Lecture Overview

* **Iterative algorithms**
  - CORDIC
  - Division
  - Square root

* **Topics covered include**
  - Algorithms and their implementation
  - Convergence analysis
  - Speed of convergence
  - The choice of initial condition
CORDIC

- To perform the following transformation

\[ y(t) = y_R + j \cdot y_I \rightarrow |y| \cdot e^{j\phi} \]

and the inverse, we use the CORDIC algorithm

**CORDIC - COordinate Rotation DIgital Computer**

CORDIC: Idea

- Use rotations to implement a variety of functions

Examples:

\[ x + j \cdot y \leftrightarrow |x^2 + y^2| \cdot e^{j\tan^{-1}(y/x)} \]

\[ z = \sqrt{x^2 + y^2} \quad z = \cos(y / x) \quad z = \tan(y / x) \]

\[ z = x / y \quad z = \sin(y / x) \quad z = \sinh(y / x) \]

\[ z = \tan^{-1}(y / x) \quad z = \cos^{-1}(y) \]
**CORDIC (Cont.)**

- How to do it?
- Start with general rotation by $\phi$
  
  \[
  x' = x \cdot \cos(\phi) - y \cdot \sin(\phi) \\
  y' = y \cdot \cos(\phi) + x \cdot \sin(\phi)
  \]

  \[
  x' = \cos(\phi) \cdot [x - y \cdot \tan(\phi)] \\
  y' = \cos(\phi) \cdot [y + x \cdot \tan(\phi)]
  \]

- The trick is to only do rotations by values of $\tan(\phi)$ which are powers of 2

---

**CORDIC (Cont.)**

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$\tan(\phi)$</th>
<th>$k$</th>
<th>$i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>45°</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>26.565°</td>
<td>$2^{-1}$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>14.036°</td>
<td>$2^{-2}$</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>7.125°</td>
<td>$2^{-3}$</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>3.576°</td>
<td>$2^{-4}$</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>1.790°</td>
<td>$2^{-5}$</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>0.895°</td>
<td>$2^{-6}$</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

- To rotate to any arbitrary angle, we do a sequence of rotations to get to that value
Basic CORDIC Iteration

\[ x_{i+1} = (K_i) \cdot [x_i - y_i \cdot d_i \cdot 2^{-i}] \]
\[ y_{i+1} = (K_i) \cdot [y_i + x_i \cdot d_i \cdot 2^{-i}] \]

\[ K_i = \cos(\tan^{-1}(2^{-i})) = 1/(1 + 2^{-2i})^{0.5} \]
\[ d_i = \pm 1 \]

The \( d_i \) is chosen to rotate by \( \pm \varphi \)

- If we don’t multiply \((x_{i+1}, y_{i+1})\) by \( K_i \) we get a gain error which is independent of the direction of the rotation
- The error converges to 0.61 - May not need to compensate for it
- We also can accumulate the rotation angle: \( z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i}) \)

Example

- Initial vector is described by \( x_0 \) and \( y_0 \) coordinates

\[ \sqrt{x_0^2 + y_0^2} \]

- We want to find \( \varphi \) and \((x_0^2 + y_0^2)^{0.5}\)
**Step 1: Check the Angle / Sign of $y_0$**

- If positive, rotate by $-45^\circ$
- If negative, rotate by $+45^\circ$

\[
\begin{align*}
  d_1 &= -1 \ (y_0 > 0) \\
  x_1 &= x_0 + y_0/2 \\
  y_1 &= y_0 - x_0/2
\end{align*}
\]

**Step 2: Check the Sign of $y_1$**

- If positive, rotate by $-26.57^\circ$
- If negative, rotate by $+26.57^\circ$

\[
\begin{align*}
  d_2 &= -1 \ (y_1 > 0) \\
  x_2 &= x_1 + y_1/4 \\
  y_2 &= y_1 - x_1/4
\end{align*}
\]
Repeat Step 2 for Each Rotation $k$

- Until $y_n = 0$

\[ x_n = A_n \cdot (x_0^2 + y_0^2)^{0.5} \]

The Gain Factor

- Gain accumulation:
  \[ G_0 = 1 \]
  \[ G_0G_1 = 1.414 \]
  \[ G_0G_1G_2 = 1.581 \]
  \[ G_0G_1G_2G_3 = 1.630 \]
  \[ G_0G_1G_2G_3G_4 = 1.642 \]

- So, start with $x_0$, $y_0$; end up with:
  \[ z_3 = 71^\circ \]
  \[ (x_0^2 + y_0^2)^{0.5} = 1.642 \ (\ldots) \]

- We did the rectangular-to-polar coordinate conversion
Rectangular-to-Polar Conversion: Summary

- Start with vector on x-axis

\[ A = |A| \cdot e^{j\varphi} \]
\[ x_0 = |A| \]
\[ y_0 = 0, \ z_0 = \varphi \]

\[ z_i < 0, \ d_i = -1 \]
\[ z_i > 0, \ d_i = +1 \]

\[ z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i}) \]

CORDIC Algorithm

\[ x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i} \]
\[ y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i} \]
\[ z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i}) \]

\[ d_i = \begin{cases} 
-1, & z_i < 0 \\
+1, & z_i > 0 
\end{cases} \]

\[ d_i = \begin{cases} 
-1, & y_i > 0 \\
+1, & y_i < 0 
\end{cases} \]

**Rotation mode**
(rotate by specified angle)
Minimize residual angle

**Vectoring mode**
(align with the x-axis)
Minimize y component

Result
\[ x_n = A_n \cdot [x_0 \cdot \cos(z_0) - y_0 \cdot \sin(z_0)] \]
\[ y_n = A_n \cdot [y_0 \cdot \cos(z_0) + x_0 \cdot \sin(z_0)] \]
\[ z_n = 0 \]

Result
\[ x_n = A_n \cdot (x_0^2 + y_0^2)^{0.5} \]
\[ y_n = 0 \]
\[ z_n = z_0 + \tan^{-1}(y_0/x_0) \]

\[ A_n = \prod (1+2^{-2^i}) \cdot 0.5 \rightarrow 1.647 \]

7.13

7.14
**Vectoring Example**

![Diagram showing vectoring example](image)

<table>
<thead>
<tr>
<th>Acc. Gain</th>
<th>Residual angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_0 = 1$</td>
<td>$\varphi = 30^\circ$</td>
</tr>
<tr>
<td>$K_1 = 1.414$</td>
<td>$\varphi = -15^\circ$</td>
</tr>
<tr>
<td>$K_2 = 1.581$</td>
<td>$\varphi = 11.57^\circ$</td>
</tr>
<tr>
<td>$K_3 = 1.630$</td>
<td>$\varphi = -2.47^\circ$</td>
</tr>
<tr>
<td>$K_4 = 1.642$</td>
<td>$\varphi = 4.65^\circ$</td>
</tr>
<tr>
<td>$K_5 = 1.646$</td>
<td>$\varphi = 1.08^\circ$</td>
</tr>
<tr>
<td>Etc.</td>
<td></td>
</tr>
</tbody>
</table>

**Vectoring Example: Best-Case Convergence**

![Diagram showing vectoring example](image)

<table>
<thead>
<tr>
<th>Acc. Gain</th>
<th>Residual angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_0 = 1$</td>
<td>$\varphi = 45^\circ$</td>
</tr>
<tr>
<td>$K_1 = 1.414$</td>
<td>$\varphi = 0^\circ$</td>
</tr>
<tr>
<td>$K_2 = 1.581$</td>
<td>$\varphi = 0^\circ$</td>
</tr>
<tr>
<td>$K_3 = 1.630$</td>
<td>$\varphi = 0^\circ$</td>
</tr>
<tr>
<td>$K_4 = 1.642$</td>
<td>$\varphi = 0^\circ$</td>
</tr>
<tr>
<td>$K_5 = 1.646$</td>
<td>$\varphi = 0^\circ$</td>
</tr>
<tr>
<td>Etc.</td>
<td></td>
</tr>
</tbody>
</table>

*In the best case ($\varphi = 45^\circ$), we can converge in one iteration*
Calculating Sine and Cosine

To calculate sin and cos:
- Start with $x_0 = 1/1.64$, $y_0 = 0$
- Rotate by $\varphi$

![Diagram showing sine and cosine](image)

To calculate sin and cos:
- Start with $x_0 = 1/1.64$, $y_0 = 0$
- Rotate by $\varphi$

### Functions

#### Rotation mode

- **sin/cos**
  - $z_0 = \text{angle}$
  - $y_0 = 0$, $x_0 = 1/A_n$
  - $x_n = A_n \cdot x_0 \cdot \cos(z_0)$
  - $y_n = A_n \cdot x_0 \cdot \sin(z_0)$
  - ($=1$)

#### Vectoring mode

- **$\tan^{-1}$**
  - $z_0 = 0$
  - $z_n = z_0 + \tan^{-1}(y_0/x_0)$
  - **Vector/Magnitude**
  - $x_n = A_n \cdot (x_0^2 + y_0^2)^{0.5}$

#### Polar $\rightarrow$ Rectangular

- $x_n = r \cdot \cos(\varphi)$
- $y_n = r \cdot \sin(\varphi)$

#### Rectangular $\rightarrow$ Polar

- $x_0 = r$
- $z_0 = \varphi$
- $y_0 = 0$
- $r = (x_0^2 + y_0^2)^{0.5}$
- $\varphi = \tan^{-1}(y_0/x_0)$
CORDIC Divider

- To do a divide, change CORDIC rotations to a linear function calculator

\[
x_{i+1} = x_i - 0 \cdot y_i \cdot d_i \cdot 2^{-i} = x_i
\]

\[
y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}
\]

\[
z_{i+1} = z_i - d_i \cdot (2^{-i})
\]

Generalized CORDIC

\[
x_{i+1} = x_i - m \cdot y_i \cdot d_i \cdot 2^{-i}
\]

\[
y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}
\]

\[
z_{i+1} = z_i - d_i \cdot e_i
\]

<table>
<thead>
<tr>
<th>(d_i)</th>
<th>Rotation</th>
<th>Vectoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_i = -1, z_i &lt; 0)</td>
<td>(d_i = -1, y_i &gt; 0)</td>
<td>(d_i = +1, y_i &lt; 0)</td>
</tr>
<tr>
<td>sign((z_i))</td>
<td>-sign((y_i))</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mode</th>
<th>(m)</th>
<th>(e_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circular</td>
<td>+1</td>
<td>tan(^{-1}(2^{-i}))</td>
</tr>
<tr>
<td>Linear</td>
<td>0</td>
<td>(2^{-i})</td>
</tr>
<tr>
<td>Hyperbolic</td>
<td>-1</td>
<td>tanh(^{-1}(2^{-i}))</td>
</tr>
</tbody>
</table>
An FPGA Implementation

- Three difference equations directly mapped to hardware
- The decision $d_i$ is driven by the sign of the $y$ or $z$ register
  - Vectoring: $d_i = -\text{sgn}(y_i)$
  - Rotation: $d_i = \text{sgn}(z_i)$
- The initial values loaded via muxes
- On each clock cycle
  - Register values are passed through shifters and add/sub and the values placed in registers
  - The shifters are modified on each iteration to cause the desired shift (state machine)
  - Elementary angle stored in ROM
- Last iteration: results read from reg


Iterative Sqrt and Division

- Inputs:
  - $a$ (14 bits), reset (active high)
- Outputs:
  - $zs$ (16 bits), $zd$ (16 bits) Total: 32 bits

Iterative $1/\sqrt{Z}$: Simulink XSG Model

- **User defined parameters**
  - Wordlength (#bits, binary pt)
  - Quantization, overflow
  - Latency, sample period

- **The choice of initial condition**
  - Determines # iterations
  - and convergence...

\[ x_s(k+1) = \frac{x_s(k)}{2 \cdot (3 - Z \cdot x_s^2(k))} \]

\[ x_s(k) = \frac{1}{\sqrt{N}} \lim_{k \to \infty} \]

\[ y_s(k+1) = \frac{y_s(k)}{2 \cdot (3 - y_s^2(k))} \]

\[ y_s(k) = \frac{1}{\sqrt{N}} \lim_{k \to \infty} \]

\[ e_s(k) = y_s(k) - 1 \]

\[ e_s(k+1) = \frac{1}{2} \cdot e_s(k)^2 \cdot (3 + e_s(k)) \]
**Quadratic Convergence: 1/N**

\[ x_d(k+1) = x_d(k) \cdot (2 - N \cdot x_d(k)) \quad x_d(k) \to \frac{1}{N_{k \to \infty}} \]

\[ y_d(k+1) = y_d(k) \cdot (2 - y_d(k)) \quad y_d(k) \to 1_{k \to \infty} \]

\[ e_d(k) = y_d(k) - 1 \]

\[ e_d(k+1) = -e_d(k)^2 \]

**Initial Condition: 1/sqrt(N)**

\[ y_s(k+1) = \frac{y_s(k)}{2} \cdot (3 - y_s(k)^2) \quad y_s(k) \to 1_{k \to \infty} \]

Convergence: \( 0 < y_s(0) < \sqrt{3} \)

Conv. stripes: \( \sqrt{3} < y_s(0) < \sqrt{5} \)

Divergence: \( y_s(0) > \sqrt{5} \)
Initial Condition: $1/N$

\[ y_d(k+1) = y_d(k) \cdot (2 - y_d(k)) \quad \text{Convergence: } 0 < y_d(0) < 2 \]

\[ y_d(k) \rightarrow 1_{k \to \infty} \]

\[ y_d(k) \]

\[ y_d(k+1) \]

1/sqrt($N$): Convergence Analysis

\[ x_{n+1} = \frac{x_n}{2} \cdot (3 - N \cdot x_n^2) \]

\[ x_n \rightarrow \frac{1}{\sqrt{N}}_{n \to \infty} \]

\[ x_n = \frac{y_n}{\sqrt{N}} \]

\[ y_{n+1} = \frac{y_n}{2} \cdot (3 - y_n^2) \quad [3] \]

Error:

\[ e_n = 1 - y_n \]

\[ e_{n+1} = \frac{3}{2} \cdot e_n^2 - \frac{1}{2} \cdot e_n^3 \]

$y_{n+1} = \frac{y_n}{2} \cdot (3 - y_n^2)$

$y_n \to 1_{n \to \infty}$

$0 < y_0 < \sqrt{3} \Rightarrow$ convergence

$\sqrt{3} < y_0 < \sqrt{5} \Rightarrow$ conv. stripes

$y_0 > \sqrt{5} \Rightarrow$ divergence

$\Rightarrow$ divergence

$\Rightarrow$ conv. stripes

$\Rightarrow$ convergence
Choosing the Initial Condition

Initial Condition > sqrt(5) Results in Divergence
1/sqrt(): Picking Initial Condition

\[ x_{n+1} = \frac{x_n}{2} \cdot (3 - N \cdot x_n^2) \]

\[ x_n \to \frac{1}{\sqrt{N}} \quad \text{as } n \to \infty \]

- **Equilibriums**: \( 0, \pm \frac{1}{\sqrt{N}} \)

- **Initial condition**
  - Take: \( V(x_n) = \left( x_n - \frac{1}{\sqrt{N}} \right)^2 \) (6.1)
  - Find \( x_0 \) such that:
    \[ V(x_{n+1}) - V(x_n) < 0 \quad \forall n \quad (n = 0,1,2,...) \] (6.2)
  - Solution: \( S = \{ x_0 : V(x_0) < a \} \) (6.3)

  “Level set” \( V(x_0) = a \) is a convergence bound
  \( \Rightarrow \) Local convergence (3 equilibriums)

---

**Descending Absolute Error**

- **sqrt**: \( V_s(x_0) = \frac{x_0}{4} \cdot (x_0 - 1)^2 \cdot (x_0 + 1) \cdot (x_0^2 + x_0 - 4) \)
  - Initial condition, \( x_0 \):
    \[ V_s(x_0) = x_0 \cdot (x_0 - 1)^2 \cdot (x_0 - 2) \]

- **Div**: \( V_d(x_0) = x_0 \cdot (x_0 - 1)^2 \cdot (x_0 - 2) \)
  - Initial condition, \( x_0 \):

**Descending error**:

\( E(x_k) = (x_k - 1)^2 \quad V(x_k) = E(x_{k+1}) - E(x_k) < 0 \quad \forall k \)
1/sqrt(N): Picking Initial Condition (Cont.)

(6.2) \[ \frac{x_0}{2} \cdot (1 - N \cdot x_0^2) \cdot \left( \frac{5}{2}x_0 - \frac{N}{2}x_0^3 - \frac{2}{\sqrt{N}} \right) < 0 \]

Roots:
\[ x_1 = \frac{1}{\sqrt{N}} \]
\[ x_{2,3} = \frac{1}{2 \cdot \sqrt{N}} (-1 \pm \sqrt{17}) \]

Max:
\[ x_{0M} = \sqrt{\frac{5}{3N}} \]

Initial Condition Circuit

(6.3) \[ \frac{1}{\sqrt{N}} - \sqrt{a} < x_0 < \frac{1}{\sqrt{N}} + \sqrt{a} \]

\[ N \]
\[ >16 \]
\[ >4 \]
\[ >1 \]
\[ >1/4 \]
\[ >1/16 \]
\[ >1/64 \]

\[ x_0 = 1/4 \]
\[ x_0 = 1/2 \]
\[ x_0 = 1 \]
\[ x_0 = 2 \]
\[ x_0 = 4 \]
\[ x_0 = 8 \]
Left: Internal Node, Right: Sampled Output

- Internal node and output
- Zoom in: convergence in 8 iterations
- Internal divide by 2 extends range

Convergence Speed

- **# iterations required for specified accuracy**

<table>
<thead>
<tr>
<th>Target relative error (%)</th>
<th>0.1%</th>
<th>1%</th>
<th>5%</th>
<th>10%</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_0$: 50%, # iter (sqrt/div)</td>
<td>5 / 4</td>
<td>5 / 3</td>
<td>4 / 3</td>
<td>3 / 2</td>
</tr>
<tr>
<td>$e_0$: 25%, # iter (sqrt/div)</td>
<td>3 / 3</td>
<td>3 / 2</td>
<td>2 / 2</td>
<td>2 / 1</td>
</tr>
</tbody>
</table>

- **Adaptive algorithm**
  - current result $\rightarrow$ .ic for next iteration

$N_k$ $\rightarrow$ .ic $\rightarrow$ Iterative algorithm $\rightarrow$ $(y_k)\rightarrow 1/\sqrt{N}$
Summary

- Iterative algorithms can be used for a variety of DSP functions
- CORDIC uses angular rotations to compute trigonometric and hyperbolic functions as well as divide and other operations
  - One bit of resolution is resolved in each iteration
- Newton-Raphson algorithms for square root and division have faster convergence than CORDIC
  - Two bits of resolution are resolved in each iteration (the algorithm has quadratic error convergence)
- Convergence speed greatly depends on the initial condition
  - The choice of initial condition can be made to guarantee decreasing absolute error in each iteration
  - For slowly varying inputs, adaptive algorithms can use the result of the current iteration as the initial condition
  - Hardware latency depends on the initial condition and accuracy

References


Additional References

