~~Analog IC Design Revision Exam-1~~

Fast ~~ICA~~ ICA Algorithm.

kurtosis → 4th order.

EMD — Empirical Mode decomposition

EMG → 20 Hz – 450 Hz
↓
electromyography.

IMF – Intrinsic Mode functions
↓
FM & AM oscillation modes.

linear
LIP
QIP — Interpolation ← predictive
CIP — Cubic splines

ICA: – Independent Component Analysis (obj: mixed signal → independent sources)

BSS
↓
Blind source segregation technique

$m = a_1 s_1 + a_2 s_2 + a_3 s_3 + a_4 s_4 \ldots$

↓ mixed signal; ↓ sources

Take this matrix form,

$$\begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix}_{4\times1} = \begin{bmatrix} a_{11} & a_{12} & a_{21} & a_{22} \\ \ldots \end{bmatrix}_{4\times4} \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ S_4 \end{bmatrix}_{4\times1}$$

Single channel fast IC – algorithm → Swethi barcding
(SCfastIC
↓
Research papers.

✓ gramschmidth
✓ update
✓ whitening
memory

} imp.

quadIC

Each fibre generates action potentials (MUAPs)

s

will act as dif sources → we ~~get~~ detect signal which is mixed.

- Gaussianity
- Non-gaussanity

~~kurtosis~~ (4th order random variable)

~~KURTOSIS~~ PUTOSIS.

Multiple MUAPs present on EMG.

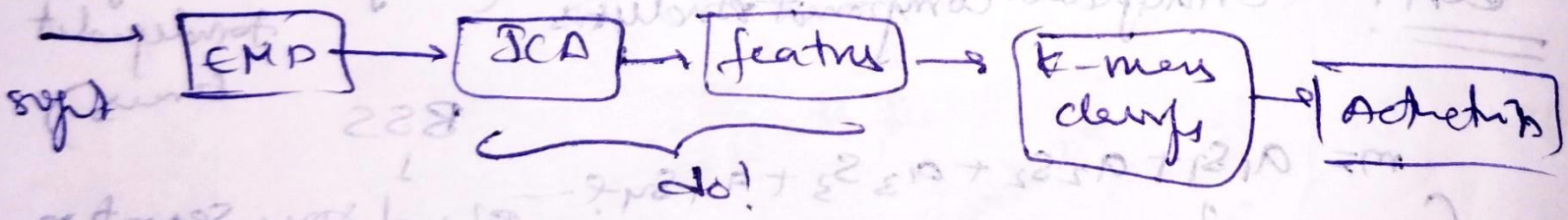
$\sum_{n=0}^{m}$ MUAP → EMG

from ICA - features are extracted

time domain specific features
wilson amplitude

RMS

Intrinsic Mode functions.



ICA algorithm - identify the sub-blocks

~~CORDIC square root~~:

Rotation mode:

rotate vector
Nor - div, sqr
gso - 2 vectors
gso - 64 vectors/ND
Updation

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \prod \cos\theta \prod \begin{bmatrix} 1 & \frac{1}{2^i} \\ -\frac{1}{2^i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

$$\prod_{i=0}^{n-1} \frac{1}{\sqrt{1+2^{-2i}}} = \frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{1+\frac{1}{2}}} \times \frac{1}{\sqrt{1+\frac{1}{4}}} \times \cdots$$

$$= \sqrt{\frac{1}{2}} \times \sqrt{\frac{2}{3}} \times \sqrt{\frac{4}{5}} \times \sqrt{\frac{8}{9}} \cdots$$

for standard cordic $\rightarrow k = 0.607252935$

fraction $\times$ 65536 = 16.16 format hexadecimal.

$1 = 65536$

$0 = 0$

$45^\circ = \frac{\pi}{4} = 0.785 \times 65536 = 51471.85$

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = 0.6072529 \begin{bmatrix} 1 & \frac{1}{2^i} \\ -\frac{1}{2^i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = k \begin{bmatrix} x_i - \frac{1}{2^i} y_i \\ \frac{-1}{2^i} x_i + y_i \end{bmatrix}$$

$$x_{i+1} = k\left[x_i \mp 2^{-i} y_i\right]$$

$$y_{i+1} = k\left[\pm 2^{-i} x_i + y_i\right]$$

20

$\theta_{i+1} = \theta_i \pm \tan^{-1}(2^{-i})$ (from rotation)

θ_{i+1} converges to 0 then y_{i+1} converges to x_{i+1}

$x_n \rightarrow k\sqrt{x^2+y}$

$y_n \rightarrow 0$

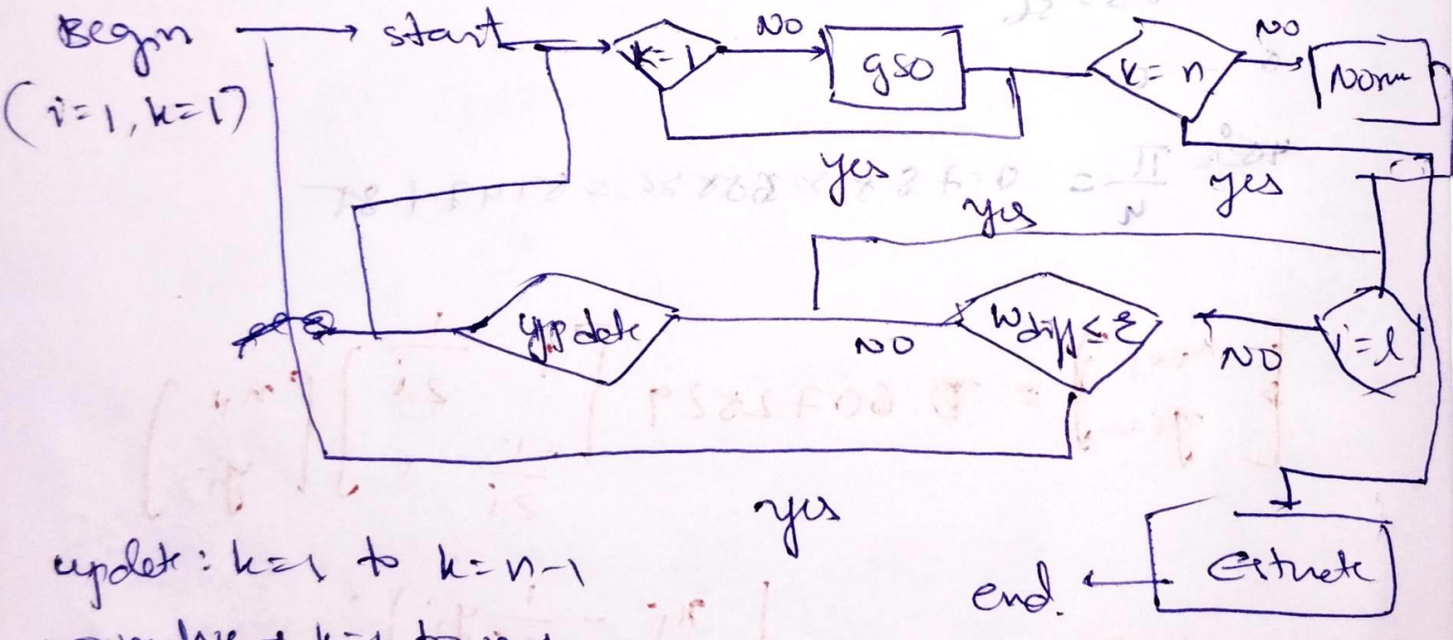
$\theta_n \rightarrow \theta_{req}$ } $\theta_{in} = \theta + \tan^{-1}(y/x)$

pipelined CORDIC — Parallel CORDIC.

CORDIC vectoring CORDIC Rotating

- vector update, orthogonalization, normalization.
vect cross product - right now not req.

2D → 7D.



update : k=1 to k=n-1

normalize → k=1 to n-1

estimate ⇒ k=1 to n-1

Eg sv — k=1 to n

convergecheck → k=1 to n-1.

Vectoring → (x_{in}, y_{in}) , $(x_{out}, y_{out}, \theta_{out})$

$y_{out} = 0$, $x_{out} = \sqrt{x_{in}^2 + y_{in}^2}$

$\theta_{out} = \tan^{-1}\left(\frac{y_{in}}{x_{in}}\right)$

In 20.12 format (fixed point)

$1 \times 4096 = 4096$

$90 \times 4096 = 368640$.

we use 4096 → as 2^{12}

4096, 0, 368640

4096×2

4096
4096

$\sqrt{2} \times 4096$

Normalization → to normalize (x_{in}, y_{in}) we can use vectoring which gives $x_{out} = \sqrt{x_{in}^2 + y_{in}^2}$

$\theta_{out} \rightarrow (\cos\theta_{out}, \sin\theta_{out}) = (x_{norm}, y_{norm})$

clk, reset, x_{in}, y_{in}, x_{out}, y_{out}, thetaout, iout;

x_{in}, y_{in} → 32 bit (20.12)

initial = 0, init = 0 → for i till N-1 for 0 & for y>0
for y<0

y>0 → $x_{next} = x_i + \frac{1}{2^i} y_i$ =

↓

$x_i + y >>> i$

>>> → shiftright
keep sign.

$y_{next} = y_i - \frac{1}{2^i} x_i$

↓

$y_i + x_i >>> i$

y >>> → +y/2

y >>> i → +$y/2^i$.

for y<0 → other case.

if y==0 → else x_next = x, ynext = 0, theta_next = theta
(maintains the same value of y=0)

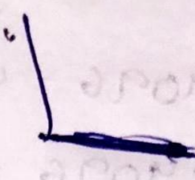
then update x <= x_next, y <= ynext

then iterate i <= i+1

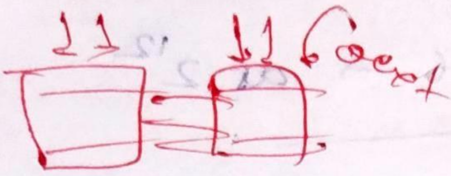
4096-1 ($\frac{1}{2}$) $\sqrt{2}+1$ $\sqrt{5}$.

Normalization means scaling (~~xxx~~ to magnitude to 1

~~two~~ $x_{nor} = \cos\theta$, $y_{nor} = \sin\theta$.



→ double pipelining:



vectoring, rotating, g80, normalization update } all in 7 D.

ganesh codes:

① absolute_value → calculates the absolute value of two signed signals x in & y in to give xout & yout.

data width we are using = 32 bits →

signed bits xin - having 1 in the sign bit → code takes 2's complement to give absolute value. (-ve)

Eg: when xin = -5 in 16 bits →

2's complement → ~x_in + 1'b1

1011 → 0100 + 1 = 0101

16'b 0 000000000000101 } 2's complement.

(sign bit) 16'b 1111111111111011 → -5

2 MUX used

xout = xin[15] ? { 1'b0, ~yin[14:0] + 1 }; xin

② vectoring

sd → signed decimal.

first stage, middle stages, last stage.

M CORDI.

Input: x_in, y_in output → xout, yout, θout.

vectoring is →

$x_{out} = M_{in}\cos(\beta+\theta) = M_{in}(\cos\beta\cos\theta - \sin\theta\sin\beta)$

$y_{out} = M_{in}\sin(\beta+\theta) = M_{in}(\sin\beta\cos\theta + \cos\beta\sin\theta)$

modulus of vector → $\sqrt{x_R^2 + y_R^2}$

β → initial angle

for anticlockwise.

$x_{in} = M_{in}\cos\beta$

$y_{in} = M_{in}\sin\beta$

$x_{out} = x_{in}\cos\theta - y_{in}\sin\theta$

$y_{out} = y_{in}\cos\theta + x_{in}\sin\theta$

for clockwise → $x_{out} = M_{in}\cos(\beta-\theta)$

$y_{out} = M_{in}\sin(\beta-\theta)$

Simplifying: $\begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix}$

e.g. anticlockwise.

$\begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix}$ → clockwise

↳ Rot(θ)

θ → can be decomposed into micro rotations → $\sum_{j=0}^{\infty} \alpha_j$

$Rot(\theta) = \prod_{j=0}^{\infty} Rot(\alpha)$

In CORDIC – we avoid multiplications – decomposition rotation into scaling operation & rotation rotations

$\prod \cos\alpha$ * $\prod \begin{bmatrix} 1 & \tan\alpha \\ -\tan\alpha & 1 \end{bmatrix}$

choosing elementary angle – for shifts

$\tan 2^{-i}$ $\quad \frac{1}{2^0}, \frac{1}{2}, \frac{1}{2^2}, \ldots, \frac{1}{2^{n-1}}$

$x_{out} = x_{in} - d\,2^{-i} y_{in}$

$y_{out} = y_{in} + d\,2^{-i} x_{in}$ $\quad d$ → sign bit

$+1, -1$ → direction

$\prod \cos\theta = \prod \frac{1}{\sqrt{1+\tan^2\theta}} = \prod \left(\frac{1}{(1+\frac{1}{2^{2i}})^{1/2}} \right)$

$\prod \frac{1}{(1+2^{-2i})^{1/2}} = \frac{1}{1.6468}$

we introduce $K(j)$ to cancle out $\prod \cos\theta$

$K(j) = \frac{1}{\prod \cos\alpha_j}$

$M(out) = K(j) \, M(in)$

when a vector is rotated then the length gets reduced by small amount → $\cos\theta$

for output adjustment → $x_{output} = K^{-1} x_{input}$

$k = K^{-1} \Rightarrow K = 1.6468$

we multiply ~~it with~~ input with 1.6468 after doing CORDIC

• Vector block for first stage → as a part of multi-stage pipeline. → clk, nreset, enable, xin, yin, xoutput, micro-rot_o, enable_next_stage, vec-micro rot

take x+y, 6 — ie 16+6=22 bit as xin & yin (to avoid overflow)

to enable next stage of pipline → start

→ active low reset

singed inputs → including → all quadrants.

→ nreset = 0 (off)

if (nreset) → xtemp_out = 0 → posedge clk & negedge
ytemp_out = 0
microrot = 0
enable_next stage = 0
vec_microrot_out_stl = 0.

(checks reset then also enable)

else → if (enable) → enable = 1

{ xtemp out = xin + yin (45 deg clk wise)

ytemp = yin − xin

macrot = 0

enable next stage = 1

vec-macrot-out-stat = 1 } allow to start macro rotation

else → enable next stage = 0

vec-macrot-out-stat = 0

→ why are we having a separate code for first stage and rethos code nothing relating anticlockwise is neutral?

as finding magnitude/angle is our goal one of the rotation is enough - we prefer clockwise

In first stage, initial large rotation is handled that sets up subsequent finer rotations - first stage - coarse rotation.

In next stages if error is positive - clockwise, else anticlockwise. → towards the desired vector direction iterate.

first stage uses - different control logic - isolates from next stage - optimizes hardware.

Breaking into discrete stages makes each stage simpler.

Vectoring goal: vector to align in with desired output direction

compute angle & magnitude of vector by putting initial y coordinate to 0 - which gives the angle of vector wrt x-axis. As rotations donot change magnitude, must give $\sqrt{x_{in}^2 + y_{in}^2}$

Convergence check is done → error should be less than

vectoring first stage in 2D ✓

2D → one vector vectoring → similarly → 2D ~~2~~ vectors

vectoring 4 pt can be similarly done for any no. of samples like 1024 in 2D only.

In double pipelining → rotating & vectoring are done at the same time - implenting two levels of pipelining for 2D vectoring:

pipelining: to enhance speed:

each of the stages are connected in sequence - allows each stage to start a new calculation as soon as prev. stage has passed its part, enabling multiple inputs to be processed in diff stages simultaneously

helps to achieve a higher clock rate - as each stage is responsible only for a portion of computation

double pipelining: breaking down each stage into even smaller sub-stages

parallel vs pipelined.

first level - primary CORDIC processor computes θ by vectoring - corresponds to iterative nature of CORDIC

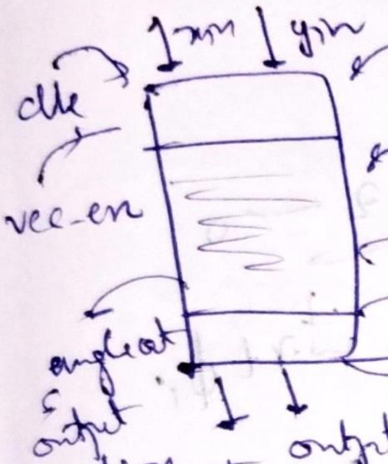
second level - Then rotations are done using rotating blocks as soon as θ is available (even partially) - we can start using those decisions to apply rotations to another vector.

Both vectoring & rotating happens at same time.

→ cordic vectoring 2D (this only we need.)

for 7D → 6 angles we get (x0, x1, x2, x3, x4, x5, x6)
↓ vectory

for 3D :
→ (x0, x1, x2) ⟨ vectory / rotaty



angle-calc-enable-in → enable to start calculating the angle of rotation

→ vec-microrot-out-start → indicates start of micro rotations

→ quad-out → quadrant.

→ micro-angle o → rotation directions from each stage

output & angle n out

output_valid-o → if output is valid.

yout-o

Blocks : Quadrant calculation → vec + quad-check

absolute value calc

input upscaling → to match the precision reg. +6 bits

first vectory stage i=1

intermediate micro rotations (i = 2 to 15)

final vectory stage i = 16

x_last_stage_out = $\sqrt{x^2+y^2}$ ← magnitude of vect

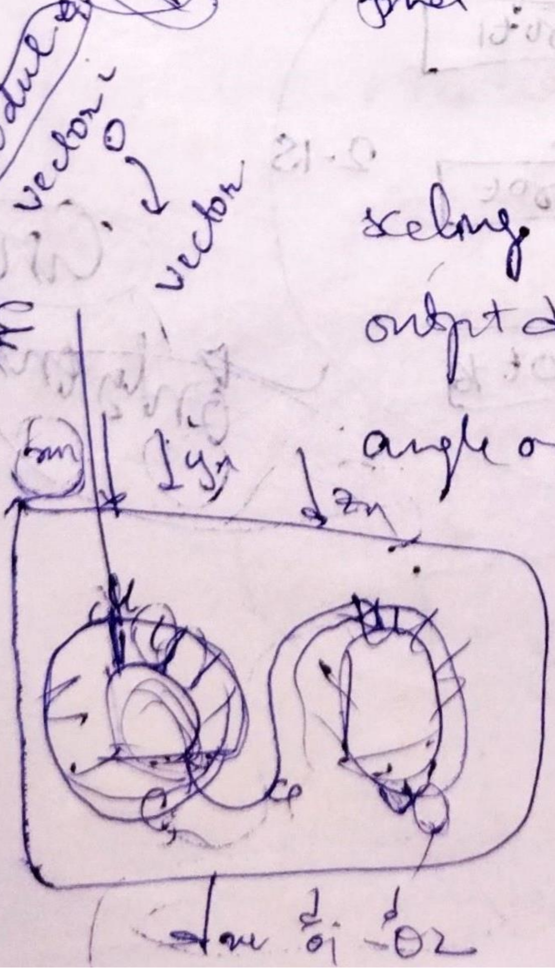
scaling

output downscaling from n+6 to n bits.

angle out final output.

vec

2. Module
vector → vector



f(a, b, sum).
a + b = sum.
f(1, 2, sum)

3D CORDIC algorithm.

(x_{in}, y_{in}, z_{in}) → coordinates (R_i, θ_i, ϕ_i)

In 2D – $(x_{in}, y_{in}) \rightarrow (R_{in}, \theta_{in})$

$R_{in} = \sqrt{x_{in}^2 + y_{in}^2 + z_{in}^2}$

$\sqrt{x_{in}^2 + y_{in}^2}$

Here

$y \rightarrow 0$, $x = \sqrt{x_{in}^2 + y_{in}^2}$

In 3D, $x = \sqrt{x_{in}^2 + y_{in}^2 + z_{in}^2}$, $y \rightarrow 0$, $z \rightarrow 0$

vector rotated by $\omega t \theta_i$ & $\omega t \phi_i$

angle α_i around z-axis & by an angle β_i