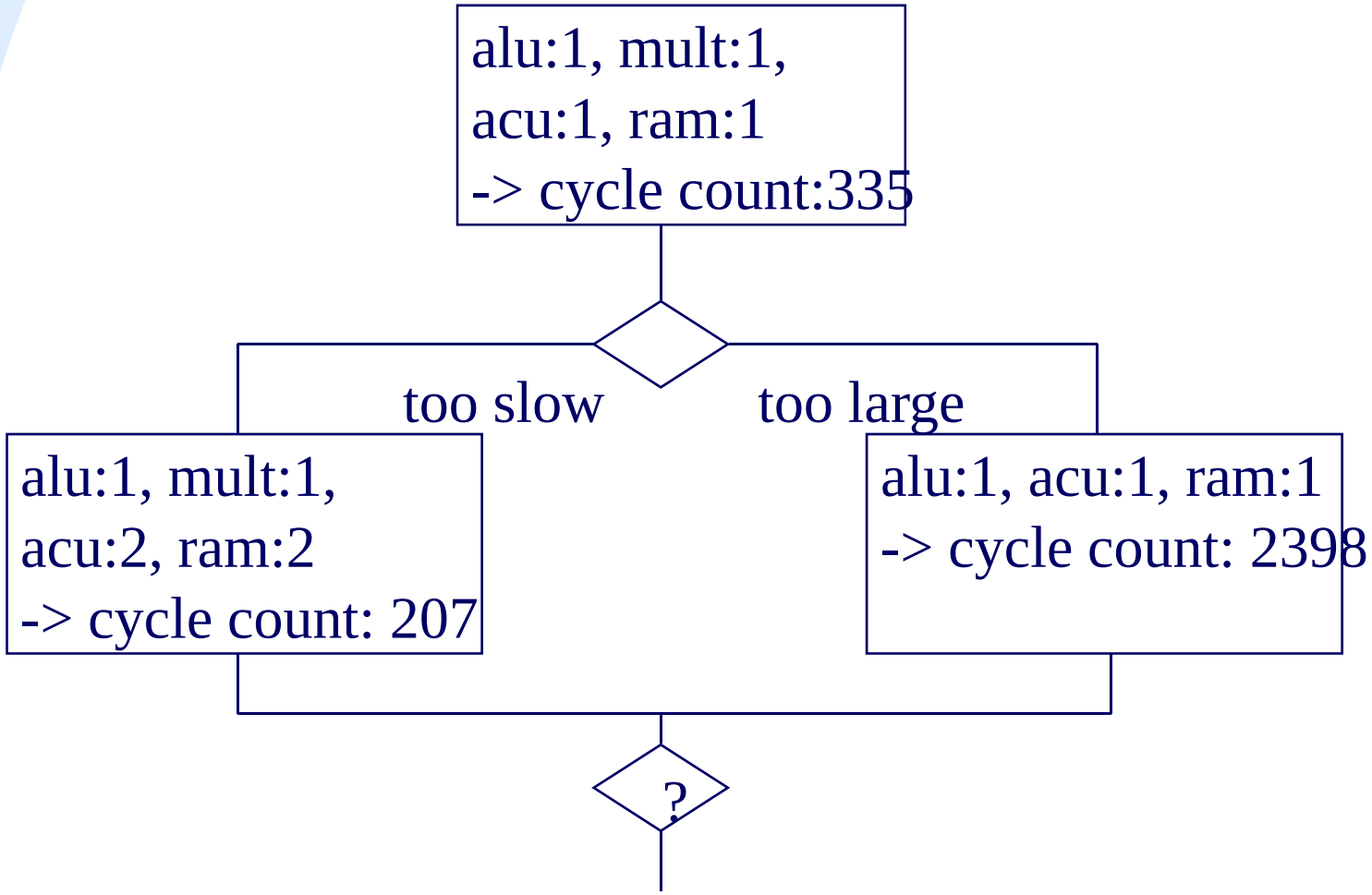
- **Arithmetic EXU:**
- **ALU: 2's comp ALU operations**
  - ACU: unsigned arithmetic modulo comp.
  - MULT
- **Memory EXU**
  - ROM and RAM
- **I/O EXU**
  - In, Out, Tri, IO(bidirectional)
- **Controller EXU**

# EXU architecture



# Multi processor architecture




alu:1, mult:1,
acu:1, ram:1
-> cycle count:335
too slow
too large
alu:1, mult:1,
acu:2, ram:2
-> cycle count: 207
alu:1, acu:1, ram:1
-> cycle count: 2398
?

# Architecture comparison

- **general purpose DSP**
  - have a fixed data-path (usually multiply and accumulate)
  - program width 16 - 32 bits
  - flexible programming including C language
  - only code generation required
- **hardwired DSP (Mistral-I, Mistral-III)**
  - have a very flexible data-path
  - not good for decision making (if ..)
  - mostly data-path generation

# Architecture comparison - cont.

- **Microcoded processor architecture**
  - flexible and multiple data-path structure
  - program width: 32 - 256 bits
  - programmable, but code space requirement is less efficient
  - need both data-path and code generations
  - good for algorithms requiring specific data-path architectures with decision making

  e.g. speech pitch extractor, speech coder

# 4. Bit Serial Architecture

- **Bit-serial, operation dedicated**
  - Use bit-serial multipliers (complexity of a parallel adder), bit-serial adders, and shift-registers
  - processes one bit of signal at one clock using a dedicated one-bit arithmetic or memory elements -> slowing down the effective $f_{system}$
  - almost hardwired (simple) control
  - efficient implementation (good for digital filters), limit in throughput
  - Limitation: hard to be applied to control intensive algorithms.

# Timing of bit-serial operation

- **LSB (least significant bit) first**
  - supply the LSB of a signal first,
  - carry propagation is allowed
  - can employ ordinary number system
  - needs large delay(latency) for multiplication, limit for high throughput application

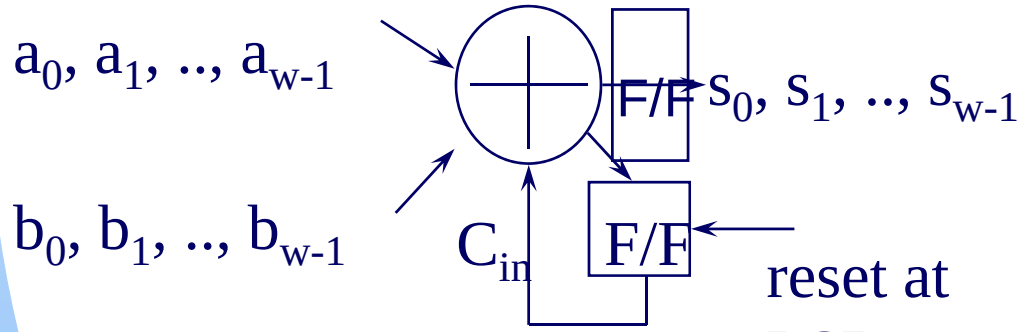
# Timing of bit-serial operation - cont.

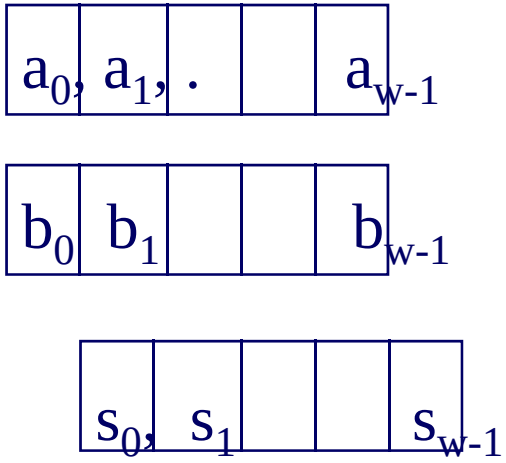
- **MSB(most significant bit) first**
  - supply the MSB of a signal first,
  - carry propagation is not allowed
  - redundant number system is used
    - Carry is propagated to only one stage
  - needs small latency, can be used for high throughput system
  - Complex and larger cell area

# Bit serial components – for ‘w’ clock/sample

Delay


Adder



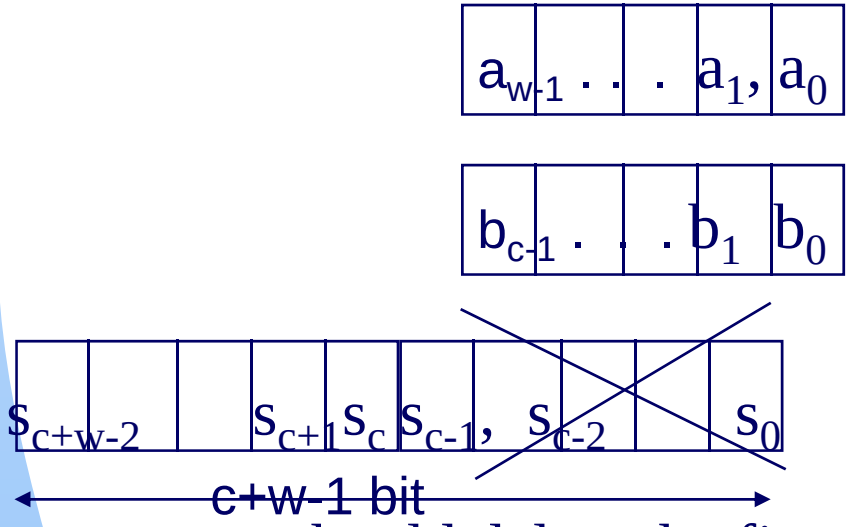

one clock latency occurred!

# Bit serial components - cont.

**scaler**

- 1 bit delay: *2
- 1 bit advance with sign extension: *0.5
  (implemented with relative delay)

**multiplier**




w bit of signal

c bit of coefficients

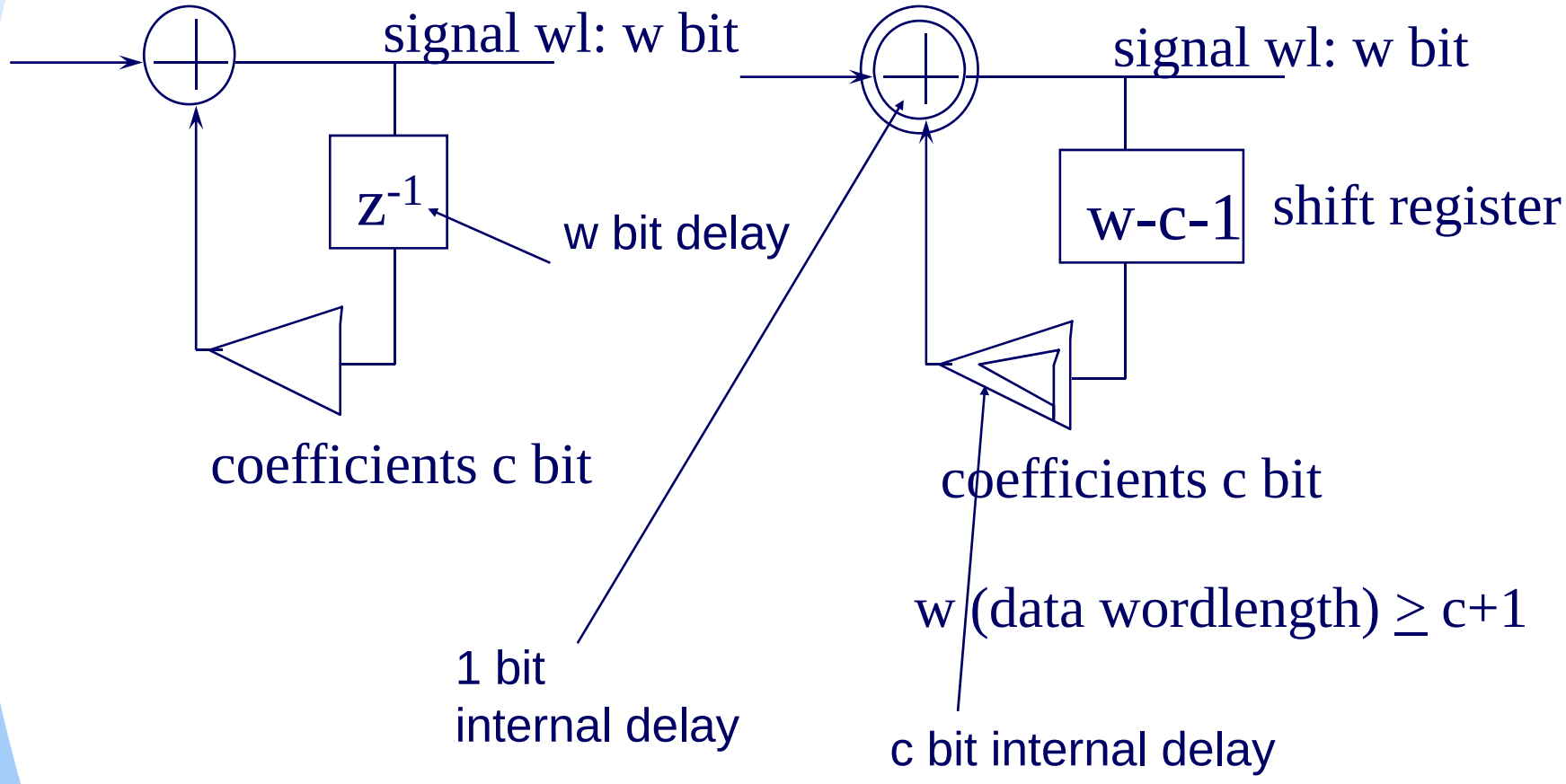
w+c-1 bit -> w bit

should delete the first c-1 bits -> total c bit latency

# Retiming and delay management

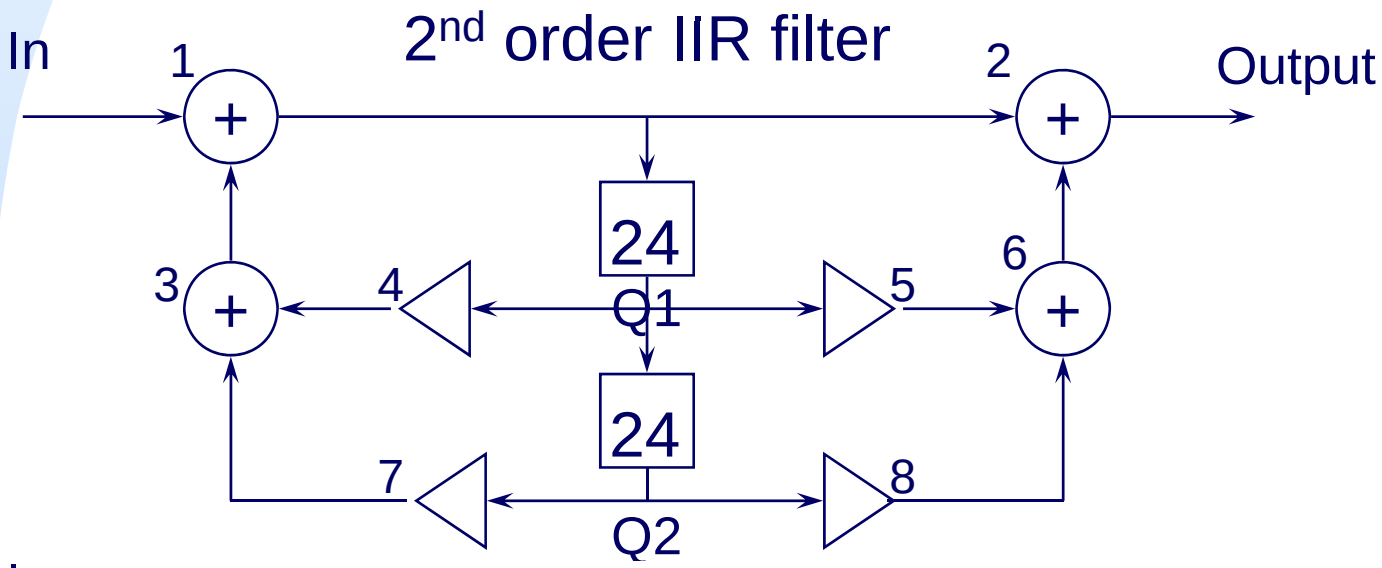
- Maximum throughput of a digital filter is determined by the number of delay blocks($z^{-1}$) and the total latency of the arithmetic blocks
- When the latency is large, the data wordlength for bit-serial implementation should be increased even if it is not needed for signal representation

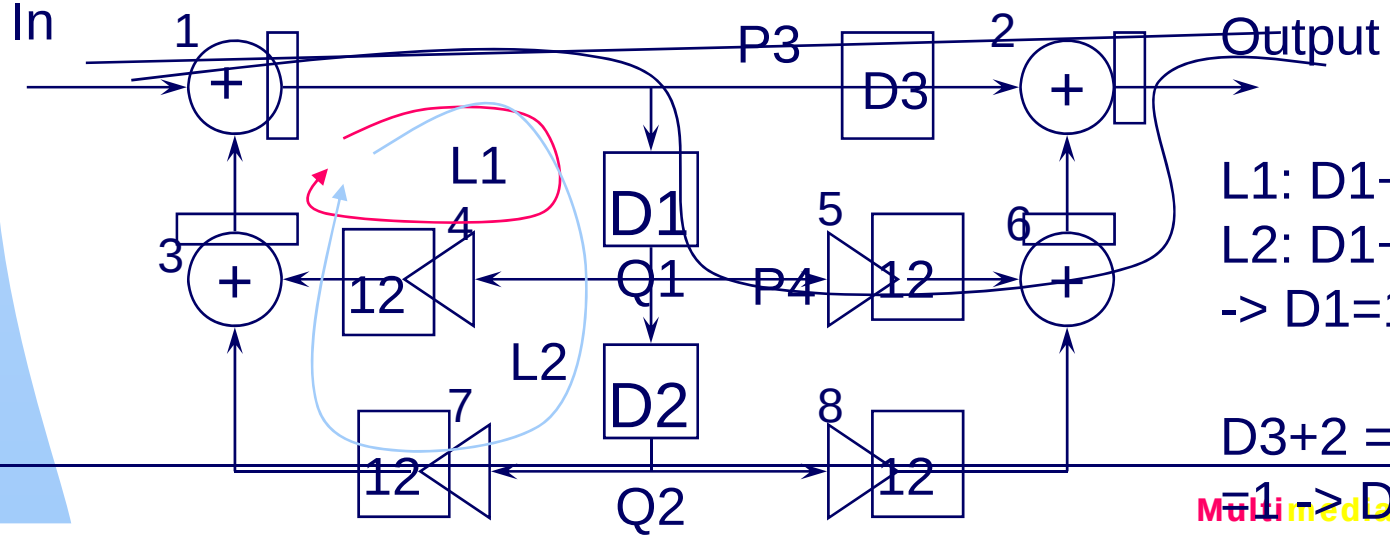
# Example - 1st order filter



# 2nd order bit-serial

Assume w=24, c=12



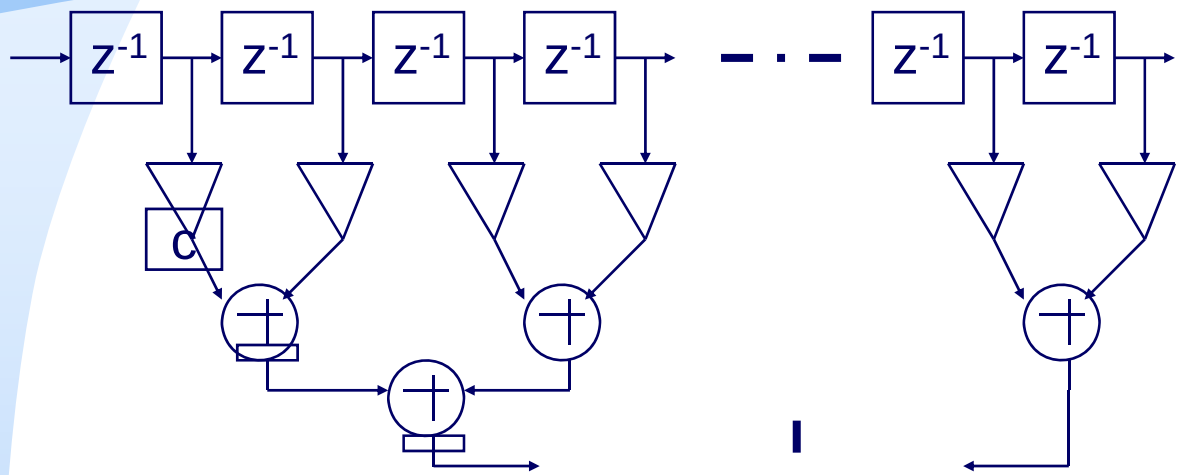

L1: D1+12+2=24
L2: D1+D2+12+2=48
-> D1=10, D2=24

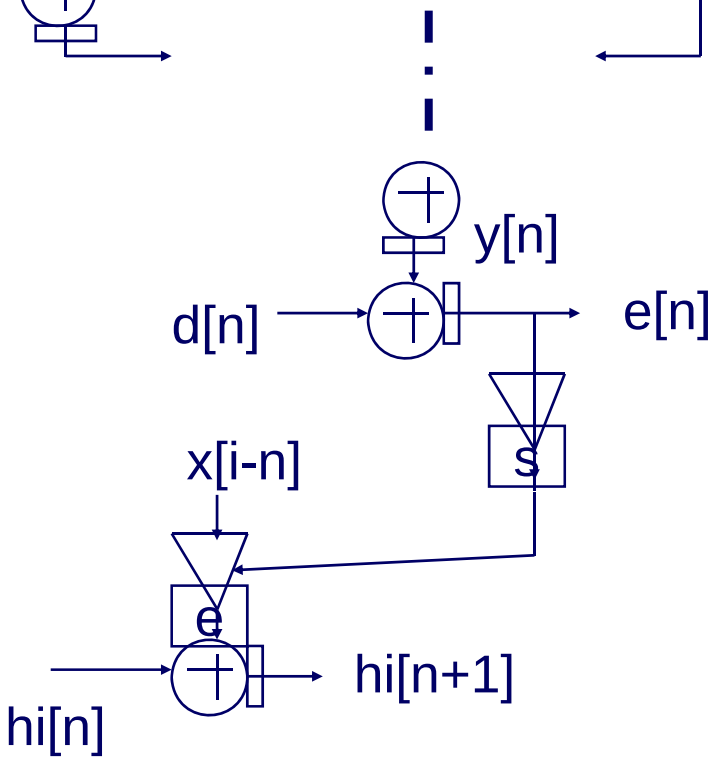
D3+2 = D1+12+3 – 24
=1 -> D3=-1

# Example - adaptive LMS digital filter

- data wl: w, coefficients wl: c,
  number of taps: N ($\log_2 N = M$),
  step size wl: s, error wl: e
- total delay in one sampling time = c+M+s+e
  -> w $\geq$ c+M+s+e

So, we may need to increase w (just for timing, not for better precision)


z-1
z-1
z-1
z-1
z-1
z-1
c


y[n]
d[n]
e[n]
x[i-n]
s
e
hi[n+1]
hi[n]

# MSB first serial processing

- **Redundant number system**
  - A Radix B RNS is allowed to possess digits from the set { -(B-1),..,-1,0,1,...(B-1)}
  - Let $X=x_{n-1}x_{n-2}...x_0$ be a n-digit radix beta number, then
  - $X=x_{n-1}B^{n-1} + x_{n-2}B^{n-2} + .. x_0B^0$
  - So, the number of values a digit is allowed to possess by the number system is (2B-1)
- **Low latency even for multiplication, thus good for feedback based systems.**
  - Not popular because the complexity of each arithmetic element is high.

- **B=4**
  +3, +2, +1, 0, -1, -2, -3
- **Representation of 9, multiple (redundant) forms**
  2,1 = 2*4+1 <- basic representation
  3,-3 = 3*4 -3
- **The carry propagation is limited to just one stage, so we can do arithmetic from the MSB**

**1,2,1 + 1,1,2 -> (0,2)x4²+ (1, -1)x4 + (1, -1) = 0,3,0,-1**

**0,2**

**1, -1 <- 0,3 is represented as 1,-1 to have a room for carry pro.**

**1, -1**

**----------**

**0,3,0,-1**

- **Why not keep propagating**
  - Because the number system has a room that prevents overflow even when there is a carry propagated from the low digit.

# Time-multiplexing and multi-rate

- **processes two or more different signals with the same operations using one hardware**
- **fsample_max = fclock_max / WL / MF,**
  where MF is the multiplexing factor
- **Example: stereo circuit**

# Bit-serial summary

- **advantages**
  - small chip area per throughput
  - Low complex control and interconnection

  <- high system clock rate, small arithmetic elements, hardwired control
- **disadvantages**
  - limited throughput (but faster than program controlled architectures)
  - High power consumption at shift registers
  - limited control capability

  -> not good for integrating general control functions

# 5. Distributed Arithmetic Architecture

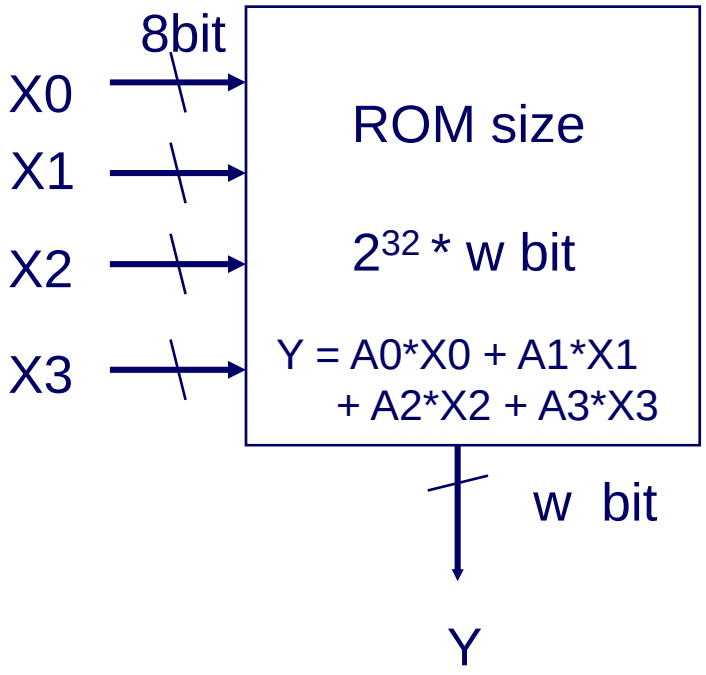
- **A special kind of bit-serial architecture**
- **ROM + accumulator based, instead of multiply + adder.**
- **Many of digital filtering algorithms are Sum of Product (multiply and accumulation) based - convolution, transformation, dot-product**
- **Distributed arithmetic computes the inner product in a bit-serial manner, using ROM and Accumulator based HW**
  - Bit-serial operation reduces the needed ROM size.
  - Still, it is needed to decompose the algorithm for high order digital filters
  - Not flexible, so not adequate for adaptive filters

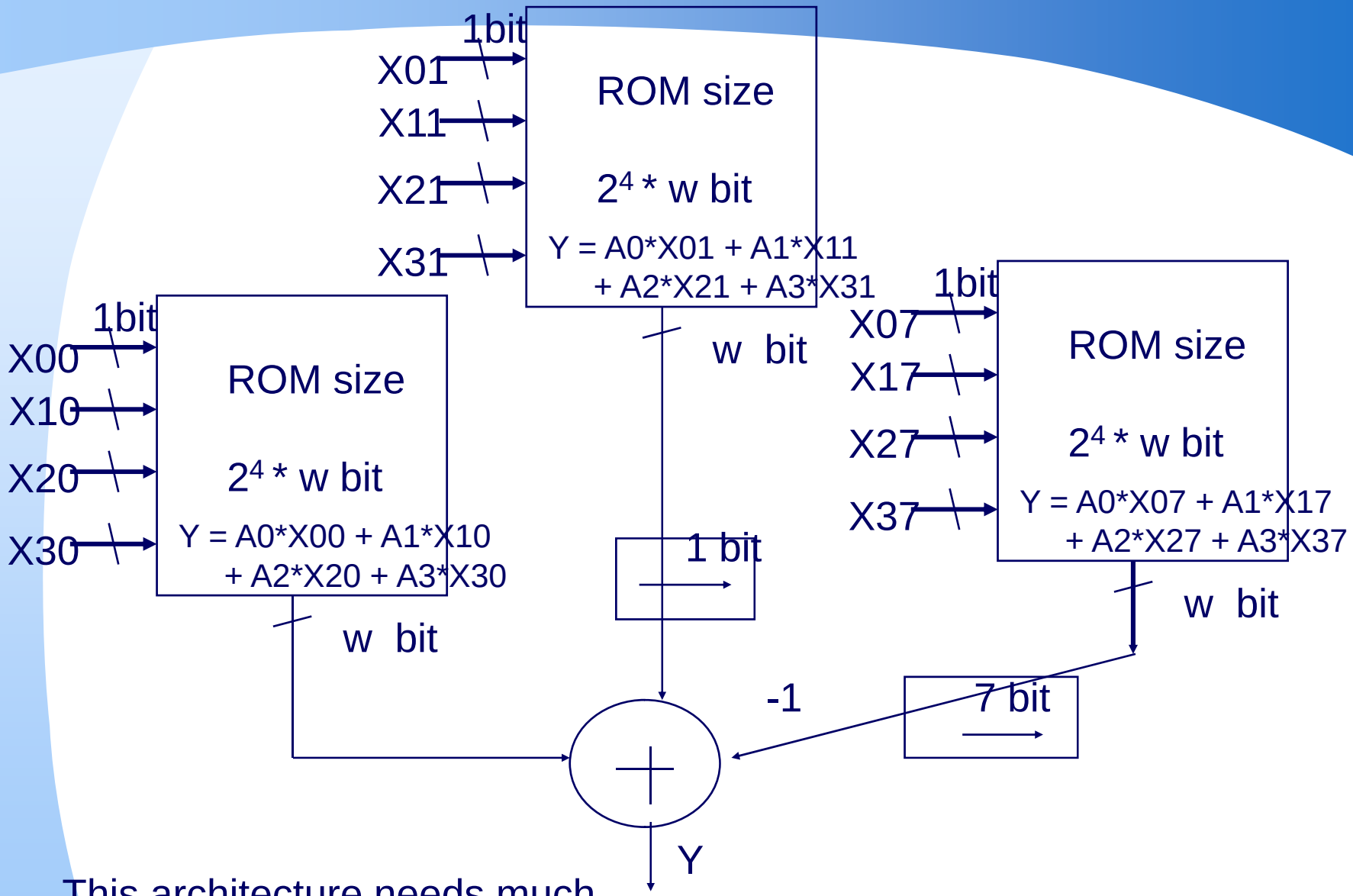
# Implementation of the sum of product

- $Y = A0*X0 + A1*X1 + A2*X2 + A3*X3$
- Xi = xi7 xi6 xi5 . . xi0
  (xi7 xi6.. is 1 or 0, assume 8 bit)
  $= (-1)*xi7*2^7 + xi6*2^6 + xi5*2^5 + .. + xi0*1$
- $Y = (-1)*2^7*(A0*x07 + A1*x17 + ..+A3*x37)$
  $+ 2^6*(A0*x06 + A1*x16 + ..+A3*x36)$
  . . .
  $+ (A0*x00 + A1*x10 + ..+A3*x30)$

# Why bit-serial?

A direct bit-parallel implementation with ROM






This architecture needs much smaller ROM (8 *16word ROM) size although it requires an 8 input adder

# Distributed arithmetic

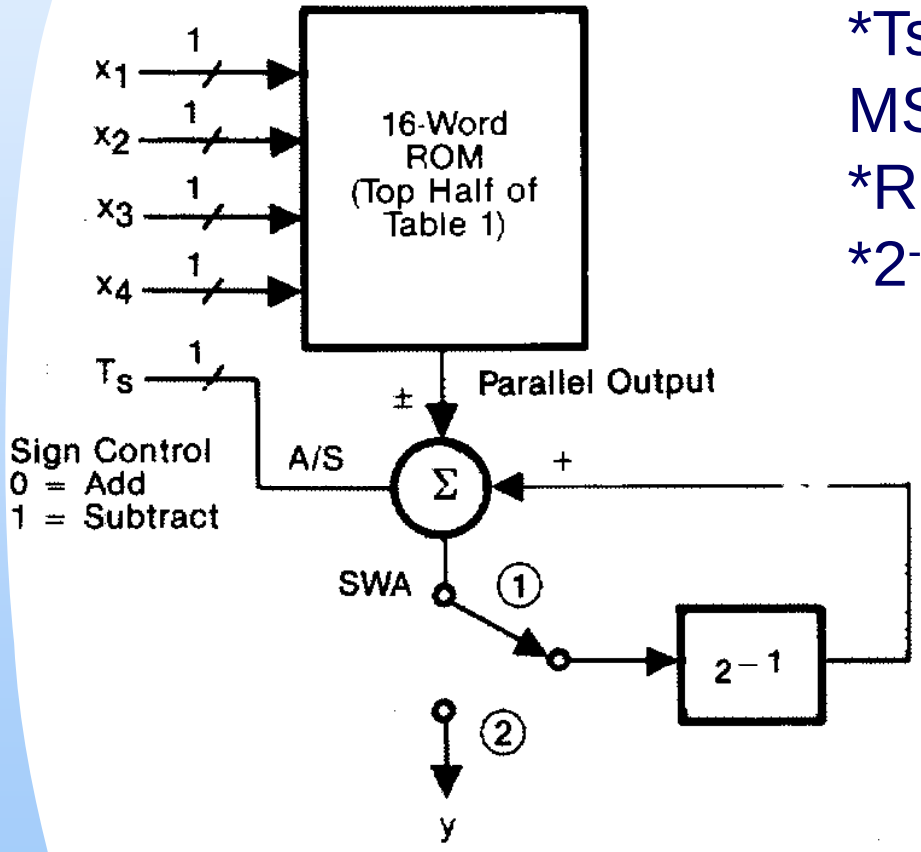



Figure 1b. Adder/Subtractor and Memory

*Ts controls add/sub: sub for MSB

*ROM size-> 16 * word length

*$2^{-1}$ is arithmetic right shift

# Distributed arithmetic (ROM Table)

| Address | b3 b2 b1 b0 | contents |
|---|---|---|
| 0 | 0 0 0 0 | 0 |
| 1 | 0 0 0 1 | A0 (=0.25) |
| 2 | 0 0 1 0 | A1 (=-0.1) |
| 3 | 0 0 1 1 | A1+A0 (=0.25-0.1) |
| | ... | |
| 15 | 1 1 1 1 | A3+A2+A1+A0 |

# Distributed arithmetic (Minimum ROM)

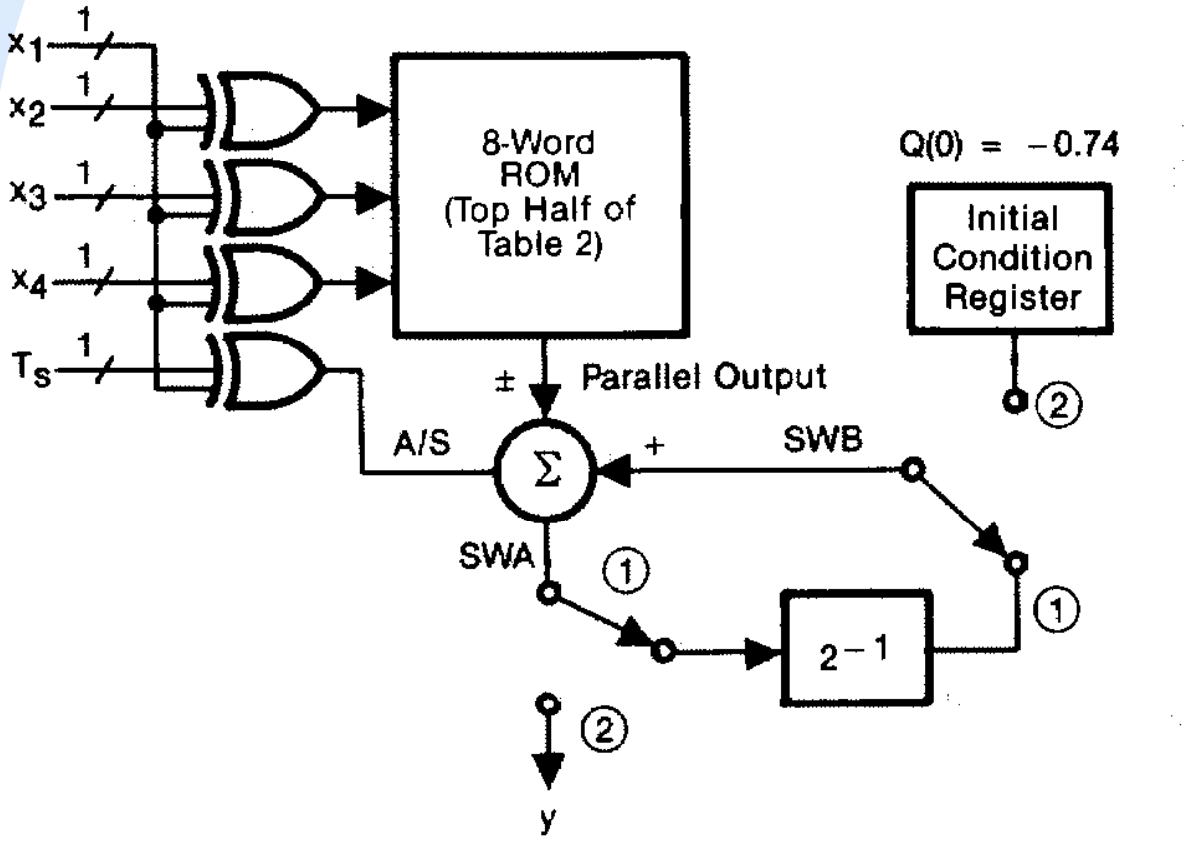



Figure 1c. Adder/Subtractor and Reduced Memory

# Speeding-up the distributed arithmetic based circuits

Apply 2-bit at a time for speed-up.
Needs two ROM-> 2 times speed

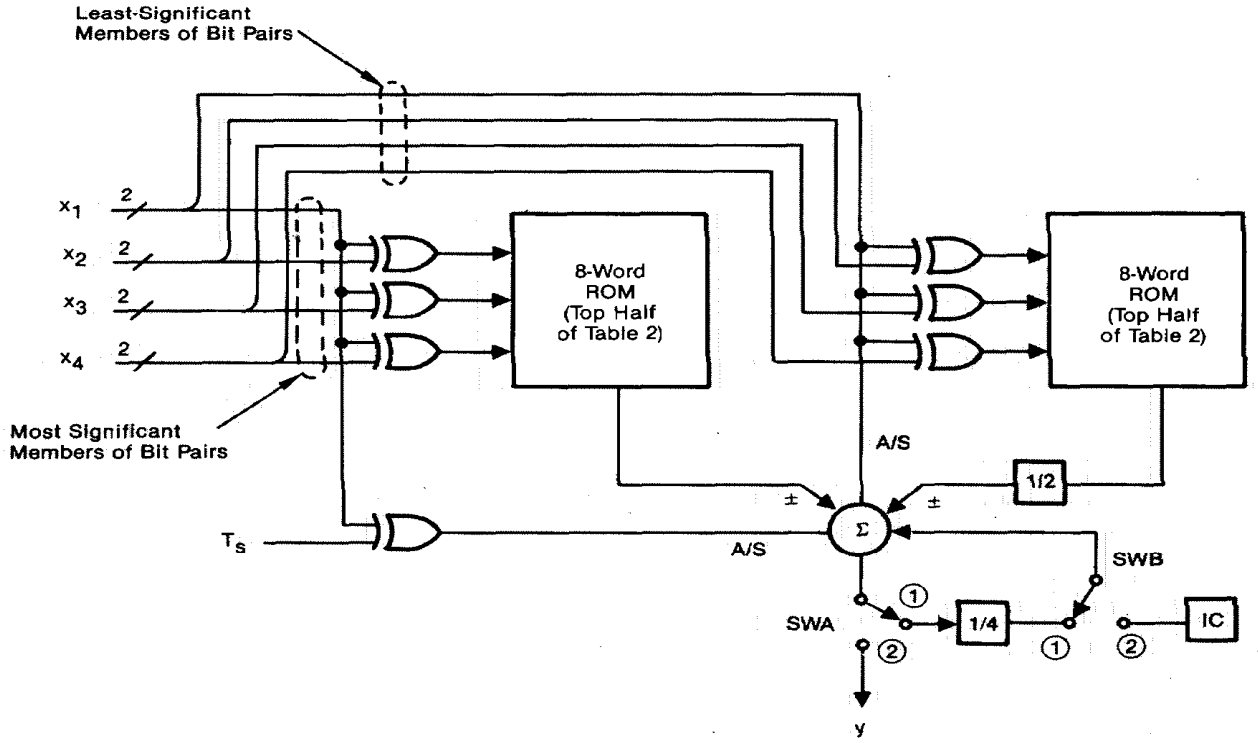



Figure 2. 2-Memory, 2 BAAT version of Figure 1C.

# Application of distributed arithmetic

- FIR filter
- DCT, IDCT -- matrix vector product
  -- no need of complex inter-connections found in efficient structure
- Distributed arithmetic is not good when the filter coefficients need to be changed. This (all bit serial arithmetic based one) is also not adequate for floating-point arithmetic.

# 6. SoC based Architecture

- HW based architecture is efficient in terms of throughput for a given silicon area and power consumption, but not flexible enough
- Today's multimedia and communication standards are very complex and need SW much.
- Today's consumers want something special for them – needs differentiation

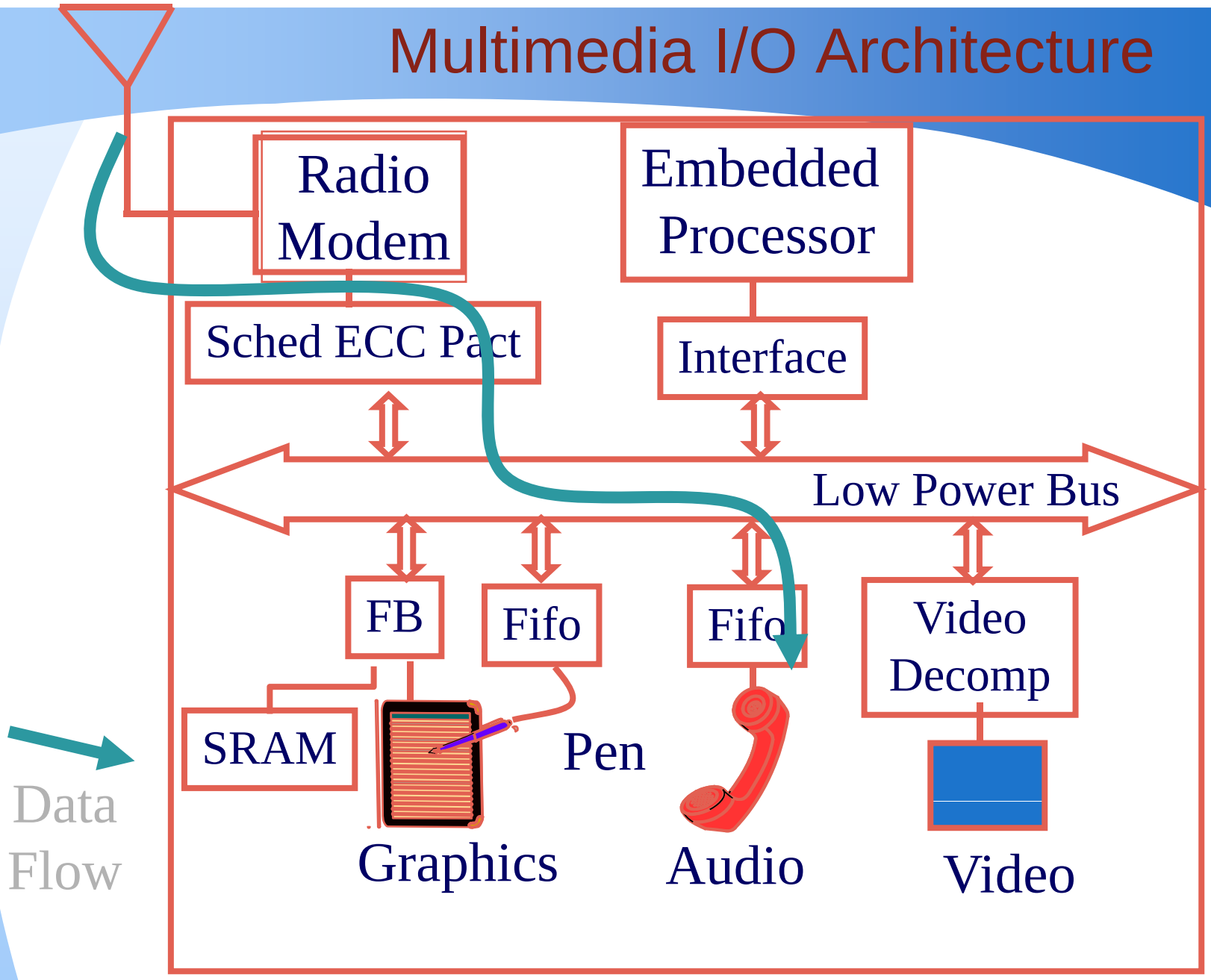
- -> Mix of CPU for programmability and HW blocks for high throughput

# Example

- TI developed C6x architecture for massive communication (not mobile) markets
- And, acquired the Amati Communications that developed ADSL technology

- But, TI's solution (C6x based ADSL) couldn't win the market. TI's solution was too expensive. The winning solution was based on CPU (such as ARM7) + HW modem blocks (FFT and ..).

# Multimedia I/O Architecture



# Concluding Remarks

- **Implementation of digital filtering algorithms requires adequate architectural choice because the system clock frequency is much different from the signal sampling clock frequency.**
  - Fully parallel
  - Bit parallel, time multiplexed
    - Distributed register based
    - Program ROM based
  - Bit serial
- **The flexibility of the architecture needs to be considered too. -> SoC architecture**