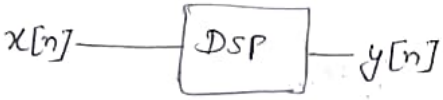
16-01-2025

# VLSI Signal Processing


Multiple Inputs Multiple outputs (MIMO)




Singale Input Single Output (SISO)

↳ Using:

① OBC

② DSP - Processors, GPGPU — CUDA

③ Custom Hardware - FPGA (semi-custom)

④ ASIC

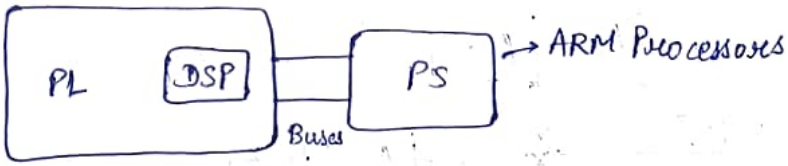
↳ Full custom

→ FPGA + dedicated DSP Unit

FPGA + Processor (P.S)

↳ PS + PL

FPGA + dedicated DSP Unit




↳ Software - Hardware Co-processing

→ Soft core

→ Hard core

- RISC V — for space application

Buses → AXI → used nowadays
↳ PLB

} AMBA (Advanced Microcontroller Bus Architecture)

# Major Functionalities of DSP

① Convolution

$$y[n] = \sum_{k=0}^{N-1} x[k]\,h[n-k]$$

→ DSP: non-terminating units
↳ I/p coming continuously.

↳ Reflection
↳ Dot product / Multiplier
↳ Summer

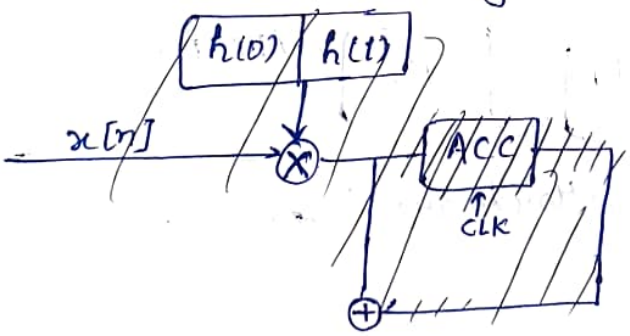
↳ Not applicable for non-linear system
e.g. Multi-rate: multiple frequencies / sampling rate
↳ Systems with multiple rates
↳ connected using upsampling / downsampling
↳ Linear time variant / invariant

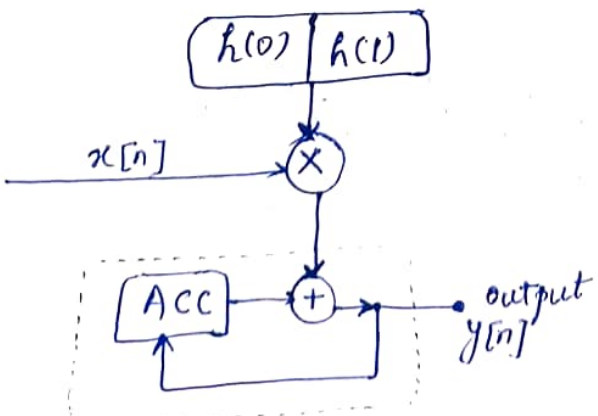
→ Convolution is valid for LTI system.

$n=0$, $y[0] = x[0]\,h[0]$ ⇝ one multiplier (zero addition)

$n=1$, $y[1] = x[0]\,h[1] + x[1]\,h[0]$ → 2 multipliers for parallel processing

↳ Addition of previous result
↳ Memory → Accumulator




Accumulation → Memory
Addition

: MAC
Multiply-Accumulate Unit

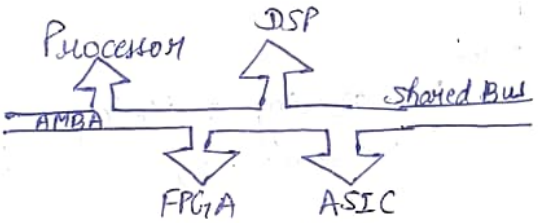
→ Sequential: only one adder & multiplier

## 2-D Convolution:

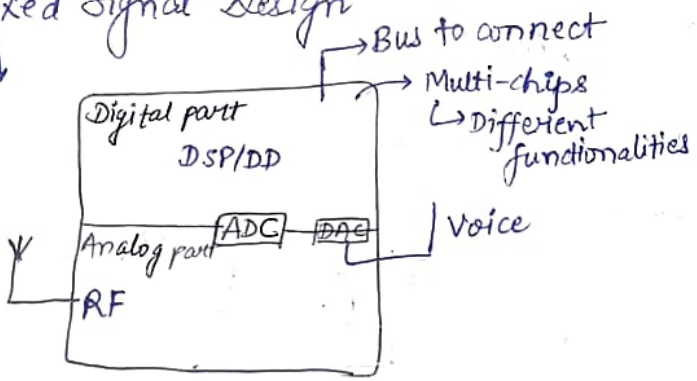
$$y[m,n] = \sum_{\substack{K=0 \\ l=0}}^{N-1} x[k,l]\, h[m-K, n-l]$$

↳ Linear independence property
↳ Apply in row, keeping column fixed.

17-01-2024




Mobile platform
↓
Mixed Signal Design
↓




Internal Buses
↳ Low throughput

SoC → System on chip

NoC → Network on chip
↳ Has SoC, more no. of processing elements
↳ Each connected with dedicated router
↳ Each can communicate faster
↳ High throughput
↳ More area

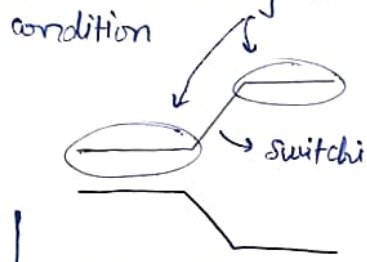
## Performance Metrics :

→ Power consumption ⇝ Provisions to reduce power through algorithm
→ Speed
→ Area

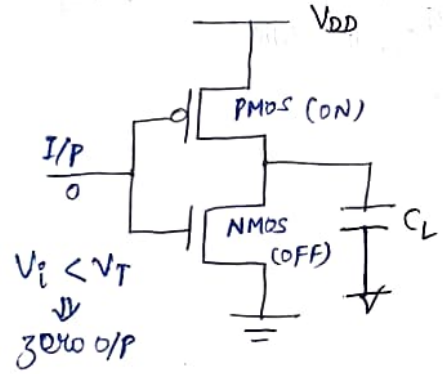
↳ To go to sleep mode
↳ Power Retention Logic
↳ Power ↓, Area ↑ (complexity ↑)

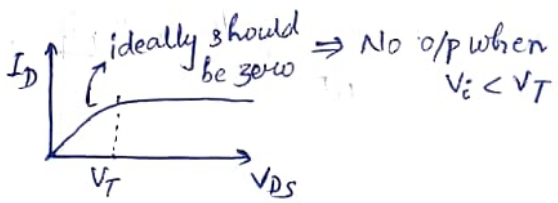
Power, $P = P_{SP} + P_{DP}$
(Static) (Dynamic)

Device in steady-state condition ↳ During switching
↳ cannot be zero, can be reduced.




↳ Zero (ideally), but leakage




$P_{DP} = C_L V_{DD}^2 f$, f: frequency of switching

or $C_L V_{DD} \cdot V \cdot f$ (V: max V)

$= C_L V_{DD} \cdot V_{DD} f$


Derivation: $i_L = -C \frac{dV}{dt}$

Worst case $i_L = -C \frac{V_{DD}}{t_p}$

$= C_L V_{DD} f$

$\therefore P_{DP} = V \times i_L = V_{DD} \times C_L \times V_{DD} \times f$

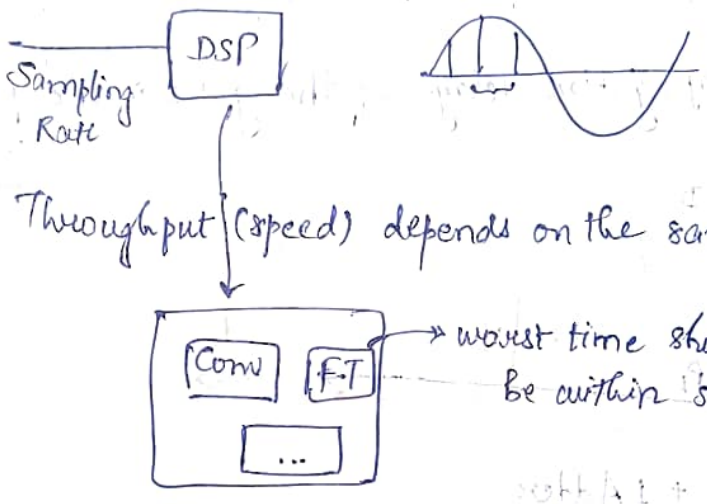
$P_{DP} \downarrow \Rightarrow f \downarrow$ [But we want high speed]

$\Rightarrow V_{DD} \downarrow$ (But only upto a level; risk of noise)

# Scaling
# Profiling

## DSP Representation

① Realtime application




② Data Driven

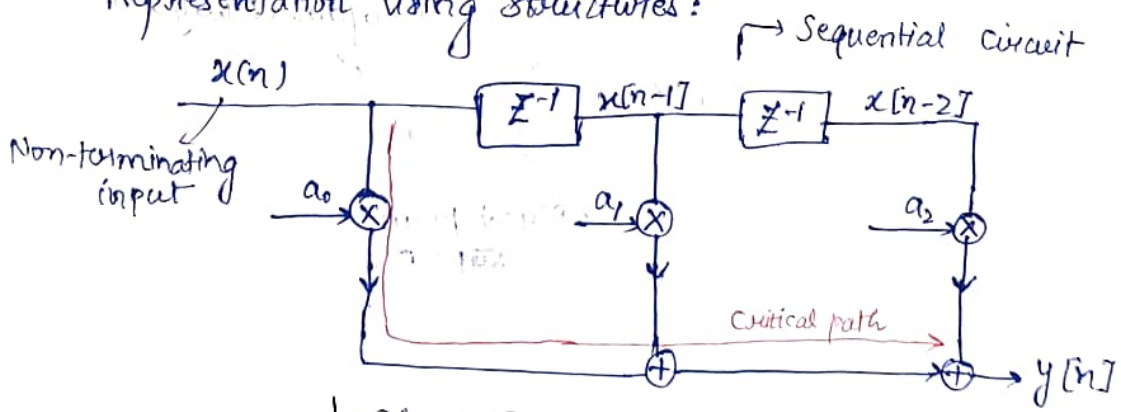
↳ Each nodes should be functional / excited only when all the data is received.

Eg. FIR filter:

$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2]$$

↳ 3-tap filter → $a_0, a_1, a_2$

Representation using structures:




↳ Direct form I structure

↳ Hardwares required: 2 adders
3 multipliers
2 delay blocks (D)

**Latency** → Delay to get the o/p after the application of i/p.

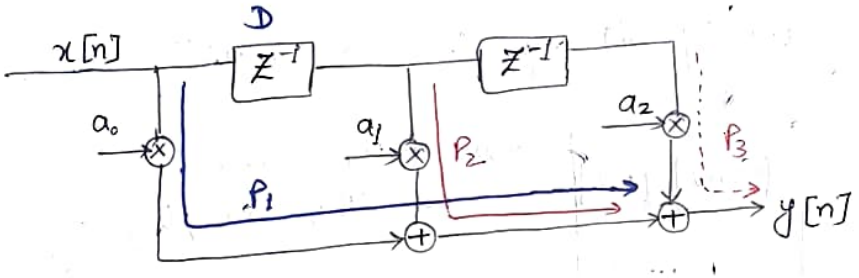
**Critical path** [in sequential circuit]: Path b/w 2 storage elements
OR, path (longest possible) b/w input & output
[in both Sequential as Combinational circuit]

↳ Consists of 2 adders + 1 multiplier in the eg.

↳ Reduce the critical path without reducing functionalities.

↳ Total propagation delays of the gates in the path

21-01-2025




$P_3$ = 1M + 1 Adder

$P_1$ = 1M + 2 Adder ✓

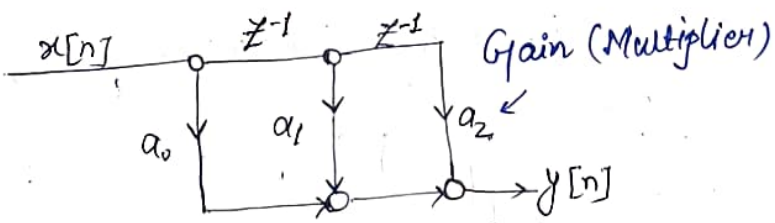
$P_2$ = 1M + 2 Adder ✓

} worst possible

→ Block Diagram Representation

**Signal Flow Graph**

↳ vertex, node




Direct form SFG

# Transposition Structure

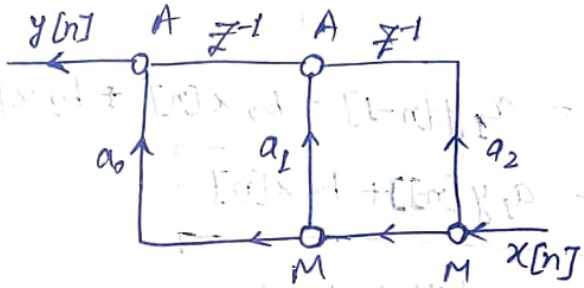
(Direct Form II)

Input → Output

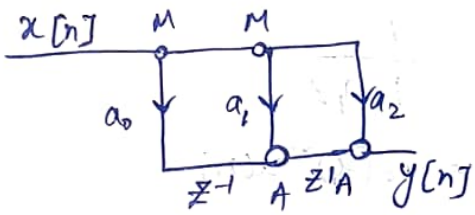
Incoming → Outgoing

Accumulation

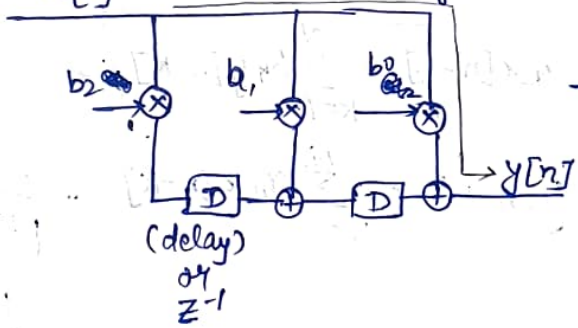
→ ⊕ ←




↓ Flip / Fold




⇓ Block diagram




→ Data Broadcast Structure

$y[n] = a_0 Z$

# Data Flow Graph (DFG)

↳ DSP runs on Data Driven property

○ Node/Actor

Multiplier

Execution / fire

IIR Filter: $y[n] = \sum_{K=1}^{M} a_K y[n-K] + \sum_{K=0}^{N-1} b_K x[n-K]$ $\left(\begin{matrix} M=N \\ \text{or} \\ M<N \end{matrix}\right)$

1st order IIR filter: $y[n] = a_1 y[n-1] + b_0 x[n] + b_1 x[n-1]$

$$\boxed{y[n] = a_1 y[n-1] + b_0 x[n]}$$

$\Rightarrow Y(z) = a_1 z^{-1} Y(z) + b_0 X(z)$

$\Rightarrow H(z) = a_1 z^{-1} H(z) + b_0$

$\Rightarrow H(z) = \dfrac{b_0}{1 - a_1 z^{-1}}$

$\Rightarrow h(n) = b_0 (a_1)^n u(n)$.

23-01-2024

IIR Filter:

$$\boxed{y[n] = \sum_{K=0}^{M} a_K x[n-K] - \sum_{K=1}^{N} b_K y[n-K]}$$

$$\sum_{K=0}^{N} b_n y[n-K] = \sum_{K=0}^{M} a_n x[n-K]$$ ... LCCDE (Linear Constant Coefficient Difference Equations)

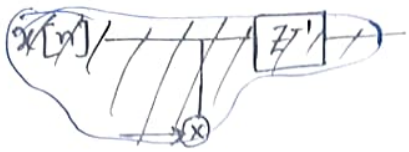
- → M=N
- → M>N
- → M<N

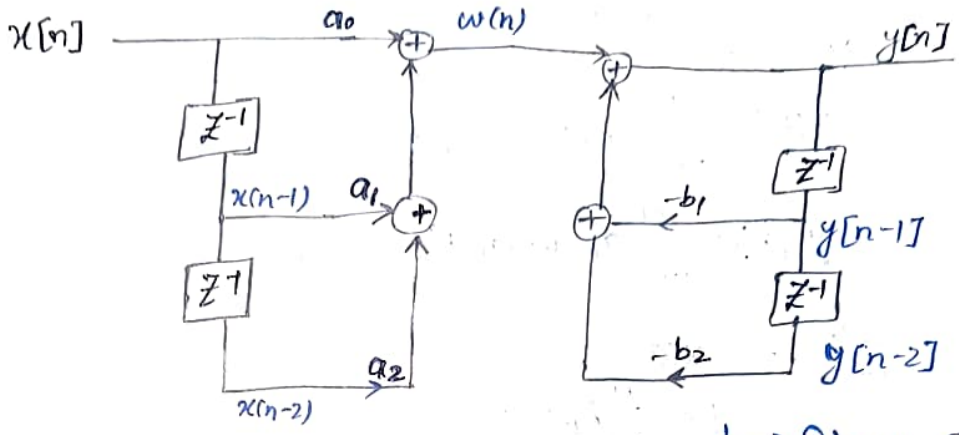
Simple form: $y[n] = b_1 y[n-1] + a_1 x[n]$

For M=2, N=2,

$$y[n] = (a_0 x[n] + a_1 x[n-1] + a_2 x[n-2]) - b_1 y[n-1] - b_2 y[n-2]$$

$\Rightarrow \boxed{b_0 = 1}$




↳ Direct Form I
↳ Non-canonical

Canonical form: No. of delay elements is equal to the order of the system.

↳ Hardware required:
- 4 adders
- 4 delay elements (DFFs)
- 5 multipliers

↳ Complex structure
↳ Reduce complexity: combine delay elements for w(n) and y(n).

$$H(z) = \frac{Y(z)}{X(z)} = \frac{Y(z)}{W(z)} \cdot \frac{W(z)}{X(z)}$$

$$= H_1(z) \cdot H_2(z)$$

$$H_1(z) = \frac{Y(z)}{W(z)}$$

$$\Rightarrow Y(z) = W(z) \cdot H_1(z)$$

$$H_2(z) = \frac{W(z)}{X(z)}$$

$$\Rightarrow W(z) = H_2(z)\, X(z)$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} a_k z^{-k}}{1 + \sum_{k=1}^{N} b_k z^{-k}}$$

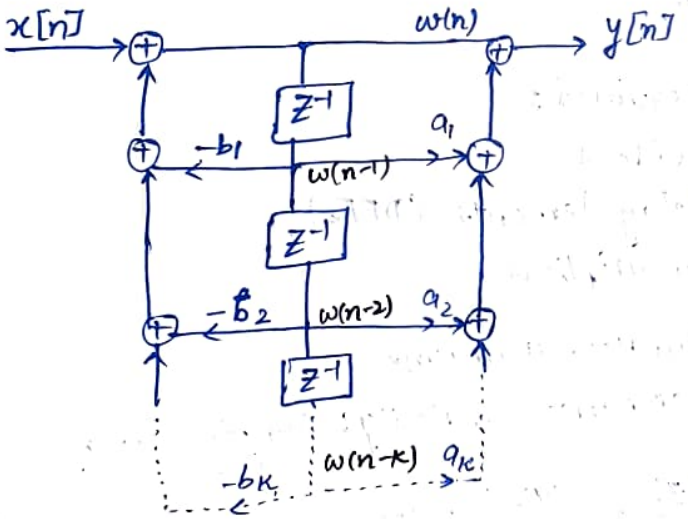
$$\frac{Y(z)}{W(z)} = \sum_{k=0}^{M} a_k \cdot z^{-k}$$

$$\Rightarrow Y(z) = \sum_{k=0}^{M} a_k \cdot z^{-k} \cdot W(z)$$

$$y[n] = \sum_{k=0}^{M} a_k \, w[n-k] \quad \ldots (2)$$

$$\frac{W(z)}{} = \frac{X(z)}{1 + \sum_{k=1}^{N} b_k z^{-k}}$$

$$\Rightarrow w[n] = x[n] - \sum_{k=1}^{N} b_k \cdot w(n-k) \quad \ldots (1)$$



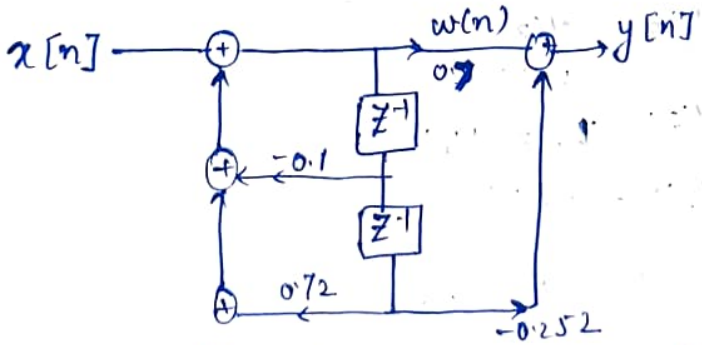

(Half of form I)

→ Less delay elements

→ Canonical

→ Direct form II

Eg. $y[n] = (-0.1\, y[n-1] + 0.72\, y[n-2]) + (0.7\, x[n] - 0.252\, x[n-2])$



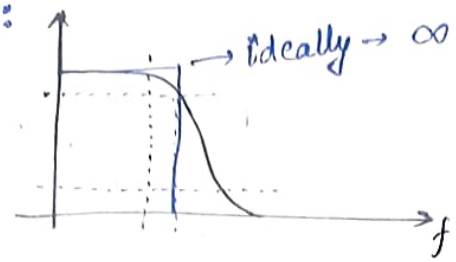

→ 2 delays

→ Filter

$$w(n) = x(n) - 0.1\, w(n-1) + 0.72\, w(n)$$

$$y(n) = 0.7\, w(n) - 0.252\, w(n-2).$$

Filter:

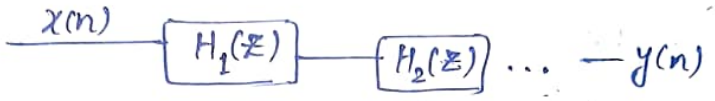



Cascade form structure:

$$H(z) = H_1(z) \cdot H_2(z) \cdots H_N(z)$$

→ To reduce quantization error

28-01-2025




$$y[n] = \frac{3}{4}y[n-1] - \frac{1}{3}y[n-2] + x[n] + \frac{1}{3}x[n-1]$$

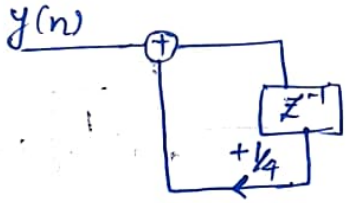
$$H(z) = \frac{1 + \frac{1}{3}z^{-1}}{1 - \frac{3}{4}z^{-1} + \frac{1}{3}z^{-2}} = \frac{1 + \frac{1}{3}z^{-1}}{\left(1 - \frac{1}{2}z^{-1}\right)\left(1 - \frac{1}{4}z^{-1}\right)}$$

$$= H_1(z) \cdot H_2(z)$$

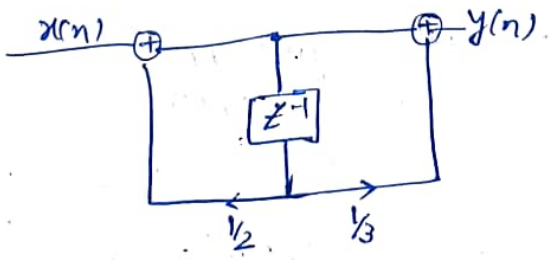
$$H_1(z) = \frac{\left(1 + \frac{1}{3}z^{-1}\right)}{\left(1 - \frac{1}{2}z^{-1}\right)}$$

$$H_2(z) = \frac{1}{\left(1 - \frac{1}{4}z^{-1}\right)}$$

$H_2$:




$H_1$:



## Parallel form:

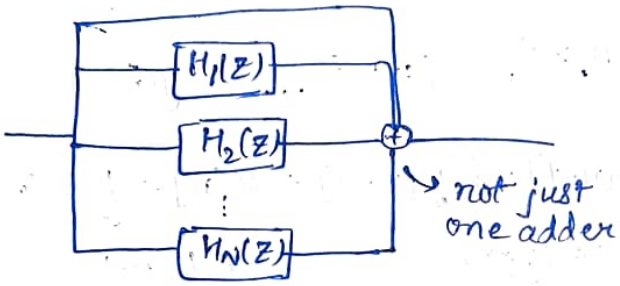
$$H(z) = C + \sum_{K=1}^{N} \frac{C_K}{1 - P_K z^{-1}}$$

(Partial Expansion)

→ for IIR as well as FIR

$$H(z) = C + \frac{C_1}{1 - P_1 z^{-1}} + \frac{C_2}{1 - P_2 z^{-1}} + \dots + \frac{C_N}{1 - P_N z^{-1}}$$

$$H(z) = C + H_1(z) + H_2(z) + \dots + H_N(z)$$




## Linear Phase System

Type-I: $h(n) = h(\cancel{n} N-n)$ [symmetric]

$h(N-n)$ (for causal), N: Even or odd

$$H(z) = \sum_{K=0}^{N-1} h(K) z^{-K}$$

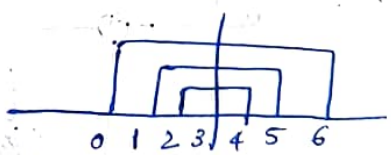
↳ Delay elements can be shared (only 4 required)

$h[0] = h[+6]$

$h[1] = h[+5]$

$h[3] = h[3]$

$h[2] = h[4]$




# SQNR = $\frac{\text{Input Signal (RMS)}}{\text{Noise (RMS)}}$

(Signal to Quantization Noise Ratio)

$= 6.02 + 1.76$ → For every one bit improvement in the signal, SQNR improves by 6 dB.

Types:
- → Analog to digital (Input) Q. error
- → Multiply & Accumulation (Product) error

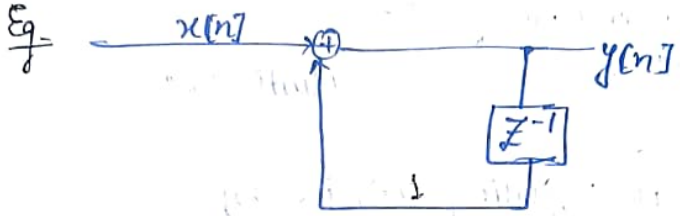
→ Truncation introduces more error.

→ Rounding .. less error.

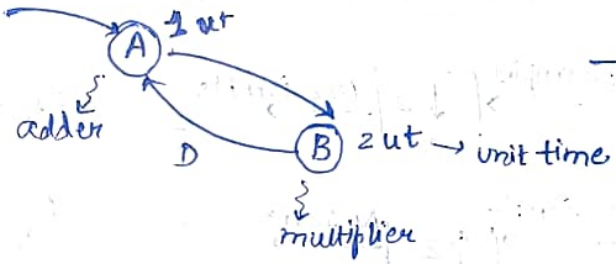
# Data Flow Graph

Nodes → Represent computations (multiply & adder)
Directed edges → Represent data paths
Edge → Delay.

Eg-




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




→ Fine DFG
↳ only adder and multiplier computations

Eg

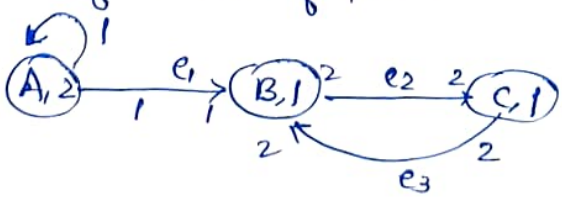

→ Course-Grain DFG

30-01-2025

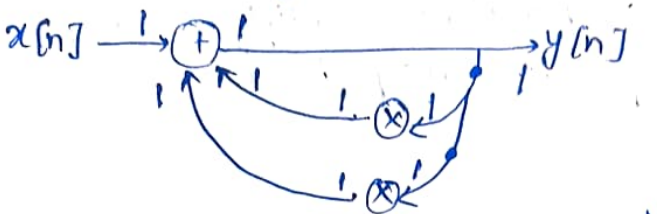
→ Exposes hidden concurrency among different parts.

## Synchronous Dataflow Graph :

↳ No. of data samples produced or consumed by each node in each execution is specified.
↳ Each tasks operate on pre-defined no. of samples and produces a fixed no. of o/ps.

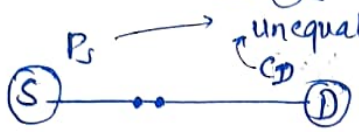


Eg. $y[n] = x[n] + a_1 y[n-1] + a_2 y[n-2]$




→ Single rate weight DFG

Eg.




→ Multi-rate

$\boxed{f_S P_S = f_D C_D}$ : firing relationship

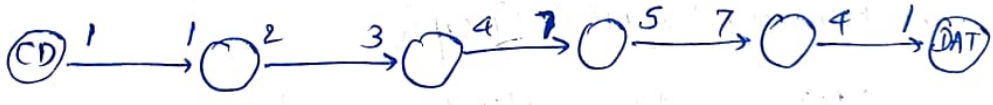
## Single-rate and Multi-rate SDFGs:

**Decimator:** N samples → [↓2] → N/2 samples ≡ $\xrightarrow{2}\bigcirc\xrightarrow{1}$

**Expander:** N/2 samples → [↑2] → N samples ≡ $\xrightarrow{1}\bigcirc\xrightarrow{2}$

Eg Format converter to store a CD-quality audio sampled at 44.1 kHz to a DAT that records a signal at 48 kHz.

$$\frac{f_{out}}{f_{CD}} = \frac{480}{441} = \frac{160}{147} = \frac{2}{1} \times \frac{4}{3} \times \frac{5}{7} \times \frac{4}{7}$$




**Homogeneous SDFG:**

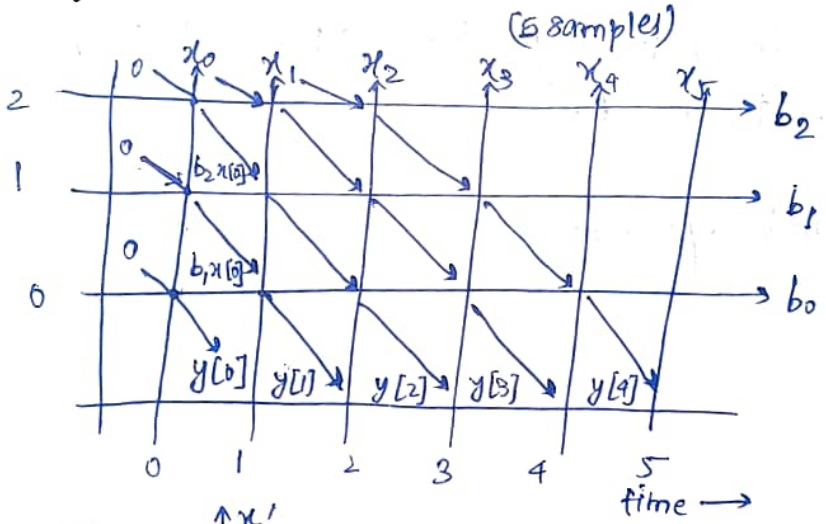
↳ Single-rate graph where each node produces one token

## Dependency Graph:

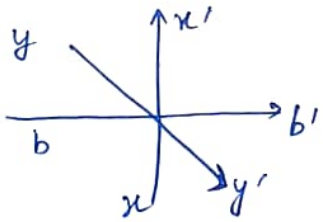
↳ Directed graph which shows the dependence of the computation in an algorithm.

nodes → computation
edges → procedure constraints among nodes

Eg: $y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2]$




$y[0] = b_0 x[0] + b_1 x[-1] + b_2 x[-2]$
$y[1] = b_0 x[1] + b_1 x[0] + b_2 x[-1]$
$y[2] = b_0 x[2] + b_1 x[1] + b_2 x[0]$
⋮




$b = b'$
$x = x'$
$y' = y + bx'$

# Implement 4×4 Matrix multiplication using ~~Cyclic~~ systolic table in Verilog.

## Transformations on a Dataflow Graph:

↳ To minimize critical path delay or no. of registers.

- Retiming
- Folding
- Unfolding
- Pipelining
- Parallel processing

## Performance Measures for Design Tradeoffs

- **Iteration period**: Time taken by system to compute all the operations in one iteration of an algorithm.
  - Measured in terms of the no. of clock cycles required to compute one o/p sample.
- **Sampling period**: Average time b/w two successive data samples.
- **Latency**: Time delay for algorithm to produce output in response to an input.
- Power
  - static and dynamic.

04-02-2025

# Number Representation

- Fixed point
- Floating point

## Fixed Point Representation

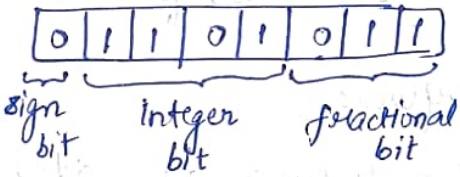
- Computer assumes any bits/no. as integer.
- An integer X is used to represent a real no. x, with

$N = m + n + 1$ bits,

N: wordlength

m: no. of integer bits (excluding sign bit)

n: no. of fractional bits.




**Q-format:** X sometimes called $Q_{m.n}$ or $Q_m$ number.

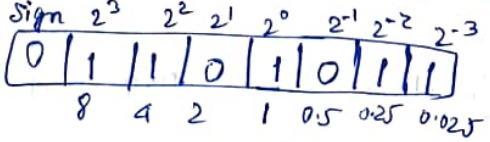
**Conversion to and from fixed point:**

Real to fixed-point: $X := \text{Round}(x \cdot 2^n)$

fixed-point to Real: $x := X \cdot 2^{-n}$

Eg $x = 13.4$ → Represent using $Q_{4.3}$ format.

$X = \text{Round}(13.4 \times 2^n) = 107 (= 01101011_2)$




$$x = \frac{107}{8} = 13.375 \neq 13.4$$

→ Dynamic Range = $\frac{\text{Max value}}{\text{Min value}} = \frac{2^{N-1}-1}{1}$ (Signed integer format)

## Range vs. Resolution for fixed-point no.s :

A Qm.n fixed point no. can represent no. in the range

$$[-2^m,\ 2^m - 2^{-n}].$$

Fractional part resolution $= 2^{-(N-1)}$

Eg. To represent $x$ in signed 8-bit variable.

$$-28.3 < x < 17.5$$

$m = 5$ to represent the integer part:

$(2^4 = 16 < 28.3 < 32 = 2^5)$

$n = 8 - 1 - m = 2$ bits left for the fractional part.

$x$ should be stored in Q5.2 format

Eg. Addition with overflow:

Two no.s in Q4.3 format are added:

$x = 12.25 \Rightarrow X = 98$ 0 1 1 0 0 0 1 0

$y = 14.75 \Rightarrow Y = 118$ + 0 1 1 1 0 1 1 0

$Z = X + Y = 216$ 1 1 0 1 1 0 0 0

This no. is out of range and will be represented as

$$216 - 256 = -40 \Rightarrow Z = -5.0$$

$(2^8)$ $\div 2^3$

## Fixed-Point Multiplication and Division

06-02-2025

If operands and results are in same Q-format:

$$z = x.y \iff Z = (X.Y)/2^n$$

$$z = x/y \iff Z = (X.2^n)/Y$$

→ Double wordlength is needed for the intermediate result.

→ Division (multip.) by $2^n$ is ~~mul~~implemented by a right (left)-shift by n-bits.

→ Lowest bits in the result are truncated (round-off noise).

→ Risk of overflow.

Eg. Two nos in Q 5.2 format:

$x = 6.25 \Rightarrow X = 25$

$y = 4.75 \Rightarrow Y = 19$

$X \cdot Y = 475$

$\Rightarrow Z = 475/2^2 = 118 \Rightarrow z = 29.5$

In Q5.3 format:

$x = 6.25 \Rightarrow X = 50$

$y = 4.75 \Rightarrow Y = 38$

$\Rightarrow Z = \frac{1900}{2^3} = 237.5 \Rightarrow z = 29.625$

With different Q-format:

In general, multiplication of 2 fixed-point no.s $Qm_1.n_1$ and $Qm_2.n_2$ gives an intermediate result in the format:

$$Qm_1+m_2.n_1+n_2$$

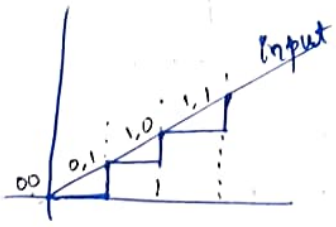
which may be then right-shifted $n_1+n_2-n_3$ steps and stored in the format

$$Qm_3.n_3$$

Quantization error in representing $H(z)$:

- Input Q.E.
- Product Q.E.
- Output Q.E.

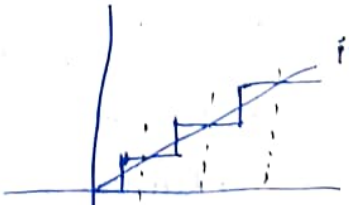
Truncation:




↳ worst-case error = $2^{-(B-1)}$

= 2(Rounding error)

Rounding:

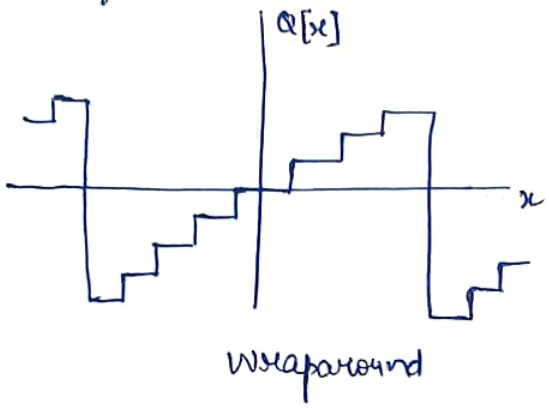



# SIMULINK → HDL TOOLBOX

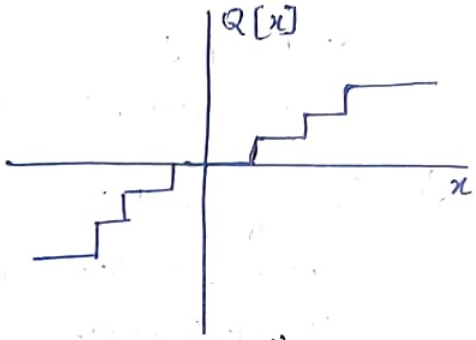
# MATLAB → fixed-point toolbox

## Overflow

- Wraparound
- Saturation




Wraparound




Saturation

## Schwartz's inequality:

$$|y[n]| \leq \sqrt{\sum_{n=-\infty}^{\infty} h^2[n] \sum_{n=-\infty}^{\infty} x^2[n]}$$

→ To find upper bound on the o/p values of a LTI system.

## Fixed Point Quantization

↳ The fractional B-bit 2's complement no. representation evenly distributes $2^B$ quantization levels b/w $-1$ and $1-2^{-(B-1)}$.

↳ Spacing b/w quantization levels : $\frac{2}{2^B} = 2^{-(B-1)} = \Delta_B$

↳ Any signal value falling b/w two levels is assigned to one of the two levels.

$e = Q[x] - x$ : quantization error.

quantization notation : $x_Q$.

## Floating Point Number

↳ $F \times r^E$ , F : fraction
E : exponent
r : radix (= 10 for decimal nos.)

In 32-bit single precision floating-point representations :

↳ Most significant bit : sign bit (S) → 0 for ⊕ve, 1 for ⊖ve

↳ Following 8 bits : biased exponent (E)

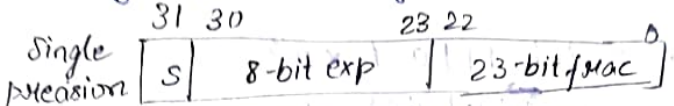
↳ Remaining 23 bits : fraction (F).

11-02-2025

# Single precision Floating Point No.

↳ Easy (and lazy) way of dealing with scaling problem.

↳ 32-bit single precision floating point.




$$x = -1^S \times 2^{exp-127} \times 1.frac$$
$$1.175 \times 10^{-38} < |x| < 1.7 \times 10^{38}$$

↳ MSB is sign bit (same as fixed point).

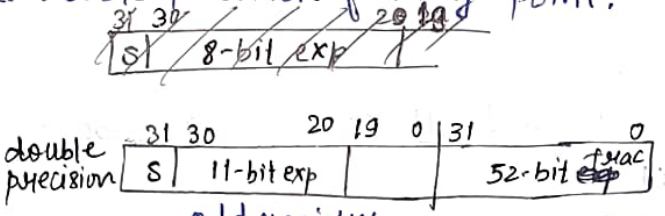
↳ 8-bit exponent in bias −127 integer format (i.e., add 127 to it).

↳ 23-bit to represent only the fractional part of the mantissa.

The MSB of the mantiss is ALWAYS '1', therefore it is not stored.

## Double Precision Floating Point No.

↳ 64-bit double precision floating point:




$$x = -1^S \times 2^{exp-1023} \times 1.frac$$
$$2.2 \times 10^{-308} < |x| < 1.7 \times 10^{308}$$

↳ MSB is sign-bit (same as fixed point).

↳ 11-bit exponent in bias −1023 integer format (i.e., add 1023 to it).

↳ 52-bit to represent only the fractional part of the mantissa.

The MSB of the mantissa is ALWAYS '1', therefore it is not stored.

## Short and Long IEEE 754 Formats: Feature

Coded no. $x = (-1)^S \cdot 2^e \cdot f$

| Feature | Single/Short | Double/Long |
|---|---|---|
| Word width in bits | 32 | 64 |
| Significand in bits | 23 + 1 hidden | 52 + 1 hidden |
| Significand range | $[1, 2-2^{-23}]$ | $[1, 2-2^{-52}]$ |
| Exponent bits | 8 | 11 |
| Exponent bias | 127 | 1023 |
| Zero (±0) | e + bias = 0, f = 0 | e + bias = 0, f = 0 |
| Subnormal | e + bias = 0, f ≠ 0; represents $\pm 0.f \times 2^{-126}$ | e + bias = 0, f ≠ 0; represents $\pm 0.f \times 2^{-1022}$ |
| Infinity (±∞) | e + bias = 255, f = 0 | e + bias = 2047, f = 0 |
| Not-a-no. (NAN) | e + bias = 255, f ≠ 0 | e + bias = 2047, f ≠ 0 |
| min | $2^{-126} \approx 1.2 \times 10^{-38}$ | $2^{-1022} \approx 2.2 \times 10^{-308}$ |
| max | $\approx 2^{128} \approx 3.4 \times 10^{38}$ | $\approx 2^{1024} \approx 1.8 \times 10^{308}$ |

Eg. Represent -13.8 using IEE 754 32-bit single precision floating point representation.

$(13.8) \rightarrow (1101.11001)$

$1101.11001 = 1.101111001 \times 2^3$

(shifted right)

Actual exponent = 3

Biased exponent = 3 + 127 = 130 = (10000010)

sign of fraction / Mantissa (s) = -ve = 1

| S | Biased exponent | Matissa/fraction |
|---|---|---|
| 1 | 10000010 | 10111001000000000000000 |

Eg. 32-bit pattern:

1 1000 0001 011 0000 0000 0000 0000 0000

S = 1 (-ve)

Biased exponent = 1000 0001 = 129

Actual exponent = 129 - 127 = 2

$1.011\ 0000\ 0000\ 0000\ 0000\ 0000 = 1 + 1\times2^{-2} + 1\times2^{-3}$

$= 1.375$

$\therefore No. = -1.375 \times 2^2 = -5.5$

| | Single/Short | Double/long |
|---|---|---|
| Ordinary no. | e+bias $\in [1,254]$ | e+bias $\in [1,2046]$ |
| | $e \in [-126, 127]$ | $e \in [-1022, 1023]$ |
| | Represents $1.f \times 2^e$ | Represents $1.f \times 2^e$ |

Examples:

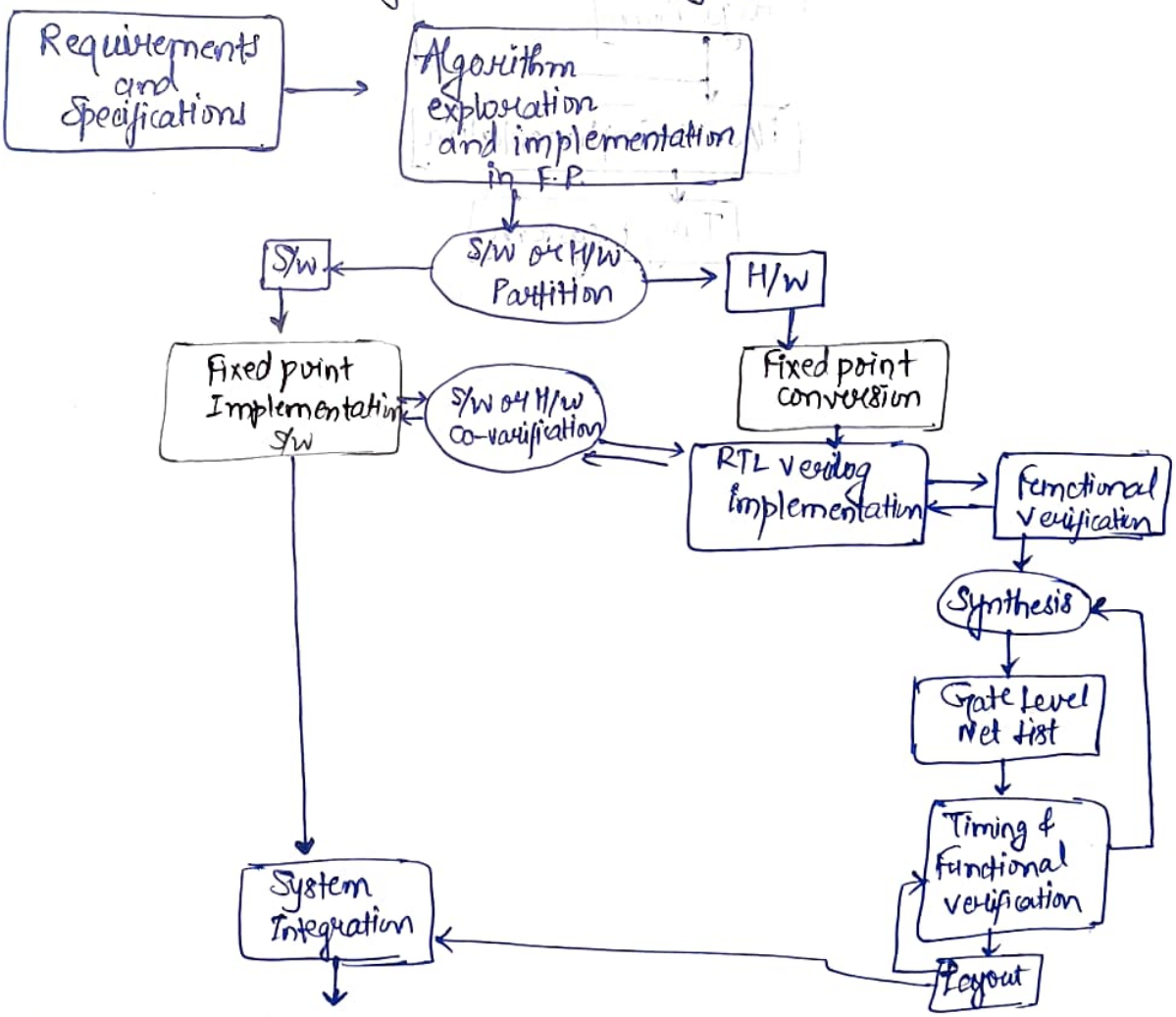
| | | | |
|---|---|---|---|
| 0 | 0 | 0...(23)...0 | = 0 |
| 1 | 0 | 0...(23)...0 | = -0 |
| 0 | 1...(8)...1 | 0...(23)...0 | = infinity |
| 1 | 1...(8)...1 | 0...(23)...0 | = -infinity |
| 0 | 1...(8)...1 | 0 0000 10...(17)...0 | = NaN |
| 1 | 1...(8)...1 | 0010b01000100101010101010 | = NaN |

→ Floating-point DSPs represent no.s with a mantissa and an exponent, nowadays following the IEEE 754.

→ The mantissa dictates the no. of precision and the exponent controls its dynamic range.

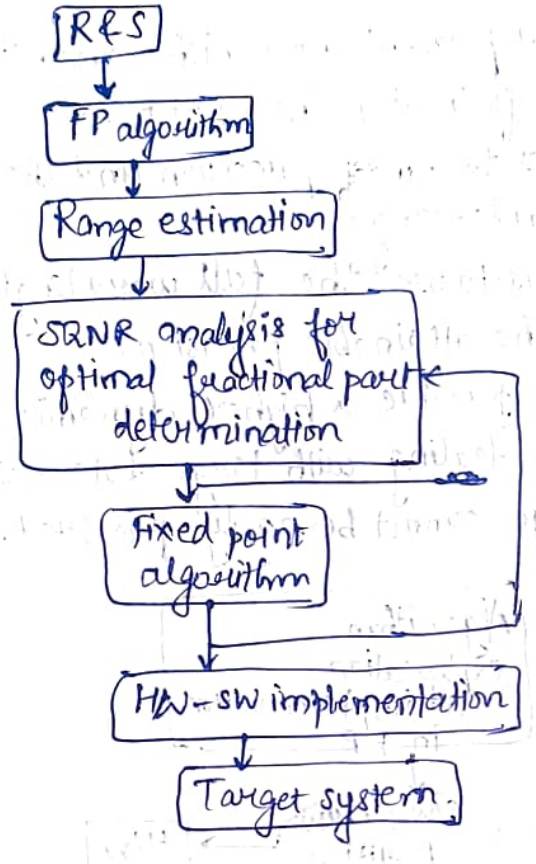
→ No.s are scaled so as to use the full word length available, hence maximizing the attainable precision.

→ Floating-point no.s provide a higher dynamic range, which can be essential when dealing with large data sets and with data sets whose range cannot be easily predicted.




## FP to Fixed Point Conversion

- Start with the algorithm (FP).
- Estimate the range.
- Determine n.
- The fractional part m requires detailed analysis of signal quantization noise.

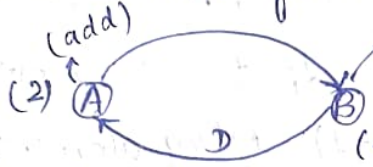

R&S
FP algorithm
Range estimation
SQNR analysis for optimal fractional part determination
Fixed point algorithm
HW-SW implementation
Target system

13-02-2025

# Loop Bound and Iteration Bound

**Iteration bound** (of DSP program) is the lower bound on the iteration of DSP program regardless of the amount of computing resources available ⇒ fundamental limit on the minimum sampling period.

**Loop bound**: Time for a loop to complete one iteration.




$y(n) = ay(n-1) + x(n)$

**Loop**: same begin & end node

**Iteration of node** ↳ execution of node exactly once

**Iteration of DFG** ↳ execution of each node exactly once

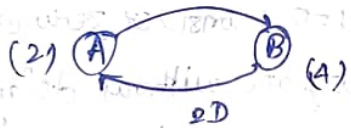
Precedence constraint:

inter-iteration : delay :— $B_n \Rightarrow A_{n+1}$

intra-iteration : no-delay : $A_n \rightarrow B_n$

iteration : $A_n \rightarrow B_n \Rightarrow A_{n+1} \rightarrow B_{n+1} \Rightarrow A_{n+2}$

Loop bound of $i^{th}$ loop: $\boxed{\frac{t_i}{w_i}}$, $t_i$ : loop computation time, $w_i$ : no. of delay in the loop.

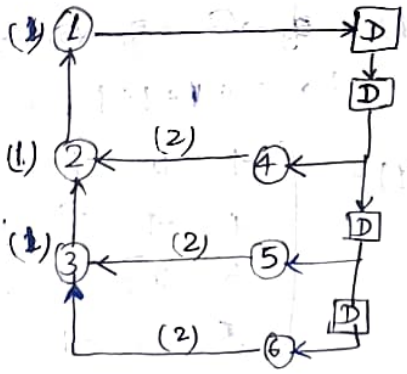



Loop bound $= \frac{6}{2} = 3$.

↳ 1 iteration of loop completed in 3 u.t.

[Iteration bound = 3 u.t.]

Eg



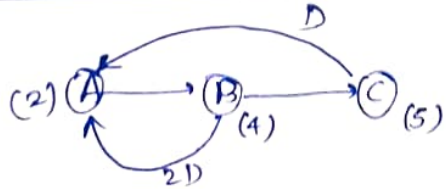

$1 \rightarrow 4 \rightarrow 2 \rightarrow 1$ : l.p. $= \frac{4}{2}$ u.t.

$1 \rightarrow 5 \rightarrow 3 \rightarrow 2 \rightarrow 1$ : l.p. $= \frac{5}{3}$ u.t.

$1 \rightarrow 6 \rightarrow 3 \rightarrow 2 \rightarrow 1$ : l.p. $= \frac{5}{4}$ u.t.

**Critical loop**: Loops in which has maxm loop bound

**Iteration bound** : Maxm loop bound : $\boxed{T_\infty = \max\left\{\frac{t_i}{w_i}\right\}}$




l.b. = Max(A → B → C → A)

# Longest path Matrix Algorithm "Iteration Bound" (LPM Method)

$L^{(m)}$, $m = 1, 2, \ldots, d$, $d$: no. of delays in the DFG
$m$: delays
↳ series of matrix

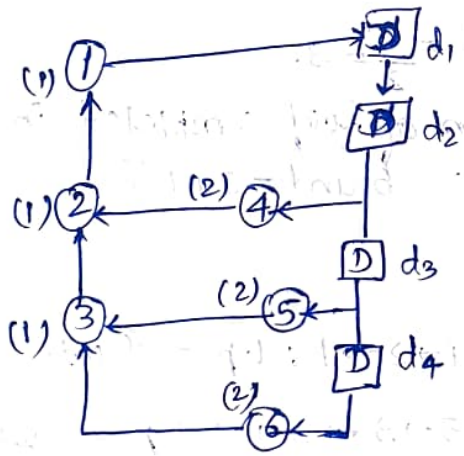
→ value of $l_{ij}^{(m)}$ : Longest computation time of all paths from delay element $d_i$ to delay element $d_j$ that passes through $(m-1)$ delay elements; if no such path, it is set to $-1$.

higher-order matrices : $l_{i,j}^{(m+1)} = \max_{k \in K} \left( -1,\ l_{i,k}^{(1)} + l_{k,j}^{(m)} \right)$

$K$: $[1, d]$

$l_{i,k}^{(1)} \neq -1$ (neither should be $-1$) ↳ $-1$, if so.

$$T_\infty = \max_{i,m \in \{1,2,\ldots,d\}} \left\{ \frac{l_{bb}^{(m)}}{m} \right\}$$




↳ $m-1=0$ ; consider zero delays

$l_{11}(1)$ : no path without delay → $-1$

$l_{12}(1)$ : no computational node → $0$

$l_{13}(1)$ : no path without delay → $-1$

⋮

$l_{31}(1)$ : $d_3 \to ⑤ \to ③ \to ② \to ① \to d_1$

↳ $2 + 1 + 1 + 1 = 5$

$$\therefore L(1) = \begin{bmatrix} -1 & 0 & -1 & -1 \\ 4 & -1 & 0 & -1 \\ 5 & -1 & -1 & 0 \\ 5 & -1 & -1 & -1 \end{bmatrix}$$

14-02-2025

$L^2$: $l_{2,3}^{(2)} = \max_{k \in \{\}} (-1, l_{2,k}^{(1)} + l_{k,3}^{(1)})$ → no valid case → $-1$

$l_{3,2}^{(2)} = \max_{k \in \{1\}} (-1, l_{3,k}^{(1)} + l_{k,2}^{(1)})$ → valid for $k=1$ → 5

$$L(2) = \begin{bmatrix} 4 & -1 & 0 & -1 \\ 5 & 4 & -1 & 0 \\ 5 & 5 & -1 & -1 \\ -1 & 5 & -1 & -1 \end{bmatrix}$$

$l_{i,j}^{2} = \max_{k \in K} (-1, l_{i,k}^{1} + l_{k,j}^{1})$, $l_{i,k}^{1}, l_{k,j}^{1} \neq -1$.

$l_{1,1}^{(2)} = \max_{k \in \{\}} (-1, l_{1,k}^{(1)} + l_{k,1})$ → $k=1$ : not valid
$k=2$ : valid → 4

$L_3$:

$l_{i,j}^{(3)} = \max_{k \in K} (-1, l_{i,k}^{(1)} + l_{k,j}^{(2)})$ ← from L(1) and L(2).

$$L(3) = \begin{bmatrix} 5 & 4 & -1 & 0 \\ 8 & 5 & 4 & -1 \\ 9 & 5 & 5 & -1 \\ 9 & -1 & 5 & -1 \end{bmatrix}$$

$L^4$:

$$L(4) = \begin{bmatrix} 8 & 5 & 4 & -1 \\ 9 & 8 & 5 & 4 \\ 10 & 9 & 5 & 5 \\ 10 & 9 & -1 & 5 \end{bmatrix}$$

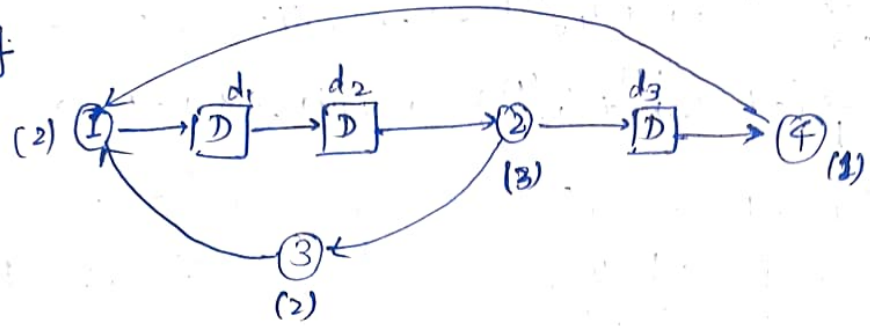
$T_\infty = \max \left\{ \frac{4}{2}, \frac{4}{2}, \frac{5}{3}, \frac{5}{3}, \frac{5}{3}, \frac{8}{4}, \frac{8}{4}, \frac{5}{4}, \frac{5}{4} \right\} = 2$

(5) → diagonal values (if not $-1$)

(4) → delay [L(m)]

[Cycle Algorithm]

Eg.

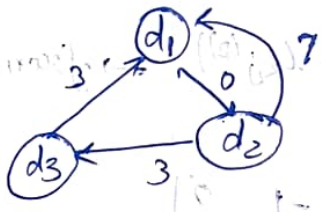



$d_1 \longrightarrow d_2 : 0$

$d_2 \rightarrow d_1 : d_2 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow d_1 = 7$

$d_2 \rightarrow d_3 : d_2 \rightarrow 2 \rightarrow d_3 = 3$

$d_3 \rightarrow d_1 : d_3 \rightarrow 4 \rightarrow 1 \rightarrow d_1 = 3$




$$L^{(1)} = \begin{bmatrix} -1 & 0 & -1 \\ 7 & -1 & 3 \\ 3 & -1 & -1 \end{bmatrix}$$

$$L^{(2)} = \begin{bmatrix} 7 & -1 & 3 \\ 6 & 7 & -1 \\ -1 & 3 & -1 \end{bmatrix}, \quad L^{(3)} = \begin{bmatrix} 6 & 7 & -1 \\ 14 & 6 & 10 \\ 10 & -1 & 6 \end{bmatrix}$$

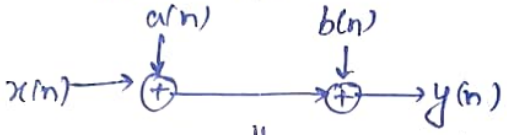
$$T_\infty = \max\left\{ \frac{7}{2}, \frac{7}{2}, \frac{6}{3}, \frac{6}{3}, \frac{6}{3} \right\} = 3.5$$

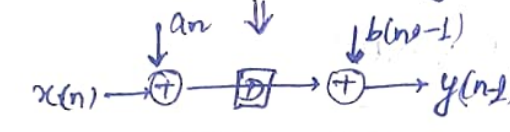
# Pipelining

↳ Reduce the critical path.

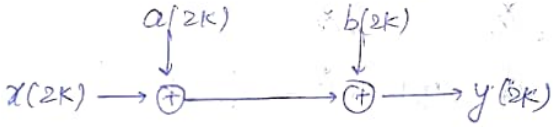
↳ Either increase the clock speed or decrease the power consumption.

↳ Inserting pi buffer without affecting functionality.

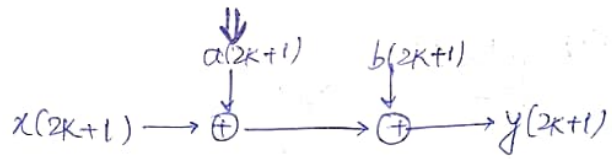



} Pipelining ⇝ latency ↑ as no of buffer ↑

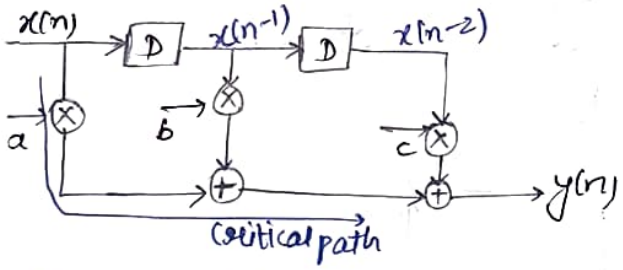



17-02-2025




} Parallel processing

Eg.




$T_M$: multiplication-time

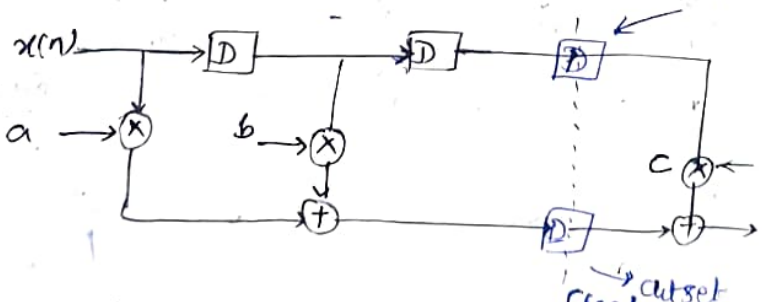
$T_A$: Addition-time

$T_{sample} \geq T_M + 2T_A$

$f_{sample} \leq \frac{1}{T_M + 2T_A}$ → For faster input rate/sample rate, direct form structure cannot be used.

## Pipelining of FIR Digital Filters

Place latches across feed-forward cutset of the graph.



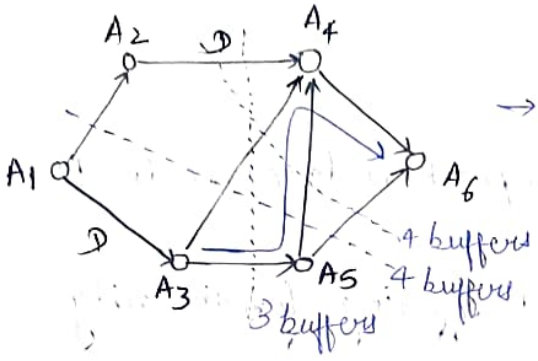

→ Performance ↑
hardware complexity ↑

→ Latency ↑

→ No. of latches ↑

→ Speed of DSP architecture (or clock period) is limited by the longest path b/w any 2 latches, or b/w an i/p and a latch or, b/w a latch and an o/p, or b/w i/p & o/p.

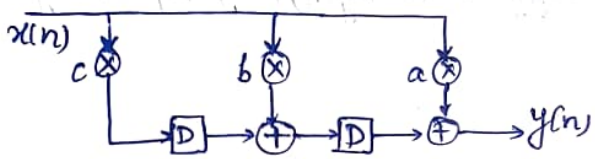
Eg.



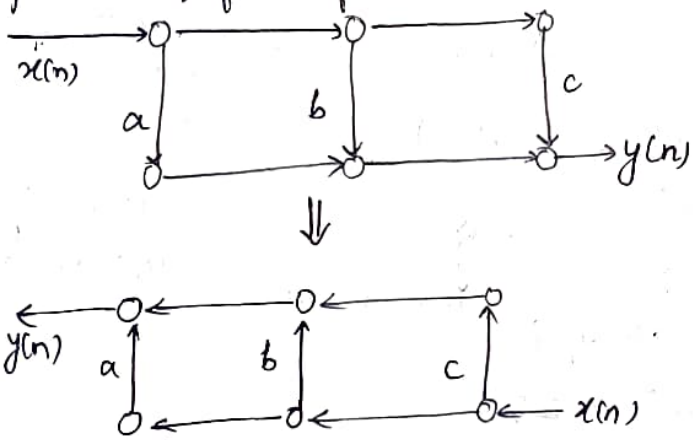

→ Only feed forward cutsets.

## Data Broadcast Structure

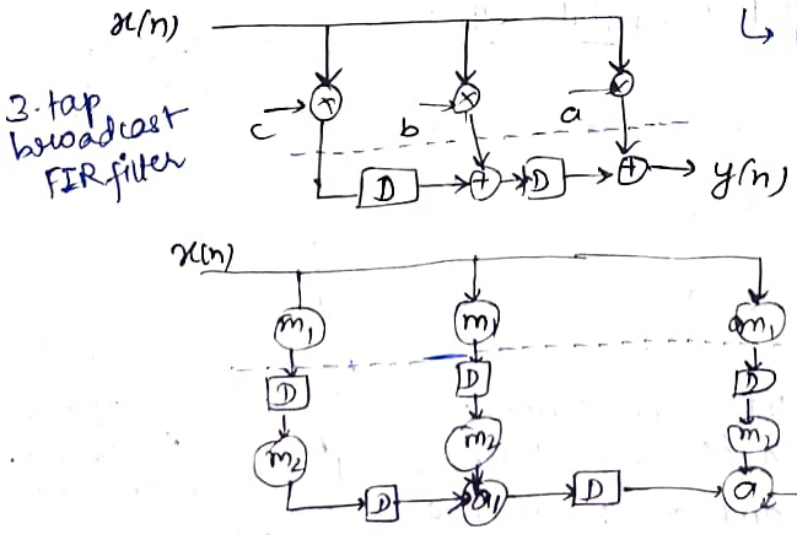
### Data Broadcast structure of FIR filters:




### Transposed SFG of FIR filters:




## Fine Grain Pipelining




Let $T_M = 10$ units, $T_A = 2$ units

↳ Multiplier broked into 2 smaller units with processing times 6 units & 4 units.

⇓

Desired clock period can be achieved in $\left(\frac{T_M + T_A}{2}\right)$.

## Parallel Processing (Block processing)

↳ Increases the sampling rate by replicating the hardware.
↳ Multiple outputs are computed in parallel in a clock period.
↳ Can be used to reduce power consumption.

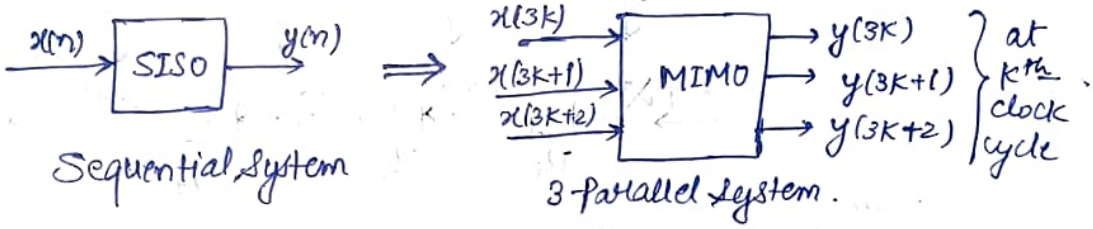
Eg SISO FIR filter

$$y(n) = ax(n) + bx(n-1) + cx(n-2)$$

Convert to MIMO (with 3 i/ps per clock cycle to get parallel system):

$$y(3K) = ax(3K) + bx(3K-1) + cx(3K-2)$$
$$y(3K+1) = ax(3K+1) + bx(3K) + cx(3K-1)$$
$$y(3K+2) = ax(3K+2) + bx(3K+1) + cx(3K).$$

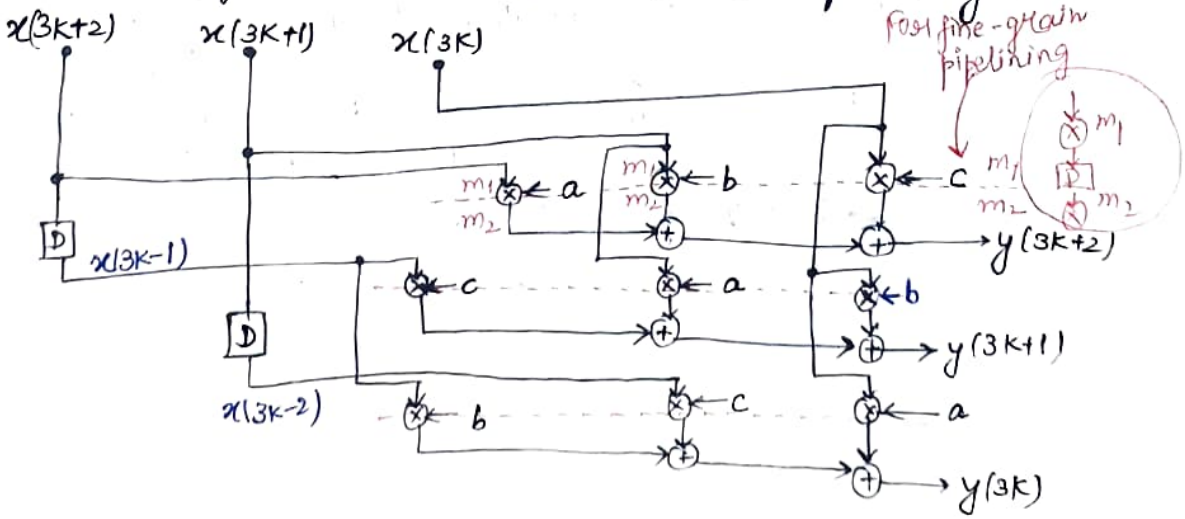
→ No. of i/p.s processed in a clock cycle : block size.




→ In MIMO structure, placing a latch at any line produces an effective delay of L (= block size) clock cycles at the sample rate. So, each delay element known as block delay or L-slow.

↳ Power ↓
Performance maintained (Critical path remains same).
Hardware resources ↑

Eg. 3-tap FIR filter with block size 3 : Parallel processing arch.



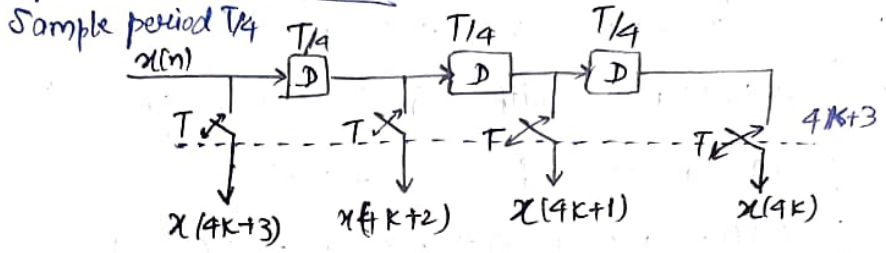
$T_{clock} \geq T_M + 2T_A$

$T_{Iteration} = T_{sample} = \frac{T_{clock}}{L} \geq \frac{T_M + 2T_A}{3}$

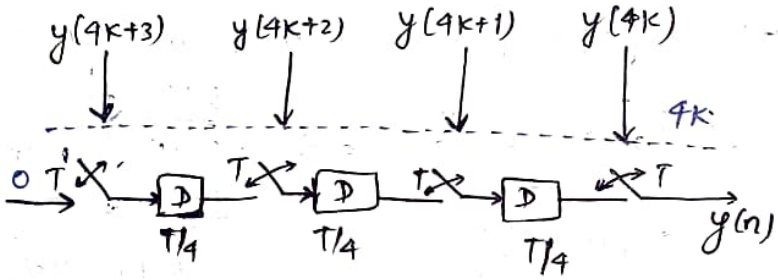
In parallel system: $T_{sample} \neq T_{clock}$

In pipelined system: $T_{sample} = T_{clock}$

Serial-to-parallel converter:




Parallel-to-serial converter:




Why to use parallel-processing:

↳ When the critical path is less than the I/O bound (system is communication bounded), then only pipelining can be used. Once this limit is reached, pipelining can no longer increase the speed.

↳ Combine pipelining with parallel processing to further increase the speed of DSP system.

↳ $T_{Iteration} = T_{sample} = \frac{T_{clock}}{L \cdot M}$ (for L block size parallel processing and M stage pipelining)

Higher speed + Lower power consumption

→ Parallel processing can also be used for reduction of power consumption while using skew clocks.

## For Low Power

Propagation delay ($T_{pd}$) of CMOS circuit:

$$T_{pd} = \frac{C_{charge} \cdot V_0}{k(V_0 - V_t)^2}$$ , $C_{charge}$: capacitance to be charged or discharged in a single clock cycle.

Power consumption in CMOS circuit:

$$P_{CMOS} = C_{total} \cdot V_0^2 \cdot f$$ , $C_{total}$: total capacitance of the CMOS ckt.

In original sequential FIR filter:

$$P_{seq.} = C_{total} \cdot V_0^2 \cdot f, \quad f = 1/T_{seq}$$

① For M-level pipelined system
- ↳ Critical path ↓ to $1/M$ times.
- ↳ charge/disch. capacitance in a single clock cycle ↓ to $1/M$ times.
- ↳ same clock speed ($f$) is maintained.

⇓

Supply voltage ↓ to $\beta V_0$ $(0 < \beta < 1)$.

Pipelined filter:

$$P_{pip} = C_{total} \cdot \beta^2 \cdot V_0^2 \cdot f = \beta^2 \cdot P_{seq.}$$ ↓ by $\beta^2$

pd: $T_{seq.} = \frac{C_{charge} \cdot V_0}{k(V_0 - V_t)^2}$, $T_{pip} = \frac{(C_{charge}/M) \cdot \beta V_0}{k(\beta V_0 - V_t)^2}$

For same clock speed,

$$\Rightarrow M(\beta V_0 - V_t)^2 = \beta (V_0 - V_t)^2 \longrightarrow \text{solve to get } \beta.$$

② For L-parallel system
- ↳ Charging capac. does not change.
- ↳ Total capac. ↑ by L times.
- ↳ To maintain same sampling rate
- ↳ Clock speed ↑ to $LT_{seq}$
- ↳ charging capac. is charged/disch. L times longer ($L.T_{seq}$)

⇓

Supply voltage ↓ to $\beta V_0$

$$T_{seq\,parallel} = \frac{(C_{charge}/L) \cdot \beta V_0}{k(\beta V_0 - V_t)^2}$$

$$\Rightarrow L(\beta V_0 - V_t)^2 = \beta (V_0 - V_t)^2 \longrightarrow \text{solve for } \beta.$$

$$\therefore P_{para} = (LC_{total})(\beta V_0)^2 \frac{f}{L} = \beta^2 . P_{seq}.$$

③ Combining Pipelining and Parallel Processing for Lower Power:

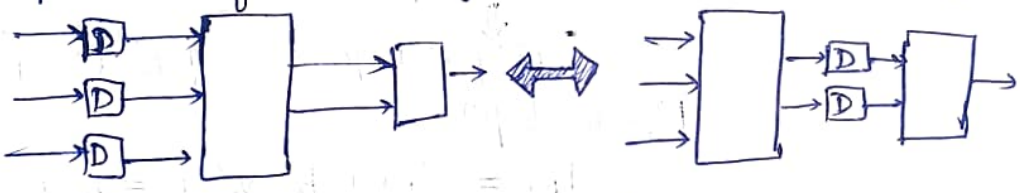
↳ Pipeling reduces the capacitance to be charged/discharged in 1 clock period, while parallel processing increases the clock period for charging/discharging the original capacitance.

$$\text{p.d.: } LT_{pd} = \frac{(C_{charge}/M)\beta V_0}{K(\beta V_0 - V_t)^2} = \frac{L \cdot C_{charge} \cdot V_0}{K(V_0 - V_t)^2}$$

$$\Rightarrow LM.(\beta V_0 - V_t)^2 = \beta (V_0 - V_t)^2 \rightarrow \text{solve for } \beta.$$

## Retiming

↳ A graph-transformation by which D-elements are relocated.



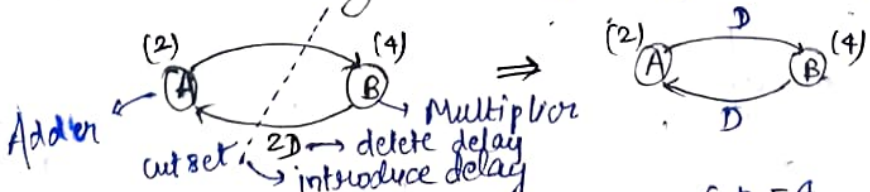
↳ Reduce critical path
↳ Improvement in Speed (clock period ↓)
↳ Power consumption ↓
↳ No. of register ↓
↳ Latency → does not alter

↳ Does not change
↳ delay in loop D
↳ iteration bound

↳ Used in logic synthesis.

Node retiming: Move a D-element from each i/p edge to each edge (or vice versa).

Cutset Retiming: One type of pipelining.




Critical path = 6
I.b. = 6/2 = 3

C.p. = 4
I.b = 6/2 = 3.

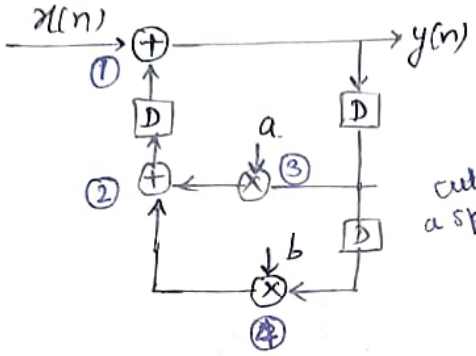
$y(n) = x(n) + y(n-2)$

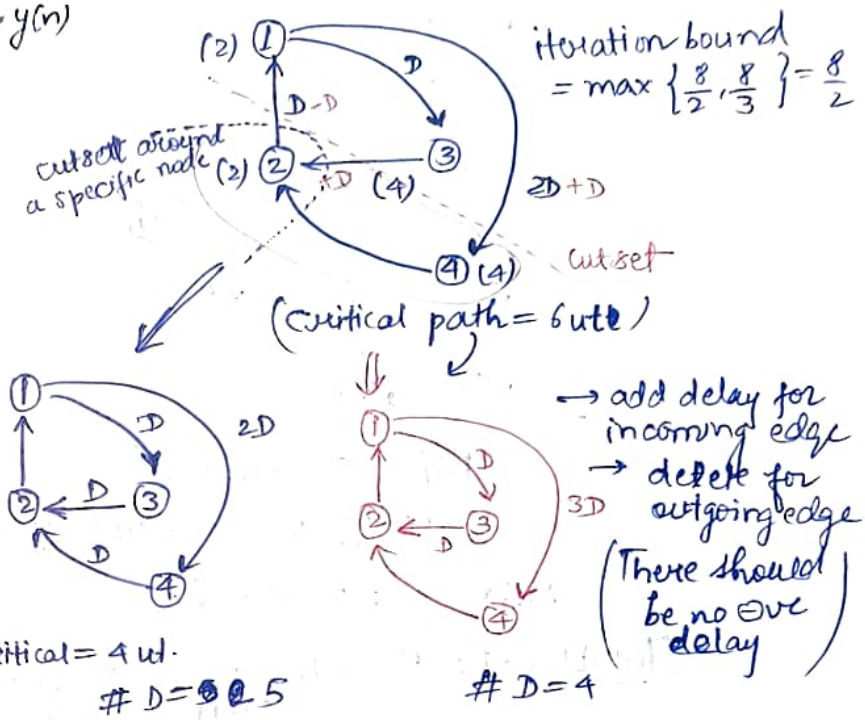
$y(n) = x(n-1) + y(n-3)$

20-02-2025

## Cutset Retiming:

↳ Add delays to edges going one way & remove from ones going the other.




**Retiming value, r(v):** No. of delays moved from each output edge of the node v to each of its i/p edges.

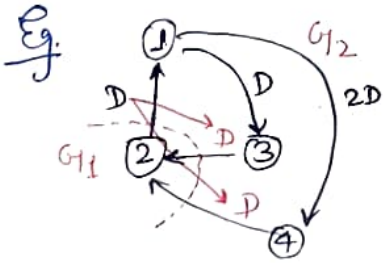
**Feasibility constraints:**

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

$w_r(e)$: after retiming; $w(e)$: before retiming; ($w$: weight/delay)

: no. of delay elements $> 0$ for each edge uv.

Eg.




Original weights

$1 \to 3 = 1$
$1 \to 4 = 2$
$2 \to 1 = 1$
$3 \to 2 = 0$
$4 \to 2 = 0$

$w_r(e) = w(e) + r(N)_{\text{receive}} - r(U)_{\text{send}}$

$1 \to 3 = 1 + 0 - 0 = 1$
$1 \to 4 = 2 + 0 - 0 = 2$
$2 \to 1 = 1 + 0 - 1 = 0$
$3 \to 2 = 0 + 1 - 0 = 1$
$4 \to 2 = 0 + 1 - 0 = 1$

Retimed weights

Choose retiming values:

$G_2 = 0 \Rightarrow r(1) = r(3) = r(4) = 0$

$G_1 = 1 \Rightarrow r(2) = 1$

**Pipeling:** A special case of cutset retiming by placing delays at feedforward cutset.

21-02-2025

## Constraints : [Bellman Ford Algorithm]

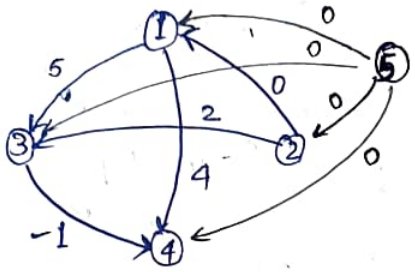
$x_1 - x_2 \leq 0$ [Edge j→i with weight k]

$x_3 - x_1 \leq 5$

$x_4 - x_1 \leq 4$

$x_4 - x_3 \leq -1$

$x_3 - x_2 \leq 2$




W, r
↓ weight ↘ retimed-vector

Reference node = 5th

$W(5-1) = W(5-2) = W(5-3) = W(5-4) = 0$

Ref. node : $r^0(v) = \{0, 0, 0, 0\}$

**Iteration-1 :** 1st node: $W_{(v=1)} = r^0(1) + W^{\infty}(1 \to 1),\ r^0(2) + W^{\infty}(2 \to 1),\ r^0(3) + W^{\infty}(3 \to 1),\ r^0(4) + W^{\infty}(4 \to 1)$

$= \{r^0(v) + W(v-1)\}$

$= \{0, \infty, 0, \infty, \infty\}$ ← for 5th node $\xrightarrow{\min} 0$

For no direct path existing ↓ Take ∞ weight ↑ For path going to itself

$W_{(v=2)} = r^0(1) + W^{\infty}(1-2),\ r^0(2) + W^{\infty}(2-2),\ r^0(3) + W^{\infty}(3-2),\ r^0(4) + W^{\infty}(4-2)$

$= \{0, \infty, \infty, \infty, \infty\} \xrightarrow{\min} 0$

$W_{(v=3)} = \{5, 2, \infty, \infty\} \xrightarrow{\min} 0$

$W_{(v=4)} = \{0, 4, \infty, -1, \infty\} \xrightarrow{\min} -1$

Iteration-2 : $r^1(V) = \{0, 0, 0, -1\}$

$V=1 : W = r^1(1) + w(1-1)$

$r^1(2) + w(2-1)$

$r^1(3) + w(3-1)$

$r^1(4) + w(4-1)$

$= \{\infty, 0, \infty, \infty\}$

$V=2 : W = \{\infty, \infty, \infty, \infty\}$

$V=3 : W = \{5, 2, \infty, \infty\}$

$V=4 : W = \{4, \infty, -1, \infty\}$

$\therefore r^2(V) = \{0, 0, 0, -1\} \rightarrow$ same as the prev.

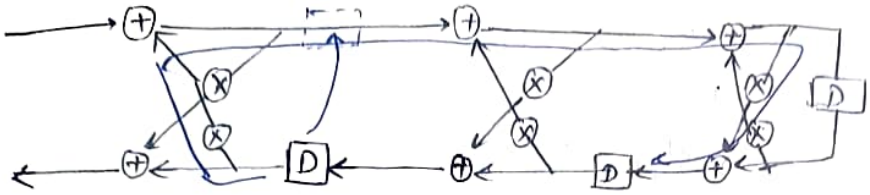
↓ stop.

$r(1) = 0$
$r(2) = 0$
$r(3) = 0$
$r(4) = -1$

} Retimed values

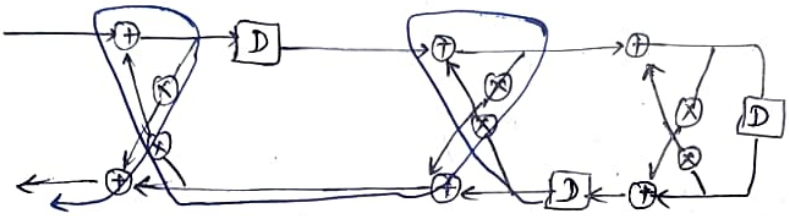
25-02-2025

# Pipelining



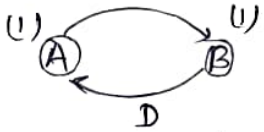
→ 3-stage Lattice filter

$T_{critical} = 4T_A + 2T_M$



→ speed ↑ but not beyond iteration bound.

# K-Slow-down and Retiming



$T_{clk} = 2ut$

$T_{iter} = 2ut$

Replace each D by KD

2 slow transformation

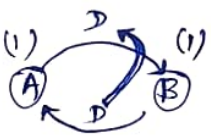
$T_{clk} = 2ut$

$T_{iter} = 2 \times 2ut = 4ut.$

→ K-1 null operation must be inserted

→ Hardware utilization is reduced (eg. 50%).

Retiming → 2-slow version

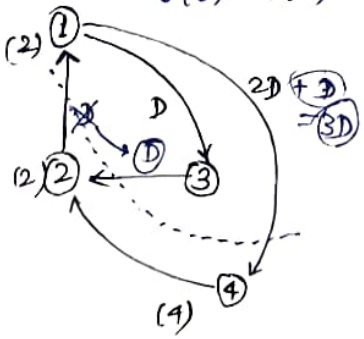


# PCDT constraints from DFG:

$w_r(e) = w(e) + r(V) - r(U) \geq 0$

$r(U) - r(V) \leq w(e)$

$T_{iter} = T_\infty = \max\left\{\frac{8}{2}, \frac{8}{3}\right\} = \frac{8}{2}.$



Nodes: (cannot be reduced)

| | | | |
|---|---|---|---|
| ① → ③ | 2+4 = 6 (computation time) | ↗ Tiger ⇝ can be reduced ✓ | $W_r(1-3) = W(1-3) + r(3) - r(1)$ $= 1 + r(3) - r(1)$ |
| ② → ① | 4 ut | X ⟶ | $W_r(2-1) = 1 + r(1) - r(2)$ |
| ① → ④ | 6 ut | ✓ ⟶ | $W_r(1-4) = 2 + r(4) - r(1)$ |
| ④ → ② | 6 ut | ✓ ⟶ | $W_r(4-2) = 0 + r(2) - r(4)$ |
| ③ → ② | 6 ut | ✓ ⟶ | $W_r(3-2) = 0 + r(2) - r(3)$ |

$W_r(1-3) \geq 1$

$W_r(2-1) \geq 0$

$W_r(1-4) \geq 1$

$W_r(4-2) \geq 1$

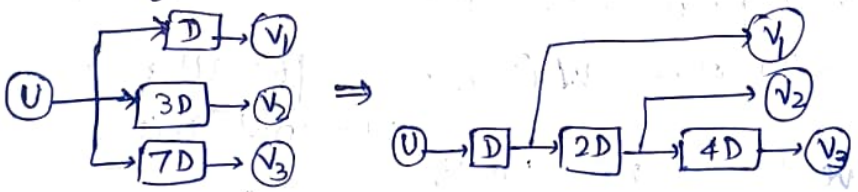
$W_r(3-2) \geq 1$

27-02-2025

## Register Minimization

↳ Find a feasible soln with min. no. of registers in the design space constrained by feasibility and critical path constraints (the refined retimed space with min. clock period).

Eg. Multiple fan-out problem




Fan out ↑
↳ driving multiple o/ps
↳ more driving current ↑

## Minimize Clock Period

Min. feasible clock period of a graph G is

$$\Phi(G) = \max \{ t(p) : w(p) = 0 \}.$$

$W(u,v)$ : Min.$^m$ no. of registers on any path $u \to v$

$D(u,v)$ : Max.$^m$ computation time among all paths from $u \to v$ with weight $W(u,v)$.

→ $M = t_{max} \cdot n$, $t_{max}$ : max$^m$ computation time of any node in G.

$n$ : no. of nodes in G.

form a new graph G' which is same as G except the edge weights replaced by

$$w'(e) = Mw(e) - t(v) \quad (e = u \to v)$$

Solve for all-pairs shortest path on G' ($S_{uv}$).

If $u \neq v$, then

$$W(u,v) = \lceil S_{uv}/M \rceil \text{ (next integer value) and}$$

$$D(u,v) = M \times W(u,v) - S_{uv} + t(v)$$

If $u = v$, $W(u,v) = 0$,

$$D(u,v) = t(u,v)$$

Then, we use $W(u,v)$, $D(u,v)$ to find if there is a soln such that $\Phi(G) \leq c$.

Construct the set of constraints:

- feasibility constraints:

$$r(u) - r(v) \leq w(e) \quad \text{for every edge in G.}$$

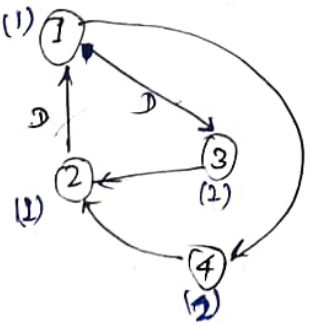
- critical path constraints:

$$r(u) - r(v) \leq W(u,v) - 1 \quad \text{for all nodes } u, v \text{ in G such that } D(u,v) > c \text{ (cycle time).}$$

Solve to get $r(\cdot)$ (retiming values)

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

Eg.

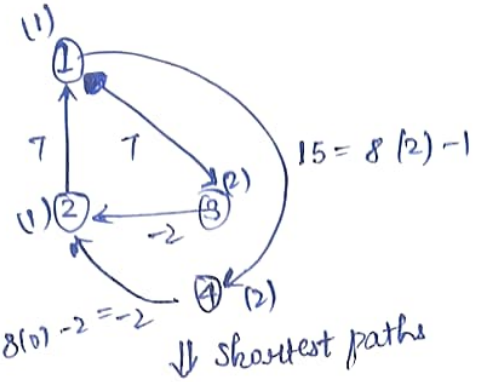



$M = 2 \times 4 = 8$

$w'(e) = M \times w(e) - t(u)$

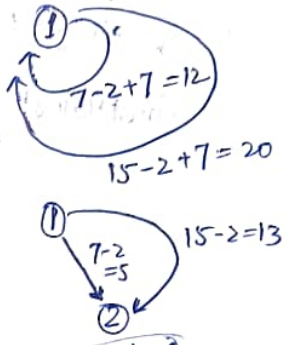
$y(n) = ay(n-2) + by(n-3) + x(n)$

⇓ Find G'




⇓ Shortest paths

| $S_{UV}$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 12 | 5 | 7 | 15 |
| 2 | 7 | 12 | 14 | 22 |
| 3 | 5 | -2 | 12 | 20 |
| 4 | 5 | -2 | 12 | 20 |




⇓ Find W(U,V), D(U,V)

If $U \neq V$, $W(U,V) = [S_{UV}/M]$ ← next integer value, $D(U,V) = M\,W(U,V) - S_{UV} + t(V)$

$U = V$, $W(U,V) = 0$, $D(U,V) = t(U,V) = t(U)$

| W(U,V) | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 2 |
| 2 | 1 | 0 | 2 | 3 |
| 3 | 1 | 0 | 0 | 3 |
| 4 | 1 | 0 | 2 | 0 |

| D(U,V) | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 1 | 4 | 3 | 3 |
| 2 | 2 | 1 | 4 | 4 |
| 3 | 4 | 3 | 2 | 6 |
| 4 | 4 | 3 | 6 | 2 |

**Feasibility constraints:**

) $r(U) - r(V) \leq w(e)$

$\therefore r(1) - r(1) \leq 1$

$r(1) - r(3) \leq 1$

$r(1) - r(4) \leq 2$

$r(3) - r(2) \leq 0$

$r(4) - r(2) \leq 0$

**Critical path constraints:**

$r(U) - r(V) \leq W(U,V) - 1$ for all vertices U, V in G such that $D(U,V) > 3$.

$\therefore r(1) - r(2) \leq 0$

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

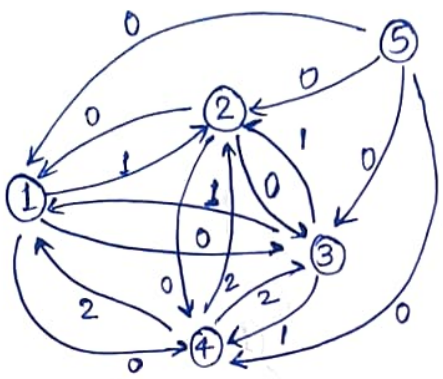
$r(2) - r(4) \leq 2$

$r(3) - r(1) \leq 0$

$r(3) - r(4) \leq 2$

$r(4) - r(1) \leq 0$

$r(4) - r(3) \leq 1.$




Constraint graph for the inequalities

28-02-2025

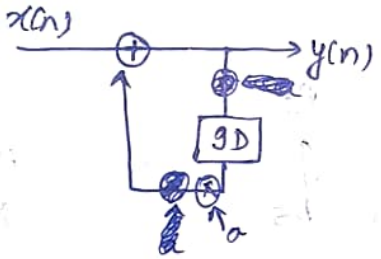
# Unfolding

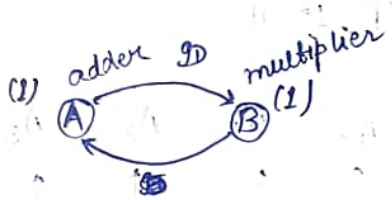
↳ Transformation technique applied to a DSP program to create a new program describing more than one iteration of the original program.

↳ Unfolding factor: J → J consecutive iterations of the original program.

Also used to design bit parallel and word parallel architectures from bit serial and word serial arch.

$$y(n) = ay(n-9) + x(n)$$




- Unfolding factor, $J = 2$.

$$y(2K) = ay(2K-9) + x(2K)$$

$$y(2K+1) = ay(2K-8) + x(2K+1)$$

$$\Rightarrow y(2K) = ay(2K-9) + x(2K)$$

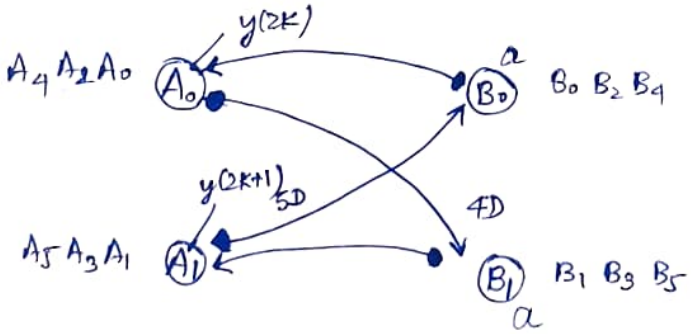
$$= ay(2(K-5)+1) + x(2K)$$

$$y(2K+1) = ay(2K-8) + x(2K+1)$$

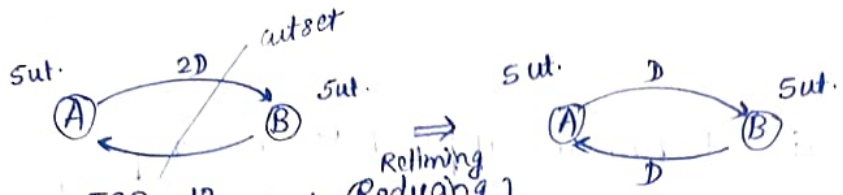
$$= ay(2(K-4)+0) + x(2K+1)$$

2-architectures: even & odd

$A_0, B_0$: even
$A_1, B_1$: odd



Eg.



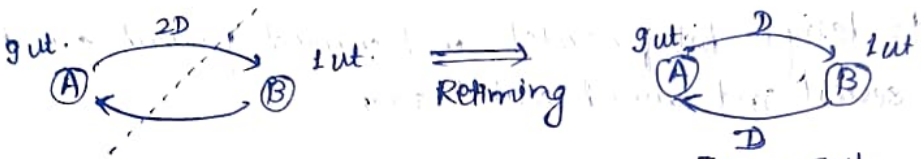

$IPB = \frac{10}{2} = 5$ ut.

$CP = 10$ ut.

$IPB = \frac{10}{2} = 5$ ut.

$CP = 5$ ut.

Eg.

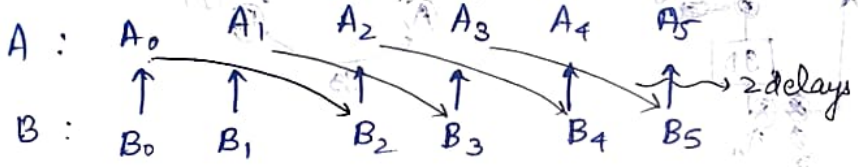



$IPB = \frac{10}{2} = 5$ ut.

$CP = 10$ ut.

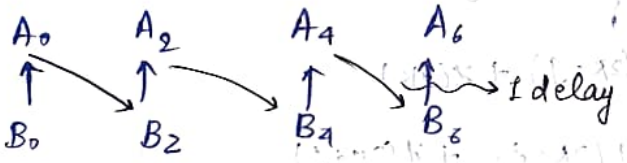
$IPB = 5$ ut.

$CP = 9$ ut. $\approx$ same

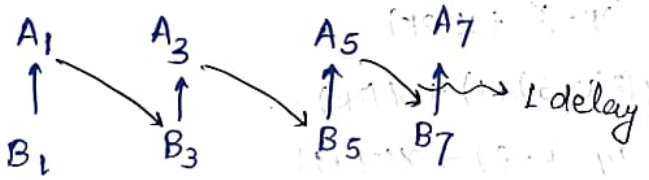


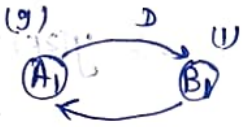

Hardware Resource ↑

$J = 2$ :



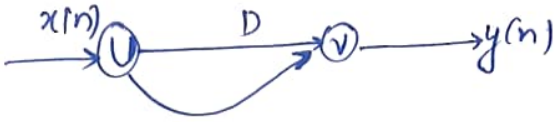





$T_{sample} = \frac{10}{2}$ ← No. of unfoldings

Eg.

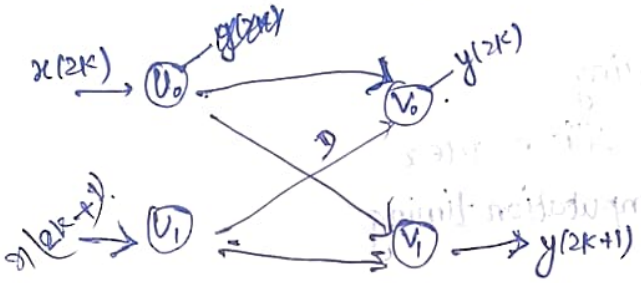



$y(n) = x(n) + x(n-1)$

$y(2K) = x(2K) + x(2K-1)$

$y(2K+1) = x(2K+1) + x(2K)$

$J = 2$ (unfolding factor)




$U_0, V_0$ : even

$U_1, V_1$ : odd

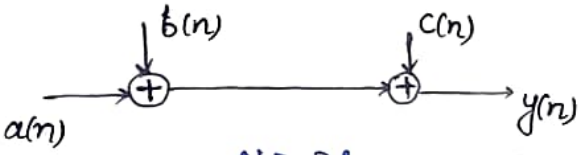
18-03-2025

# Algorithmic Transformations:

1. Unfolding
2. Retiming
3. Folding → Resource sharing
   - (i) Area reduction — silicon area
     - → Increasing computation timing
   - (ii) Control units
   - (iii) Adding of registers.

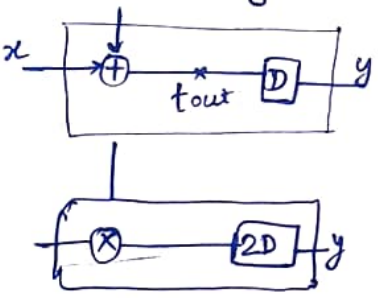
## Folding

Eg.




$N \geq 2$: no. of clock cycles required for single iteration

Pipeline stage:

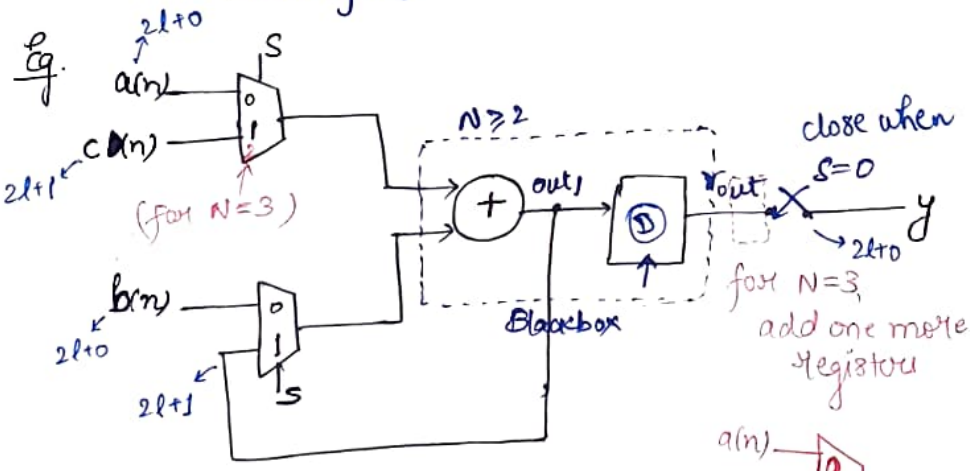


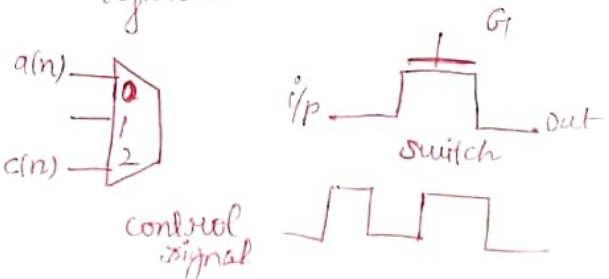

↳ Latency: 2 clock cycles

[Adder takes 1 ut.]

| t: | 0 | 1 | 2 |
|---|---|---|---|
| in 1: | $a_0$ | $a_1$ | $a_2$ |
| in 2: | $b_0$ | $b_1$ | $b_2$ |
| tout: | $a_0+b_0$ | $a_1+b_1$ | $a_2+b_2$ |
| – | $a_0+b_0$ | $a_1+b_1$ | |

Eg.



| S: | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|
| timestamp: (clk) | 0 | 1 | 2 | 3 | 4 | 5 |
| in1 | $a_0$ | $c_0$ | $a_1$ | | | |
| in2 | $b_0$ | $a_0+b_0$ | $b_1$ | | | |
| out1 | $a_0+b_0$ | $a_0+b_0+c_0$ | $a_1+b_1$ | | | |
| rout1 | – | $a_0+b_0$ | $a_0+b_0+c_0$ | | | |
| y | – | – | $a_0+b_0$ | | | |

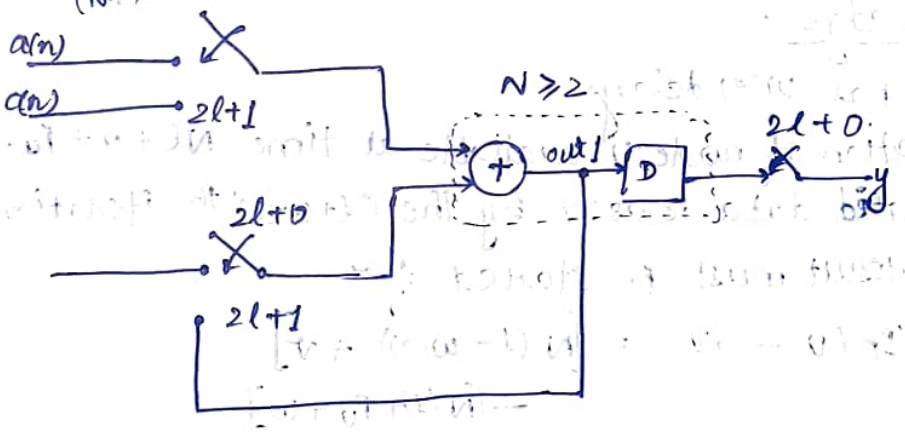
↓

Take o/p when $r_{out1} = a_i+b_i+c_i$ [Tap the output]

$\therefore N \geq 2$ : (Minm clock cycles)

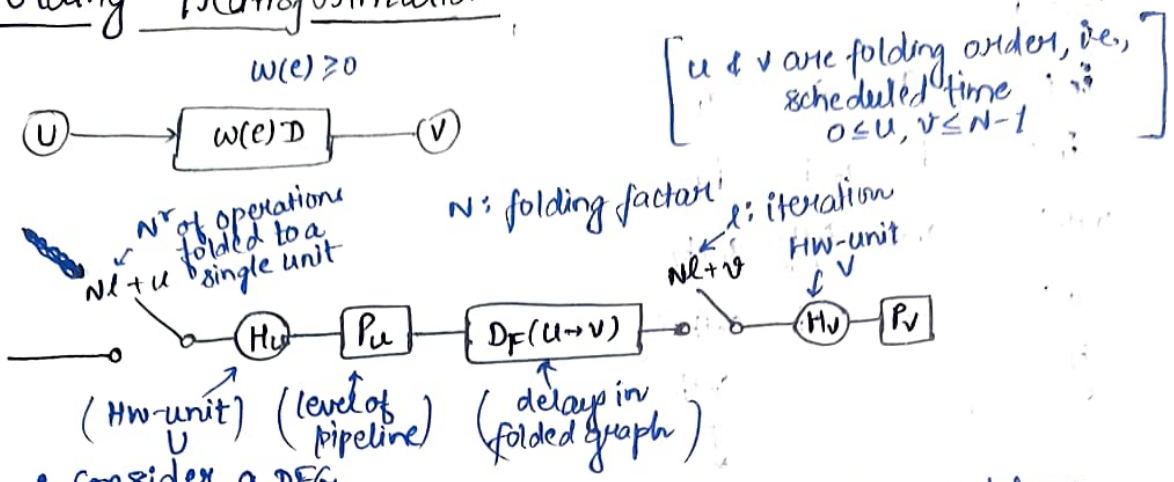
folding factor (N=2) → iteration no. $2l+0$ ← Switching (S=0)

N=2 : Folding factor



20-03-2025

# Folding Transformation

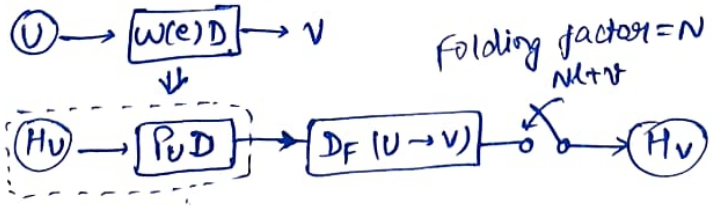



- Consider a DFG.
- An edge e connecting nodes U and V with w(e) delays.
- Executions of the $l^{th}$ iterations of U & V at time units Nl+u & Nl+v.
- u and v (folding orders) : time partitions at which the nodes are scheduled to execute & satisfy $0 \le u, v, N$.
- N: folding factor → no. of operations folded to a single $f^{n}$ unit.
- HU, HV → $f^{n}$ units to execute nodes U and V (HU pipelined by PU stages).

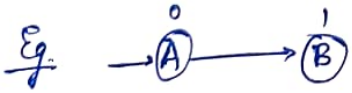
## Folding an edge :

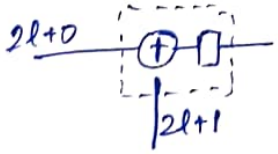
- $U \xrightarrow{e} V$ has w(e) delays.
- $l^{th}$ iteration of node U available at time Nl+u+PU.
- Generated data is used by the $(l+w(e))^{th}$ iteration of V.
- The result must be stored for

$$D_F(U \xrightarrow{e} V) = [N(l+w(e)) + v] - [N(l+P_U+u]$$
$$= Nw(e) - P_U + v - u$$




Eg.

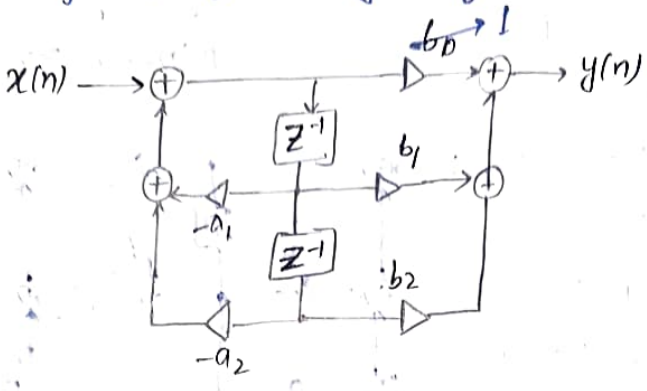



$$D_f = [2(0+0)+1] - [2\times0+1+0]$$
$$= 1-1 = 0$$

# Biquad Filter

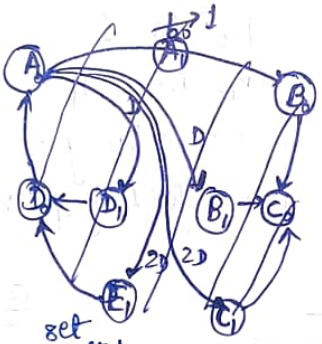
↳ 2 poles, 2 zeros digital filter




Poles: improves ↑ response

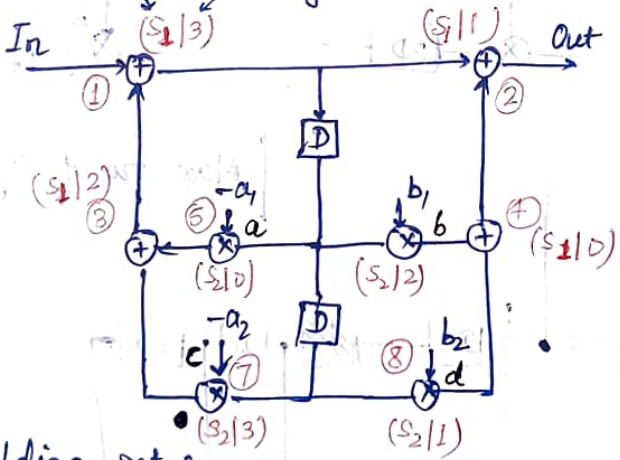
Zeros: reduce ⇒ suppress performance ↓

$$H(z) = g \frac{1 + \beta_1 z^{-1} + \beta_2 z^{-2}}{1 + \alpha_1 z^{-1} + \alpha_2 z^{-2}}$$




set

Folding order 3




**Folding set:**

↳ An order of operations executed by same hardware.

Eg $S_1 = \{A_1, \phi, A_2\}$

$A_1: (S_1|0)$, $A_2 (S_1|2)$

**Biquad filter:**

## Folding equation:

receive ↘ v ; u ↙ send

$$D_F(U \to V) = N w(e) - P_u + v - u$$

$D_F(1\to2) = 4(1) - 1 + 1 - 3 = 1$

$D_F(1\to5) = 4(1) - 1 + 0 - 3 = 0$

$D_F(1\to6) = 4(1) - 1 + 2 - 3 = 2$

$D_F(1\to7) = 4(1) - 1 + 3 - 3 = 3$

$D_F(1\to8) = 4(2) - 1 + 1 - 3 = 5$

$D_F(3\to1) = 4(0) - 1 + 3 - 2 = 0$

$D_F(4\to2) = 4(0) - 1 + 1 - 0 = 0$

$D_F(5\to3) = 4(0) - 1 + 1 - 0 = 0$

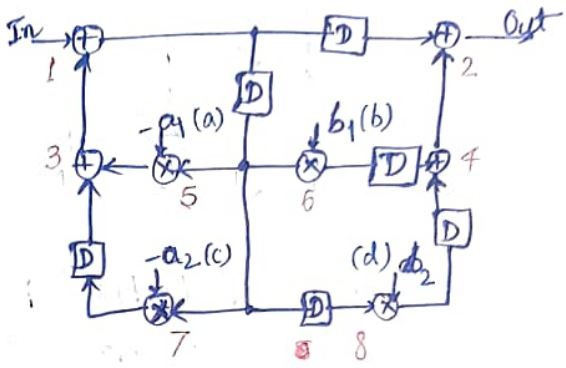
$D_F(6\to4) = 4(1) - 2 + 0 - 2 = 0$

$D_F(7\to3) = 4(1) - 2 + 2 - 3 = 1$

$D_F(8\to4) = 4(1) - 2 + 0 - 1 = 1$

## Valid folding:

$D(U \to V) > 0$




$P_u = P_A = 1$, if u is adder

$P_u = P_M = 2$, if u is multiplier

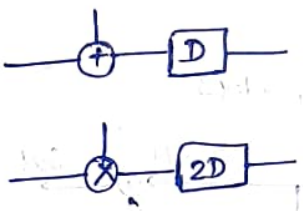
$w(e)$ : delay b/w u & v

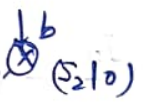
N : folding factor = 4.

$S_1 = \{ \overset{0}{4}, \overset{1}{2}, \overset{2}{3}, \overset{3}{1} \}$

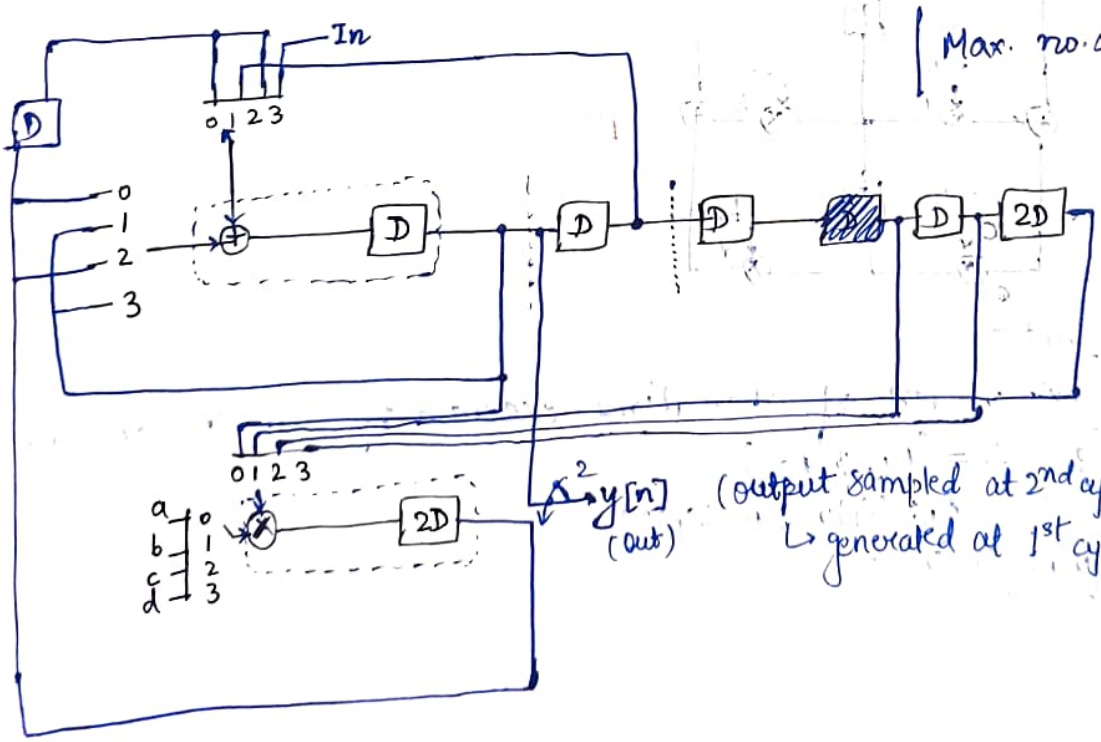
$S_2 = \{ \overset{0}{5}, \overset{1}{8}, \overset{2}{6}, \overset{3}{7} \}$
a d b c

25-03-2025



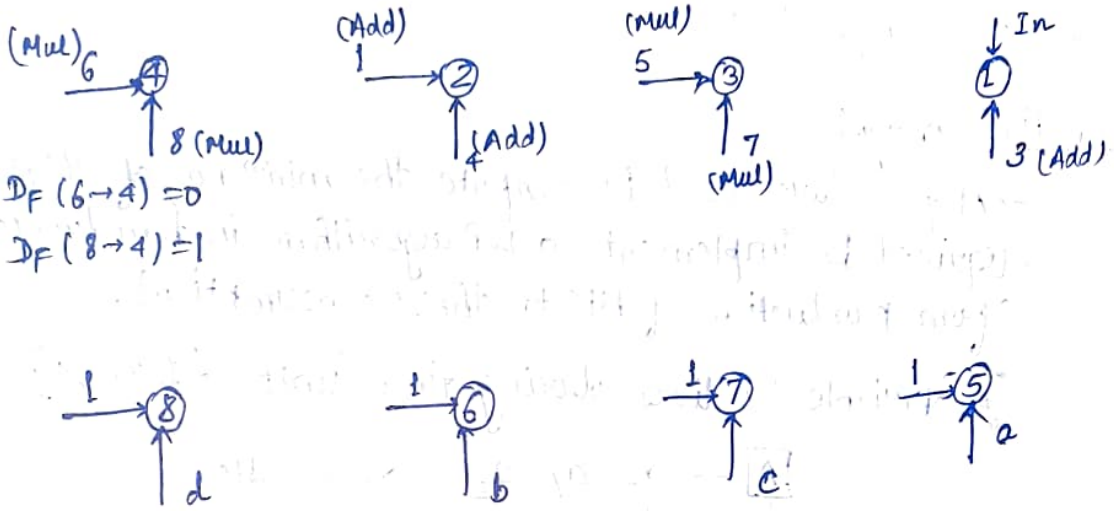




Max. no. of registers = 5




$\overset{2}{\Delta} \to y[n]$ (out) (output sampled at 2nd cycle)

↳ generated at 1st cycle ($S_1|\frac{1}{\uparrow}$)




27-03-2025

# Write verilog code to implement the biquad filter.

↳ Control unit

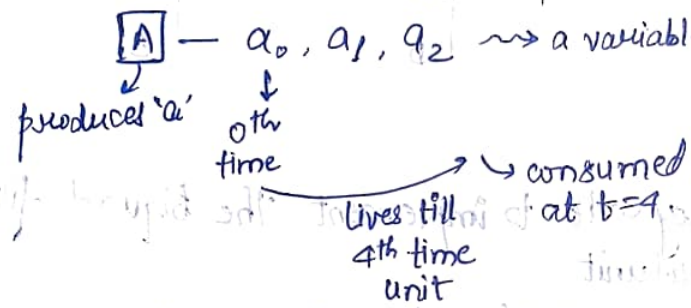
↳ ALU / Data path unit

# Register Minimization

## Lifetime Analysis

↳ A procedure used to compute the min$^m$ no. of registers required to implement a DSP algorithm in hardware (from production of bits to their consumption).

Eg. variable 'a' lives during time unit $\{1, 2, 3, 4\}$




b lives during $\{2, 3, 4, 5, 6, 7\}$

c lives during $\{5, 6, 7\}$

time stamp : $\{1, 2, 3, 4, 5, 6, 7\}$

| | | | | | | |
|---|---|---|---|---|---|---|
| a | a,b | a,b | a,b | b,c | b,c | b,c |
| 1 | 2 | 2 | 2 | 2 | 2 | 2 |

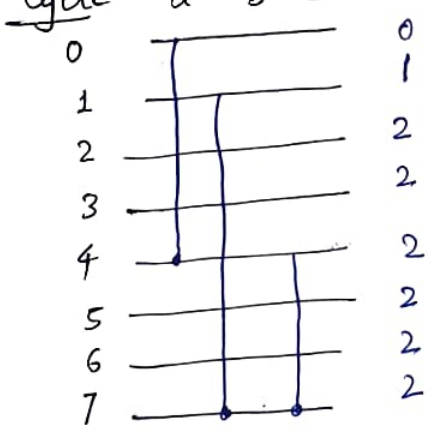
⇝ variables living

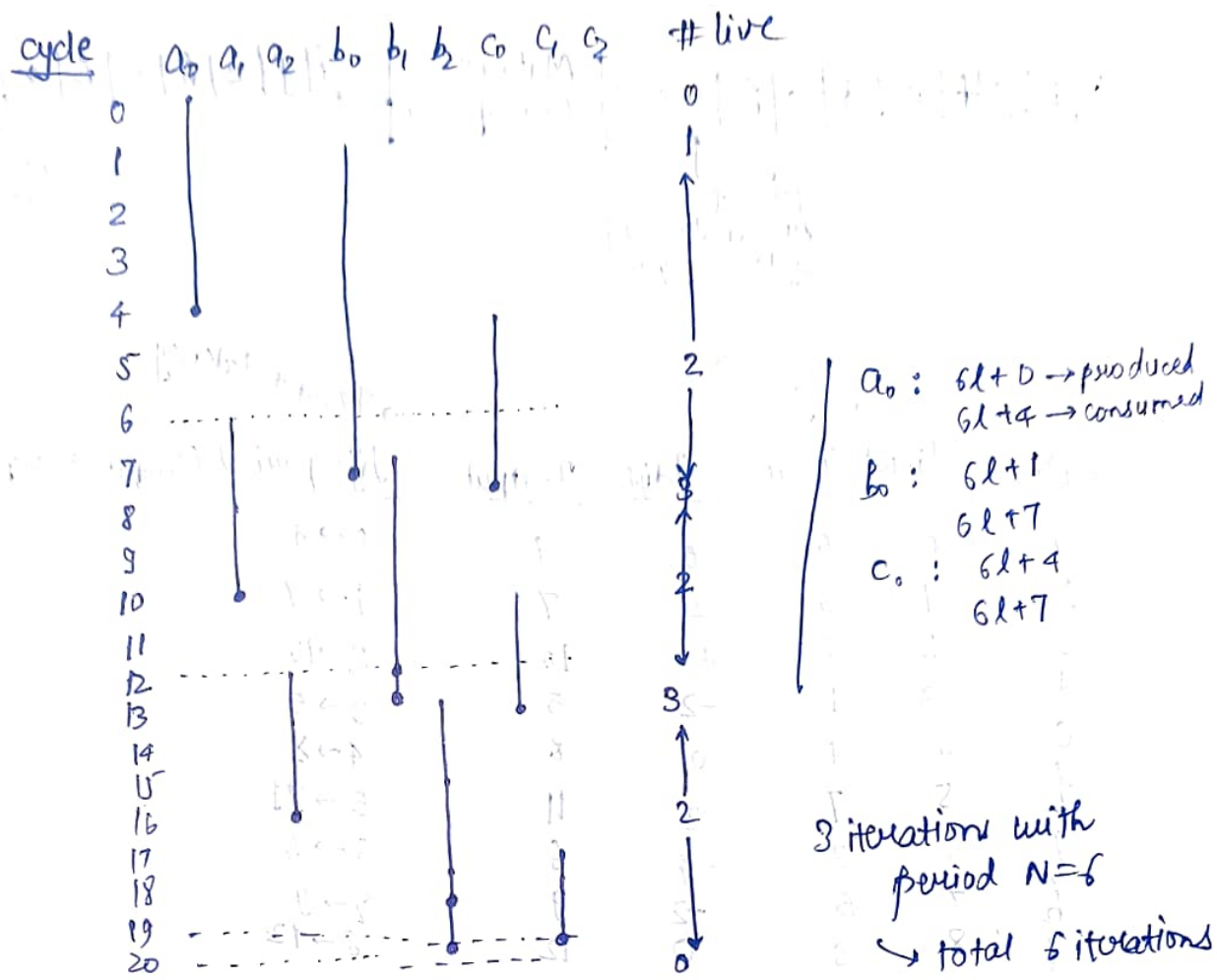
⇝ registers required.

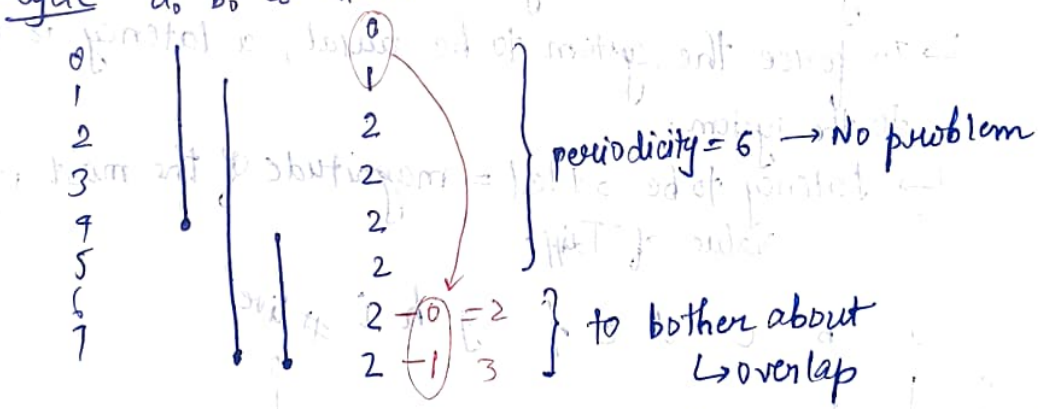
Min$^m$ ~~Total~~ no. of Registers required $= \max\{1, 2, 2, 2, 2, 2, 2\}$

$= 2$.

## Linear lifetime chart






## 3×3 Matrix Transpose

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \Longrightarrow \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

1 iteration → 9 clock cycles

i | h | g | f | e | d | c | b | a → [Matrix Transposer] → i | f | c | h | e | b | g | d | a

1st clk cycle (t=0) ... 9th clock cycle

1st clk cycle ... 9th clock cycle

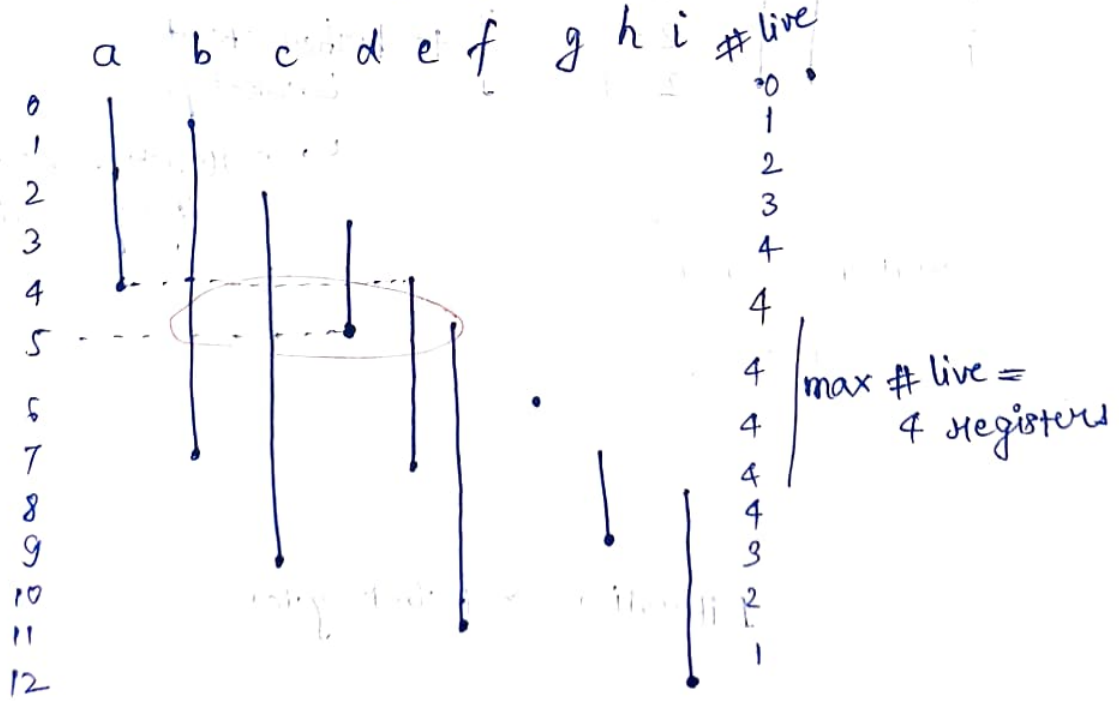
out before in (non-causality)

| Sample | Tinput | Tzout (temporary) | Tdiff | Toutput → Tzout + Latency | Life period (Tinput → Toutput) |
|---|---|---|---|---|---|
| a | 0 | 0 | 0 | 4 | 0 → 4 |
| b | 1 | 3 | 2 | 7 | 1 → 7 |
| c | 2 | 6 | 4 | 10 | 2 → 10 |
| d | 3 | 1 | -2 | 5 | 3 → 5 |
| e | 4 | 4 | 0 | 8 | 4 → 8 |
| f | 5 | 7 | 2 | 11 | 5 → 11 |
| g | 6 | 2 | -4 | 6 | 6 → 6 |
| h | 7 | 5 | -2 | 9 | 7 → 9 |
| i | 8 | 8 | 0 | 12 | 8 → 12 |

Illegal → add latency

↳ To force the system to be causal, a latency is added to the system.

↳ Latency to be added = magnitude of the most negative value of Tdiff.



28-03-2025

# Forward-Backward Register Allocation Technique

Steps:

① Determine the minm no. of registers using lifetime analysis.

② Input each variable at the time step corresponding to the beginning of its lifetime. If multiple variables are input in a given cycle, these are allocated to multiple registers with preference given to the variable with the longest lifetime.

③ Each variable is allocated in a forward manner until it is dead or it reaches the last register. In forward allocation, if the register 'i' holds the variable in the current cycle, then register (i+1) holds the same variable in the next cycle. If (i+1)th register is not free then use the first available forward register.

④ Being periodic, the allocation repeats in each iteration. So hash out the register $R_j$ for the cycle $l+N$ if it holds a variable during cycle $l$.

⑤ For variables that reach the last register and are still alive, they are allocated in a backward manner on a first come first serve basis.

⑥ Repeat steps 4 and 5 until, the allocation is complete.

Eg. Register minimization:

$a = \{1, 2, 3, 4\}$

$b = \{2, 3, 4, 5, 6, 7\}$

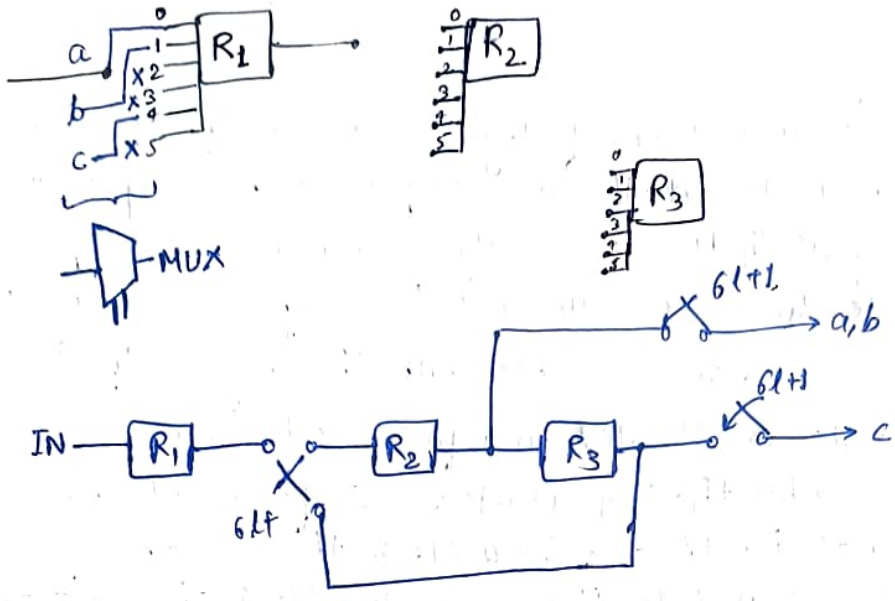
$c = \{5, 6, 7\}$ $[N=6]$

Minm no. of registers = 3. [Single iteration]

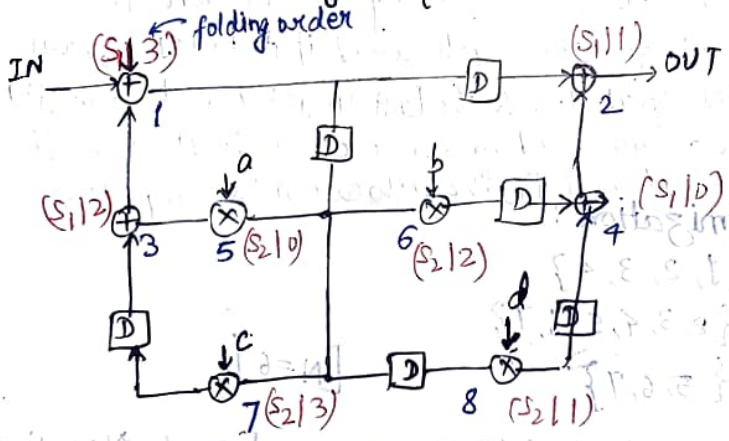
Register Allocation:

| cycle | input | $R_1$ | $R_2$ | $R_3$ | output |
|---|---|---|---|---|---|
| 0 | a → ready by 1st clock cycle | | → ready before 1st clock cycle | | |
| 1 | b | a | | | |
| 2 | ready by 1st clk cycle | b | a | | |
| 3 | | | b | a | |
| 4 | c | | ⓐ | b | a |
| 5 | | c | b | | |
| 6 | | | c | b | |
| 7 | Prepare for next input ($d_1$) | | ⓑ | ⓒ | b, c |

Forward → preference ↓ Backward




## Register Minimization of Biquad filter




One entry for each ~~input~~ node:

$$T_{input} = u + P_u$$

$$T_{output} = u + P_u + \max_v \{D_F(u \to v)\}$$

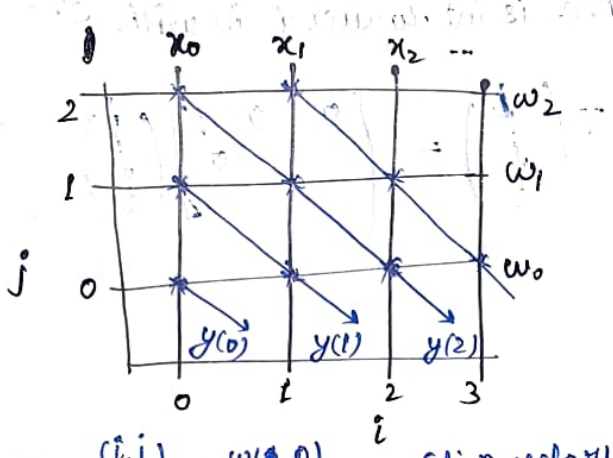
01-04-2025

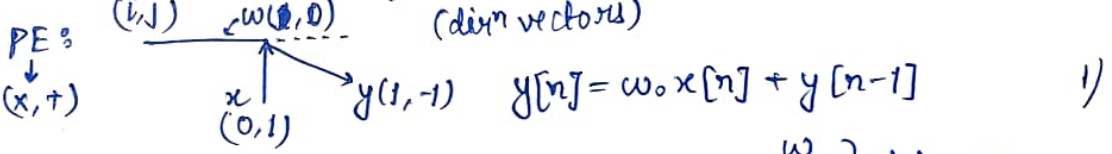
# Systolic Array

→ Systolic architectures are designed by using linear mapping on regular dependence graphs (DGs).

**Regular Dependence Graph**: The presence of an edge in a certain direction at any node in the DG represents the presence of an edge in the direction at all nodes in the DG.

Eg. $y[n] = w_0 x[n] + w_1 x[n-1] + w_2 x[n-2]$



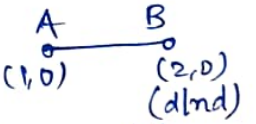

Linear mapping: $(i,j) \xrightarrow{T} (j', t')$

w, x, y } delay or no delay

$j' = i+j$, $t' = i$

**Basic vectors:**

→ Projection vector / iteration vector: $d = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$




→ Two nodes that are displaced by 'd', multiples of 'd' are executed by the same processor.

→ Processor space vector: $P^T = (P_1 \ P_2)$ | $(0,1) \rightarrow (1,0)$

Any node with index $I^T = (i, j)$ would be executed by processor

$$P^T I = (P_1 \ P_2) \begin{pmatrix} i \\ j \end{pmatrix}$$

→ Scheduling vector: $S^T = (S_1, S_2)$

Any node with index I would be executed at time $S^T I$.

→ Hardware Utilization Efficiency: $HUE = \frac{1}{|S^T d|}$

If A is mapped as processor of B.

⇒ can't be executed at same time.

$S^T I_A \neq S^T I_B$, i.e., $S^T d \neq 0$.

Edge mapping: $P^T e$ is introduced in with $S^T e$ delay.

$$\begin{pmatrix} i' \\ j' \\ t' \end{pmatrix} = T \begin{pmatrix} i \\ j \\ t \end{pmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & & \\ 0 & & \end{bmatrix} \begin{pmatrix} i \\ j \\ t \end{pmatrix}$$

# FIR Filter Design $B_1$ (Broadcast inputs, Move results, weights stay)

Distance vector, $d^T = (1, 0)$

$p^T = (0, 1)$ [choose] $[\because p^T d = 0]$ [or $p^T = (0, K)$]

$s^T = (1, 0)$ $[\because s^T d \neq 0]$ [or $s^T = (K_1, K_2)$, $K_1 \neq 0$, $K_2 \rightarrow$ any]

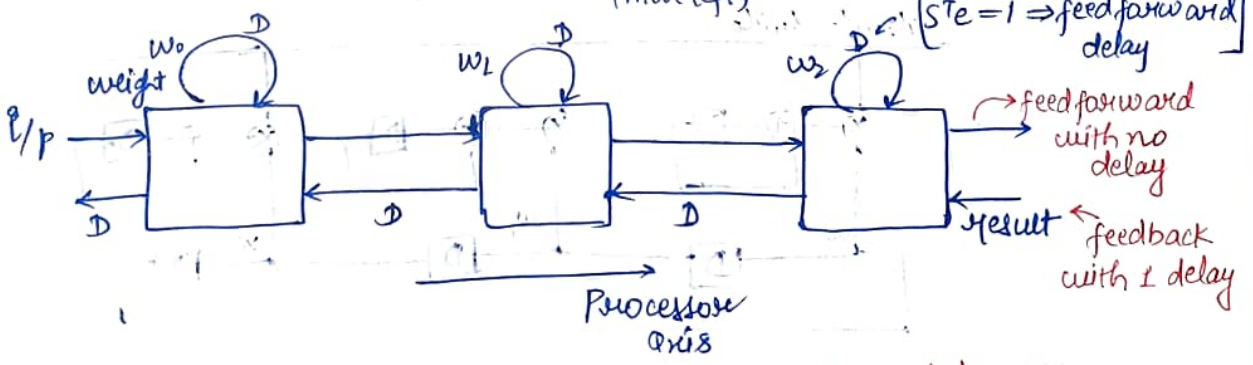
Any node with index $I^T = (i, j)$

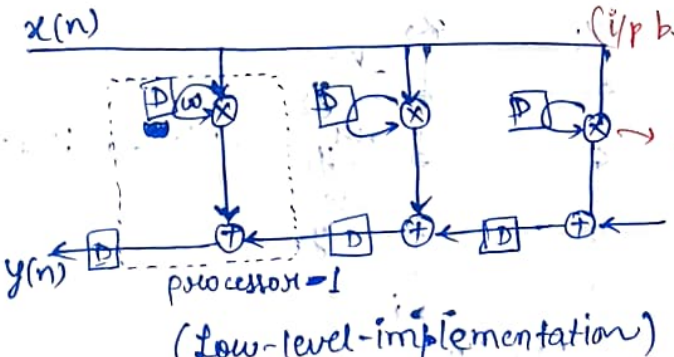
- is mapped to processor $p^T I = j$.
- is executed at time $s^T I = i$.

Since, $s^T d = 1$, we have $HUE = \frac{1}{|s^T d|} = 1$

Edge mapping:

| e | $p^T e$ | $s^T e$ ← delay or no delay |
|---|---|---|
| $w^T (1, 0)$ | 0 → stationary | 1 → delay |
| i/p $(0, 1)$ | 1 (move right) | 0 |
| result $(1, +1)$ | −1 → b/w path (move left) | +1 |




(Low-level-implementation)

(i/p broadcasted) → Direct form 2 or data broadcast structure

multiplied by i/p, delayed and then multiplied

→ Result is staying, weight is moving, i/p is being broadcast

## Design $B_2$:

$d^T = (1, -1)$

$p^T = (1, 1)$

$s^T = (1, 0)$

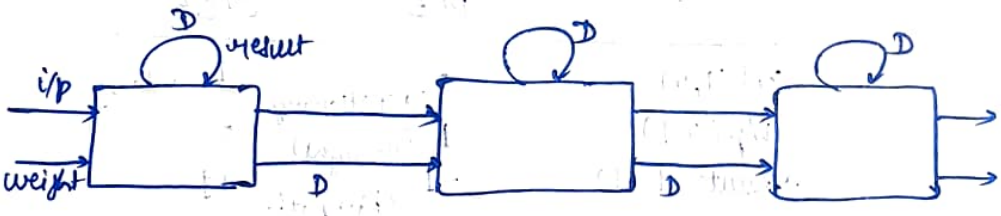
$p^T I = i + j$

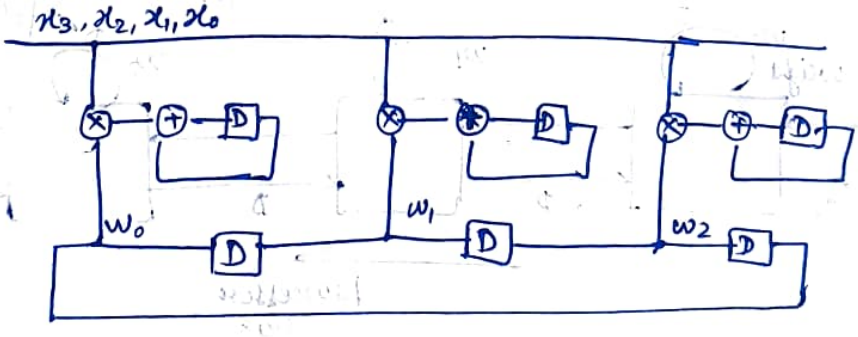
$s^T I = i$

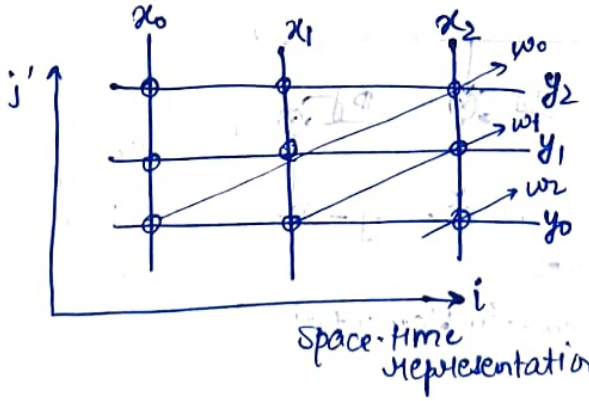
$HUE = \frac{1}{s^T d} = 1$

03-04-2025

| e | $p^T e$ | $s^T e$ |
|---|---|---|
| $w^T(1,0)$ | 1 → (move right) | 1 |
| i/p $(0,1)$ | 1 ↗ | 0 |
| result $(1,-1)$ | 0 → (stays) | 1 |




$(i, j) \rightarrow (j', t')$

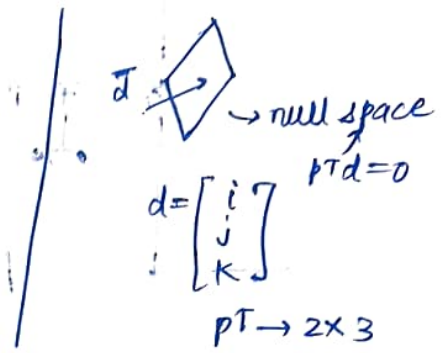
Space-time representation

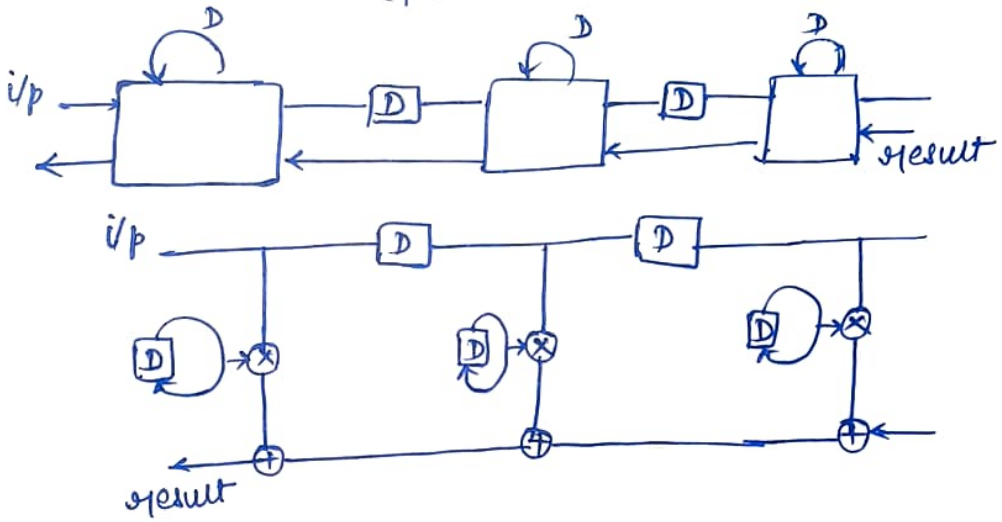
## Design F :

$d^T = (1,0)$, $p^T(0,1)$, $s^T = (i,1)$

$|s^T d| = 1 \Rightarrow HVE = 1$

| e. | $p^T e$ | $s^T e$ |
|---|---|---|
| $w^T(1,0)$ | 0 | 1 |
| i/p (0,1) | 1 | 1 |
| result (1,-1) | -1 (move left) | 0 |





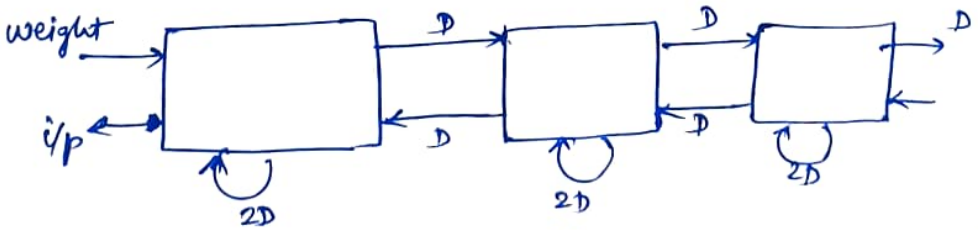


## Design $R_1$ :

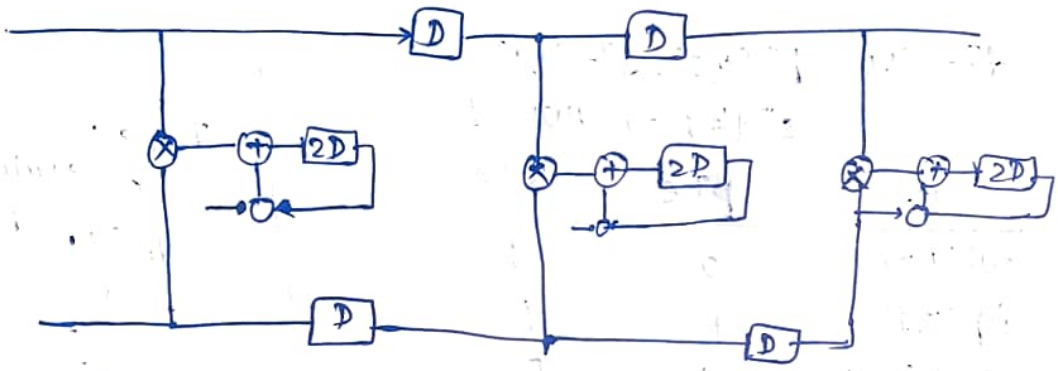
$d^T = (1,-1)$, $p^T = (1,1)$, $s^T = (1,-1)$

$|s^T d| = 2 \Rightarrow HVE = 1/2$.

| e | $p^T e$ | $s^T e$ |
|---|---|---|
| $w^T(1,0)$ | 1 | 1 |
| i/p (0,-1) | -1 | 1 |
| result (1,-1) | 0 | 2 |




Hardware utilization $= \frac{1}{s^T d} = \frac{1}{2} = 50\%$ utilization.


D
D
2D
2D
2D
D
D

07-04-2025

# CONVOLUTION

## Linear Convolution

$$y[n] = x[n] * h[n] = \sum x[n]\, h[n-k]$$
$$= \sum h[n]\, x[n-k]$$

## Circular Convolution

↳ For periodic signals

$x[n] \circledast h[n]$ → modulo operations


## Fast convolution

Systolic
↳ Latency ↑
↓
Throughput ↑

Operations:
Addition
Multiplication → computationally intensive (repeated addition)
↳ More hardware resource
↳ More time
↳ Reduce multiplication at the expense of increasing addition.

"Arithmetic Strength Reduction"

## Polynomial Multiplication

$x[n] = [3, 5]$ $\quad h[n] = [1, 2, 4]$
↓ $3x+5$ $\quad$ ↓ $x^2+2x+4$

| | 1 | 2 | 4 |
|---|---|---|---|
| 3 | 3 | 6 | 12 |
| 5 | 5 | 10 | 20 |

$y[n] = x(n) * h(n) = [3, 6+5, 10+12, 20]$
$y[n] = [3, 11, 22, 20]$

Polynomial wise: $(3x+5)(x^2+2x+4)$

$= 3x^3 + 6x^2 + 12x + 5x^2 + 10x + 20$

$= 3x^3 + 11x^2 + 22x + 20$

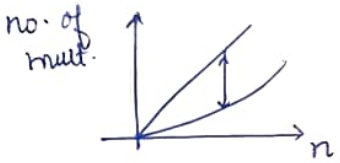
$x[n] \longleftrightarrow X[z]$

$h[n] \longleftrightarrow H[z]$

$Y[z] = X[z] \cdot H[z]$

No. of multiplications: $3 \times 2 = 6$

No. of additions: 2




## Algorithms:

1. Cook and Toom Algorithm
2. ~~Winograd~~ Winograd
   - ↳ Chinese Remainder theorem → Lagrange interpolation

## Complex Multiplication:

$(a+jb)(c+jd) = e+jf$

$$\begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} c & -d \\ d & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

→ 4 multiplications
↳ 2 additions.

$ac - bd = ac - ad + ad - bd$

$= a(c-d) + d(a-b)$

$M_1$ $A_1$ $M_4$ $M_2$ $A_2$

$ad + bc = bc + bd - bd + ad$

$= b(c+d) + d(a-b)$

$M_3$ $A_3$ $A_5$

↳ 3 multiplications
+
5 additions

$$\begin{bmatrix} e \\ f \end{bmatrix} = \underset{2\times 3}{\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}} \underset{3\times 1}{\begin{bmatrix} (c-d)a \\ (c+d)b \\ d(a-b) \end{bmatrix}}$$

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} c-d & 0 & 0 \\ 0 & c+d & 0 \\ 0 & 0 & d \end{bmatrix} \begin{bmatrix} a \\ b \\ a-b \end{bmatrix}$$

↓ Post-computation matrix — no need to multiply with it, just choose operands

↓ pre-computed

↓ computed on-the-fly

↳ 3 multiplications + 3 additions (reduced from 5)

## Lagrange interpolation polynomial :

$$y = f(x)$$

$$P_0(x) = \frac{(x-x_1)(x-x_2)\cdots(x-x_N)}{(x_0-x_1)(x_0-x_2)\cdots(x_0-x_N)}$$

if $x_0$ is known

Substitute $x = x_0$,

$$P_0(x) = 1, \quad x = x_0$$
$$= 0, \quad x \neq x_0.$$

In general, $P_i(x) = \prod_{\substack{j=0 \\ i\neq j}}^{N} \frac{x-x_j}{x_i-x_j}$

$$P(x_j) = 1, \quad i = j$$
$$= 0, \quad i \neq j$$

$$\therefore f(x) = y_0 P_0(x) + y_1 P_1(x) + \cdots + y_n P_n(x)$$

Lagrange interpolation formula : $f(x) = \sum_{i=0}^{N} y_i \cdot P_i(x)$

08-04-2025

## Lagrange interpolation formula

$$y[n] = h(n) \cdot x(n)$$

$$\boxed{S(P) = h(P) \cdot x(P)}$$

$2 \times 2$

$$h = [h_0 \; h_1], \quad x = [x_0 \; x_1]$$

## Cook Toom Algorithm:

1. Choose $L+N-1$ different real numbers $\beta_0, \cdots, \beta_{L+N-2}$. (small values)
2. Compute $h(\beta_i) + x(\beta_i)$.
3. Compute $s(\beta_i) = h(\beta_i) \cdot x(\beta_i)$.
4. Compute $s(P)$.

Eg. $h(P) = h_0 + h_1 P$

$x(P) = x_0 + x_1 P$

$$S(P) = S_0 + S_1 P + S_2 P^2$$

$$= \underset{M_1}{h_0 x_0} + \underset{M_2}{(h_1 x_0 + h_0 x_1)} P + \underset{M_3}{h_1 x_1} P^2$$

($A_1$, $A_2$, $A_3$ mark the additions)

→ 4 Multip. + 3 Additions

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \end{bmatrix} = \begin{bmatrix} h_0 x_0 \\ h_1 x_0 + h_0 x_1 \\ h_1 x_1 \end{bmatrix}$$

$$= \underset{3 \times 2}{\begin{bmatrix} h_0 & 0 \\ h_1 & h_0 \\ 0 & h_1 \end{bmatrix}} \underset{2 \times 1}{\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}}$$

$3 \times 2$

or, $\boxed{S = T\,x}$

## Cook-Toom Algorithm:

1. $L+N-1 = 2+2-1 = 3$

Given that $\beta_0 = 0$

$\beta_1 = 1$

$\beta_2 = -1$

② $h(\beta_i)$

| $\beta_i$ | $h(\beta_i)$ | $x(\beta_i)$ | $S(\beta_i) = h(\beta_i) \cdot x(\beta_i)$ |
|---|---|---|---|
| $\beta_0$ | $h_0$ | $x_0$ | (M1) $h_0 x_0$ |
| $\beta_1$ | $h_0 + h_1$ | $x_0 + x_1$ | (M2) $(h_0+h_1)(x_0+x_1)$ (A1) → on-the-fly add. |
| $\beta_2$ | $h_0 - h_1$ | $x_0 - x_1$ | (M3) $(h_0-h_1)(x_0-x_1)$ (A2) → on-the-fly add. |

# $y = P_0 \cdot y_0 + P_1 \cdot y_1$

$S(\beta_0)$ $S(\beta_1)$

Lagrange interp.

↳ $x \to P$

$x_i \to \beta_i$, $y \to S(P)$

$$S(P) = S(\beta_0) \cdot \frac{(P-\beta_1)(P-\beta_2)}{(\beta_0-\beta_1)(\beta_0-\beta_2)} + S(\beta_1) \cdot \frac{(P-\beta_0)(P-\beta_2)}{(\beta_1-\beta_0)(\beta_1-\beta_2)} + S(\beta_2) \cdot \frac{(P-\beta_0)(P-\beta_1)}{(\beta_2-\beta_0)(\beta_2-\beta_1)}$$

$$= -S(\beta_0) \cdot \frac{(P-1)(P+1)}{} + S(\beta_1) \cdot \frac{(P)(P+1)}{2} + \frac{S(\beta_2) \cdot (P) \cdot (P-1)}{2}$$

$$= S(\beta_0) + \left(\frac{S(\beta_1)}{2} - \frac{S(\beta_2)}{2}\right) \cdot P + \left(-S(\beta_0) + \frac{S(\beta_1)}{2} + \frac{S(\beta_0)}{2}\right) P^2$$

$\therefore S(P) = S_0 + S_1 P + S_2 P^2$

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \end{bmatrix} = \begin{bmatrix} S(\beta_0) \\ \frac{S(\beta_1)}{2} - \frac{S(\beta_2)}{2} \\ -S(\beta_0) + \frac{S(\beta_1)}{2} + \frac{S(\beta_0)}{2} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} h(\beta_0) \cdot x(\beta_0) \\ \frac{h(\beta_1) \cdot x(\beta_1)}{2} \\ \frac{h(\beta_2) \cdot x(\beta_2)}{2} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} h(\beta_0) & 0 & 0 \\ 0 & \frac{h(\beta_1)}{2} & 0 \\ 0 & 0 & \frac{h(\beta_2)}{2} \end{bmatrix} \begin{bmatrix} x_0 \\ x_0 + x_1 \\ x_0 - x_1 \end{bmatrix}$$

$(3 \times 3)$ $(3 \times 1)$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} h(\beta_0) & 0 & 0 \\ 0 & \frac{h(\beta_1)}{2} & 0 \\ 0 & 0 & \frac{h(\beta_2)}{2} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

(3×3) (3×2)

↓ Post comp. (add.)

↓ Pre-computation (addition)

$\therefore S = \underline{A} \underset{\text{diagonal matrix}}{H} \underline{B} X$

(post-computation) Pre-computation (preaddition)

→ $H_0 = h_0$

$H_1 = (h_0 + h_1)/2$ } (pre-computed)

$H_2 = (h_0 - h_1)/2$ }

---

$X_0 = x_0$

$X_1 = x_0 + x_1$

$X_2 = x_0 x_1$

---

$S_0 = H_0 X_0$

$S_1 = H_1 X_1$

$S_2 = H_2 X_2$

$\therefore$ $S_1 = S_0$

$S_1 = S_1 - S_2$ } + ③ Additions

$S_2 = -S_0 + S_1 + S_2$ }

$\therefore$ Total → 3M + 5A → 1M reduction, 4A increment.

(ignoring additions in pre-computation)

Conventionally, 4M + 1A

Generally,

$S = Tx = AHBx$.

Since $T = AHB$, CT algorithm provides a way to factorize the convolution matrix T into 3 multiplying matrices and the total no. of multiplications is determined by the non-zero elements on the main diagonal of the matrix H.

[ C and D contain only small integers. ]

↳ Still, the no. of additions has increased.

⇒ Use modified CT algorithm.

# Modified Cook-Toom Algorithm

$$S'(p) = S(p) - S_{L+N-2} \cdot p^{L+N-2}$$
$$\hookrightarrow L+N-3$$

① Choose $L+N-2$ real nos. $\beta_0, \dots, \beta_{L+N-3}$.

② Compute $h(\beta_i)$ and $x(\beta_i)$; $i = 0, \dots, L+N-3$.

③ Compute $S(\beta_i) = h(\beta_i) \cdot x(\beta_i) \quad \forall\ i = 0, \dots, L+N-3$.

④ Compute $S'(\beta_i) = S(\beta_i) - S_{L+N-2}\, \beta_i^{L+N-2}$, $i = 0, \dots, L+N-3$.

⑤ $$S'(p) = \sum_{i=0}^{L+N-3} S'(\beta_i) \cdot \frac{\prod_{j \neq i} (p - \beta_j)}{\prod_{j \neq i} (\beta_i - \beta_j)}$$

⑥ $$S(p) = S'(p) + S_{L+N-2} \cdot p^{L+N-2}.$$

## 2×2 convolution Algorithm using Modified CT with

$\beta_0 = 0,\ \beta_1 = -1.$

$x(p) = x_0 + x_1 p$
$h(p) = h_0 + h_1 p$

$S'(p) = S(p) - h_1 x_1 p^2$ at $\beta_0 = 0, \beta_1 = -1.$

$S'(\beta_i) = h(\beta_i) \cdot x(\beta_i) - h_1 x_1 \beta_i^2$

$H_0 = h_0,\ H_1 = h_0 - h_1,\ H_2 = h_1$
$X_0 = x_0,\ X_1 = x_0 - x_1,\ X_2 = x_1$

$\beta_0 = 0 \Rightarrow h(\beta_0) = h_0,\ x(\beta_0) = x_0$
$\beta_1 = 1 \Rightarrow h(\beta_1) = h_0 - h_1,\ x(\beta_1) = x_0 - x_1$

$S'(\beta_0) = h(\beta_0)\, x(\beta_0) - h_1 x_1 \beta_0^2 = h_0 x_0$

$S'(\beta_1) = h(\beta_1) x(\beta_1) - h_1 x_1 \beta_1^2 = (h_0 - h_1)(x_0 - x_1) - h_1 x_1$

$S'(p) = S'(\beta_0) + p\,(S'(\beta_0) - S'(\beta_1))$

$\therefore S(p) = S'(p) + h_1 x_1 p^2$

$= s_0 + s_1 p + s_2 p^2 = S_0 + S_1 p$

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_0 & 0 & 0 \\ 0 & h_0 - h_1 & 0 \\ 0 & 0 & h_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$

$S_0 = H_0 X_0,\ S_1 = H_1 X_1,\ S_2 = H_2 X_2$

$\Rightarrow S_0 = s_0,\ S_1 = s_0 - s_1,\ S_2 = s_2.$

Large size algo.
↳ winograd algorithm

CT
↳ less efficient

## Chinese Remainder Theorem (CRT) for integer:

↳ non-negative integer → uniquely

Unknown : $x$

$x \,\%\, m_0 \to a_0 \to$ Remainder
$x \,\%\, m_1 \to a_1$
⋮
$x \,\%\, m_n \to a_n \to$ Remainder

- Remainder → w.r.t. modulus under the condition
  $m_0, m_1, m_2, \dots m_n \to$ Relatively prime (coprime)

and, $x < m_0 m_1 m_2 \dots m_n$

$x \equiv a_1 \bmod m_1$
$x \equiv a_2 \bmod m_2$
$\vdots$
$x \equiv a_n \bmod m_n$

→ set of congruence → solve simultaneously to get the value of $x$.

→ $x$ is smaller than the product of moduli
$x < m_1 \cdot m_2 \cdots m_n$

**CRT for integer :**

$C_i = R_{m_i}[C]$, for $i = 0, 1, 2, \ldots, k$.

$C_i$ → Remainder, $m_i$ → moduli, $C$ → unknown integer

$R_{m_i}[C]$ → Remainder when $C \div m_i$

GCD of $[m_0, m_1, \ldots, m_n] = 1$, i.e., $m_i$'s are relatively prime.

$C \ll m_0, m_1, \ldots, m_n$.

$$C = \sum_{i=0}^{k} C_i N_i M_i \bmod (M), \quad M = \prod_{i=0}^{k} m_i,$$

$$M_i = \frac{M}{m_i},$$

$N_i$ is the solution of the equation $N_i M_i + n_i m_i = GCD(M_i, m_i) = 1$.

Solve using Euclidean GCD algorithm.

If $A \neq B$,

$GCD(A, B) = G' = AR + BS = 1$

**Eg.** Given $m_0 = 3, m_1 = 4, m_2 = 5$

$M = 3 \times 4 \times 5 = 60$

$M_0 = \frac{M}{m_0} = 20, \quad M_1 = \frac{M}{m_1} = 15, \quad M_2 = \frac{M}{m_2} = 12.$

$N_i M_i + n_i m_i = GCD(M_i, m_i)$.

$i=0$ : $N_0(20) + n_0(3) = GCD(20, 3) = 1$

$\Rightarrow N_0 = -1, n_0 = 7$

$i=1$ : $N_1(15) + n_1(4) = GCD(15, 4) = 1$

$\Rightarrow N_1 = -1, n_1 = 4$

$i=2$ : $N_2(12) + n_2(5) = 1$

$\Rightarrow N_2 = -2, n_2 = 5.$

$$C = \sum_{i=0}^{2} (C_i N_i M_i) \bmod M$$

Let $C = 17$ : $17 \bmod 3 = 2 \rightarrow C_0$

$17 \bmod 4 = 1 \rightarrow C_1$

$17 \bmod 5 = 2 \rightarrow C_2$

$\therefore C = (C_0 N_0 M_0 + C_1 N_1 M_1 + C_2 N_2 M_2) \bmod M$

$= (2 \times 20 \times -1 + 1 \times -1 \times 15 + 2 \times -2 \times 12) \bmod 60$

$= (40 - 15 - 48) \bmod 60 = (-103) \bmod 60 = \boxed{17}$.

## CRT algorithm for polynomial:

$$c^i(p) = R_{m^i(p)}[C(p)], \quad i = 0, 1, \ldots, K.$$

$c^i(p)$ → remainder polynomial; $m^i(p)$ → moduli polynomial; $C(p)$ → unknown polynomial

$R_{m^i(p)} \rightarrow$ remainder $\left[\frac{C(p)}{m^i(p)}\right]$.

$M(p) = \prod_{i=0}^{K} m^i[p]$ → relatively prime, $M^i(p) = \frac{M(p)}{m^i(p)}$.

$$s(p) = h(p) \times x(p)$$

$$c[p] = \sum_{i=0}^{K} \left( c^i(p) N^i(p) M^i(p) \right) \bmod M(p)$$

$N^i(p) \cdot M^i(p) + n^i(p) \cdot m^i(p) = GCD[M^i(p), m^i(p)] = 1$,

provided that the degree of $c[p]$ should be less than the degree of $M[p]$.

→ The remainder of polynomial w.r.t. moduli polynomial, $p^i + f(p)$ when degree of $f(p) \le i-1$ can be obtained by substituting $p^i$ by $-f(p)$ within polynomial.

Eg. poly. $5x^2+3x+5$.

ⓐ moduli → $(x+2)$ $[x=-2]$

remainder $= 5(-2)^2 + 3(-2) + 5$

$= 19$

ⓑ moduli → $(x^2+2)$, rem. $= 5(-2) + 3x + 5 = 3x - 5$ $[x^2 = -2]$

11-04-2025

Eg. Consider 2×3 linear convolution. Construct an effective realization using Winograd algorithm with

$h(p) = h_0 + h_1 p$

$x(p) = x_0 + x_1 p + x_2 p^2$

Given: $m(p) = p(p-1)(p^2+1)$

$m^{(0)}(p) = p$, $m^{(1)}(p) = p-1$, $m^{(2)}(p) = p^2+1$

$\therefore m(p) = m^{(0)}(p) + m^{(1)}(p) p + m^{(2)}(p) \cdot p^2$

$$M^{(i)}(p) = \frac{m(p)}{m^{(i)}(p)}$$

$\Rightarrow M^{(0)}(p) = (p-1)(p^2+1), = p^3 - p^2 + p - 1 = p(\underbrace{p^2 - p + 1}_{\text{quotient}}) - \underbrace{1}_{\text{remainder}}$

$M^{(1)}(p) = p(p^2+1)$,

$M^2(p) = p(p-1)$.

and,

$$N^{(i)}(p)M^{(i)}(p) + n^{(i)}(p)m^{(i)}(p) = 1, \quad i = 0, 1, 2.$$

1st iteration (i=0):

$$\Rightarrow N^0(p)M^0(p) + n^0(p)m^0(p) = 1$$

$$\Rightarrow N^0(p)(p-1)(p^2+1) + n^0(p).p = 1$$

$$\Rightarrow N^0(p) = -1,$$

$$n^0(p) = p^2 - p + 1.$$

| i | $m^i(p)$ | $M^i(p)$ | $n^i(p)$ | $N^i(p)$ |
|---|---|---|---|---|
| 0 | $p$ | $p^3-p^2+p-1$ | $p^2-p+1$ | $-1$ |
| 1 | $p-1$ | $p^3+p$ | $-\frac{1}{2}(p^2+p+2)$ | $\frac{1}{2}$ |
| 2 | $p^2+1$ | $p^2-p$ | $-\frac{1}{2}(p-2)$ | $\frac{1}{2}(p-1)$ |

Obtained using:

$M(p) = PQ + R$

$$\begin{array}{r|l} & p^2-p+1 \\ \hline p & p^3-p^2+p-1 \\ & p^3 \\ \hline & -p^2 \\ & -p^2 \\ \hline & p-1 \\ & p \\ \hline & -1 \end{array}$$

$$M^0(p) = m^0(p)(p^2-p+1) + (-1)$$

$$\Rightarrow m^0(p)(p^2-p+1) - M^0(p) = 1,$$

and, $N^0(p).(p^3-p^2+p-1) + n^0(p).p = 1$

$$\Rightarrow m^0(p)(p^2-p+1) - M^0(p) = N^0(p)(p^3-p^2+p-1) + n^0(p).p$$

$$\therefore N^{(0)}(p) = -1$$

$$n^0(p) = p^2-p+1$$

2nd iteration : (i=1)

$$M^1(p) = m^1(p)\cdot(p^2+p+2) + 1$$

$$\downarrow \div 2$$

$$\Rightarrow \frac{M^1(p)}{2} = \frac{m^1(p)}{2}\cdot(p^2+p+2) + 1$$

$$\Rightarrow 1 = \frac{M^1(p)}{2} - \frac{m^1(p)}{2}\cdot(p^2+p+2)$$

and, $N^1(p)\cdot M^1(p) + n^1(p)\cdot m^1(p) = 1$

$$\Rightarrow N^1(p)\cdot M^1(p) + n^1(p)\cdot m^1(p) = \frac{M^1(p)}{2} - \frac{m^1(p)}{2}(p^2+p+2)$$

$\therefore N^1(p) = 1/2$

$n^1(p) = -1/2\,(p^2+p+2)$

$p-1 \;\big|\; p^2+p+2$

$p^3+p$

$p^3-p^2$

$+p^2+p$

$p^2-p$

$2p$

$2p-2$

$2$

3rd iteration : (i=2)

$$M^2(p) = m^2(p)\,(1) + (-p-1)$$

$$\Rightarrow -M^2(p) + m^2(p)\,(-p) = 1$$

and, $N^2(p)\,M^2(p) + m^2(p)\,m^2(p) = 1$

$\Rightarrow$ Since remainder is a fn of p.

$\therefore m^{(2)}(p) = [(p+1)]\,Q + r$

$\Rightarrow m^{(2)}(p) = [(p+1)]\,(-p+1) + 2$ → not a fn of p → stop!

$p^2+1 \;\big|\; 1$

$p^2-p$

$p^2+1$

$-p-1$

$-(p+1) \;\big|\; -p+1$

$p^2+1$

$p^2+p$

$-p+1$

$-p-1$

$2$

$$\Rightarrow \frac{m^2(p)}{2} + \frac{(p+1)(-p+1)}{2} = 1.$$

$$\Rightarrow +(p+1) = m^2(p)\cdot 1 - M^2(p)$$

$$\therefore \frac{m^2(p)}{2} + (-p+1)\left[\frac{m^2(p) - M^2(p)}{2}\right] = 1$$

and, $N^2(p)\,M^2(p) + n^2(p)\,m^2(p) = 1$

$$\Rightarrow N^2(p) = \frac{1}{2}(p-1),\quad n^2(p) = \frac{1}{2}(-p+2).$$

Now,

$$\left.\begin{aligned} h^{(i)}(p) &= h(p) \bmod m^i(p) \\ x^i(p) &= x(p) \bmod m^i(p) \end{aligned}\right\} \text{ for } i=0,1,2.$$

$$\Rightarrow h^0(p) = (h_0 + h_1 p) \bmod p = h_0$$

$$h^1(p) = (h_0 + h_1 p) \bmod (p-1) = h_0 + h_1$$

$$h^2(p) = (h_0 + h_1 p) \bmod (p^2+1) = (h_0 + h_1 p)$$

degree > degree of $(h_0 + h_1 p)$

$\Rightarrow$ Rem. $= (h_0 + h_1 p)$

and,

$$x^0(p) = (x_0 + x_1 p + x_2 p^2) \bmod p = x_0$$

$$x^1(p) = (x_0 + x_1 p + x_2 p^2) \bmod (p-1) = x_0 + x_1 + x_2$$

$$x^2(p) = (x_0 + x_1 p + x_2 p^2) \bmod (p^2+1) = x_0 + x_1 p - x_2$$

$$= (x_0 - x_2) + x_1 p$$

and,

$$s^i(p) = h^i(p) \cdot x^i(p), \text{ for } i=0,1,2.$$

$$\Rightarrow s^0(p) = h_0 x_0 = s_0^{(0)} \quad \rightarrow 1 \text{ mult.}$$

$$s^1(p) = (h_0 + h_1)(x_0 + x_1 + x_2) = s_0^{(1)} \quad \rightarrow 1 \text{ mult.}$$

$$s^2(p) = (h_0 + h_1 p)(x_0 - x_2 + x_1 p)$$

$$= \underbrace{h_0(x_0 - x_2) - h_1 x_1}_{s_0(2)} + \underbrace{(h_0 x_1 + h_1(x_0 - x_2))}_{s_1(2)} p \quad \rightarrow 4 \text{ mult.}$$

↘ not a fn of $p$

Rewrite,

$$\left.\begin{aligned} s_0(2) &= h_0(x_0 + x_1 - x_2) - h_1 x_1 \\ s_1(2) &= h_0(x_0 + x_1 - x_2) - (h_1 - h_0)(x_0 - x_2) \end{aligned}\right\} \text{ 3 mult.}$$

$\therefore$ Total mult. required $= 5$.

In matrix form,

$$\begin{bmatrix} s_0(2) \\ s_1(2) \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} h_0 & 0 & 0 \\ 0 & h_1 - h_0 & 0 \\ 0 & 0 & h_0 + h_1 \end{bmatrix} \begin{bmatrix} x_0 + x_1 - x_2 \\ x_0 - x_2 \\ x_1 \end{bmatrix}$$

Now,

$$s(p) = \sum_{i=0}^{2} s^{(i)}(p) N^{(i)}(p) m^{(i)}(p) \bmod m(p)$$

$$s_0 + s_1 p + s_2 p^2 + s_3 p^3 = \left[ s^{(0)}(p)(p^3 - p^2 + p - 1) + s^{(1)}(p)(p^3 + p) + \frac{s^{(2)}(p)}{2}(p^3 - 2p^2 + p) \right] \bmod [p^4 - p^3 + p^2 - p]$$

$$\Rightarrow S_0 + S_1 p + S_2 p^2 + S_3 p^3 = p^0 S_0^{(0)} + p^1 \left[ -S_0^{(0)} + \frac{S_0^{(1)}}{2} + \frac{S_0^{(2)}}{2} + \frac{S_1^{(2)}}{2} \right]$$
$$+ p^2 \left[ S_0^{(0)} - S_0^{(2)} \right]$$
$$+ p^3 \left[ -S_0^{(0)} + \frac{S_0^{(1)}}{2} + \frac{S_0^{(2)}}{2} - \frac{S_1^{(0)}}{2} \right]$$

In matrix form,

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 1 & 1 \\ 1 & 0 & -2 & 0 \\ -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} S_0^{(0)} \\ \frac{1}{2} S_0^{(1)} \\ 1/2\, S_0^{(2)} \\ 1/2\, S_1^{(2)} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 2 & 1 & -1 \\ 1 & 0 & -2 & 0 & 2 \\ -1 & 1 & 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} h_0 & 0 & 0 & 0 & 0 \\ 0 & \frac{h_0+h_1}{2} & 0 & 0 & 0 \\ 0 & 0 & h_0/2 & 0 & 0 \\ 0 & 0 & 0 & (h_1-h_0)/2 & 0 \\ 0 & 0 & 0 & 0 & \frac{h_0+h_1}{2} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

(final form)

Using Winograd algorithm,

for 2×3 convolution,

multiplication required = 5

addition required = 11.

whereas, in direct form,

mult. req. = 6

add. req. = 2.

15-04-2025

## Linear Winograd:

Winograd algorithm also called Minimal Filtering algorithm:

g: filter
d: input
y: output

$$Y = A^T [(Gg) \odot (B^T d)]$$

$\odot$ ← dot/pointwise mult.

each dimension

$$G = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

## 1 D CNN:

F [2, 3]
i/p ↙ ↘ filter

$g: [g_0\ g_1\ g_2\ g_3]$

↓ Rearrange

$$F = \begin{bmatrix} g_0 & g_1 & g_2 \\ g_1 & g_2 & g_3 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix}$$

$$= \begin{bmatrix} f_0 g_0 + f_1 g_1 + f_2 g_2 \\ f_0 g_1 + f_1 g_2 + f_2 g_3 \end{bmatrix} \longrightarrow \text{6 multiplications + 4 additions}$$

$$m_1 = (g_0 - g_2).f_0$$

$$m_2 = (g_1 + g_2)\left(\frac{f_0 + f_1 + f_2}{2}\right)$$

$$m_3 = (g_2 - g_1)\left(\frac{f_0 - f_1 + f_2}{2}\right)$$

$$m_4 = (g_1 - g_3)f_2$$

$$F = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix} \rightarrow$$ 4 multiplications (reduced) + 8 additions (increased) [excluding precomputed $f_i$'s]

**2D CNN:** $A^T[(G_1 g G_1^T) \odot (B^T d B)]$

$$f(x,y) \circledast g(x,y) = \sum_{\forall k} \sum_{\forall l} f(x-k, y-l) \cdot g(x,y)$$

↳ Same as 1D operated 2 times; column- and row-wise and then added.

\# $\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} e \\ f \end{bmatrix}$ : inner product

outer product : reduce the ~~access~~ power consumption and computation complexity.

$$\begin{matrix} ae + bf \\ ce + df \end{matrix}$$

17-04-2025

\# Iterative fast convolution Method [in book (Parhi)].
Heuristic
\# ~~Heuristic~~ based fast convolution (in book)

Arithmetics :

① Canonical sign Digit Arithmetic
② Distributed Arithmetic

# Distributed Arithmetic (DA)

↳ ~~Multiplication~~ No direct multiplication.
↳ For inner product and MAC calculation.

→ A multiply and Accumulate (MAC) operation:

$$y = A_1 x_1 + A_2 x_2 + \dots + A_K x_K$$
($A_K$: const., $x_K$: no.)

$$= \sum_{K=1}^{K} A_K x_K$$

-1 → Tertiary bit/
0 Redudant bit
1

→ DA technique is bit-serial in nature. ⟹ Slow
→ When no. of elements in a vector is nearly same as the wordsize ⟹ Fast.
→ Replaces explicit multiplications by ROM look-ups.
↳ efficient technique for FPGAs.

## Formulation of DA:

$$y = \sum_{K=1}^{K} A_K x_K$$

Let $x_K$ be an N-bit scaled 2's complement no.

$$|x_K| < 1$$

$$x_K : \{b_{K0}, b_{K1}, b_{K2}, \dots, b_{K(N-1)}\} \longrightarrow 2^s \text{ complement.}$$

where $b_{K0}$ is a sign bit

$$\therefore x_K = -b_{K0} + \sum_{n=1}^{N-1} b_{Kn} 2^{-n}$$

$$\Rightarrow y = \sum_{K=1}^{K} A_K \left[ -b_{K0} + \sum_{n=1}^{N-1} b_{Kn} 2^{-n} \right]$$

$$= -\sum_{K=1}^{K} (b_{K0} \cdot A_K) + \sum_{K=1}^{K} \sum_{n=1}^{N-1} (A_K \cdot b_{Kn}) 2^{-n}$$

$$= -\sum_{K=1}^{K} (b_{K0} \cdot A_K) + \sum_{K=1}^{K} \left[ (A_K \cdot b_{K1}) 2^{-1} + (A_K \cdot b_{K2}) 2^{-2} + \dots + (A_K \cdot b_{K(N-1)}) 2^{-(N-1)} \right]$$

$$= -\left[ b_{10} \cdot A_1 + b_{20} \cdot A_2 + \dots + b_{K0} \cdot A_K \right]$$
$$+ \left[ (b_{11} \cdot A_1) 2^{-1} + (b_{12} \cdot A_1) 2^{-2} + \dots + (b_{1(N-1)} \cdot A_1) 2^{-(N-1)} \right]$$
$$+ \left[ (b_{21} \cdot A_2) 2^{-1} + (b_{22} \cdot A_2) 2^{-2} + \dots + (b_{2(N-1)} \cdot A_2) 2^{-(N-1)} \right]$$
$$\vdots$$
$$+ \left[ (b_{K1} \cdot A_K) 2^{-1} + (b_{K2} \cdot A_K) 2^{-2} + \dots + (b_{K(N-1)} \cdot A_K) 2^{-(N-1)} \right]$$

$$= -[b_{10} \cdot A_1 + b_{20} \cdot A_2 + \dots + b_{K0} \cdot A_K]$$
$$+ [(b_{11} \cdot A_1) + (b_{21} \cdot A_2) + \dots + (b_{K1} \cdot A_K)] 2^{-1}$$
$$+ [(b_{12} \cdot A_1) + (b_{22} \cdot A_2) + \dots + (b_{K2} \cdot A_K)] 2^{-2}$$
$$\vdots$$
$$+ [(b_{1(N-1)} \cdot A_1) + (b_{2(N-1)} \cdot A_2) + \dots + (b_{K(N-1)} \cdot A_K)] 2^{-(N-1)}$$

$$= -\sum_{K=1}^{K} (b_{K0}) \cdot A_K + \sum_{n=1}^{N-1} [b_{1n} \cdot A_K + b_{2n} \cdot A_2 + \dots + b_{Kn} \cdot A_K] 2^{-n}$$

$$\therefore \boxed{y = -\sum_{K=1}^{K} A_K \cdot (b_{K0}) + \sum_{n=1}^{N-1} \left[\sum_{K=1}^{K} A_K \cdot b_{Kn}\right] 2^{-n}}$$

Or, $y = \sum_{n=1}^{N-1} \left[\sum_{K=1}^{K} A_K b_{Kn}\right] 2^{-n} + \sum_{K=1}^{K} A_K (-b_{K0})$

$\sum_{K=1}^{K} A_K b_{Kn}$ → $2^K$ possible values

$\sum_{K=1}^{K} A_K (-b_{K0})$ → $2^K$ possible values

With sign bit as input, we can store it in ROM of size $= 2 \times 2^K$.

Eg. No. of taps, $K = 4$

Fixed coeff, $A_1 = 0.72$, $A_2 = -0.3$, $A_3 = 0.95$, $A_4 = 0.11$

We need $2^K = 2^4 = 16$-words ROM.

ROM: Address and contents:

| $b_{1n}$ | $b_{2n}$ | $b_{3n}$ | $b_{4n}$ | Contents |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | $A_4 = 0.11$ |
| | | ⋮ | | |
| 0 | 1 | 0 | 1 | $A_2 + A_4 = -0.19$ |
| | | ⋮ | | |
| 1 | 1 | 1 | 1 | $A_1 + A_2 + A_3 + A_4 = 1.48$ |

$$\sum_{K=1}^{4} A_K b_{Kn} = A_1 b_{1n} + A_2 b_{2n} + A_3 b_{3n} + A_4 b_{4n}$$

↳ MAC without explicit multiplier

↳ Size of ROM grows exponentially with each added i/p address line.

↳ For each element in a vector, we have an address line.
For $K = 16 \Rightarrow 2^{16}$ (i.e., 64K) of ROM → 16 add. lines.

↳ Offset binary coding to reduce ROM size

$$Y_l = \sum_{k=1}^{K} A_k b_{kn}$$

For $K = 3$,

$$Y_l = A_1 b_{1n} + A_2 b_{2n} + A_3 b_{3n}$$

If bits $\equiv$ 0, 0, 0 : $Y_l = 0$

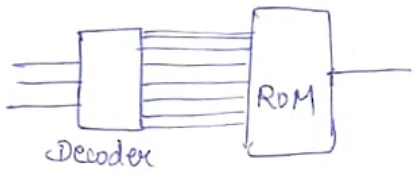
bits $\equiv$ 0, 0, 1 : $Y_l = A_3$

bits $\equiv$ 1, 1, 1 : $Y_l = A_1 + A_2 + A_3$.

⇓

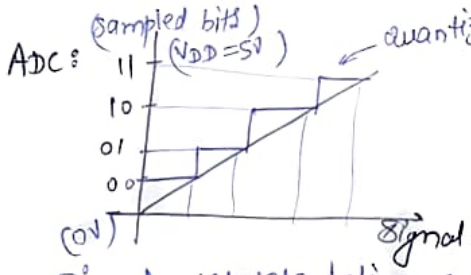
ROM can be used be store the bits.

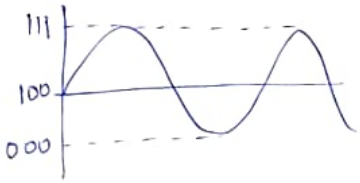
↳ size $= 2 \times 2^K$



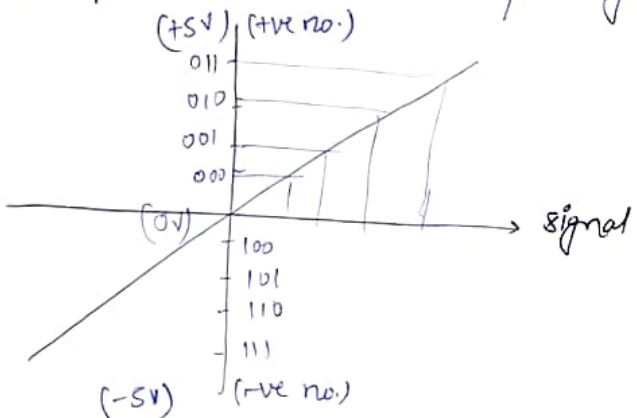

# Lookup Table
↳ Used in FPGA

# Binary offset Coding Technique
↳ Used in ADC

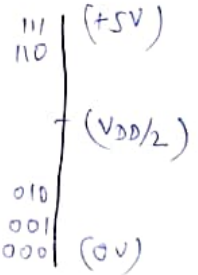



Signed representation : 3rd bit for sign.



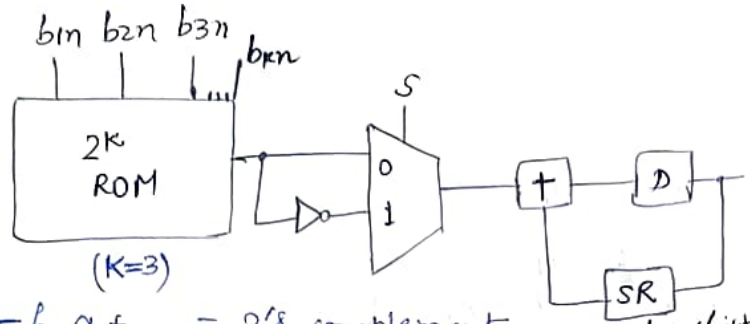

Binary offset coding
↓
Symmetric




→ size of ROM : $2^2$
(2-to-4 decoder)
(not $2^3$)

22-04-2025

# Bit serial : DA for FIR Filter




$-b_0 a + \ldots$ = 2's complement

$$\left( \begin{array}{c} \text{sign} \\ + \\ (\text{sum})2^{-1} \\ + \\ (\text{sum})2^{-2} \\ + \\ \vdots \end{array} \right)$$

$$FT = \int f(x)\, e^{-j2\pi ft}$$

↳ Gives frequency component magnitude and phase.

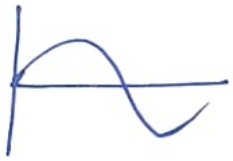
$e^{-j2\pi ft}$
↳ orthogonal basis fn

# Reduction in Filters and Transformations

Fourier transform: Orthogonal transform

cosine transform → Real transform

↳ Discrete cosine transform (DCT)
(eg., jpeg compression)



sine → F → ω (time is lost)

Discrete wavelet transform → Both spatial and frequency components

Different transform → gives additional information (eg. noise)

$$\text{DCT}: X(K) = e(K) \sum_{n=0}^{N-1} x(n) \left[\cos \frac{(2n+1)K}{2N} \pi\right], \quad K=0, 1, \ldots, N-1$$

$$\text{IDCT}: x(n) = \frac{2}{N} \sum_{K=0}^{N-1} e(K)\, X(K) \cos\left[\frac{(2n+1)K}{N} \pi\right], \quad n=0, 1, \ldots, N-1.$$

$$e(K) = \begin{cases} 1/\sqrt{2}, & K=0 \\ 1, & \text{otherwise} \end{cases}$$

$C_i = \cos\left(\frac{i\pi}{16}\right)$

$X(K) = e(K) \sum_{n=0}^{7} x(n) \cos\left(\frac{2n+1}{16}\right) K\pi, \quad K=0, 1, 2, \ldots, 7.$

$$\begin{pmatrix} X(0) \\ X(1) \\ \vdots \\ X(7) \end{pmatrix} = \begin{pmatrix} C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4 & C_4 \\ C_1 & C_3 & C_5 & C_7 & -C_7 & -C_5 & -C_3 & -C_1 \\ C_2 & C_6 & -C_6 & -C_2 & -C_2 & -C_6 & C_6 & C_2 \\ C_3 & & & & & & & \\ C_4 & & & & & & & \\ C_5 & & & & & & & \\ C_6 & & & & & & & \\ C_7 & & & & & & & \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ \vdots \\ x(7) \end{pmatrix}$$

(K ↓, n →)

↳ Reduce 8×8=64 coeff. to 8 coeff. (for 8-point DFT)

$C_4 = \cos\left(\frac{4\pi}{16}\right)$

$$X(0) = \frac{1}{\sqrt{2}} \sum_n 1 = \frac{1}{\sqrt{2}} \cos\left(\frac{4\pi}{16}\right) \sum_n 1 = C_4 \sum_n 1$$

**Reduction**: Using Trigonometric identities

$$\cos\left(\frac{i\pi}{16} \pm 2n\pi\right) = \cos\frac{i\pi}{16} = C_i$$

$$\cos\left(\frac{i\pi}{16} \pm (2n+1)\pi\right) = -\cos\frac{i\pi}{16} = -C_i$$

$$\cos\left(\frac{9\pi}{16}\right) = -\cos\left(\frac{7\pi}{16}\right)$$
$$(C_9) \quad -(C_7)$$

# Butterfly algorithm → for DFT & CFT calculations.

24-04-2025

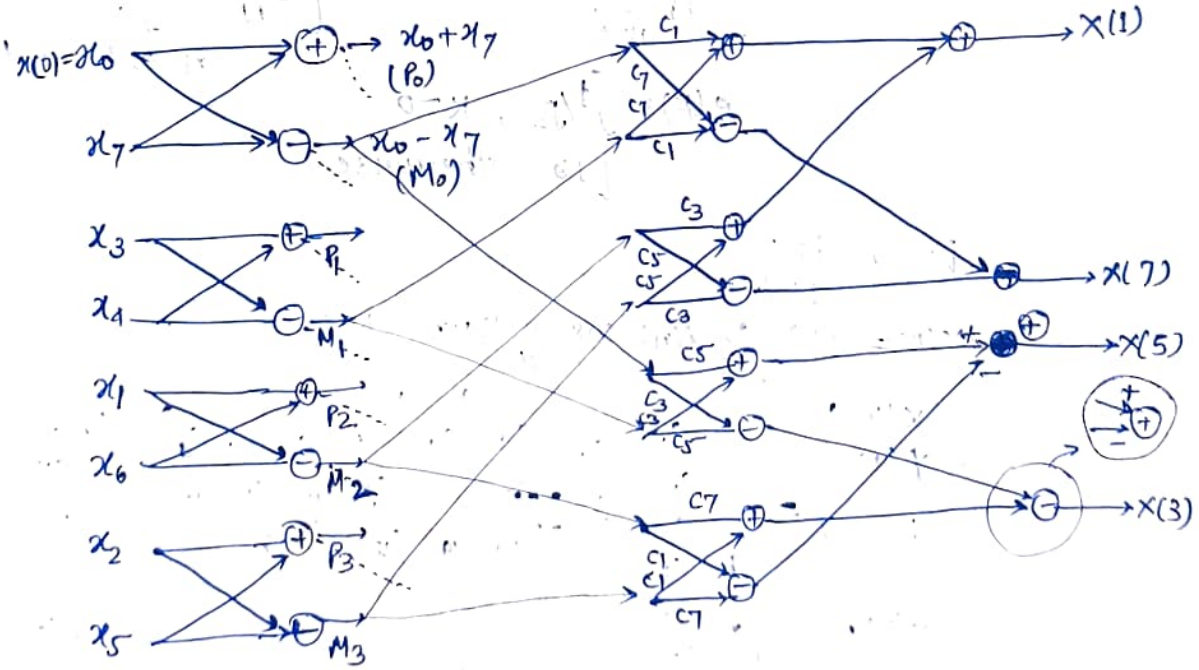
Butterfly blocks types → only adder
→ Adder + multiplier

# orthogonal transform
↳ inverse possible

★ → Deccimation in time FFT algorithm
↳ for endsem

$$X(1) = M_0 C_1 + M_1 C_7 + M_2 C_3 + M_3 C_5$$

$$M_0 = x_0 - x_7 \qquad P_0 = x_0 + x_7$$
$$M_1 = x_3 - x_4 \qquad P_1 = x_3 + x_4$$
$$M_2 = x_1 - x_6 \qquad P_2 = x_1 + x_6$$
$$M_3 = x_2 - x_5 \qquad P_3 = x_2 + x_5$$




$$M_{10} = P_0 - P_1$$
$$M_{11} = P_2 - P_3$$
$$M_{100} = P_{10} - P_{11}$$

$$P_{10} = P_0 + P_1$$
$$P_{11} = P_2 + P_3$$
$$P_{100} = P_{10} + P_{11}$$

$$X(7) = M_0 C_7 - M_1 C_1 - M_2 C_5 + M_3 C_3$$

$$X(3) = M_0 C_3 - M_1 C_5 - M_2 C_7 - M_3 C_1$$

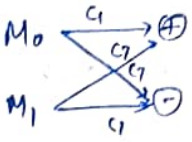
$$X(5) = M_0 C_5 + M_1 C_3 - M_2 C_1 + M_3 C_7$$

$$X(2) = M_{10} C_2 + M_{11} C_6$$

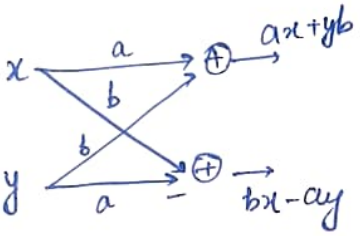
$$X(6) = M_{10} C_6 - M_{11} C_2$$

$$X(4) = M_{100} C_4$$

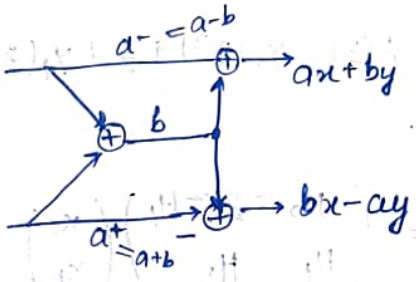
$$X(0) = P_{100} C_4.$$




→ 4 mult. + 2 add.




≡




↳ Rotation matrix (eg., cos, sin, -cos, -sin).

↳ 4 mult. + 2 add.

↳ 3 multiplier + 3 adder

[ $a^+$, $-a^-$ → pre-computed ]

Rotation matrix:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

## Formulation of Parallel (odd-even) Filters using Polyphase Decomposition:

→ area ↑ computation time ↓

Polyphase structure:

$$Y(z) = H(z)\, X(z) = \left(\sum_n h(n) z^{-n}\right)\left(\sum_n x(n) z^{-n}\right)$$

$$X(z) = \sum_{n=0}^{\infty} x(n) z^{-n}$$

$$= x_0 + x_1 z^{-1} + x_2 z^{-2} + \dots$$

$$= (x_0 + x_2 z^{-2} + x_4 z^{-4} + \dots) + (x_1 z^{-1} + x_3 z^{-3} + \dots)$$

$$= X_0(z^{+2}) + z^{-1} X_1(z^{+2})$$

(polyphase components)

Similarly,

$$H(z) = H_0(z^2) + z^{-1} H_1(z^2)$$

$$Y(z) = X_0(z^{+2}) H_0(z^2) + z^{-1}\left[X_0(z^2) H_1(z^2) + X_1(z^2) H_0(z^2)\right] + z^{-2} X_1(z^2) H_1(z^2)$$

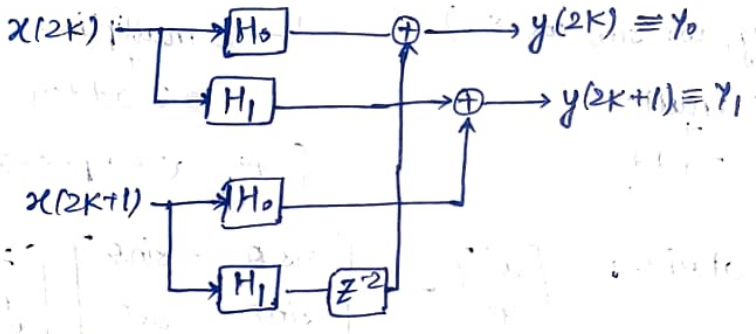
$$Y_0(z^2) = X_0(z^2) H_0(z^2) + z^{-2} X_1(z^2) H_1(z^2)$$

$$Y_1(z^2) = \left[X_0(z^2) H_1(z^2) + X_1(z^2) H_0(z^2)\right]$$

$$\therefore \quad Y(z) = Y_0(z^2) + z^{-1} Y_1(z^2)$$

$$Y = HX$$

$$\Rightarrow \begin{pmatrix} Y_0 \\ Y_1 \end{pmatrix} = \begin{pmatrix} H_0 & z^{-2} H_1 \\ H_1 & H_0 \end{pmatrix} \begin{pmatrix} X_0 \\ X_1 \end{pmatrix}$$




Rank-order Filter
↳ X

26-04-2025

# Coordinate Rotation Digital Computer (CoRDiC)

↳ To find trigonometric functions, hyperbolic functions, Square roots.

Taylor series:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

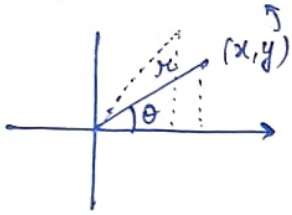
DSP chip:
↳ TMS 6455
↳ Tiger shark

Iterative algorithm:
↳ Binary search
↳ Strength reduction.

## Cordic Rotation Mode (Vectoring Algorithm)

Given the coordinates, how to find θ?

↳ (Linear rotation)




$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Eg. For angle, $\theta = 70^\circ$

Rotate by $-45^\circ \Rightarrow -45^\circ + 70^\circ = 25^\circ$

$\theta \rightarrow +ve$

Rotate by $-22.5^\circ \Rightarrow -22.5 + 25^\circ = 2.5^\circ$

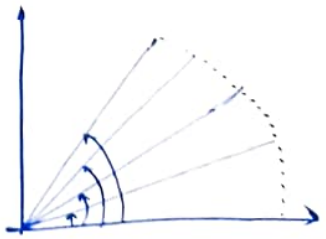
$-11.25^\circ \Rightarrow -11.25 + 2.5^\circ = -8.75^\circ$

$+5.625^\circ \Rightarrow +5.625\ 8.75^\circ = -3.125^\circ$

$+2.8125^\circ \Rightarrow 2.8.125^\circ - 3.125^\circ = -0.3125^\circ$

$+1.40625^\circ \Rightarrow 1.40625 - 0.3125^\circ = 1.09375^\circ$

$45^\circ + 22.5^\circ + 11.25^\circ - 5.625^\circ - 2.8125^\circ =$



$\tan\theta = 2^{-i}$, if $\theta$

$\theta i = 2^{-i}$

$\tan\theta^i = 2^{-i}$

$\hat{x}_2 = x_1 - y_1 \tan\theta = x_1 - y_1 2^{-i}$

$\hat{y}_2 = y_1 + x_1 \tan\theta = y_1 - x_1 2^{-i}$

iteration: $x^{i+1} = x^i - d_i 2^{(-i)} \cdot y^i$ $\Big\rangle$ $d = \pm 1$

$y^{i+1} = y^i + d_i (2^{-i} x^i)$

$z^{i+1} = z^i - d_i \theta(i)$

28-04-2025

# Canonical Signed Digit Representation (CSD)

↳ Rep. no. of non-zero bits minimal

Shift add

Ternary representation : $\{0, 1, -1\}$

Binary : $\{0, 1\}$

Eg. $x \times 30 = x \times 3 \times 10^1 + 0$ (decimal)

$y = x \times 11110$ (binary)

$y = 2^4 x + 2^3 x + 2^2 x + 2^1 x$

↳ 4 mult. + 3 add.

↳ actually shifting ⇒ $0010 \times 2$

$0100$

$C = 32 - 2$

$= 100000 - 10$

$y = Cx$

$y = x \times 2^5 - x \times 2^1$

↳ 2 mult. + 1 sub. (or add.)

[2's complement]

**Property:**

① No 2 consecutive bits in CSD are non-zero.

② Minimal no. of non-zero bits.

③ It is unique.

④ Range of no. : $\{-\frac{4}{3}, \frac{4}{3}\}$

⑤ W-bit CSD $\{-1, 1\}$,

the avg. no. of non-zero bits $= \frac{w}{3} + \frac{1}{9} + O(2^{-w})$ → on avg.

↓ order of

# 15-bit no. → $1/3$ (~30%) → 20% reduction $1/2$

(probability in CSD of giving non-zero bits) (probability in binary)

Eg. Binary → 1010111 (consecutive non-zero)

Not allowed

LSB

0111

↓

10001 → (-1)

∴ 1011001

not allowed

↓

11 01 001

↓

∴ 101 01 001

add, subtract

$2^7 - 2^5 - 2^3 - 2^0$

Binary → CSD converter

011

↓

101

Represent negative by 2's complement.

↳ MSB ≡ sign bit

Eg. Convert 1111 0 to CSD.

↓

011110

↓

100010

↓

1000T0 → 30

$2^5 - 2^1 = 30$

T000T0 → -30

↔ 011110

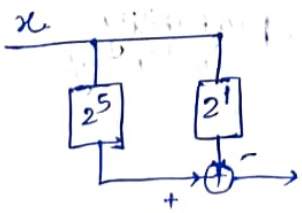
↓

1000T0 → 30

T00010 (if leading 0 is sign bit)

↘

-30



## Finite wordlength:

Horner's rule for precision system

$x \cdot 2^{-5} + x^{-3}$

↳ right-shift by 5.

$x \rightarrow$ 8 bit : 0 111 1111

0111 1111 → | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 11111 $(2^{-5})$

\+ 0 0 0 0 1 1 1 1 111 $(2^{-3})$

$(x(2^{-2}) + x)2^{-3}$

$x \times (0.10100\bar{1}0010\bar{1}00\bar{1})$

$= x \times (2^{-1} + 2^{-3} - 2^{-6} + 2^{-9} - 2^{-11} - 2^{-14})$

$= 2^{-1}\left[(x + 2^{-2}x) + 2^{-5}((-x + 2^{-3}x) - 2^{-5}(x + 2^{-3}))\right]$

⟶ Precision gets improved.

# Sub-expression elimination