THE CORDIC ALGORITHM AND CORDIC ARCHITECTURES

Implementation of Digital Signal Processing

Sabih H. Gerez
University of Twente

COURSE PROGRESS

March 5, 2021
Algorithm transformations
[Par95]
March 5 – March
12, 2021
The CORDIC Algorithm
[And98] and [Lee00]
March 5, 2021
Transformation
Addendum
March 16,
2018
CORDIC
March 16,
2018
Polyphase implementation of
multirate filters
[Lam02] and [Vau09]
The part of the theory on
downsampling is
compensatory, the part on
up-sampling is optional.
Polyphase
Implementation
March 16,
2018
March 12, 2021
March 12, 2021
March 12, 2021
March 19, 2021
March 19, 2021
March 19, 2021
Multiphaseless filter design
[Heu00], [Vau07], [Aks14] and [Kok13]
Software synthesis
Sections I and II of [Bla00]
Code generation
Sections III and IV of [Bla00]
Case study: simultaneous design
of processor and compiler
[Good85]
Multiphaseless Filter
Design
Software Synthesis
Code Generation
案件 study: simultaneous design
of processor and compiler

PROJECT PROGRESS POLL

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Title</th>
<th>Max. points</th>
<th>Nominal Load</th>
<th>Start after lecture of</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP</td>
<td>Mapping Data-Flow Graphs to RTL Designs</td>
<td>30</td>
<td>30 hours</td>
<td>February 19, 2021</td>
</tr>
<tr>
<td>TRA</td>
<td>Data-Flow-Graph Transformations</td>
<td>10</td>
<td>10 hours</td>
<td>May 5, 2021</td>
</tr>
<tr>
<td>GFS</td>
<td>The GFSK Receiver</td>
<td>60</td>
<td>60 hours</td>
<td>December 12, 2021</td>
</tr>
</tbody>
</table>

• Please respond on your progress on Project MAP:
  - A: I did not start at all;
  - B: I have worked on the “pen and paper” part of MAP;
  - C: I have also done some practical work of MAP on the xoc2 server;
  - D: I am (almost) finished MAP, but have not/hardly started work on TRA.
  - E: I have (almost) finished both MAP and TRA.

OUTLINE

• CORDIC algorithm:
  – Rotation and vectoring modes
• CORDIC architectures
  – Introduction to Project GFS
• Applications of CORDIC
REFERENCES


WHAT IS CORDIC?

- CORDIC: abbreviation of coordinate rotation digital computer.

- First publication by Volder, 1959.

- A method from the field of computer arithmetic allowing for the efficient implementation of a wide range of computations.

VECTOR ROTATIONS (1)

- Consider a sequence of rotations of a vector \((x^{(i)}, y^{(i)})\)^T rotated by \(\alpha_i\) to give vector \((x^{(i+1)}, y^{(i+1)})\)^T.

- So: 
  \[
  \begin{bmatrix}
  x^{(i+1)} \\
  y^{(i+1)}
  \end{bmatrix} = 
  \begin{bmatrix}
  \cos(\alpha_i) & -\sin(\alpha_i) \\
  \sin(\alpha_i) & \cos(\alpha_i)
  \end{bmatrix} 
  \begin{bmatrix}
  x^{(i)} \\
  y^{(i)}
  \end{bmatrix}
  \]

- After rewrite: 
  \[
  \begin{bmatrix}
  x^{(i+1)} \\
  y^{(i+1)}
  \end{bmatrix} = \cos(\alpha_i) \begin{bmatrix} 1 & -\tan(\alpha_i) \\ \tan(\alpha_i) & 1 \end{bmatrix} \begin{bmatrix}
  x^{(i)} \\
  y^{(i)}
  \end{bmatrix}
  \]

- If \(\tan(\alpha_i)\) is chosen such that \(\tan(\alpha_i) = d_i2^{-i}\), with \(d_i = \pm 1\), then the rotations can be executed without multiplications except for initial factor \(\cos(\alpha_i) = \frac{1}{\sqrt{1 + 2^{-2i}}}\).

VECTOR ROTATIONS (2)

- If \(\tan(\alpha_i) = d_i2^{-i}\), this means: \(\alpha_i = d_i\arctan(2^{-i})\).

- For an arbitrary angle \(-\pi/2 \leq \alpha \leq \pi/2\), the angle can then be decomposed as:

  \[
  \alpha = \sum_{i=0}^{n} d_i\arctan(2^{-i})
  \]

- Angles involved:

  \[
  \begin{array}{cccccccccc}
  i & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
  2^{-i} & 1 & 1/2 & 1/4 & 1/8 & 1/16 & 1/32 & 1/64 & 1/128 & 1/256 \\
  \arctan(2^{-i})[\text{deg}] & 45.0 & 26.6 & 14.0 & 7.1 & 3.6 & 1.8 & 0.9 & 0.4 & 0.2
  \end{array}
  \]
VECTOR ROTATION EXAMPLE (1)

- The 8 subsequent rotations for a rotation of 15 degrees are:

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^{-i}</td>
<td>1</td>
<td>1/2</td>
<td>1/4</td>
<td>1/8</td>
<td>1/16</td>
<td>1/32</td>
<td>1/64</td>
<td>1/128</td>
<td>1/256</td>
</tr>
<tr>
<td>\arctan(2^{-i})</td>
<td>45.0</td>
<td>26.6</td>
<td>14.0</td>
<td>7.1</td>
<td>3.6</td>
<td>1.8</td>
<td>0.9</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>d_i</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>\sum \alpha_i</td>
<td>45.0</td>
<td>18.4</td>
<td>4.4</td>
<td>11.5</td>
<td>15.1</td>
<td>13.3</td>
<td>4.2</td>
<td>14.7</td>
<td>14.9</td>
</tr>
</tbody>
</table>

- The arctangent values can be precomputed and stored in a look-up table (LUT), say \( L(i) \).
- The \( d_i \) depend on the required rotation angle.

ANGLE ACCUMULATION

- Keep track of total rotation angle in an angle accumulator:

\[
z^{(i+1)} = z^{(i)} - d_i L(i)
\]

- The angle accumulator can be used to determine \( d_i \):
  - Initialize \( z^{(0)} = \alpha \).
  - Factor \( d_{i+1} \) becomes 1 when \( z^{(i)} \geq 0 \) and -1 otherwise.

CORDIC EQUATIONS SUMMARY

- Original equations were:

\[
\begin{bmatrix}
    x^{(i+1)} \\
    y^{(i+1)}
\end{bmatrix} = \begin{bmatrix}
    1 & -\tan(\alpha_i) \\
    \tan(\alpha_i) & 1
\end{bmatrix} \begin{bmatrix}
    x^{(i)} \\
    y^{(i)}
\end{bmatrix}
\]

- Making use of the special values for the tangent, leaving out the multiplication by the cosine and combining with angle accumulation, one gets:

\[
\begin{align*}
x^{(i+1)} &= x^{(i)} - d_i 2^{-i} y^{(i)} \\
y^{(i+1)} &= d_i 2^{-i} x^{(i)} + y^{(i)} \\
z^{(i+1)} &= z^{(i)} - d_i L(i)
\end{align*}
\]
ROTATION-MODE CORDIC

- Goal is to rotate vector by angle $\alpha$.
- Initialization:
  - $x^{(0)} = x$
  - $y^{(0)} = y$
  - $z^{(0)} = \alpha$
- Final result:
  - $x^{(n)} = K (x \cos(\alpha) - y \sin(\alpha))$
  - $y^{(n)} = K (x \sin(\alpha) - y \cos(\alpha))$
  - $z^{(n)} = 0$
- Where:
  - $K = \prod_{i=1}^{n} \sqrt{1 + 2^{-2i}}$
  - $K$ converges to 1.647.
- Conclusion: the result vector is rotated but scaled version of original vector.

VECTORING-MODE CORDIC

- Determine $d_i$ by an alternative rule: $d_i = -1$ when $y^{(i)} > 0$ and $d_i = +1$ when $y^{(i)} \leq 0$.
- Initialization:
  - $x^{(0)} = x$
  - $y^{(0)} = y$
  - $z^{(0)} = 0$
- Final result:
  - $x^{(n)} = K \sqrt{x^2 + y^2}$
  - $y^{(n)} = 0$
  - $z^{(n)} = \arctan \left( \frac{y}{x} \right)$
- This means that the initial vector has been rotated (and scaled) onto the X-axis, while the angle with the X-axis has been computed as well.

BASIC APPLICATIONS OF CORDIC

- **Arctangent, vector-magnitude** calculation and **rectangular-to-polar conversion**: direct result of vectoring-mode CORDIC.
- **Polar-to-rectangular conversion**, i.e. from $(r, \theta)$ to $(x, y)$:
  - Set $x^{(0)} = r$, $y^{(0)} = 0$, and $z^{(0)} = \theta$ in rotation mode.
  - Result will be $x = x^{(n)} = Kr \cos(\theta)$, $y = y^{(n)} = Kr \sin(\theta)$.
  - Correction for scaling by $K$ may be necessary (does not require a full-fledged multiplier as $K$ is constant).
- **Sine or cosine** calculation:
  - See above, set $x^{(0)} = 1/K$. Then $x^{(n)} = \cos(\theta)$ and $y^{(n)} = \sin(\theta)$.

ARCHITECTURE ITERATIVE CORDIC

Controller should take care of initializations, add/subtract decisions, number of iterations, etc.
The iterative architecture requires one clock cycle per iteration. It requires a *barrel shifter* to shift operand over a variable number of positions.

One can also *unroll* the architecture to perform all operations in a single clock cycle:
- Amounts to instantiate new hardware for each iteration.
- Possibly adding *pipelining* if the *critical path* becomes too long.
- The barrel shifter is no longer necessary: each stage in the hardware has a fixed shift which costs just wires.
- One could also unroll the architecture partially.

**DESIGN EXAMPLE: GFSK RECEIVER**

- What is GFSK?
  - *Gaussian frequency shift keying*
  - Method for digital transmission based on frequency modulation (FM).
  - To transmit a `1` carrier frequency is slightly increased and to transmit a `0` the frequency is slightly decreased (or vice versa).
  - The transition steps are smoothed by a Gaussian filter.
  - Found in many standards such as Bluetooth and DECT.
  - Proposed version uses parameters not related to any standard.

**GFSK RECEIVER DESIGN APPROACH**

- Model entire system: transmitter, receiver, and a channel adding noise (AWGN).
- Leave out analog circuitry for upconversion to RF and downconversion back to IF.
- Use **IT++** to set up testbench.
- The testbench computes bit error rates (BERs) for different signal-to-noise ratios (SNRs).
- Goal is to preserve BER performance when designing hardware.

*Modulated signal has 16 samples per transmitted bit.*
IMPLEMENTATION ASPECTS

- Projects focus on designing in Arx.
- Testbenches for generated C++ and VHDL will be provided.
- As C++ and VHDL behave exactly the same, most simulations will be done in C++ (simulation speed in e.g. BER simulations is important).
- C++ testbenches make use of IT++, an open-source library for telecom/signal processing:
  - [http://itpp.sourceforge.net](http://itpp.sourceforge.net)
  - It provides Matlab-style programming in C++, so vectors, matrices, etc. and lots of powerful functions to manipulate them.

GFSK: MODULATION IN FORMULAE

- The modulated signal: $s(t) = A \cos(\omega_{IF} t + \phi(t))$
  - where:
    - $A$ is the constant amplitude
    - $\omega_{IF}$ is the intermediate frequency (acts as carrier frequency)
    - $\phi(t)$ is the phase deviation, derived from the bit stream
- The phase deviation:
  $$\phi(t) = h \pi \int_{-\infty}^{t} \sum_{i} a_i g(\tau - iT) d\tau$$
  - where:
    - $h$ is the modulation index
    - $g(t)$ is a Gaussian-filtered square wave
    - $a_i$ is 1 for a transmitted 1 and -1 for a transmitted 0.

DEMODULATOR BLOCK DIAGRAM

The 16 samples per transmitted bit are first reduced to 4 and later back to 1.
Digital downconversion is a common operation in digital radio receivers. It is used to shift the carrier frequency of a radio signal (e.g. from IF to baseband) or correct for frequency offset.

This is done by multiplying an input signal by a sine and cosine of some frequency. Think of the GFSK demodulator.

Loehning, et al. Figure 1

This solution requires just one CORDIC.

Loehning, et al. Figure 2

IMAGE REJECTION MIXER: “complex” input with in-phase (I) and quadrature (Q) component. CORDIC replaces 4 multipliers!