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 $\cosh x$, $\sinh x$, $\tanh^{-1} x$, and e^x. By combining the identities $\ln(x) = 2\tanh^{-1}\frac{x-1}{x+1}$ and $x^y = e^{y\ln(x)}$, we can calculate x^y. The expanded hyperbolic CORDIC algorithm is given by:

$$
\begin{aligned}
&For\ i \le 0: \\
&\quad X_{i+1} = X_i + \delta_i Y_i\left(1-2^{i-2}\right) \\
&\quad Y_{i+1} = Y_i + \delta_i X_i\left(1-2^{i-2}\right) \\
&\quad Z_{i+1} = Z_i - \delta_i tanh^{-1}\left(1-2^{i-2}\right) \\
&For\ i > 0: \\
&\quad X_{i+1} = X_i + \delta_i Y_i 2^{-i} \\
&\quad Y_{i+1} = Y_i + \delta_i X_i 2^{-i} \\
&\quad Z_{i+1} = Z_i - \delta_i tanh^{-1}(2^{-i})
\end{aligned}
$$

$$
\begin{aligned}
&Rotation:\ \delta_i = -1\ if\ z_i < 0, \\
&\qquad\qquad\quad +1\ else \\
&Vectoring:\ \delta_i = -1\ if\ x_i y_i \ge 0, \\
&\qquad\qquad\qquad +1\ else
\end{aligned}
$$

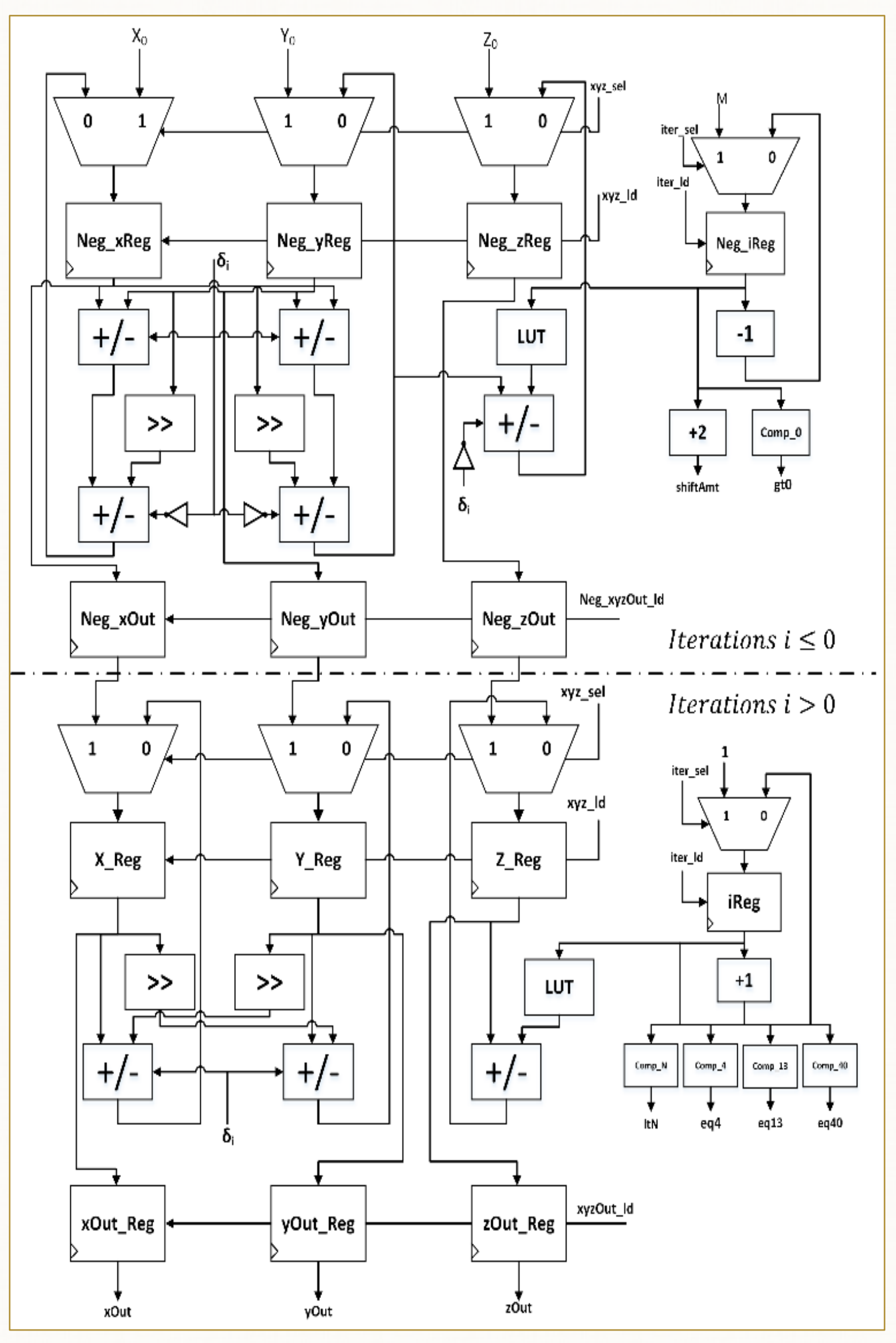

Figure 1 : Expanded Hyperbolic CORDIC Architecture.

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

1. Load $x_0 = x+1, y_0 = x-1, z_0 =$ 0 onto the CORDIC engine in vectoring mode, so that $z_n =$ $0.5\ln x$. A floating-point shifter generates $\ln x$ and a floating-point multiplier computes $y \ln x$ which is fed back to the CORDIC engine for the second operation.
2. Load $x_0 = y_0 = 1/A_n$ and $z_0 =$ $y\ln x$ onto the CORDIC engine in rotation mode so that $y_n = e^{y\ln x} = x^y$.

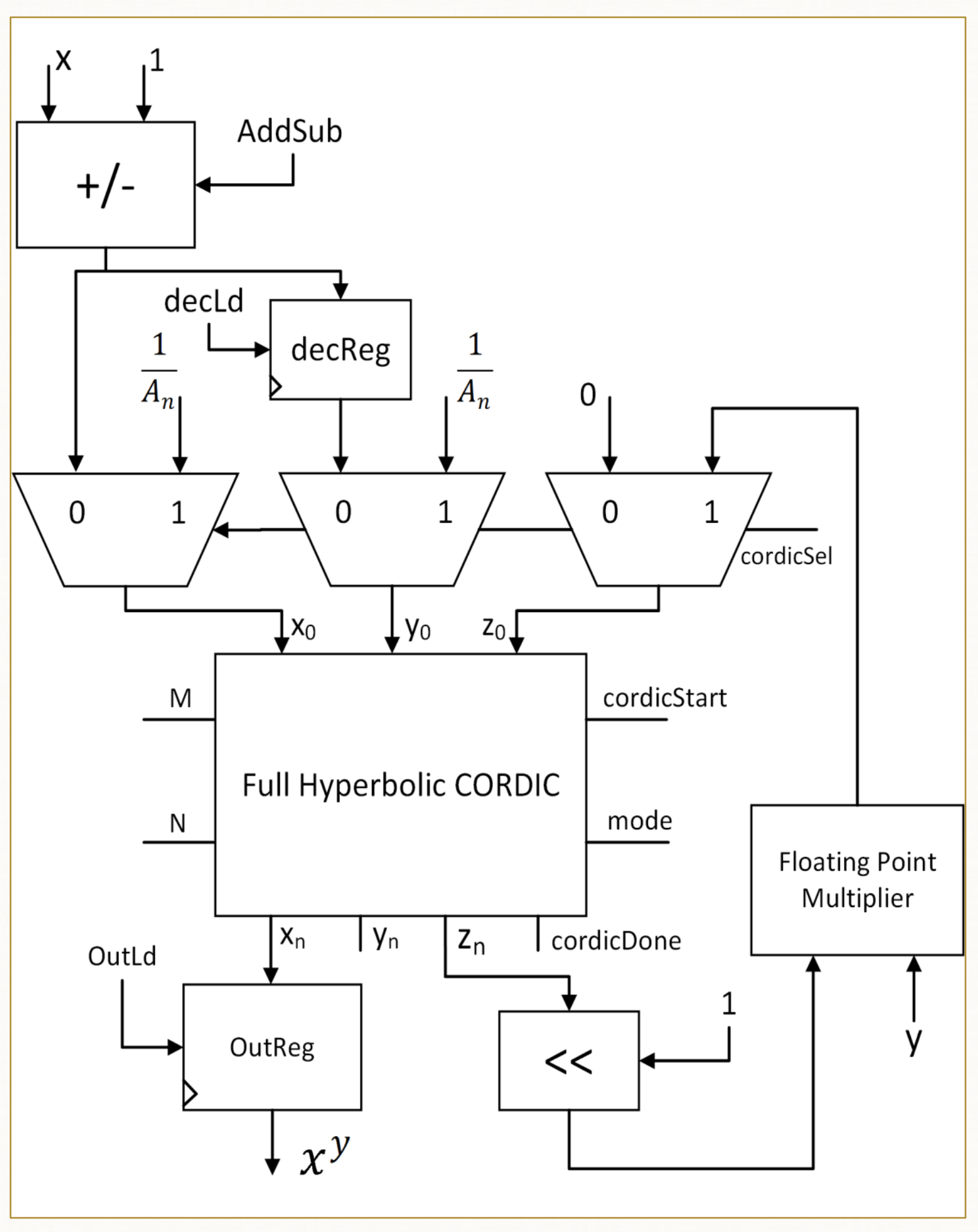

Figure 2 : Full Architecture of x^y Implementation.

Setup and Design Space Exploration

B	EW	FW	Min	Max	Dyn. Range
16	6	9	9.313×10^{-10}	4.291×10^{9}	373 dB
20	6	13	9.313×10^{-10}	4.295×10^{9}	373 dB
24	7	16	2.168×10^{-19}	1.845×10^{19}	759 dB
28	7	20	2.168×10^{-19}	1.845×10^{19}	759 dB
32	8	23	1.175×10^{-38}	3.403×10^{38}	1529 dB
36	9	26	3.455×10^{-77}	1.158×10^{77}	3071 dB
40	9	30	3.455×10^{-77}	1.158×10^{77}	3071 dB
44	9	34	3.455×10^{-77}	1.158×10^{77}	3071 dB
48	10	37	2.983×10^{-154}	1.341×10^{154}	6153 dB
52	10	41	2.983×10^{-154}	1.341×10^{154}	6153 dB
56	11	44	2.225×10^{-308}	1.798×10^{308}	12318 dB
60	11	48	2.225×10^{-308}	1.798×10^{308}	12318 dB
64	11	52	2.225×10^{-308}	1.798×10^{308}	12318 dB

Table 1 : Floating Point Formats Tested

A complete design space exploration was performed by varying parameters such as number representation and number of iterations. The floating point formats tested are listed to the left. Each given format was tested with M = 5, and N from 8, 12, …, 52.

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.

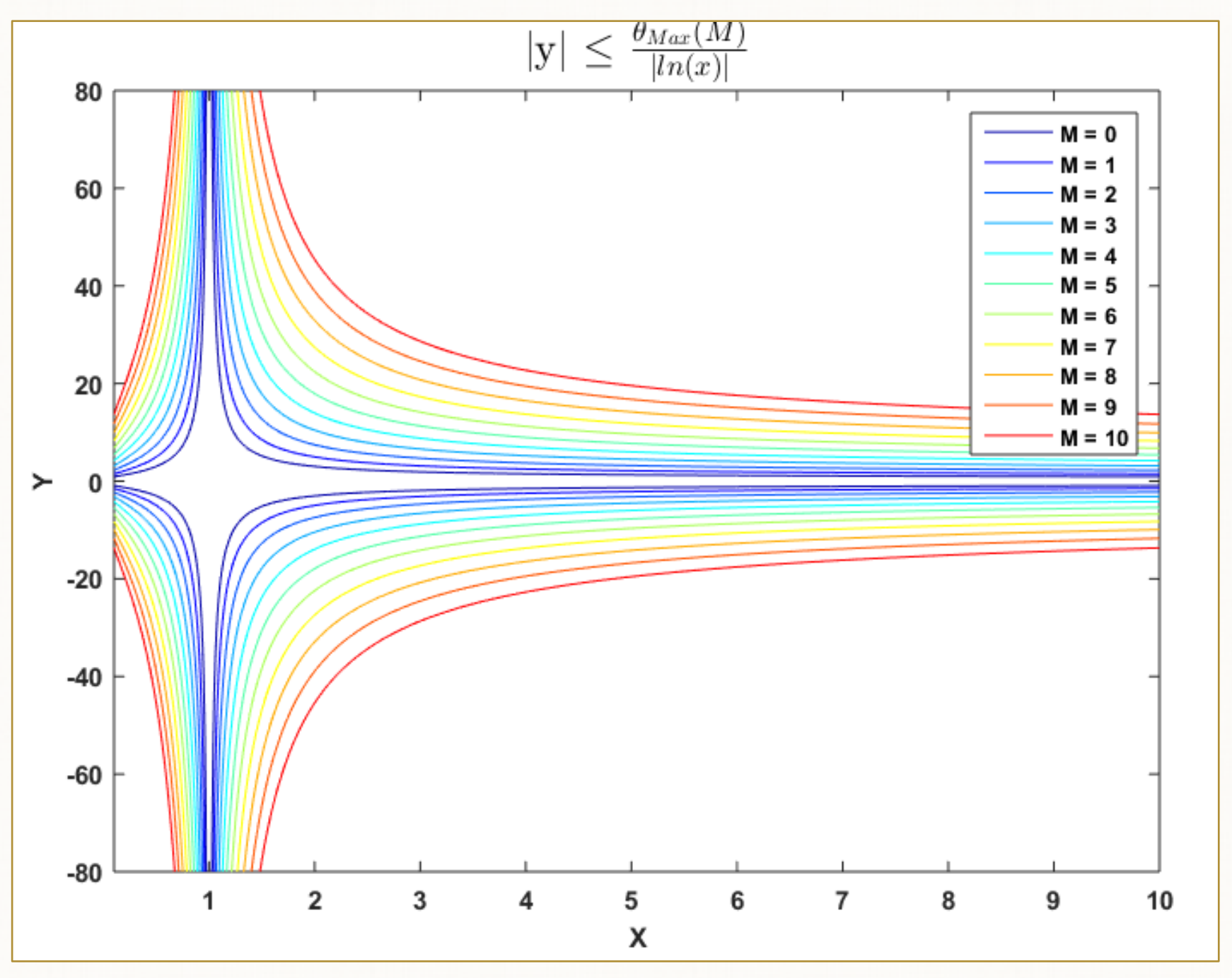

Figure 3: Range of Convergence Plot for x^y

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

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.

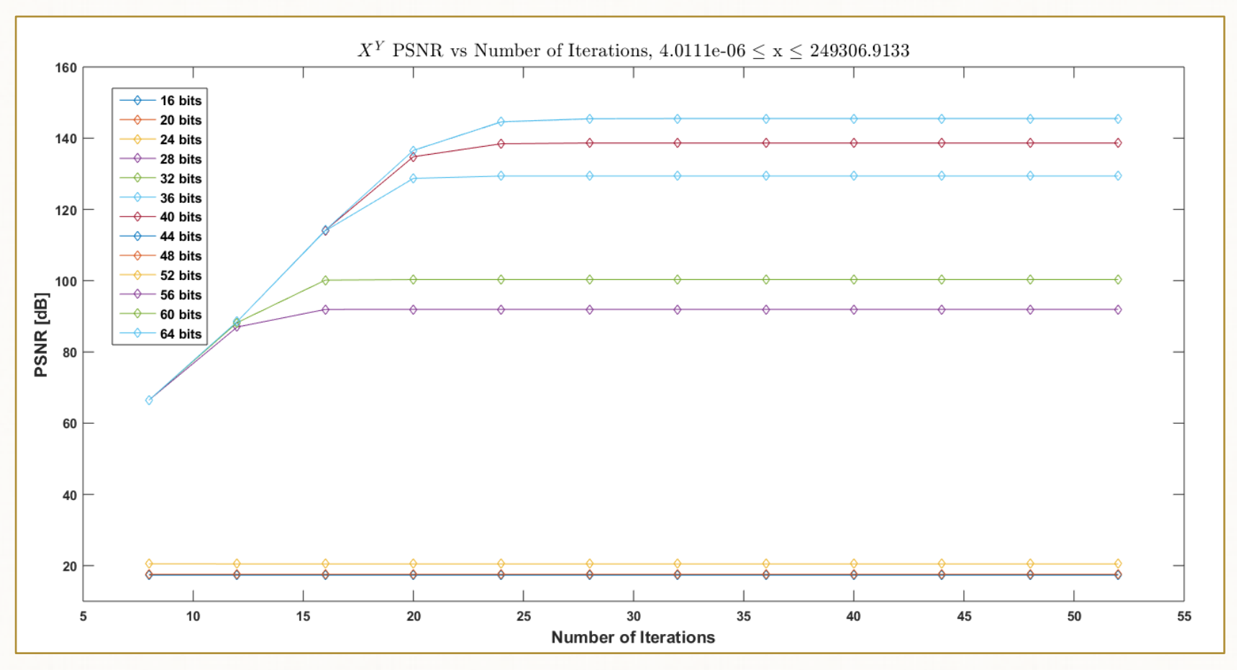

Figure 4 : x^y Architecture - Peak Signal-to-Noise Ratio (PSNR) vs. Number of iterations.

Function	Number of Iterations (N)						
	8	12	16	20	24	32	52
x^y	0.288	0.352	0.432	0.496	0.560	0.688	1.024

Table 2: Execution Time (μs) vs N for x^y Architecture.

The Pareto front shows the optimal hardware profiles for our x^y 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 $\ge 100dB$, 32 bits with 20 iterations provides the implementation with minimum resources.

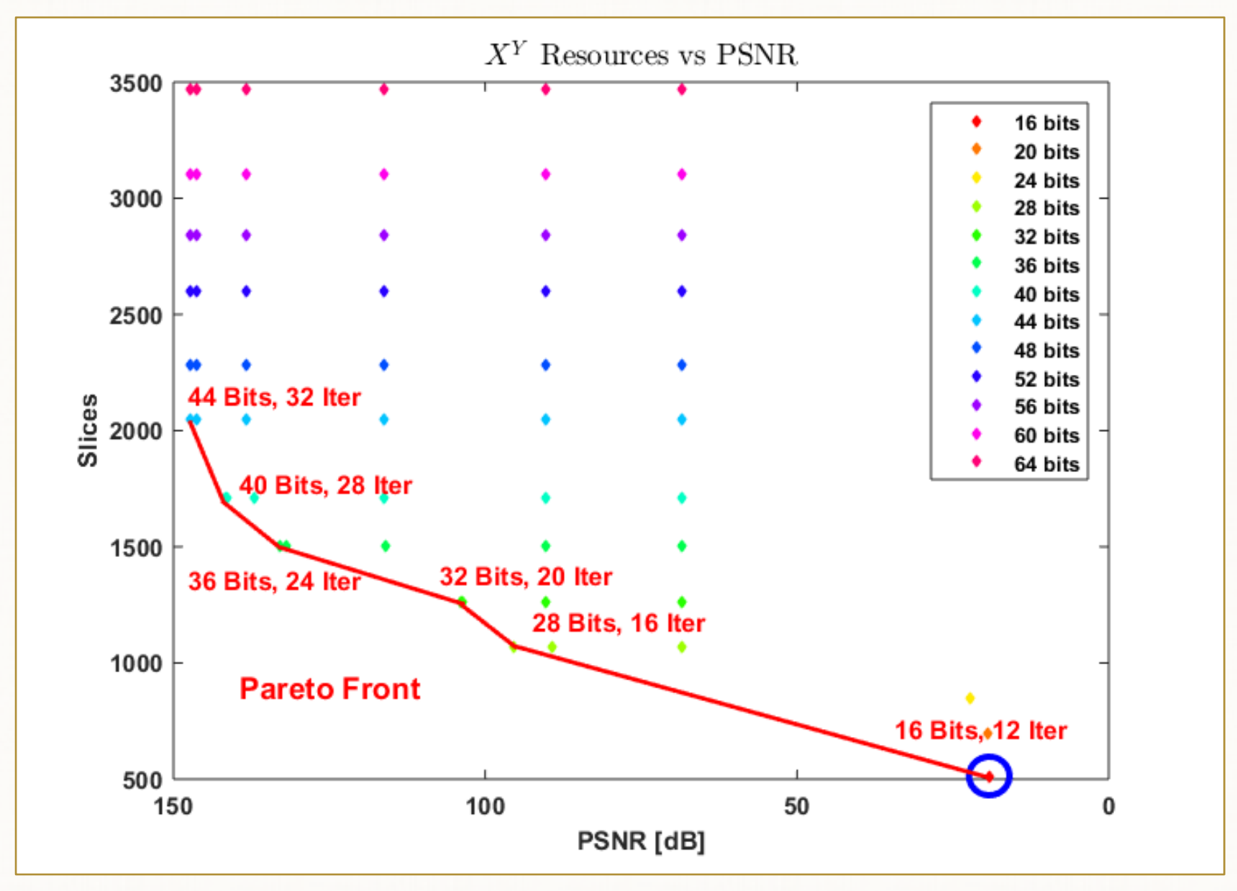

Figure 5: x^y Resources vs. PSNR with Pareto Front

Conclusion

A fully parameterized floating point iterative architecture for x^y 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 x^y. 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 EEC-1263133.