More about CORDIC
Implementing functions and DCT

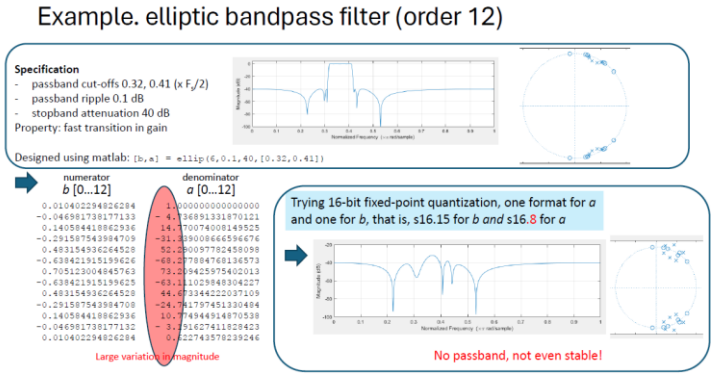
Signal Processing Systems Fall 2025

Lecture 7 (Monday 17.11.)

Outline

- Three-valued CORDIC
  - Skipping elementary rotations
- Unified CORDIC
  - System for fixed-point implementation of various functions
- Other function implementation techniques
- DCT transform
  - Application in image/video coding, efficient implementations
  - MDCT transform & audio coding

About Quiz 2 answers

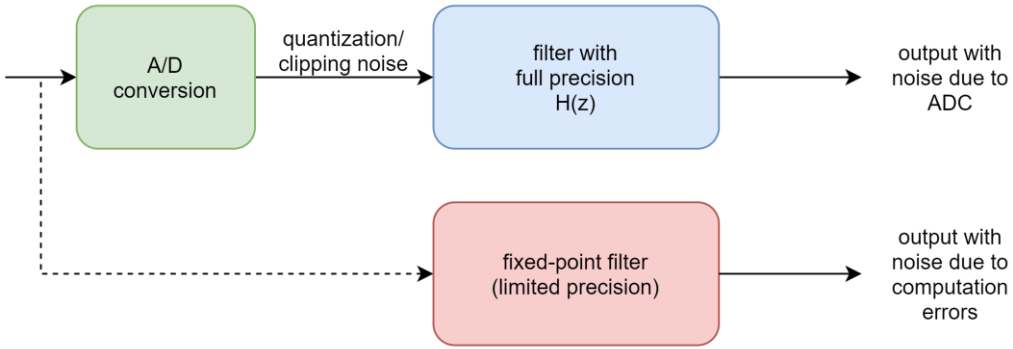


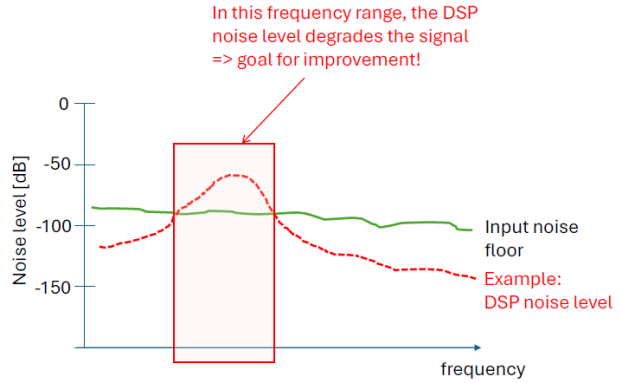
Lecture 5 Slides 26-29

1. High-order IIR filter implementation, problem with direct realization

Demonstrated in the lecture that especially the feedback coefficients can have large variation in values => for limited work length, bad quantization
SOS –structure used to avoid large variation

2. Input noise floor, how to consider in signal processing design





Lecture 5 Slide 17,
Demo: Slides 50-51

3. Word length computation for FIR output (accumulator)

Coefficients s16.15, signal 12-bit, 61 coefficients, $\sum_{k=0}^{60}|h_k| = 1.3$

Based on number of coefficients $\log_2 61 = 5.931$ => 6 guard bits => word length 16+12+6 = 34
But based on the sum 1.3 => only one guard bit => 16+12+1 = 29

1. Three-valued CORDIC

Reconsider Givens transform implementation.

The rotation decision d_i is either -1 or +1. It can increase the absolute value of the remaining angle z_i.

In **3-valued CORDIC**, some rotations can be skipped so that the value $|z_i|$ decreases monotonically.

Problem: **the gain is not constant** for specific number of effective iterations ($i: d_i \neq 0$). It must be possible to deal with varying gains.

Can be used to optimize implementations, when only some small set of rotation angles has to be implemented AND one can manage the gain compensation issue **(remember: gain depends on what elementary rotations are done)**. E.g., JPEG's DCT implementation

Implementation note: It is even possible that some elementary rotation is repeated. For example, rotation by 28 degrees obtained quite well with 14.04 degree rotations. Has gain $a_2 a_2 = 17/16$.

Table from previous lecture:

i	z_i [deg]	d_i	$\arctan(2^{-i})$ [deg]	z_{i+1} [deg]
0	+40.00	+1	45.00	−5.00
1	−5.00	−1	26.57	+21.57
2	+21.57	+1	14.04	+7.53
3	+7.53	+1	7.13	+0.40
4	+0.40	+1	3.58	−3.18
5	−3.18	−1	1.79	−1.39
6	−1.39	−1	0.90	−0.49
7	−0.49			

i	2-valued z_i [deg]	2-valued d_i	3-valued z_i [deg]	3-valued d_i	$\arctan(2^{-i})$ [deg]
0	+40.00	+1	+40.00	+1	45.00
1	−5.00	−1	−5.00	0	26.57
2	+21.57	+1	−5.00	0	14.04
3	+7.53	+1	−5.00	−1	7.13
4	+0.40	+1	+2.13	+1	3.58
5	−3.18	−1	−1.45	−1	1.79
6	−1.39	−1	+0.34	0	0.90
7	−0.49		+0.34		

$$A_N = \prod_{i=0}^{N-1} a_i,$$

$$a_i = \begin{cases} \sqrt{1 + 2^{-2i}} & \text{if } d_i = \pm 1 \\ 1 & \text{otherwise} \end{cases}$$

2. Unified CORDIC

Givens transform and polar transform represent different applications for CORDIC. In fact, the formulae for CORDIC iteration can be written in a generic form.

PRESENTED IN LECTURE 6

$$x_{i+1} = x_i - m \cdot d_i \cdot y_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + d_i \cdot x_i \cdot 2^{-i}$$
$$z_{i+1} = z_i - d_i \cdot \varepsilon_i$$

SYSTEM: Depending on the choice of m and ε_i three CORDIC **systems** can be recognized.

MODE: The target of the decision d_i made in the iteration i specifies the CORDIC **mode**:

- Rotation mode: take z_i closer to zero
- Vectoring mode: take y_i ($A_i y_i$) closer to zero

System	m	ε_i
circular	1	$\arctan(2^{-i})$
hyperbolic	-1	$\operatorname{arctanh}(2^{-i})$
linear	0	2^{-i}

Other variables provide results of interest

- Rotation mode: $A_i x_i$, $A_i y_i$
- Vectoring mode: $A_i x_i$, z_i

Unified CORDIC systems and modes

Six combinations can be derived from systems and modes.

System	Mode	
	rotation	vectoring
circular	init: (x, y, ϕ) $x_i/A_i \to x\cos\phi - y\sin\phi$ $y_i/A_i \to x\sin\phi + y\cos\phi$	init: $(x, y, 0)$ $x_i/A_i \to \sqrt{x^2 + y^2}$ $z_i \to \tan^{-1} y/x$
hyperbolic	init: (x, y, ϕ) $x_i/A_i \to x\cosh\phi + y\sinh\phi$ $y_i/A_i \to x\sinh\phi + y\cosh\phi$	init: $(x, y, 0)$ $x_i/A_i \to \sqrt{x^2 - y^2}$ $z_i \to \tanh^{-1} y/x$
linear	init: $(x, 0, z)$ $y_i \to x \cdot z$	init: $(x, y, 0)$ $z_i \to y/x$

$z_i \to$ zero $\qquad$ $y_i \to$ zero

$$x_{i+1} = x_i - m \cdot d_i \cdot y_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + d_i \cdot x_i \cdot 2^{-i}$$
$$z_{i+1} = z_i - d_i \cdot \varepsilon_i$$

System	m	ε_i
circular	1	$\arctan(2^{-i})$
hyperbolic	-1	$\operatorname{arctanh}(2^{-i})$
linear	0	2^{-i}

1. Circular (trigonometric) system

- **Givens transform** = circular system, rotation mode
- **Polar transform** = circular system, vectoring mode
- Some notable applications: signal modulation and evaluation of sine and cosine

System	Mode	
	rotation	vectoring
circular	init: (x, y, ϕ) $x_i/A_i \rightarrow x\cos\phi - y\sin\phi$ $y_i/A_i \rightarrow x\sin\phi + y\cos\phi$	init: $(x, y, 0)$ $x_i/A_i \rightarrow \sqrt{x^2 + y^2}$ $z_i \rightarrow \tan^{-1} y/x$

- signal modulation (polar-to-Cartesian mapping): in the Givens transform, let x corresponds to an incoming signal $s(n)$, and set $y = 0$. Then,

$$x'(n) = s(n)\cos\phi$$
$$y'(n) = s(n)\sin\phi \qquad (18)$$

It can be seen that x' and y' correspond to modulations of x by cosine and sine waveforms when ϕ is incremented according to a specific angular frequency ($\phi = n\omega$).

- evaluation of sine and cosine functions: if we set $x = 1/A_N$ (gain compensation factor), and $y = 0$ then we can get values of sine and cosine functions by evaluation of shift-add stages; gain compensation is done already in initialization of the algorithm.

Givens:

$$x' = x\cos\phi - y\sin\phi$$
$$y' = y\cos\phi + x\sin\phi$$

Example of doing gain compensation as a pre-processing step!

2. Hyperbolic system

- Provides means to compute many functions
  - Exponential function
    - Rotation mode
    - Initialization (1,1,α)
    - $A_i y_i \rightarrow A_i(\cosh\alpha + \sinh\alpha) = A_i e^{\alpha}$
  - Logarithm
    - Vectoring mode
    - Initialization (α+1, α-1, 0)
    - $z_i \rightarrow \operatorname{arctanh}\frac{\alpha-1}{\alpha+1} = \frac{1}{2}\ln\alpha$
  - Square root
    - Vectoring mode
    - Initialization (α+1/4, α-1/4, 0)
    - $A_i x_i \rightarrow A_i\sqrt{(\alpha+1/4)^2-(\alpha-1/4)^2} = A_i\sqrt{\alpha}$

System	Mode	
	rotation	vectoring
hyperbolic	init: (x, y, ϕ) $x_i/A_i \rightarrow x\cosh\phi + y\sinh\phi$ $y_i/A_i \rightarrow x\sinh\phi + y\cosh\phi$	init: $(x, y, 0)$ $x_i/A_i \rightarrow \sqrt{x^2 - y^2}$ $z_i \rightarrow \tanh^{-1} y/x$

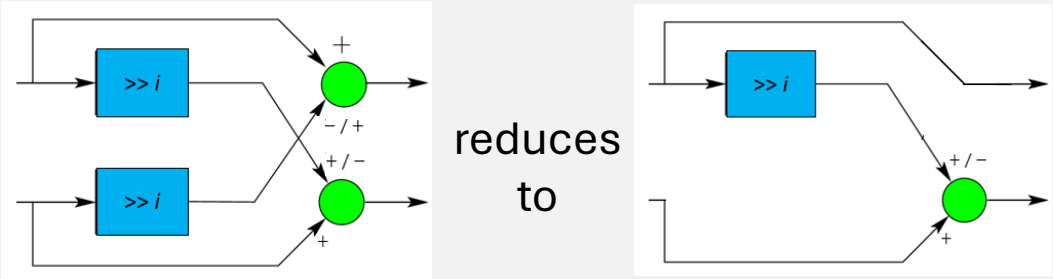
$$x_{i+1} = x_i - m \cdot d_i \cdot y_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + d_i \cdot x_i \cdot 2^{-i}$$
$$z_{i+1} = z_i - d_i \cdot \varepsilon_i$$

$$m = -1, \varepsilon_i = \operatorname{arctanh} 2^{-i}$$

See
unified_cordic_demo.xls

3. Linear system

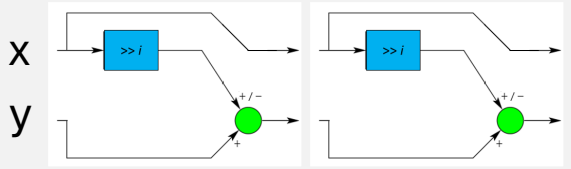
- In linear system, $m = 0$ and therefore $x_{i+1}=x_i$. Then,



$$x_{i+1} = x_i - m \cdot d_i \cdot y_i \cdot 2^{-i}$$
$$y_{i+1} = y_i + d_i \cdot x_i \cdot 2^{-i}$$
$$z_{i+1} = z_i - d_i \cdot \varepsilon_i$$

System	Mode	
	rotation	vectoring
linear	init: $(x, 0, z)$ $y_i \to x \cdot z$	init: $(x, y, 0)$ $z_i \to y/x$

- Definition says also that $\varepsilon_i = 2^{-i}$
- Consider vectoring mode with initialization (x, y, 0). Proof that implements **division** y/x:
  1. $y_N = y + x\sum_{i=0}^{N-1} d_i 2^{-i}$
  2. $z_N = -\sum_{i=0}^{N-1} d_i 2^{-i}$
  3. Substitute 2 to 1 => $y_N = y - xz_N$
  4. $z_N = \frac{y-y_N}{x} \to \frac{y}{x}$ as $y_N \to 0$



% Shifts of x added to/subtracted from y
% z initialized to zero

% Vectoring mode takes y_N closer to zero

3. Implementing functions in DSP

Alternatives

- **approximating functions** which are derived from Taylor series and other expansions. Examples:

$$
\begin{aligned}
e^x &= 1 + x/1! + x^2/2! + x^3/3! + \ldots \\
\ln x &= 2(z + z^3/3 + z^5/5 + z^7/7 + \ldots) \text{ where } z = (x-1)/(x+1)
\end{aligned}
$$

- **convergence computation** of recursive equation systems: basically two recursive formulas, one formula converges to a constant, the other to the result. Example: if $q^{(0)}$ is an approximation of $\sqrt{x}$, it can refined by evaluation of

$$q^{(i+1)} = 0.5(q^{(i)} + x/q^{(i)})$$

https://www.fpgarelated.com/showarticle/1347.php

where i is the iteration index; $q^{(i+1)} - q^{(i)}$ converges to zero.

- **lookup tables** and interpolation
- **coordinate rotation** (CORDIC)

Example: sine computation (1)

Assume that first four terms of Taylor expansion provide sufficient approximation

$$sin(x) = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7$$

Requires 7 multiplications and 3 additions.

- multiplications x2 = x*x, x3 = x2*x, x5 = x2*x3, x7 = x2*x5, -1/3!*x3, 1/5!*x5, -1/7!*x7

Multiplications can be reduced by applying **Horner's method:**

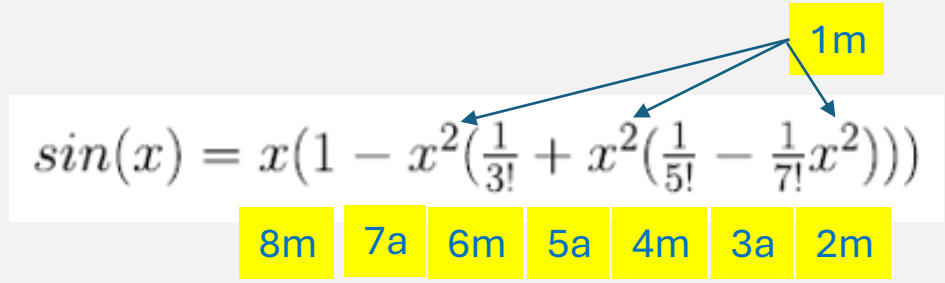
$$\begin{aligned} &a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n \\ &\quad = a_0 + x\Big(a_1 + x\Big(a_2 + x\big(a_3 + \cdots + x(a_{n-1} + x\,a_n)\cdots\big)\Big)\Big) \end{aligned}$$

We can write

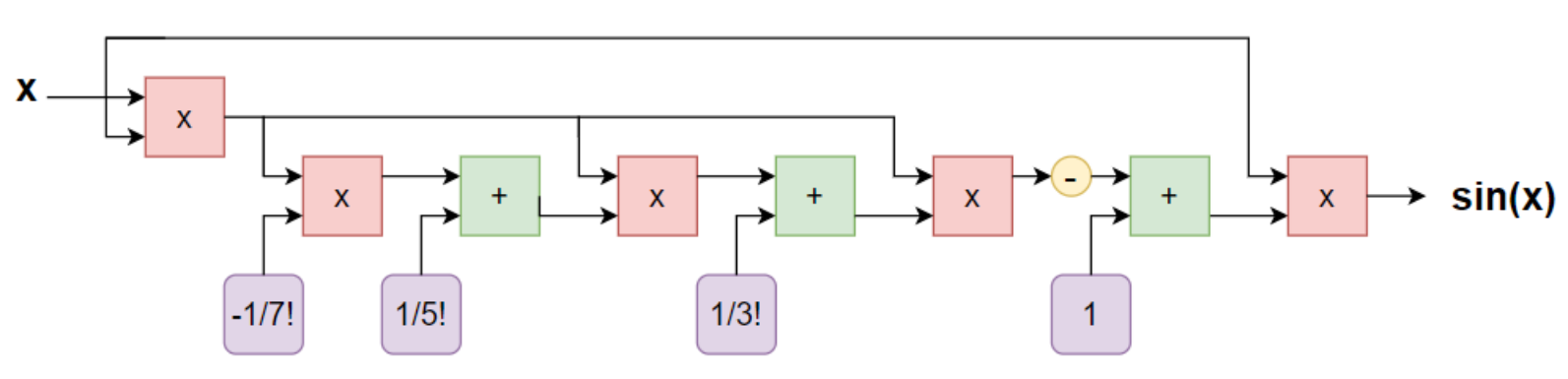
$$sin(x) = x(1 - x^2(\frac{1}{3!} + x^2(\frac{1}{5!} - \frac{1}{7!}x^2)))$$

Requires 5 multiplications and 3 additions.

8 steps in computation



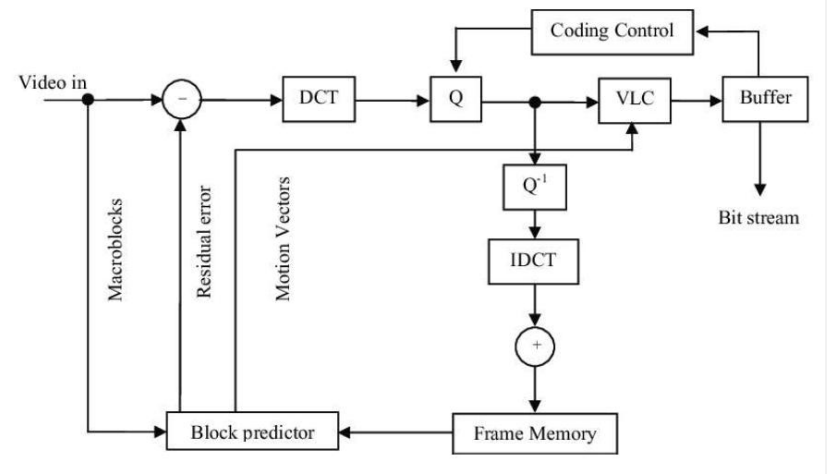
Example: sine computation (2)



- Staged computation: multiplications and additions put into pipeline stages
- Single-instruction-multiple-data (SIMD): vectorized computation of multiple sine values at the same time
- A rule of thumb in scalar software implementation: use hardware multiplier if it is available and series expansion of a function
- Benefits of CORDIC best achieved in hardware/SIMD implementation

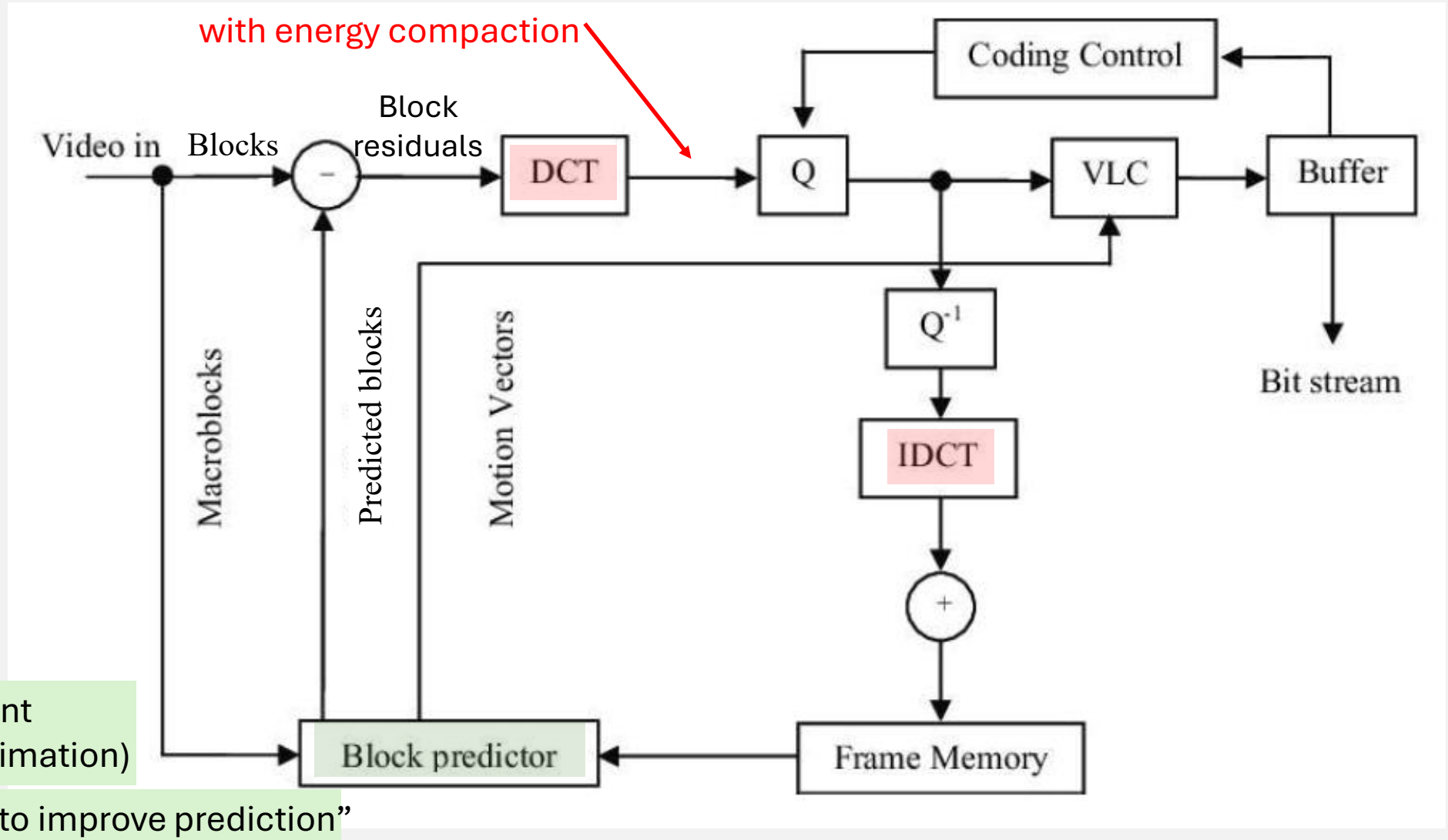
4. Discrete Cosine Transform (DCT)

- DCT is very common in **image and video coding** standards such as JPEG, MPEG-2, H.264 and HEVC.
- DCT is also the technique under most standardized **audio codecs** such as MP3 and AAC, and open source contributions like Ogg Vorbis.
- The general idea behind using transform based coding is signal **energy compaction** into a small number coefficients, which are **quantized separately** and using some form of **entropy coding**.
- Quantization control
  - Important coefficient => more bits => less quantization noise
  - Adapt to available space in output bit stream
- Trying to achieve best quality with the available bits



Another issue: How to minimize the number of additions/multiplications needed?
Many blocks to consider in codecs, focusing on DCT in the following.

Transform based motion compensated video encoder



Prediction of block content
(utilizes video motion estimation)

”compensate for motion to improve prediction”

Definitions of DCT

- In literature, eight different DCT types are recognized
  - DCT types I-IV can be derived for real even DFTs of even order
  - Other four can be derived for real even DFTs of odd order
- In practice, only few of these types are important
- In the following, we focus on **DCT-II**, which is used in JPEG coding and which is also popular in video codecs
- We can represent DCT transforms as matrix-vector multiplications
- In the case of DCT-II,

$$y_k = \sum_{n=0}^{N-1} x_n \alpha_k \cos\left(\frac{\pi}{2N}(2n+1)k\right)$$

where α_k are scaling factors. Typically, they are chosen so that the transform becomes orthogonal i.e. $\mathbf{Q}^{-1} = \mathbf{Q}^T$.

$$\mathbf{y} = \mathbf{Q}\mathbf{x}$$

$$\mathbf{Q} = \begin{bmatrix} Q_{0,0} & Q_{0,1} & \cdots & Q_{0,N-1} \\ Q_{1,0} & Q_{1,1} & \cdots & Q_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{N-1,0} & Q_{N-1,1} & \cdots & Q_{N-1,N-1} \end{bmatrix}$$

DCT-II :

$$Q_{k,n} = \alpha_k \cos\left(\frac{\pi}{N}(n+\frac{1}{2})k\right)$$

DCT-II and image coding

2-D DCT

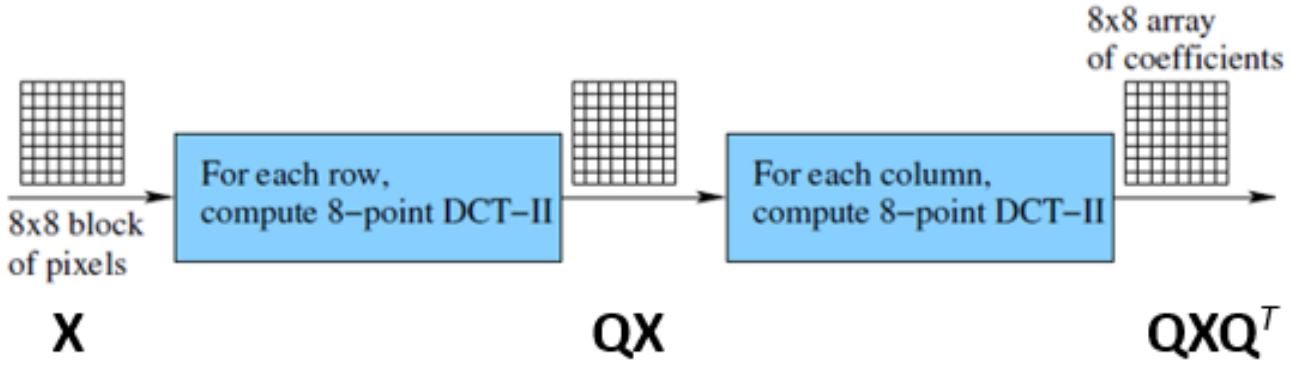
2-D discrete transforms are important in image processing and coding applications. Using matrix notation, two-dimensional forward DCT (2-D DCT) can be defined as

$$\mathbf{Y} = \mathbf{Q}\mathbf{X}\mathbf{Q}^T \tag{8}$$

where $\mathbf{X}$ denotes $N \times N$ data matrix, and $\mathbf{Y}$ is the $N \times N$ coefficient matrix. Note that this is a *separable* transform, which means that Y can be obtained by performing 1-D DCT for each column vector of $\mathbf{X}$ first, and then computing 1-D DCT of each row vector of the result. Thus, if we have an efficient solution for computing 1-D DCT, it can be readily applied to computation of 2-D DCT.

Assuming that $\mathbf{Q}$ is orthogonal, the inverse transform (2-D IDCT) is

$$\mathbf{X} = \mathbf{Q}^T\mathbf{Y}\mathbf{Q}. \tag{9}$$



Basis images

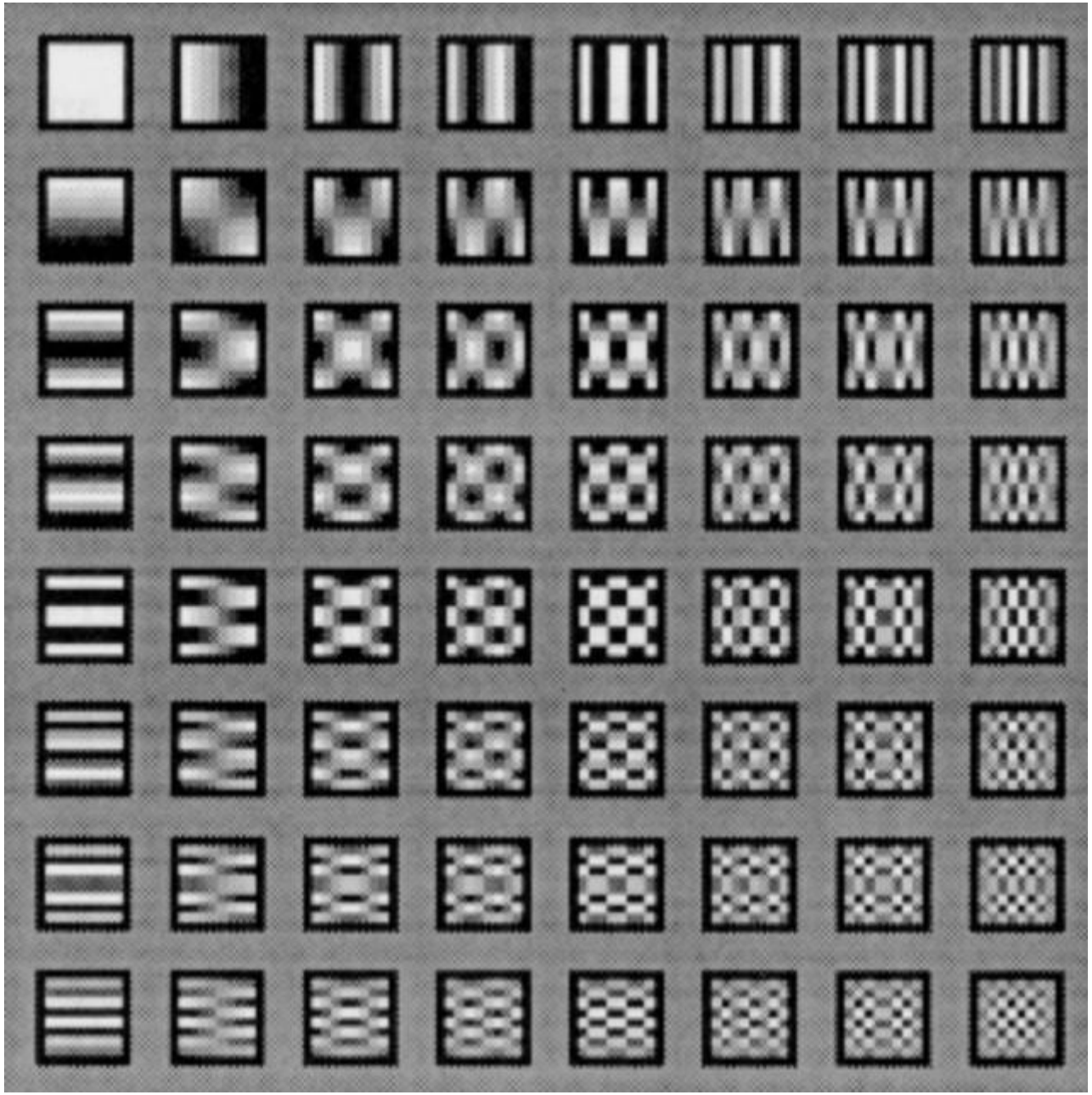
8-bit 2-D DCT represents the encoded image in terms of these 64 basis images.

These images can be obtained by multiplying columns of $\mathbf{Q}$ and rows of $\mathbf{Q}^T$.

Element X_{ij} of matrix $\mathbf{Q}$ is weighting the image formed by multiplication

$$\mathbf{column}(\mathbf{Q}, i) \times \mathbf{row}(\mathbf{Q}^T, j)$$

We see horizontal variation change in rows and vertical variation change in columns.



Source: https://users.cs.cf.ac.uk/dave/Multimedia/node231.html

DCT-II and image coding

Support on using DCT
(from natural image statistics)

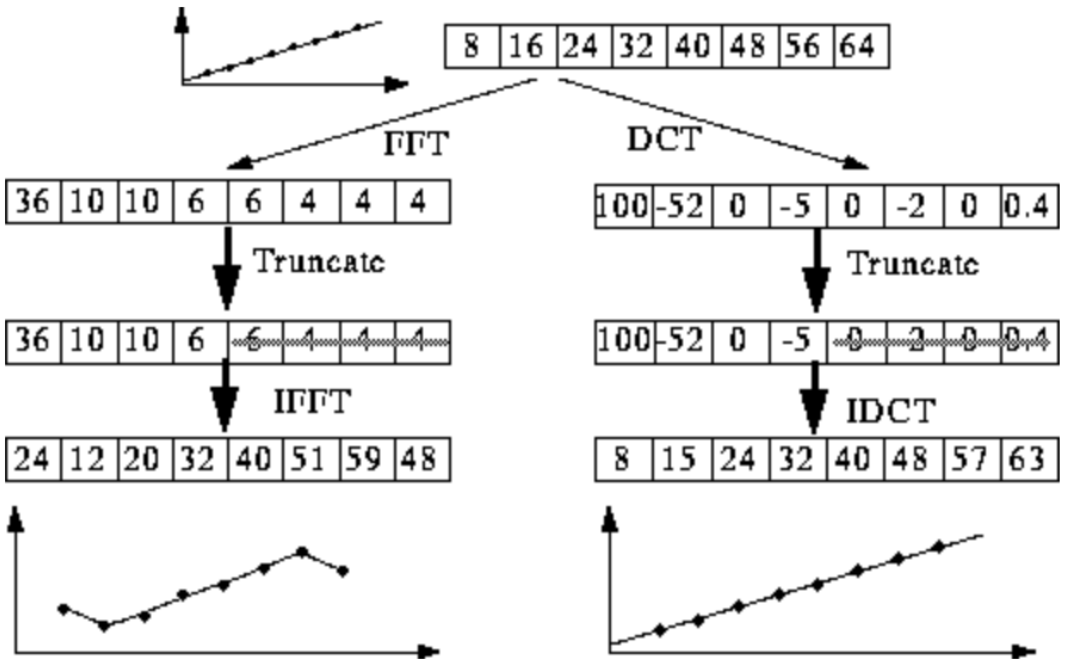
The optimal transform for an N-point random vector $\mathbf{X}$ having the data covariance matrix $\psi = E\{\mathbf{X}\mathbf{X}^T\}$ is the Karhunen-Loéve transform (KLT) whose basis vectors are the eigenvectors of ψ. However, determination of those basis vectors is computationally complex. It has been observed that the covariance matrix

$$\psi = \begin{bmatrix} 1 & \rho & \rho^2 & \cdots & \rho^{N-1} \\ \rho & 1 & \rho & \cdots & \rho^{N-2} \\ \rho^2 & \rho & 1 & \cdots & \rho^{N-3} \\ \vdots & & & \ddots & \\ \rho^{N-1} & \rho^{N-2} & \rho^{N-3} & \cdots & 1 \end{bmatrix}$$

is a relatively good model for the statistical dependency of neighboring pixels in natural images. Because its eigenvectors are approximated relatively well with the DCT-II basis vectors (Ahmed *et al.* 1974), the DCT-II provides a good deterministic transformation for energy compaction.

- Why DCT not FFT?

  DCT is similar to the Fast Fourier Transform (FFT), but can approximate lines well with fewer coefficients



DCT/FFT Comparison

Source: https://users.cs.cf.ac.uk/dave/Multimedia/node231.html

Computing DCT

$$X(i) = c(i) \sum_{k=0}^{3} x(k) \cos\left(\frac{i(k+\frac{1}{2})\pi}{N}\right)$$

- In practice, efficient structures for computing the DCT must be used
- Before considering practically important 8-point DCT-II, let us take a look at some ideas using 4-point DCT-II. Matrix representation of it on the right.
  - We see that this computation requires 16 multiplications and 12 additions

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \cos\frac{\pi}{8} & \cos\frac{3\pi}{8} & \cos\frac{5\pi}{8} & \cos\frac{7\pi}{8} \\ \cos\frac{2\pi}{8} & \cos\frac{6\pi}{8} & \cos\frac{10\pi}{8} & \cos\frac{14\pi}{8} \\ \cos\frac{3\pi}{8} & \cos\frac{9\pi}{8} & \cos\frac{15\pi}{8} & \cos\frac{21\pi}{8} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

- Considering various equalities of cosine values in the matrix, we can write it in the form, where only four values (and their negations) appear in the matrix
  - It is possible to reorganize computations so that the number of multiplications is reduced
  - E.g. X(0) could be computed using just one multiplication

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} a(2) & a(2) & a(2) & a(2) \\ a(1) & a(3) & -a(3) & -a(1) \\ a(2) & -a(2) & -a(2) & a(2) \\ a(3) & -a(1) & a(1) & -a(3) \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

4-point DCT: Reducing operation count

- Step 1: doing additions/subtractions before multiplications
  - 6 multiplications and 12 additions needed
  - Note: subtractions considered as additions (similar computational complexity)

- Step 2: computing some pairwise additions and sharing the results
  - 6 multiplications and 8 additions are needed

$$X(0) = [x(0) + x(1) + x(2) + x(3)] \cdot a(2)$$
$$X(1) = [x(0) - x(3)] \cdot a(1) + [x(1) - x(2)] \cdot a(3)$$
$$X(2) = [x(0) + x(3) - (x(1) + x(2))] \cdot a(2)$$
$$X(3) = [x(0) - x(3)] \cdot a(3) - [x(1) - x(2)] \cdot a(1)$$

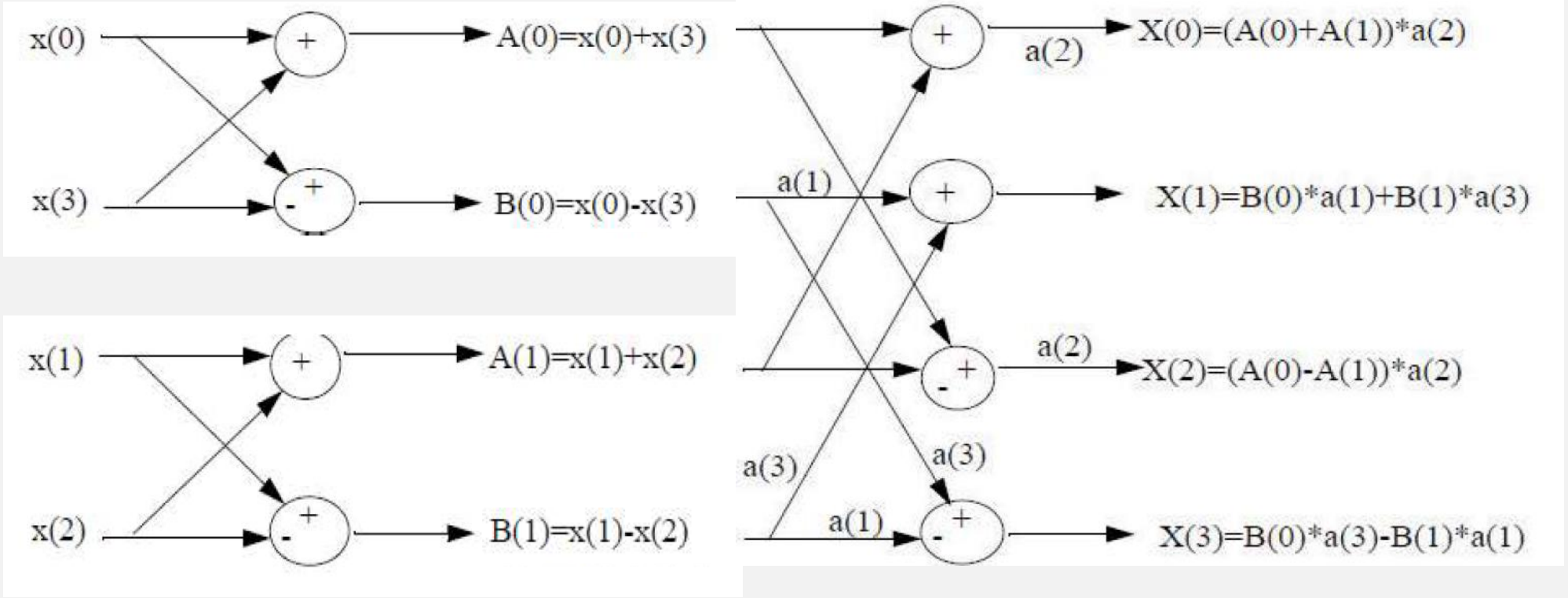
$$X(0) = [A(0) + A(1)] \cdot a(2)$$
$$X(1) = B(0) \cdot a(1) + B(1) \cdot a(3)$$
$$X(2) = [A(0) - A(1)] \cdot a(2)$$
$$X(3) = B(0) \cdot a(3) - B(1) \cdot a(1)$$

where

$$A(0) = x(0) + x(3)$$
$$A(1) = x(1) + x(2)$$
$$B(0) = x(0) - x(3)$$
$$B(1) = x(1) - x(2)$$

Butterfly diagrams

The result can be summarized using signal flow graphs.



$$a(1) = \cos\frac{\pi}{8} = 0.9239, a(2) = \frac{1}{\sqrt{2}} = 0.7071, a(3) = \cos\frac{3\pi}{8} = 0.3827$$

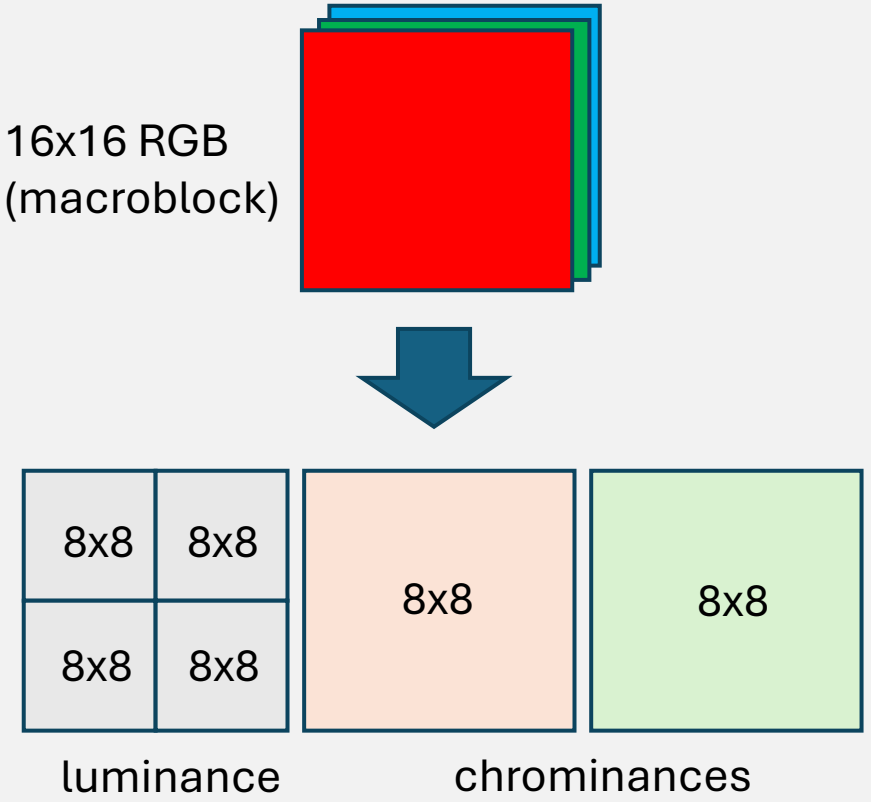
DCT: applications & implementation

8-point DCT-II

- Computation based on direct matrix multiplication requires 64 multiplications and 56 additions

$$\mathbf{y} = \mathbf{Q}\mathbf{x}$$

- H.263 case: VGA video 640x480, 30 frames/s, 8x8 blocks, 4:2:0 sampling (chroma components sampled at half rate)
- Computational complexity analysis
  - 4 luminance and 2 chroma blocks per 16x16 macroblock
  - 1200 macroblocks in VGA frame
  - 16 DCT transforms per 8x8 block (separable 2-D DCT => 8x8)
  - 30 x 1200 x 6 x 16 x [64 + 56] ops / s = 415 Mops
  - **In practice, too large value**
- Several computationally optimized algorithms have been proposed to cut the computation time or/and gate counts needed for implementation
- The algorithm selection depends on the availability of the multipliers and registers on the implementation platform



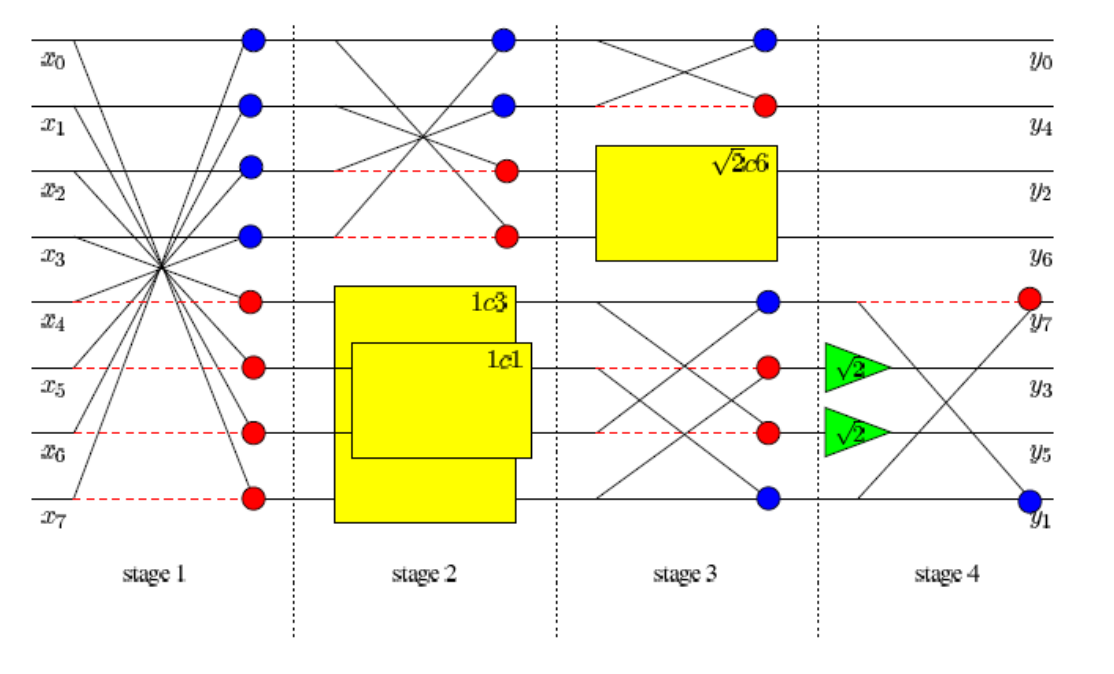
The best known fast algorithms:

* **Chen**'s algorithm : 16 multiplications and 26 additions.
* **Loeffler**'s algorithm: 11 multiplications and 29 additions

How much savings with Chen’s algorithm

- With DCT implemented using matrix multiplication
  - 30 frames/sec x 1200 Mblocks/frame x 6 (blocks/Mblock) x 16 dct/block x [64 mply + 56 add] ops/dct = 415 Mops
- With Chen
  - 30 frames/sec x 1200 Mblocks/sec x 6 (blocks/Mblock) x 16 dct/block x [16 mply + 26 add] ops/dct = 145 Mops (35%)

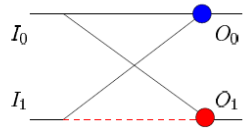
Loeffler's structure



Complexity: 11 multiplications and 29 additions

Elementary operations in the signal flow graph

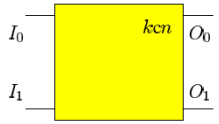
- *butterflies* both add and subtract their inputs:



$$O_0 = I_0 + I_1$$
$$O_1 = I_0 - I_1$$

Implementation of this operation requires 2 additions

- *rotators* perform Givens transforms for scaled inputs:



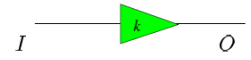
$$O_0 = I_0 k \cos(\pi n/2N) + I_1 k \sin(\pi n/2N)$$
$$O_1 = -I_0 k \sin(\pi n/2N) + I_1 k \cos(\pi n/2N)$$

Direct implementation of this operation with multipliers requires 3 multiplications and 3 additions, because

$$\begin{aligned} O_0 &= aI_0 + bI_1 = a(I_0 + I_1) + (b - a)I_1 \\ O_1 &= -bI_0 + aI_1 = a(I_0 + I_1) - (a + b)I_0 \end{aligned}$$

where $a = k\cos(\pi n/2N)$ and $b = k\sin(\pi n/2N)$. Note that a, $(b - a)$ and $(a + b)$ are all constants. Note also that there is no scaling if $k = 1$;

- *gain blocks* scale their input,

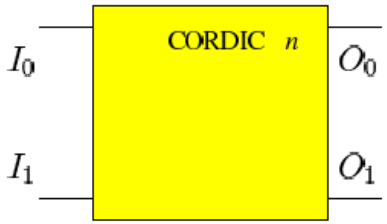


$$O = I\,k$$

and require one multiplication.

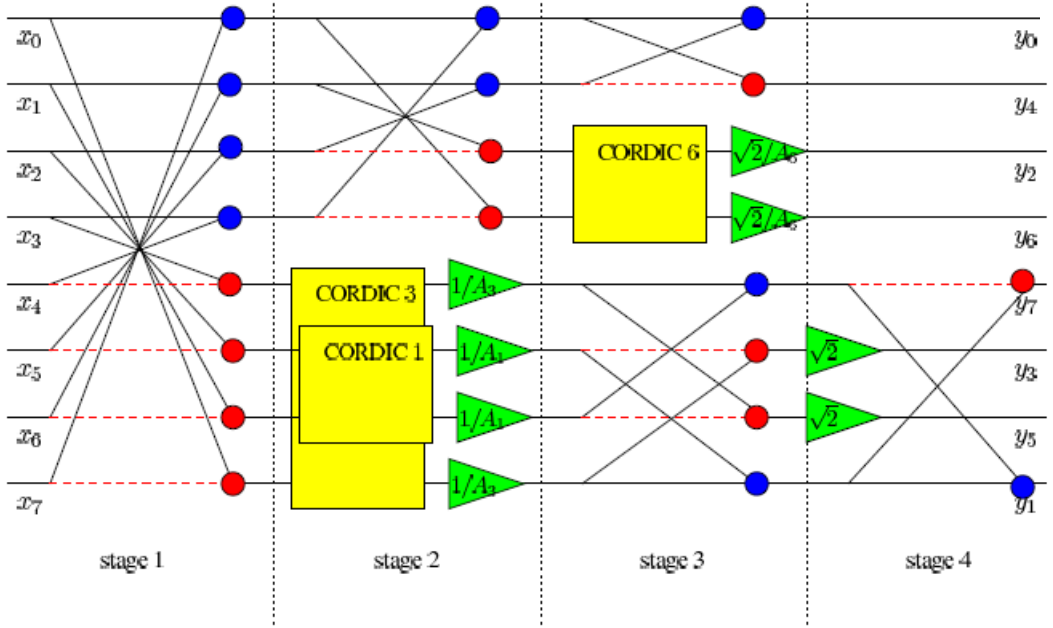
Embedding CORDIC

Rotators can be substituted by CORDIC shift-add blocks. For gain compensation, six multipliers must be added.



$$O_0 = I_0 A \cos(-\pi n/2N) - I_1 A \sin(-\pi n/2N)$$
$$O_1 = I_0 A \sin(-\pi n/2N) + I_1 A \cos(-\pi n/2N)$$

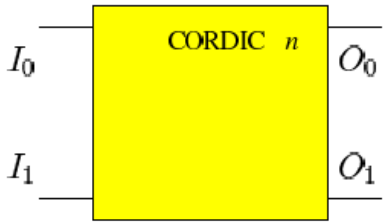
A denotes the gain of the CORDIC shift-add chain



Complexity: 8 multiplications, 20 additions + CORDIC blocks (it is possible to use 3-valued CORDIC)
- some reduction

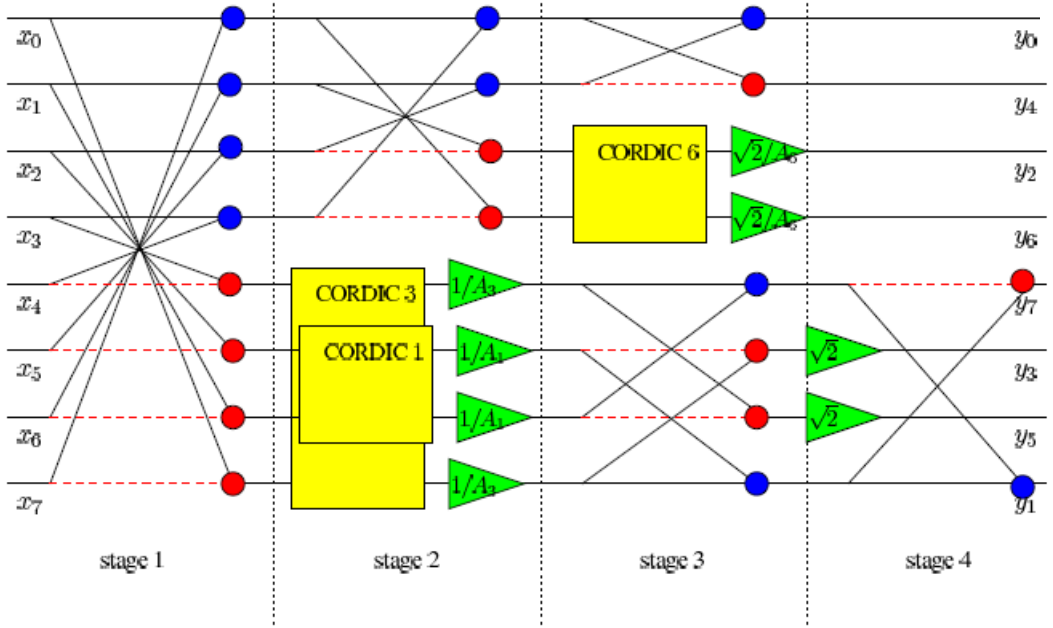
Embedding CORDIC

Rotators can be substituted by CORDIC shift-add blocks. For gain compensation, six multipliers must be added.



$$O_0 = I_0 A \cos(-\pi n/2N) - I_1 A \sin(-\pi n/2N)$$
$$O_1 = I_0 A \sin(-\pi n/2N) + I_1 A \cos(-\pi n/2N)$$

A denotes the gain of the CORDIC shift-add chain

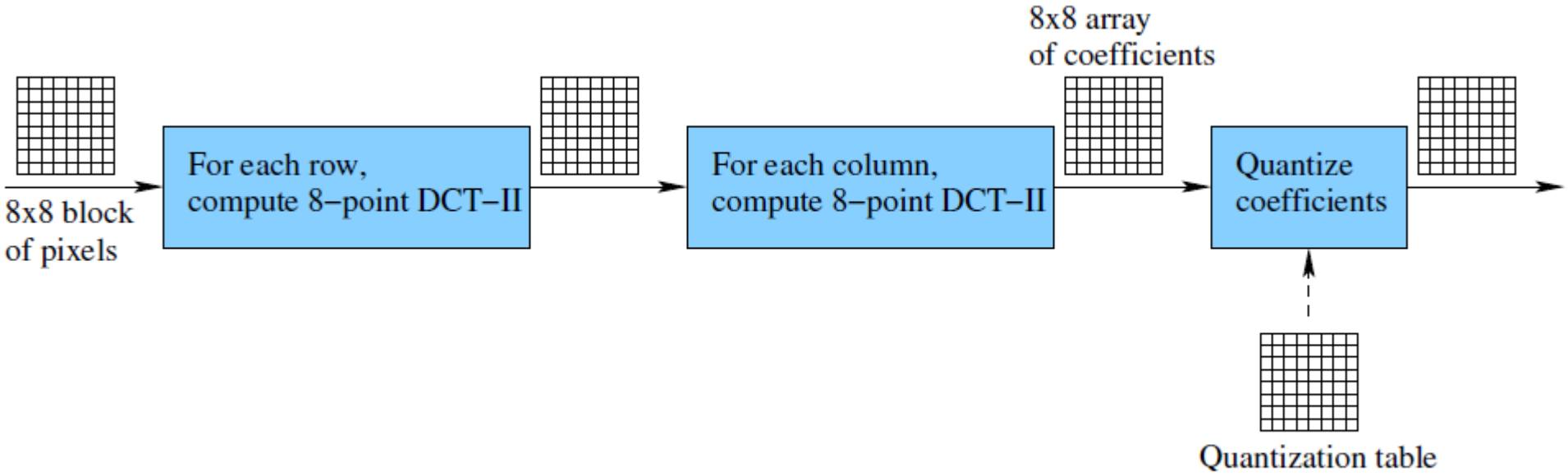


Complexity: 8 multiplications, 20 additions + CORDIC blocks (it is possible to use 3-valued CORDIC)
- some reduction

It's interesting to consider removal of gain blocks  from this. Then, some outputs are scaled. Is it possible to get along with such outputs? Considering JPEG ...

JPEG coding

In JPEG coding, a quantization table containing multipliers for each coefficient is used to control the quantization quality.



*Quantization in JPEG compression

https://dev.to/marycheung021213/understanding-dct-and-quantization-in-jpeg-compression-1col

```
# Define quantization matrix
q_mat=np.array([[16,11,10,16,24,40,51,61],
                [12,12,14,19,26,58,60,55],
                [14,13,16,24,40,57,69,56],
                [14,17,22,29,51,87,80,62],
                [18,22,37,56,68,109,103,77],
                [24,35,55,64,81,104,113,92],
                [49,64,78,87,103,121,120,101],
                [72,92,95,98,112,100,103,99]])

# Quantization
def Compress_2(img,N):
    img_dct=np.zeros((img.shape[0]//N*N,img.shape[1]//N*N))
    for m in range(0,img_dct.shape[0],N):
        for n in range(0,img_dct.shape[1],N):
            block=img[m:m+N,n:n+N]
            # DCT
            coeff=cv2.dct(block)
            # Quantization
            q_coeff=np.around(coeff*255/q_mat)
```

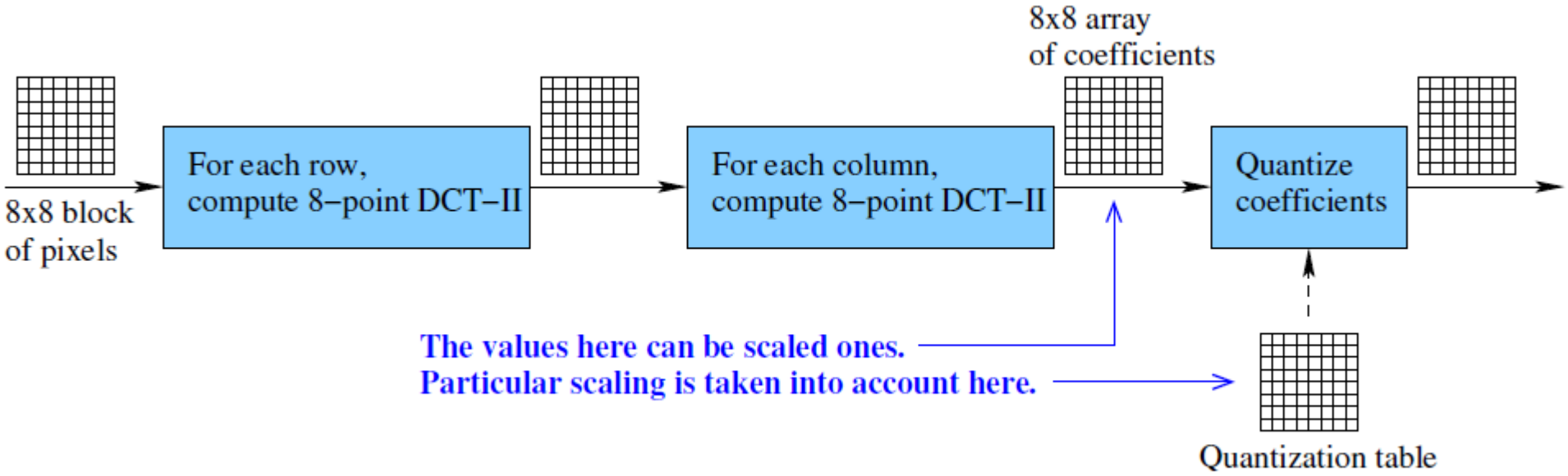
The larger the coefficient in the array
the less precision for corresponding coefficients

The code on the left uses division, but
in more efficient implementation we
would use multiplication, of course.
Perhaps try to use shift-add logic.

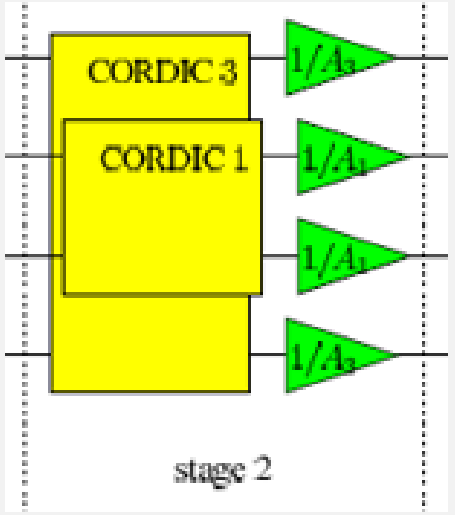
JPEG coding

In JPEG coding, a quantization table containing multipliers for each coefficient is used to control the quantization quality.

We can modify those multipliers so that the varying scaling involved in the DCT output is compensated away!

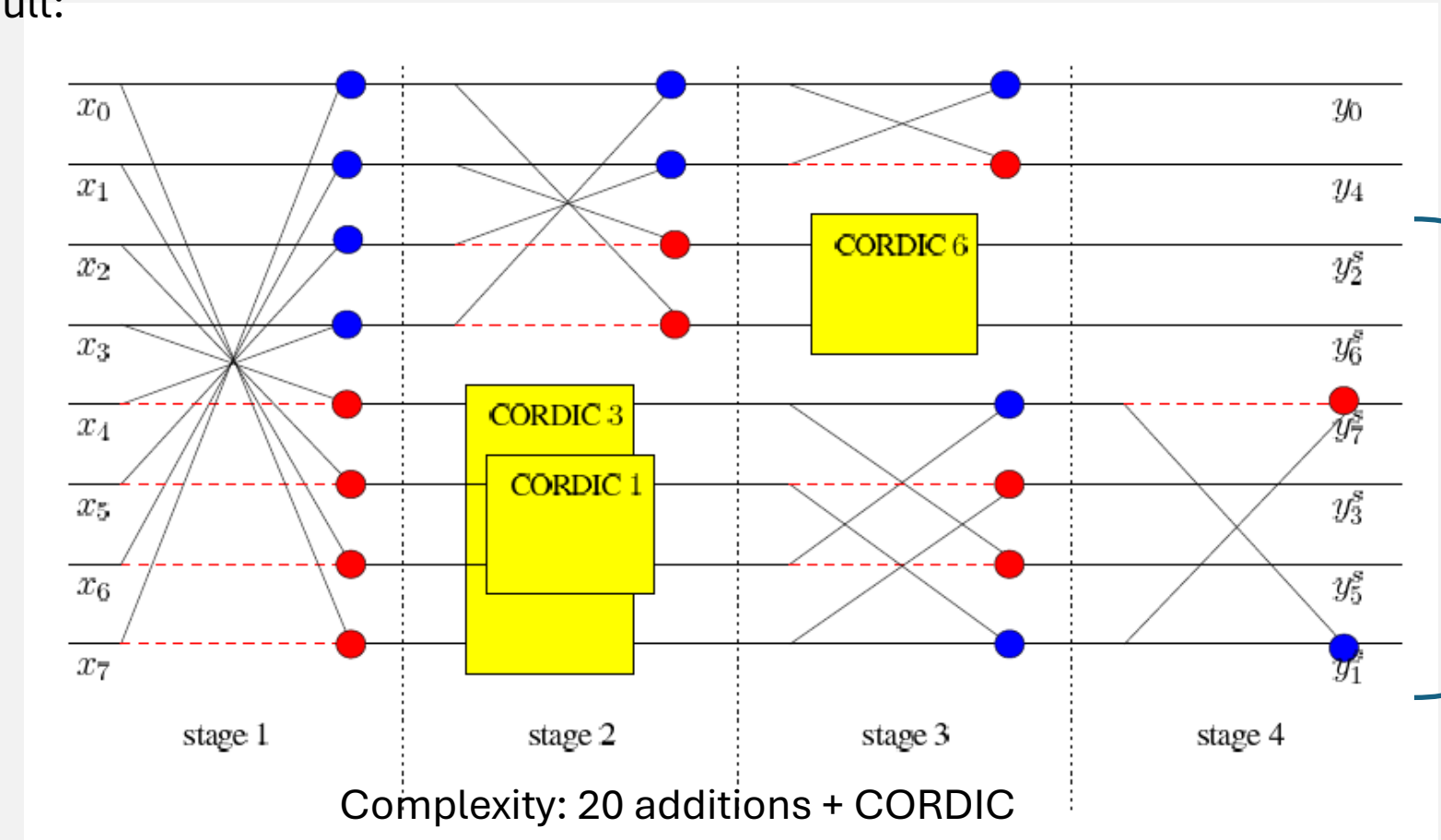


Result: Simplified Loeffler (for JPEG)



The gains of stage 2 blocks CORDIC 1 and CORDIC 3 must be equal ($A_1 = A_3$) in order to take the associated gain compensations out.

Result:



Scaled outputs

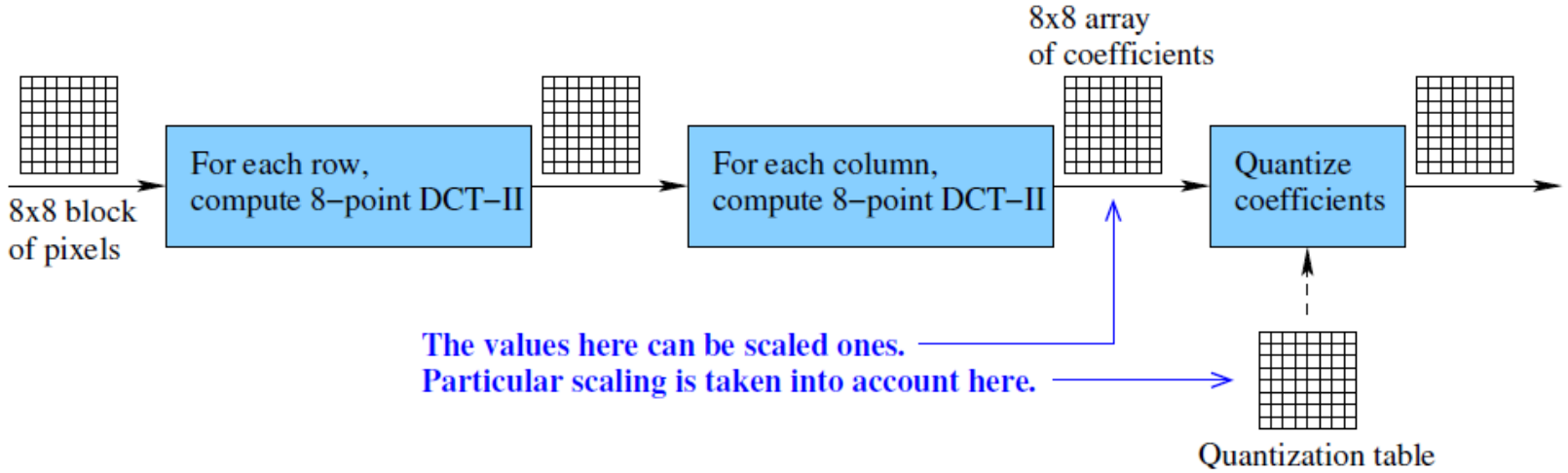
$$y_1^s = A_3 y_1$$

e.g, $y_2^s = \frac{A_6}{\sqrt{2}} y_2$

$$y_3^s = \frac{A_3}{\sqrt{2}} y_3$$

Complexity: 20 additions + CORDIC

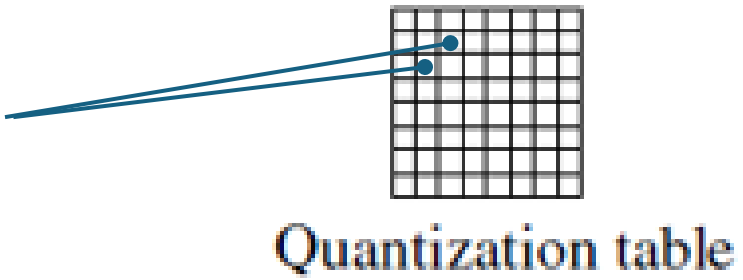
JPEG coding: taking into account scaling



$$y_1^s = A_3 y_1$$

$$y_2^s = \frac{A_6}{\sqrt{2}} y_2$$

Scale multiplier (1,2) and (2,1) by $\frac{\sqrt{2}}{A_3 A_6}$



More info on JPEG

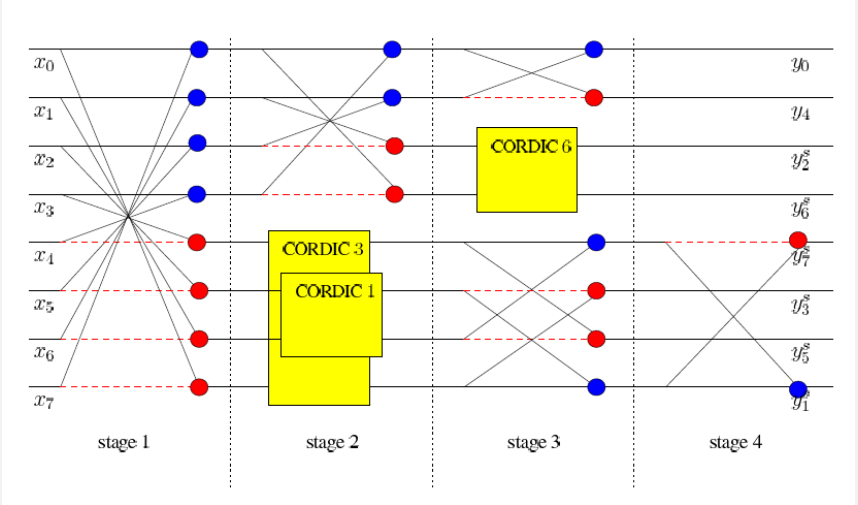
- https://parametric.press/issue-01/unraveling-the-jpeg/

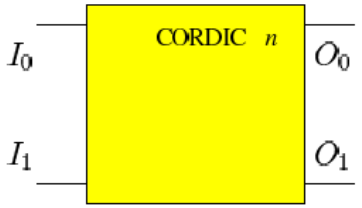
The three layers of JPEG compression

1. Chrominance Subsampling
2. Discrete Cosine Transform & Quantization
3. Run-Length, Delta & Huffman Encoding

DT3 Problem 3

- Considers implementation of CORDIC 1/3/6 blocks
- Try to find a short rotation sequence using 3-valued CORDIC
- Rotations:
  - CORDIC 1 = -11.25 degrees
  - CORDIC 3 = -33.75 degrees
  - CORDIC 6 = -67.50 degrees





$$O_0 = I_0 A\cos(-\pi n/2N) - I_1 A\sin(-\pi n/2N)$$
$$O_1 = I_0 A\sin(-\pi n/2N) + I_1 A\cos(-\pi n/2N)$$

A denotes the gain of the CORDIC shift-add chain

$N = 8$

5. Modified DCT (MDCT)

- Transform which is used in audio codecs
- MDCT and its inverse are defined as follows

In modified DCT (MDCT), N samples $\{x_0, x_1, \ldots, x_{N-1}\}$ are mapped to $N/2$ real-valued transform coefficients

$$y_k = (2/N) \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi}{2N}(2n + 1 + N/2)(2k + 1)\right) \tag{10}$$

where $k = 0, 1, \ldots, (N/2 - 1)$.

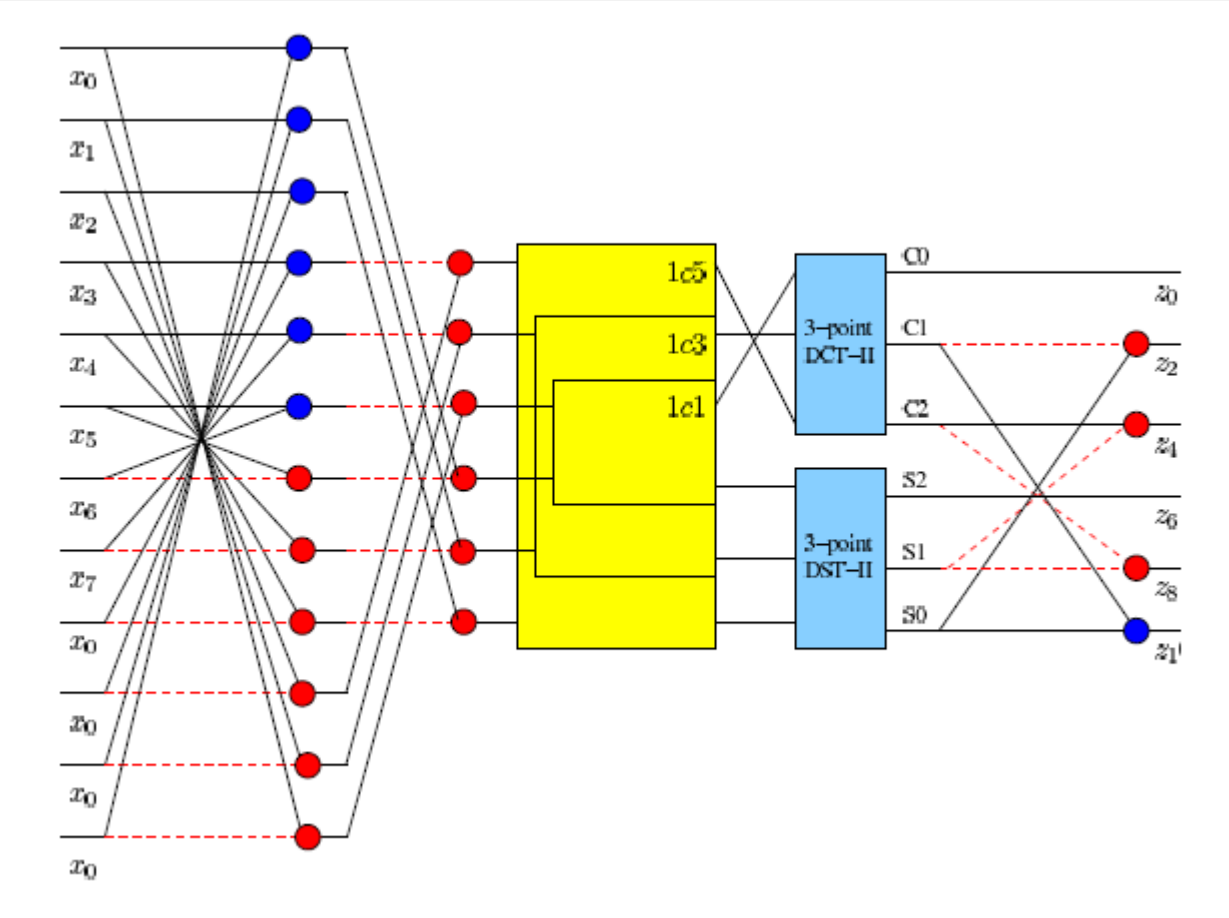
In inverse MDCT (IMDCT), N/2 transform coefficients are mapped to N samples $\hat{x}_n$ according to

$$\hat{x}_n = \sum_{k=0}^{N/2-1} y_k \cos\left(\frac{\pi}{2N}(2n + 1 + N/2)(2k + 1)\right), \tag{11}$$

where $n = 0, 1, \ldots, (N - 1)$.

Structures for MDCT

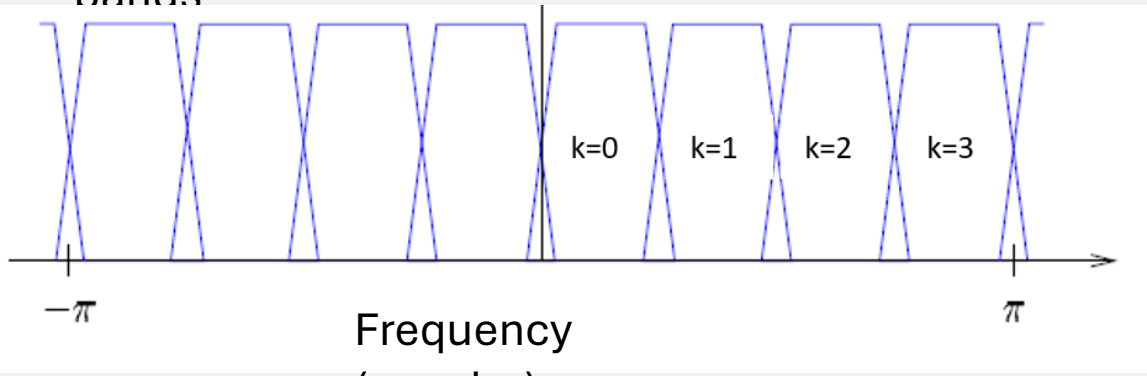
- Common implementations have been based on FFT algorithms and DCT/DST of type IV
- Britanak & Rao (2002) provide the structure on the right for 12-point MDCT
- Again, CORDIC is a possible implementation technique for rotator blocks



MDCT is a filter bank

$$y_k = (2/N) \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi}{2N}(2n + 1 + N/2)(2k + 1)\right)$$

Filter bank: output coefficients y_k correspond to particular frequency bands



- For N input samples there are only N/2 output coefficients
  - The input x_n cannot be recovered
- This is solved by using overlapping analysis windows, so-called time-domain alias cancellation (TDAC)

Time-domain Alias Cancellation (TDAC)

The MDCT can be applied to a stream of signal samples so that perfect reconstruction of the original signal is possible for unquantized transform coefficients. The overall procedure called *time-domain alias cancellation (TDAC)* is illustrated in Fig. 4.

At the encoder side, signal samples are grouped to overlapped segments, and data within each window is properly weighted. So, if u_n $(n = 0, 1, \ldots, N-1)$ represents an input sample within a segment, corresponding x_n in the definition (10) is computed using

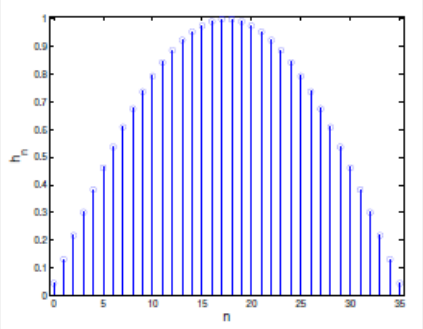
$$x_n = g_n u_n \tag{12}$$

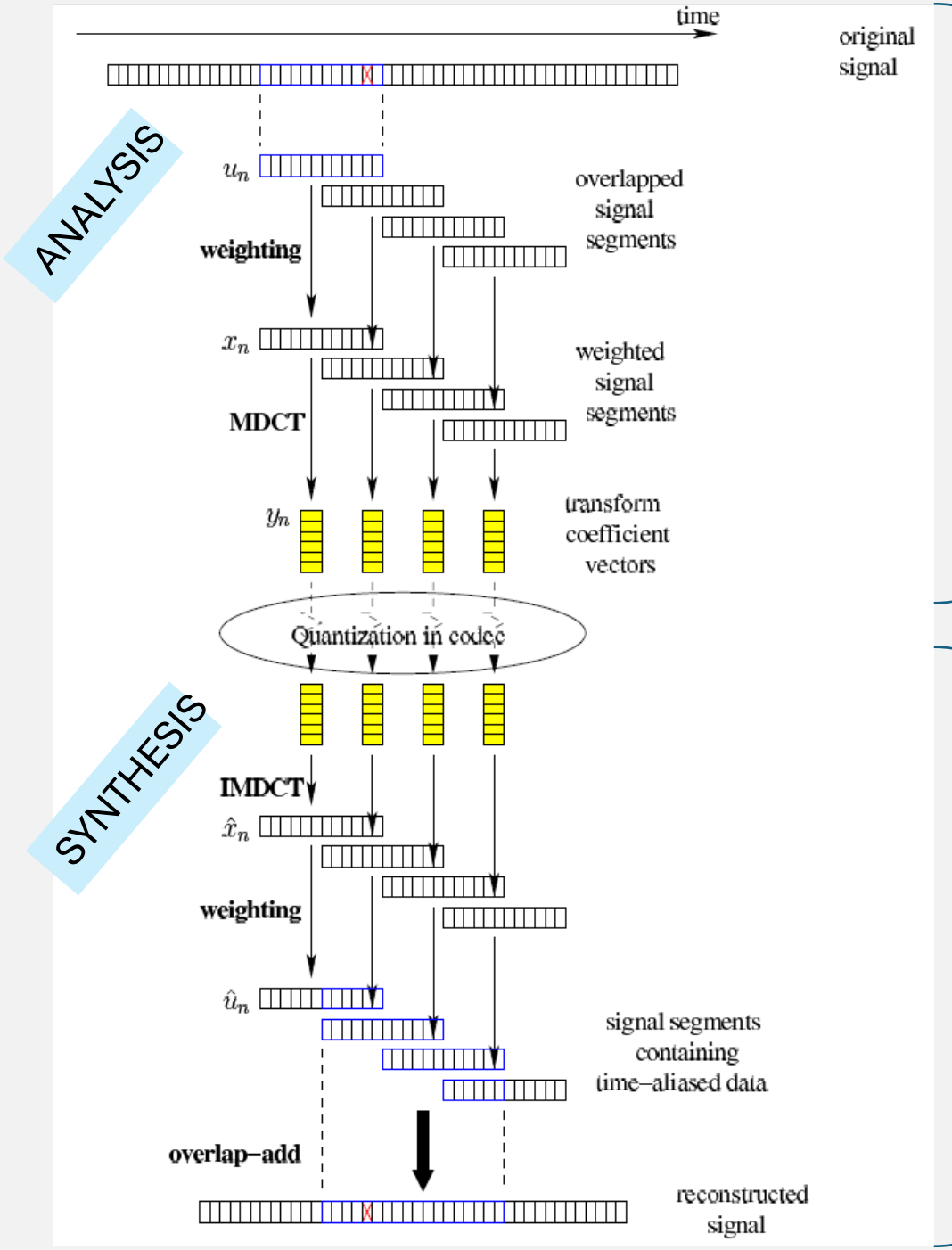
where g_n $(n = 0, 1, \ldots, N-1)$ are the analysis window coefficients. At the decoder size, outputs of the IMDCT are weighted according to

$$\hat{u}_n = h_n \hat{x}_n \tag{13}$$

where h_n $(n = 0, 1, \ldots, N-1)$ are the synthesis window coefficients. Then, adding overlapping samples $\hat{u}_n$ gives reconstruction of the original signal.

Example of sine-based weighting used in MP3 standard (N = 36)

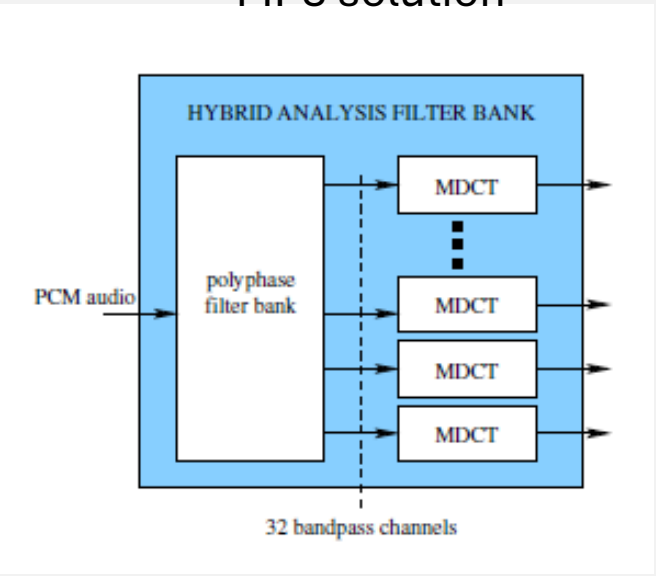


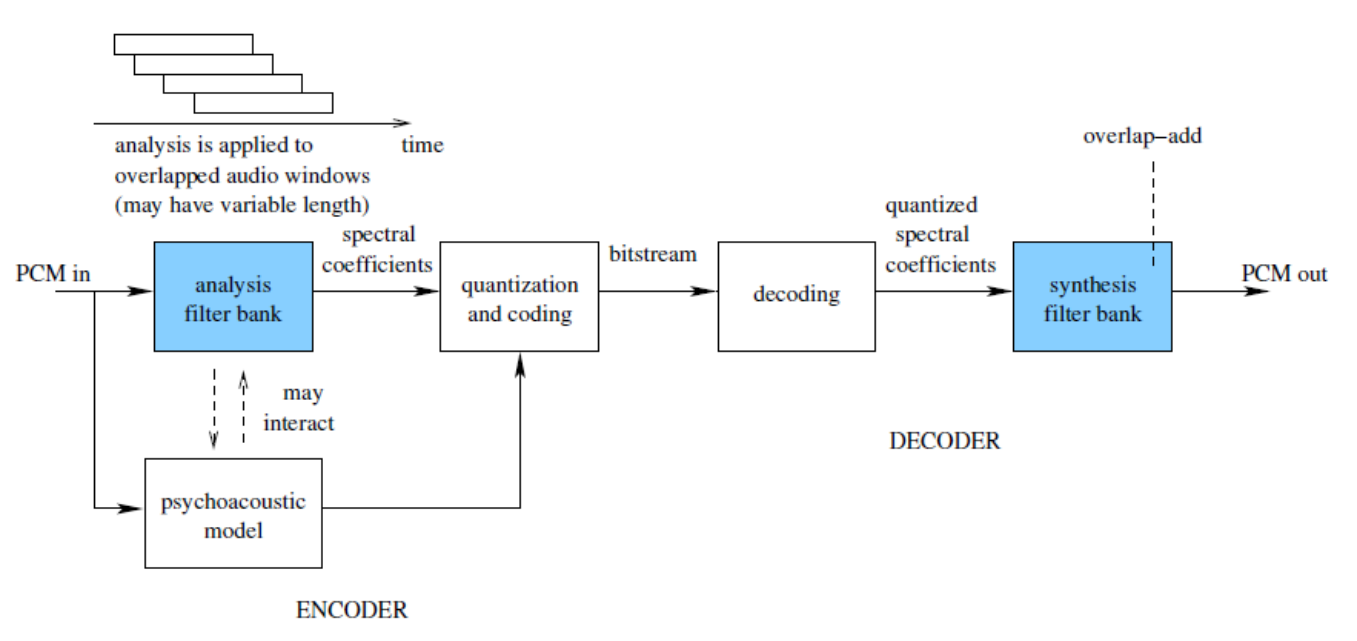


Components of an audio codec

- MDCT and IMDCT can be used to implement **analysis and synthesis filter banks**
  - There are several kinds of filter banks that can be used
  - For example, MP3 uses a hybrid of polyphase and MDCT filter banks
  - AAC and Ogg Vorbis use only MDCT for spectral decomposition
- In parallel with analysis filter bank runs a **psychoacoustic model**
  - Its purpose is to model human's capability for hearing
  - For example, strong amplitude at certain frequencies can mask nearby frequencies
  - Controls MDCT coefficient quantization

MP3 solution





Summary

- CORDIC is applicable to fixed-point implementations of many kinds of functions
  - Other alternatives: Taylor series approximations, look-up tables, …
- DCT is an important transform in image/video/audio coding
  - High data rates require optimized implementations
- Optimized implementation:
  - Find data flow structures, which avoid redundant computation
  - Prefer simple operations (addition, arithmetic shifts, …)
  - See opportunities to simplify (e.q. JPEG quantization & CORDIC gain compensation)

- Filter banks revisited in future lectures