Implementation of Hyperbolic CORDIC with AXI Full Interface

Sagar Vaidya
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mail: sagarvaidya@oakland.edu

***Abstract* — CORDIC stands for COORDINATE ROTATION DIGITAL COMPUTER. There are three basic CORDIC algorithms including Circular CORDIC, Linear CORDIC, Hyperbolic CORDIC.**

Hyperbolic CORDIC is used to compute hyperbolic functions in efficient and fast way.

I. INTRODUCTION

The hyperbolic CORDIC algorithm allows computation of hyperbolic functions. Basic hyperbolic CORDIC algorithm is a fixed-point architecture with an expanded range of convergence, and a scale-free fixed-point hardware.

It can be used in powering architecture generally that takes high cost of computation where FPGAs can be a solution if designed efficiently.

This project report will mainly focus on design of a hyperbolic CORDIC design and verification.

II. METHODOLOGY

A. This report presents hardware design for the hyperbolic CORDIC algorithm with simple range of convergence.

B. The reference of Circular CORDIC is used from the course tutorials and extended it to implement hyperbolic functions.

This is extension to the original CORDIC equations that allows for the computation of hyperbolic functions, where i is the index of the iteration (i = 1, 2, 3, …). The following iterations must be repeated to guarantee convergence: i = 4, 13, 40, … , k, $3k$ + 1.

$$x_{i+1} = x_i - \delta_i x_i 2^{-i}$$
$$y_{i+1} = y_i - \delta_i x_i 2^{-i}$$
$$z_{i+1} = z_i + \delta_i \theta_i,\ \theta_i = tanh^{-1}(2^{-i})$$

$$Rotation: \delta_i = +1\ if\ z_i < 0;\ -1, otherwise$$
$$Vectoring: \delta_i = +1\ if\ x_i y_i \geq 0;\ -1, otherwise$$

Depending on the mode of operation, the quantities X, Y and Z converge to the following values, for sufficiently large N:

Rotation Mode	Vectoring Mode
$x_n = A_n(x_1 coshz_1 + y_1 sinhz_1)$ $y_n = A_n(y_1 coshz_1 + x_1 sinhz_1)$ $z_n = 0$	$x_n = A_n\sqrt{x_1^2 - y_1^2}$ $y_n = 0$ $z_n = z_1 + tanh^{-1}(y_1/x_1)$

$A_n \leftarrow \prod_{i=1}^{N}\sqrt{1-2^{-2i}}$ (this includes the repeated iterations $i = 4, 13, 40, ...$,). For $N \rightarrow \infty$, $A_n \cong 0.8$

III. EXPERIMENTAL SETUP

With a proper choice of the initial values $x1$, $y1$, $z1$ and the operation mode, the above functions can be directly computed.

IV. RANGE OF CONVERGENCE

The basic range of convergence, obtained by a method developed by X. Hu et al, "Expanding the Range of Convergence of the CORDIC Algorithm", results in:

Rotation Mode:	$\lvert z_{in}\rvert \leq \theta_N + \sum_{i=i_{in}}^{N} \theta_i$	▫ Circular: $i_{in} = 0$, $z_{in} = z_0$, $\alpha_{in} = tan^{-1}(y_0/x_0)$ ▫ Linear: $i_{in} = 1$, $z_{in} = z_1$, $\alpha_{in} = y_1/x_1$ ▫ Hyperbolic: $i_{in} = 1$, $z_{in} = z_1$, $\alpha_{in} = tanh^{-1}(y_1/x_1)$. Note that in the summation, we must repeat the terms $i = 4, 13, 40,$
Vectoring Mode:	$\lvert\alpha_{in}\rvert \leq \theta_N + \sum_{i=i_{in}}^{N} \theta_i$	

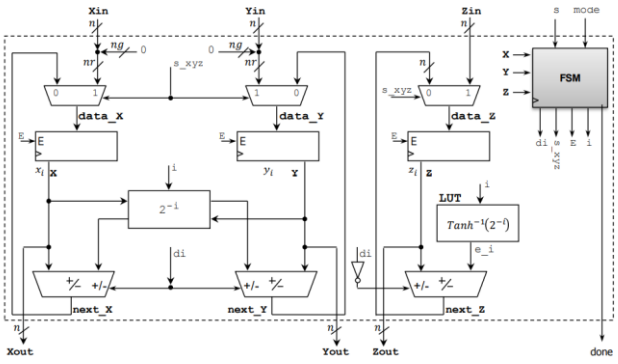
▪ **Hyperbolic**: $\theta_N + \sum_{i=1}^{N}\theta_i = tanh^{-1}(2^{-N}) + \sum_{i=1}^{N} tanh^{-1}(2^{-i}) = 1.182\ (N \rightarrow \infty)$

Rotation	$\lvert z_1\rvert \leq 1.182$	This is the limitation imposed to the input argument of the hyperbolic functions. Note that the full domain of the functions $sinh$ and $cosh$ is $(-\infty, \infty)$.
Vectoring	$\lvert tanh^{-1}(y_1/x_1)\rvert \leq 1.182 \rightarrow \lvert y_1/x_1\rvert \leq 0.807$	This is the limitation imposed to the ratio of the input arguments of the hyperbolic functions. Note that the domain of $tanh^{-1}$ is $(-1,1)$.

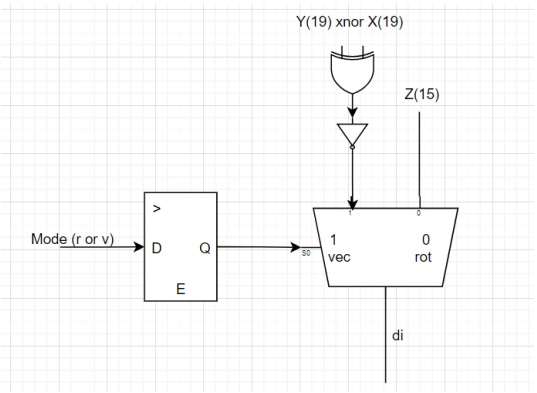
V. HARDWARE BLOCK DIAGRAM

Here the LUT holds the $\theta i = tanh -1\ (2 -i\ )$ values for i = 1,2, … , N.

The FSM is more complex as it has to account for the repeated iterations. After $N - 1 + v$ (v: # of repeated iterations) clock cycles, the result is obtained in the registers X, Y and Z, and a new process can be started.



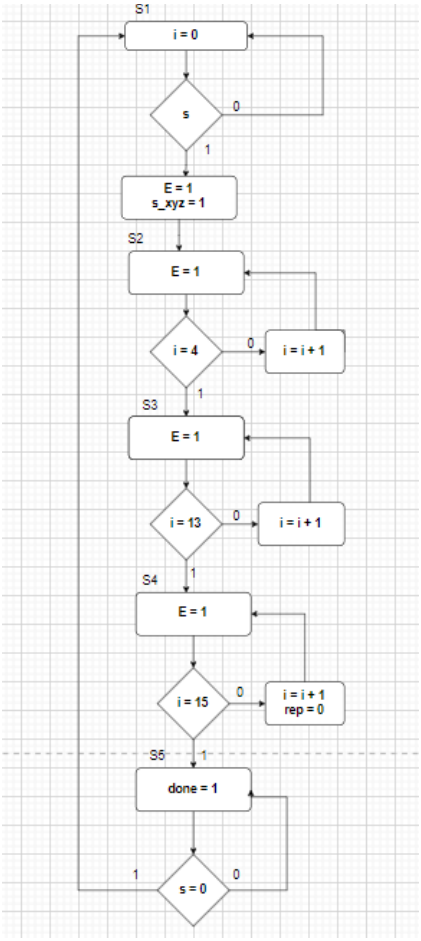
V. CONTROL DIAGRAM



The control diagram above depicts mode selection of the hyperbolic CORDIC based on assignment of input data in the word.

Vectoring and Rotating modes are selected based on MUX inputs.

VIII. FSM



This is the state machine representation of the Hyperbolic CORDIC.

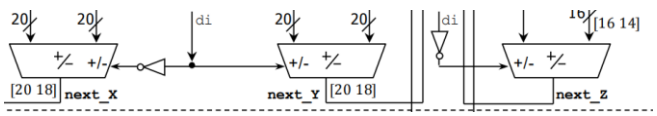
This FSM implements 16 iterations starting from 1st (not 0th).

Important part of this FSM is that it has to repeat iterations 4 and 13 as per the hyperbolic CORDIC algorithm. Hence, i = 4 and i = 13 conditions repeat the state S2 and S3 respectively. When i becomes 15, it goes back to state S1.

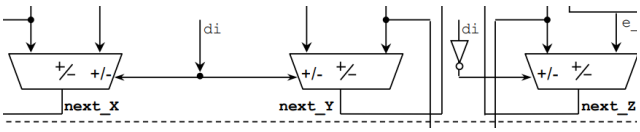
X. Few other differences

- Adder/Subtractor Logic

Circular CORDIC implementation is as below –

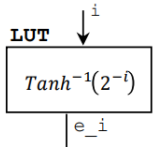


Hyperbolic CORDIC implementation is as below –



This shows the difference between add/sub selector bit 'di' for x, y, z.

- Look-up Table (LUT)

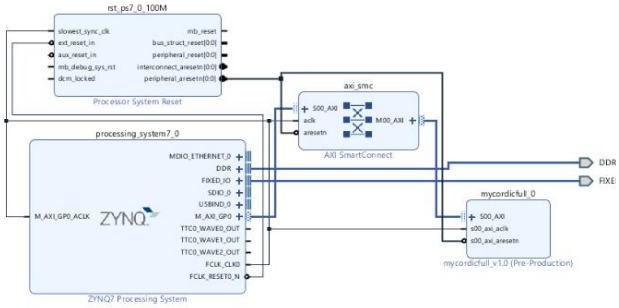


This is the LUT used for hyperbolic CORDIC that has 16 (0r 15) iterations of tanh outputs.

V. Generation of AXI Full Interface and Integration

AXI Full interface is generated as per the course tutorial by generating AXI full IP and repackaging it.
This IP is then used in creation of block design for hyperbolic CORDIC and generating VHDL source.
It is then synthesized, implemented and generated bitstream.

V. Control Diagram



This is the block design that is generated in Vivado project by importing AXI interface IP generated (mycordicfull) and Zybo board.

V. Software Routine

Software routine is implemented in SDK to verify the functionality of the design on Zybo board.
Steps involved are as below –

- Initialize base address based on AXI base address
- Write first set of input data using AXI interface write API defined in the design
- This includes writing 2 words sequentially
- Add a small delay to process the data
- Read 2 words using AXI interface read API defined in the design
- Repeat these steps for additional sets of inputs and outputs
- Build the SDK, connect the board and run on the hardware to see the output in serial console

V. Simulation of AXI Full Interface

Generated AXI interface is then simulated in Vivado by adding the testbench for simulation that uses AXI read/write
APIs on the input/output dataset.
Below if the result of AXI simulation.

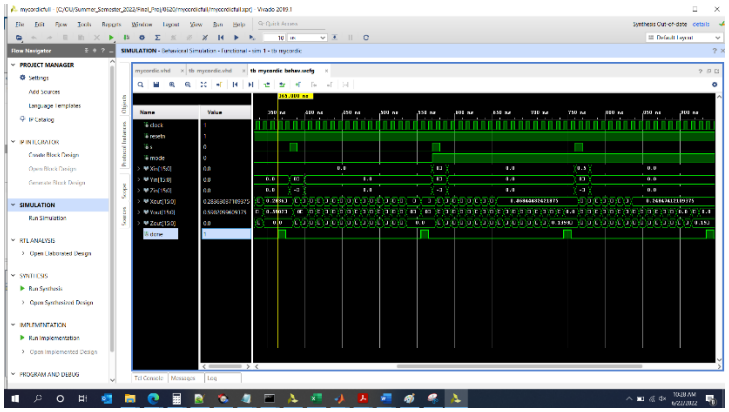
- Read 2 words using AXI interface read API defined in the design
- Repeat these steps for additional sets of inputs and outputs
- Build the SDK, connect the board and run on the hardware to see the output in serial console

V. Data Set

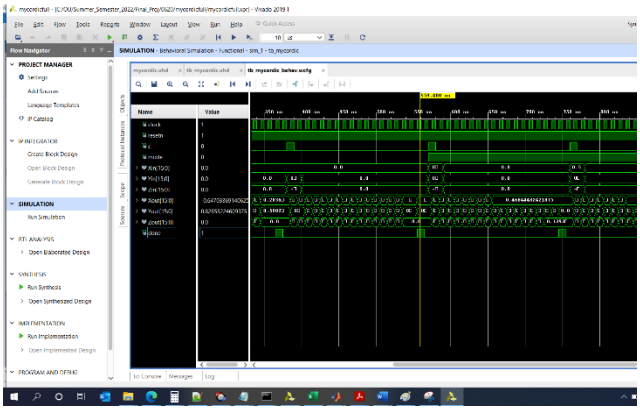
		Input Data			Output Data	
An=0.8	x0	y0	z0	xN	yN	zN
M=0	0	(1/2An)2800	(pi/6)2182	(0.273927)1188	(0.570119)247C	0
	0	(1/2An)2800	(-pi/3)BCFA	(-0.624684)D806	(0.800143)3335	0
M=1	(0.9)3999	(0.7)2CCC	(-0.9)(C667)	(0.452548)1CF6	0	(0.139721)08F1
	(0.5)2000	(0.4)1999	s	(0.24)0F5C	0	(0.198612)0CB6

This is the dataset that is simulated in below output of the simulation.
Generated AXI interface is then simulated in Vivado by adding the testbench for simulation that uses AXI read/write
APIs on the input/output dataset.
Below if the result of AXI simulation.

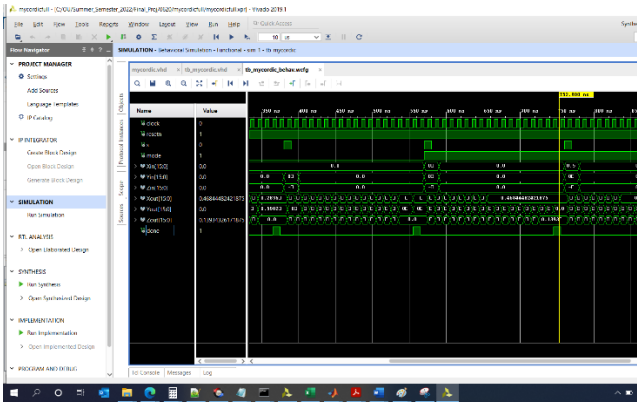
Rotating mode dataset 1:



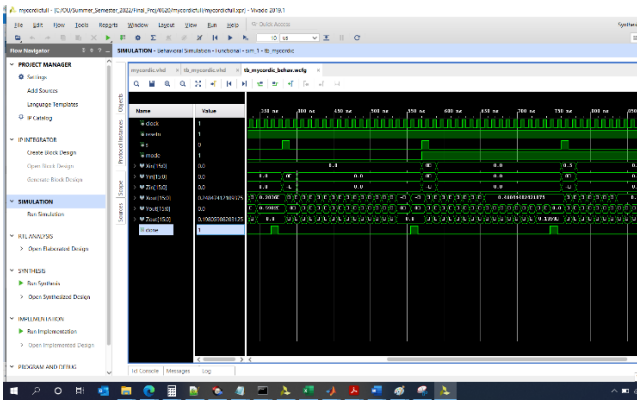
Rotating mode dataset 2:



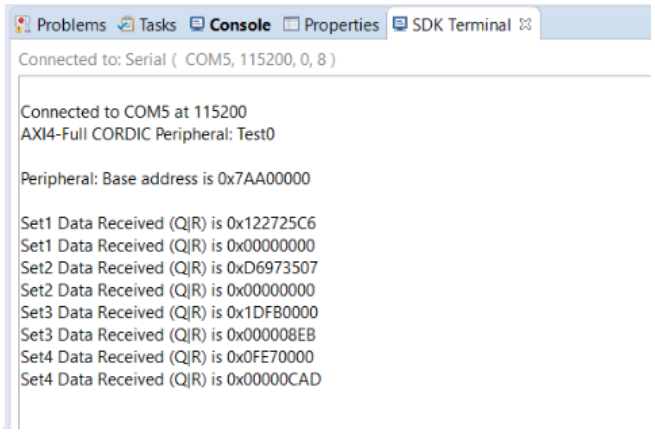
Vectoring mode dataset 1:



Vectoring mode dataset 2:



On running the test program in SDK, below output is displayed in the serial console of the hardware –



This verifies that the output of the hardware hyperbolic CORDIC design and the calculated output in the above dataset match with very little deviation.

V. Conclusion

Designing a hyperbolic CORDIC hardware software interface needs good mathematics and electronics knowledge.

These mathematical operators and its VHDL implementation is useful in different applications, such as computing (x^y) powering formulae, sinh, cosh, tanh of the given angles, logarithmic computations and many more.

Components implemented in VHDL are generic and
can be adapted easily to any application listed but not limited to the above ones.

V. References

[1] Digital Library - Arithmetic Cores (oakland.edu)

[2] https://moodle.oakland.edu/pluginfile.php/7667929/mod_resource/content/5/Notes%20-%20Unit%203.pdf

[3] Circular CORDIC implementation with AXI Full interface from Lab 3 assignment