Hardware Implementations of Fixed-Point Atan2

Florent de Dinechin  Matei Istoan

Université de Lyon, INRIA, INSA-Lyon, CITI-Lab

ARITH22
Methods for Computing $\text{atan2}$ in Hardware

Yet another arithmetic function . . .

- . . . that is useful in *telecom* (to recover the phase of a signal)
  (12–24 bits of precision)
- . . . and in general for *cartesian to polar* coordinate transformation
- and an *interesting* function, nonetheless
Common Specification

- **target function**

\[ f(x, y) = \frac{1}{\pi} \arctan \left( \frac{y}{x} \right) \]

- **input: fixed-point format**

\[ \arctan \left( \frac{ky}{kx} \right) = \arctan \left( \frac{y}{x} \right) \]

- **output: fixed-point format and binary angles**
Common Specification

- target function

\[ f(x, y) = \frac{1}{\pi} \arctan \left( \frac{y}{x} \right) \]

- input: fixed-point format

\[
\arctan \left( \frac{ky}{kx} \right) = \arctan \left( \frac{y}{x} \right)
\]

- output: fixed-point format and binary angles
Common Specification

- **target function**
  \[ f(x, y) = \frac{1}{\pi} \arctan \left( \frac{y}{x} \right) \]

- **input: fixed-point format**
  \[
  \arctan \left( \frac{ky}{kx} \right) = \arctan \left( \frac{y}{x} \right) 
  \]

- **output: fixed-point format and binary angles**

\[
\begin{align*}
\alpha &= \text{atan2}(y, x) \\
(y, x) &= (0, 1) \\
\end{align*}
\]
Common Specification

- target function

\[ f(x, y) = \frac{1}{\pi} \arctan \left( \frac{y}{x} \right) \]

- input: fixed-point format

\[ \arctan \left( \frac{ky}{kx} \right) = \arctan \left( \frac{y}{x} \right) \]

- output: fixed-point format and binary angles

\[ (-1, 0) \rightarrow (0, 1) \rightarrow (1, 0) \rightarrow (-1, 0) \]
A Meaningful Comparison

3 different methods for evaluating $\text{atan2}$ in hardware

- same accuracy specification: $f(x, y)$ computed with last-bit accuracy (faithful rounding)
- same implementation effort
A Meaningful Comparison

3 different methods for evaluating $\text{atan2}$ in hardware

- same accuracy specification: $f(x, y)$ computed with last-bit accuracy (faithful rounding)
- same implementation effort

Target platform: **FPGAs** (Field Programmable Gate Arrays)
Hello FPGAs!

Island-style homogeneous FPGAs
First Method: An Unrolled CORDIC

\[
\begin{align*}
\alpha_0 &= 0 \\
(x_0, y_0) &\rightarrow (x_0, y_0) \\
\end{align*}
\]

\[
\begin{align*}
x_{i+1} &= x_i - 2^{-i} s_i y_i \\
y_{i+1} &= y_i + 2^{-i} s_i x_i \\
\alpha_{i+1} &= \alpha_i - s_i \arctan 2^{-i} \\
\end{align*}
\]

\[
\begin{align*}
x_n &\rightarrow K \sqrt{x^2 + y^2} \\
y_n &\rightarrow 0 \\
\alpha_i &\rightarrow \arctan \frac{y}{x} \\
\end{align*}
\]
CORDIC Iteration: Datapath Implementation

\[ p = w - 1 - \lceil \log_2 (w - 1) \rceil \text{ bits for the } x_i \text{ and } y_i \text{ datapath} \]

- We can stop updating \( x_i \) when \( 2^i - 1 > p \) (unrolled operator)

\[ \alpha = 1 + \lceil \log_2 ((w - 1) \times 0.5) \rceil \text{ guard bits for the } \alpha_i \text{ datapath} \]

\[ \arctan (2^{-i+1}) \]
CORDIC Iteration: Datapath Implementation

\[ p = w - 1 - \lceil \log_2 \epsilon w - 1 \rceil \text{ bits for the } x_i \text{ and } y_i \text{ datapath} \]

- We can stop updating \( x_i \) when \( 2^i - 1 > p \) (unrolled operator)

\[ g_\alpha = 1 + \lceil \log_2 \left( \left( w - 1 \right) \times 0.5 \right) \rceil \text{ guard bits for the } \alpha_i \text{ datapath} \]

- \( \arctan \left( 2^{-i+1} \right) \)
CORDIC Iteration: Accurate Datapath Implementation

\[ \tilde{x}_i = x_i + \varepsilon_i^x \]

\[ \tilde{y}_i = y_i + \varepsilon_i^y \]

\[ x_{i+1} = x_i + \varepsilon_i^x - s_i 2^{-i} \varepsilon_i^y + u_i^x \]

\[ y_{i+1} = y_i + \varepsilon_i^y - s_i 2^{-i} \varepsilon_i^y + u_i^y \]

\[ p = w - 1 - \lceil \log_2 \varepsilon_{w-1} \rceil \] bits for the \( x_i \) and \( y_i \) datapath

- we can stop updating \( x_i \) when \( 2i - 1 > p \) (unrolled operator)
CORDIC Iteration: Accurate Datapath Implementation

\[ \tilde{x}_i = x_i + \varepsilon^x_i \]

\[ \tilde{\alpha}_i = \alpha_i + \varepsilon_{\arctan}(2^{-i}) \]

\[ p = w - 1 - \left\lceil \log_2 \varepsilon_{w-1} \right\rceil \text{ bits for the } x_i \text{ and } y_i \text{ datapath} \]

\[ g_\alpha = 1 + \left\lceil \log_2((w - 1) \times 0.5) \right\rceil \text{ guard bits for the } \alpha_i \text{ datapath} \]
Hello, again, FPGAs!
Polynomial Approximations

Polynomial approximation, and their derivatives (bipartite etc.):

- the straight-forward solution for implementing **univariate** functions
- problem: area asymptotically exponential in the input width...
  for a bivariate function, we **double the input width**.
- solutions:
  - range reduction?
  - multiple consecutive **one-input** functions?
Polynomial Approximations

Polynomial approximation, and their derivatives (bipartite etc.):

- the straight-forward solution for implementing univariate functions
- problem: area asymptotically exponential in the input width...
  for a bivariate function, we double the input width.

- solutions:
  - range reduction?
  - multiple consecutive one-input functions?
Polynomial Approximations

Polynomial approximation, and their derivatives (bipartite etc.):

- the straight-forward solution for implementing univariate functions
- problem: area asymptotically exponential in the input width... for a bivariate function, we double the input width.
- solutions:
  - range reduction?
  - multiple consecutive one-input functions?
The $\frac{1}{x}$ and $\text{arctan}(x)$ Functions

\[
\text{arctan}\left(\frac{y}{x}\right) = \text{arctan}(y \times \frac{1}{x})
\]

**reciprocal** function

**arctangent** function
Range Reductions – Symmetry and Parity

$$\arctan \left( \frac{y}{x} \right) = -\frac{\pi}{2} - \arctan \left( \frac{|x|}{|y|} \right)$$
Range Reductions – Scaling

\[
\arctan \left( \frac{2^s y}{2^s x} \right) = \arctan \left( \frac{y}{x} \right)
\]

\[s = 0\]
\[s = 1\]
\[s = 2\]
\[s = 3\]

**Normalized domain**
The $\frac{1}{x}$ and $\arctan(x)$ Functions – Reduced Domain

**reciprocal** function on $[0.5, 1)$  

**arctangent** function on $[0, 1)$

Now we can evaluate them with tables, or multipartite tables, or polynomial approximators, etc.

(all available as faithful FloPoCo operators)
Reciprocal-Multiply-Arctangent
Reciprocal-Multiply-Arctangent

\[ \frac{1}{\pi} \arctan(z) \]

\[ \varepsilon_{recip} = r - \frac{1}{x} \]

\[ \varepsilon_{mult} = z - yr \]

\[ \varepsilon_{atan} = \alpha - \frac{1}{\pi} \arctan(z) \]
Reciprocal-Multiply-Arctangent

\[ \frac{1}{\pi} \arctan(z) \]

\[ \varepsilon_{\text{recip}} = r - \frac{1}{x} \]

\[ \varepsilon_{\text{mult}} = z - yr \]

\[ \varepsilon_{\text{atan}} = \alpha - \frac{1}{\pi} \arctan(z) \]
Reciprocal-Multiply-Arctangent: Datapath Dimensioning

Goal: minimize architecture cost, such that $|\varepsilon_{\text{total}}| < \text{ulp} = 2^{-w}$

$|\varepsilon_{\text{total}}| < \frac{1}{3} |\varepsilon_{\text{recip}}| + \frac{1}{3} |\varepsilon_{\text{mult}}| + |\varepsilon_{\text{atan}}|$
Delays, Delays, Delays
The *arctan* \( \left( \frac{y}{x} \right) \) Function
First Order Bi-variate Polynomial Approximation

\[ \alpha \approx T_1(x, y) = ax + by + c \]
First Order Bi-variate Polynomial Approximation

\[ \alpha \approx T_1(x, y) = ax + by + c \]
Second Order Bi-variate Polynomial Approximation

\[ \alpha \approx T_2(x, y) = ax + by + c + dx^2 + ey^2 + fxy \]

Goal:
\[ |\varepsilon_{\text{total}}| < \text{ulp} = 2^{-w} \]

\[ \varepsilon_{\text{total}} = \varepsilon_{\text{meth}} + \varepsilon_{\text{rnd}} + \varepsilon_{\text{final}} \]
Second Order Bi-variate Polynomial Approximation

\[ \alpha \approx T_2(x, y) = ax + by + c + dx^2 + ey^2 + fxy \]

Goal: \( |\varepsilon_{total}| < \text{ulp} = 2^{-w} \)

\[ \varepsilon_{total} = \varepsilon_{meth} + \varepsilon_{rnd} + \varepsilon_{final\_rnd} \]
## Comparisons: Logic-only Synthesis

<table>
<thead>
<tr>
<th>Bitwidth</th>
<th>LUT</th>
<th>Latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CORDIC</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>173</td>
<td>9.3</td>
</tr>
<tr>
<td>12</td>
<td>435</td>
<td>14.6</td>
</tr>
<tr>
<td>16</td>
<td>734</td>
<td>19.7</td>
</tr>
<tr>
<td>24</td>
<td>1504</td>
<td>31.0</td>
</tr>
<tr>
<td>32</td>
<td>2606</td>
<td>43.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bitwidth</th>
<th>LUT</th>
<th>Latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Taylor degree 1</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>207</td>
<td>12.64</td>
</tr>
<tr>
<td>12</td>
<td>1258</td>
<td>14.74</td>
</tr>
<tr>
<td>16</td>
<td>37744</td>
<td>20.20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bitwidth</th>
<th>LUT</th>
<th>Latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Taylor degree 2</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>356</td>
<td>13.72</td>
</tr>
<tr>
<td>12</td>
<td>469</td>
<td>14.75</td>
</tr>
<tr>
<td>16</td>
<td>1509</td>
<td>17.90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bitwidth</th>
<th>Method</th>
<th>LUT</th>
<th>Latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RecipMultAtan</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>degree 0</td>
<td>175</td>
<td>11.8</td>
</tr>
<tr>
<td>12</td>
<td>degree 0</td>
<td>683</td>
<td>16.2</td>
</tr>
<tr>
<td>12</td>
<td>degree 1</td>
<td>443</td>
<td>19.0</td>
</tr>
<tr>
<td>16</td>
<td>degree 1</td>
<td>1049</td>
<td>19.1</td>
</tr>
<tr>
<td>24</td>
<td>degree 2</td>
<td>2583</td>
<td>35.2</td>
</tr>
<tr>
<td>32</td>
<td>degree 2</td>
<td>6190</td>
<td>40.7</td>
</tr>
<tr>
<td>32</td>
<td>degree 3</td>
<td>5423</td>
<td>50.8</td>
</tr>
</tbody>
</table>
Logic-only Synthesis: Area

![Graph showing the comparison of different methods for fixed-point Atan2 implementation based on area consumption. The methods compared are Taylor 1, Taylor 2, CORDIC, and RecipMultAtan. The graph plots the number of LUTs against bitwidth, showing how the area requirements grow with increasing bitwidth for each method.]
Logic-only Synthesis: delay

![Graph comparing latency (ns) vs bitwidth for different algorithms: CORDIC, Taylor 1, Taylor 2, and RecipMultAtan. The graph shows a clear trend with latency increasing as bitwidth increases for all algorithms.](image-url)
# Comparisons: 16-bit Pipelined Architectures

<table>
<thead>
<tr>
<th>Method</th>
<th>LUT + Reg.</th>
<th>BRAM + DSP</th>
<th>Speed cycles@freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORDIC</td>
<td>816 + 44</td>
<td>0+0</td>
<td>2@191</td>
</tr>
<tr>
<td></td>
<td>799 + 202</td>
<td></td>
<td>5@274</td>
</tr>
<tr>
<td></td>
<td>796 + 336</td>
<td></td>
<td>8@389</td>
</tr>
<tr>
<td>RecipMultAtan 1</td>
<td>320 + 51</td>
<td>2+1</td>
<td>2@112</td>
</tr>
<tr>
<td></td>
<td>315 + 68</td>
<td></td>
<td>3@199</td>
</tr>
<tr>
<td>RecipMultAtan 2</td>
<td>425 + 199</td>
<td>0+5</td>
<td>10@130</td>
</tr>
<tr>
<td></td>
<td>432 +250</td>
<td></td>
<td>14@253</td>
</tr>
<tr>
<td>Taylor degree 2</td>
<td>331 + 53</td>
<td>4+6</td>
<td>1@135</td>
</tr>
<tr>
<td></td>
<td>327 + 103</td>
<td></td>
<td>3@144</td>
</tr>
<tr>
<td></td>
<td>329 + 140</td>
<td></td>
<td>5@220</td>
</tr>
</tbody>
</table>
Conclusions

Very unlike "Fixed-Point Trigonometric Functions on FPGAs“ (in HEART 2013)

CORDIC?
  • efficient
  • scales well

Polynomial Approximations?
  • limited by memory requirements
  • no unique optimal solution
  • Could it be saved by better bivariate polynomial approximation?
Questions?

Thank you for your attention!

Questions?
Hidden Frame
First Method: CORDIC
The CORDIC Iteration: Datapath Dimensioning

**The $x_i$ datapath:**

\[
\tilde{x}_{i+1} = x_i + \varepsilon_i^x - s_i 2^{-i} \varepsilon_i^y + u_i^x
\]

\[
\tilde{x}_{i+1} = x_i - s_i 2^{-i} y_i + u_i^x
\]

\[
= x_i + \varepsilon_i^x - s_i 2^{-i} (y_i + \varepsilon_i^y) + u_i^x
\]

\[
= x_{i+1} + \varepsilon_i^x - s_i 2^{-i} \varepsilon_i^y + u_i^x
\]

- using the error bound $\bar{\varepsilon}_i$ for $\varepsilon_i^x$ and $\varepsilon_i^y$, and $u_i^x < 2^{-p}$

\[
\bar{\varepsilon}_{i+1} = \bar{\varepsilon}_i (1 + 2^{-i}) + 2^{-p}
\]

\[
p = w - 1 - \lceil \log_2 \bar{\varepsilon}_{w-1} \rceil \text{ bits for the } x_i \text{ and } y_i \text{ datapath}
\]

- *we can stop updating $x_i$ when $2^i - 1 > p$ (useful because unrolled operator)*
The CORDIC Iteration: Datapath Dimensioning (2)

The $\alpha_i$ datapath:

- $\varepsilon_{\text{atan}(2^{-i})} = 2^{-p_\alpha} - 1$ (or 0.5 $ulp$ on the $p_\alpha$ precision)
- $\varepsilon_{\text{final\_round}} = 2^{-w}$

\[ g_\alpha = 1 + \lceil \log_2((w - 1) \times 0.5) \rceil \]

extra guard bits needed
**Goal:** \( |\varepsilon_{total}| < ulp = 2^{-w} \)

\[
\varepsilon_{total} = \varepsilon_{meth} + \varepsilon_{final\_rnd} + \varepsilon_{rnd}
\]

- **method error:** \( \varepsilon_{meth} \)
  - due to neglected terms of Taylor series
  - constraint on \( k \implies k \geq \lceil \frac{w+1}{3} \rceil \)
- **rounding errors:** \( \varepsilon_{rnd} = \varepsilon_a \delta x + \varepsilon_b \delta y + \varepsilon_c + \varepsilon_d \delta x^2 + \varepsilon_e \delta y^2 + \varepsilon_f \delta x \delta y \)
  - depends on \( \delta x, \delta y \implies \varepsilon_{round} \) depends on \( k \)
  - \( \varepsilon_{method} + \varepsilon_{round} < 2^{-w-1} \)

\( \implies \) number of **guard bits** \( g \) \iff **compromise** between size of tables and size of multipliers
The arctan \( \frac{y}{x} \) Function