Floating Point CORDIC-based Architecture for Powering Computation

Joshua Mack, Sam Bellestri, Daniel Llamocca
jmack2545@email.arizona.edu sdbellestri@crimson.ua.edu llamocca@oakland.edu

Abstract
This work presents an architecture for powering computation in floating point arithmetic that is based on an expanded hyperbolic CORDIC algorithm, where the user can select the 2-D domain of convergence that suits their application. The fully parameterized hardware implementation allows us to explore trade-offs among design parameters (numerical format, number of iterations), resource usage, accuracy, and execution time. We carry out an exhaustive design space exploration and generate Pareto-optimal realizations in the resource-accuracy space. Our approach allows us to select optimal hardware realizations that meet or exceed accuracy requirements.

Key Contributions
• Design Space Exploration
• Pareto-Optimal Realizations based on accuracy and resource usage
• Fully customizable architecture validated on an FPGA

Methodology and Architectures
Hyperbolic CORDIC provides two modes of operation (rotation and vectoring) that allow for the direct computation of cosh x, sinh x, tanh−1 y, and eλ. By combining the identities ln(x) = 2 tanh−1 x and xλ = eλ ln x, we can calculate xλ.

The expanded hyperbolic CORDIC algorithm is given by:

For i ≤ 0:

X_i+1 = X_i + δ_i Y_i (1 − 2i^2)
Y_i+1 = Y_i + δ_i X_i (1 − 2i^2)
Z_i+1 = Z_i − δ_i tanh x_i (1 − 2i^2)

For i > 0:

X_i+1 = X_i + δ_i Y_i Z_i^2
Y_i+1 = Y_i + δ_i X_i Z_i^2
Z_i+1 = Z_i − δ_i tanh x_i (2i^2)

Rotation: δ_i = 1 if x_i < 0, −1 else
Vectoring: δ_i = −1 if x_i y_i ≥ 0, 1 else

Using CORDIC, we implement a fully customizable xλ engine in VHDL.
CORDIC has specific advantages in hardware due to its shift and add nature. The xλ architecture executes two consecutive operations.

1. Load x₀ = x + 1, y₀ = x − 1, z₀ = 0 onto the CORDIC engine in vectoring mode, so that x₀ = 0.5 ln x. A floating-point shifter generates ln x and a floating-point multiplier computes y₀ x which is fed back to the CORDIC engine for the second operation.

2. Load x₀ = y₀ = 1, z₀ = y₀ ln x onto the CORDIC engine in rotation mode so that y₀ = eλ ln x = xλ.

Figure 1: Expanded Hyperbolic CORDIC Architecture.

Figure 2: Full Architecture of xλ Implementation.

Setup and Design Space Exploration

<table>
<thead>
<tr>
<th>R</th>
<th>FW</th>
<th>PW</th>
<th>Min</th>
<th>Max</th>
<th>Dyn. Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>6</td>
<td>9</td>
<td>3.913 x 10^-18</td>
<td>4.291 x 10^-10</td>
<td>573 dB</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>13</td>
<td>3.913 x 10^-18</td>
<td>4.295 x 10^-10</td>
<td>373 dB</td>
</tr>
<tr>
<td>30</td>
<td>7</td>
<td>16</td>
<td>2.168 x 10^-18</td>
<td>1.945 x 10^-10</td>
<td>759 dB</td>
</tr>
<tr>
<td>40</td>
<td>7</td>
<td>20</td>
<td>2.168 x 10^-18</td>
<td>1.945 x 10^-10</td>
<td>759 dB</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>23</td>
<td>1.375 x 10^-18</td>
<td>3.603 x 10^-10</td>
<td>1529 dB</td>
</tr>
<tr>
<td>60</td>
<td>9</td>
<td>26</td>
<td>3.455 x 10^-17</td>
<td>1.510 x 10^-10</td>
<td>6371 dB</td>
</tr>
<tr>
<td>70</td>
<td>9</td>
<td>30</td>
<td>3.455 x 10^-17</td>
<td>1.510 x 10^-10</td>
<td>6371 dB</td>
</tr>
<tr>
<td>80</td>
<td>9</td>
<td>34</td>
<td>3.455 x 10^-17</td>
<td>1.510 x 10^-10</td>
<td>6371 dB</td>
</tr>
<tr>
<td>90</td>
<td>10</td>
<td>37</td>
<td>2.983 x 10^-16</td>
<td>1.341 x 10^-10</td>
<td>6153 dB</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>41</td>
<td>2.983 x 10^-16</td>
<td>1.341 x 10^-10</td>
<td>6153 dB</td>
</tr>
<tr>
<td>110</td>
<td>11</td>
<td>44</td>
<td>2.225 x 10^-16</td>
<td>7.190 x 10^-11</td>
<td>6218 dB</td>
</tr>
<tr>
<td>120</td>
<td>11</td>
<td>48</td>
<td>2.225 x 10^-16</td>
<td>7.190 x 10^-11</td>
<td>6218 dB</td>
</tr>
</tbody>
</table>

Table 1: Floating Point Formats Tested
CORDIC does not converge for all values of x and y.

If chosen values of x and y are bounded by the given corresponding curve for a chosen value of M, the expanded CORDIC algorithm will converge.

In our testing, we choose a range of test points evenly distributed within this range for M = 5.

For our accuracy metric, we use PSNR, defined as $PSNR (dB) = 10 \log_{10} \left( \frac{\text{max}^2}{\text{MSE}} \right)$

Results
The execution times in Table 2 and the number of slices in Figure 5 are for a Xilinx® Zynq-7000 XC7Z010-1CLG400 SoC running at 125 MHz.

The Pareto front shows the optimal hardware profiles for our xλ architecture. We note that 44 bits with 32 iterations provides the highest accuracy at the expense of a large amount of resource usage. In addition, we consider the Pareto point circled in blue to have too poor of accuracy to be considered.

As a result, the case of 28 bits with 16 iterations provides the smallest hardware implementation that gives a usable architecture at the expense of lower accuracy.

If we restrict to accuracy ≥ 100dB, 32 bits with 20 iterations provides the implementation with minimum resources.

Table 2: Execution Time (μs) vs N for xλ Architecture.

Figure 3: Range of Convergence Plot for xλ

Figure 4: xλ Architecture - Peak Signal-to-Noise Ratio (PSNR) vs. Number of Iterations.

Figure 5: xλ Resources vs. PSNR with Pareto Front

Conclusion
A fully parameterized floating point iterative architecture for xλ was presented and thoroughly validated. Floating point arithmetic features high accuracy and large dynamic range at the expense of resources. The expanded CORDIC approach allows for customized bounds on the domain of xλ. We extracted the Pareto-optimal set of architectures from the multi-objective design space. Further work will explore other arithmetic representations and enhanced versions of the expanded CORDIC algorithm such as scale-free hyperbolic CORDIC that requires fewer iterations for the same region of convergence.

This work was sponsored in part by the National Science Foundation (NSF) under award number ECC-1263133.