Generation Of Trignometric Functions Using CORDIC

Guhan Rajasekar(22410), Harivignesh(23409), DESE, IISc

I INTRODUCTION AND MOTIVATION

- This report is a summary on generation of various trigonometric functions using CORDIC (*Coordinate Rotation Digital Computer*)algorithm.
- Motivation behind using this algorithm is to demonstrate the feasibility of generating various mathematical functions in a hardware friendly manner. When we say hardware friendly, we mean avoiding usage of multiplier blocks of FPGA and generating the functions using shift and add operations.
- There are other methods to generate trigonometric functions like *Taylor's Series Expansion , Look Up Table Based Generation etc*. But these techniques are not hardware friendly and are very resource intensive. Hence this provides us clear motivation to use CORDIC to implement the said functionalities with economic usage of the available hardware resources.
- In this project, we demonstrate the implementation of sin, cos, tan, tan inverse, tan hyperbolic and tan hyperbolic inverse functions with CORDIC.

II BACK GROUND STUDY

2.1 Generalized Equations Of CORDIC

- The generalized equations of CORDIC are given as :

$$x^{(i+1)} = x^{(i)} - \mu d_i(2^{-i}y^{(i)}) \tag{1}$$

$$y^{(i+1)} = y^{(i)} + d_i(2^{-i}x^{(i)}) \tag{2}$$

$$z^{(i+1)} = z^{(i)} - d_i e^i \tag{3}$$

$$Circular\ Rotations : \mu = 1, e^i = tan^{-i}2^{-i} \tag{4}$$

$$Linear\ Rotations : \mu = 0, e^i = 2^{-i} \tag{5}$$

$$Hyperbolic\ Rotations : \mu = -1, e^i = tanh^{-i}2^{-i} \tag{6}$$

2.2 Modes Of CORDIC Algorithm

- CORDIC operates in any one of the following two modes:
  - *Rotation Mode*:d_i = signum(z^i)
  - *Vectoring Mode*: d_i = -signum($x^i y^i$)

2.3 Convergence of CORDIC

- For Sin and Cos function implementation, CORDIC algorithm works fine when the angle lies between -90^o and 90^o. For angles lying outside this range, we apply standard trignometric identities to get the desired result.
- Elemental rotations using Hyperbolic CORDIC will not converge. Convergence is guaranteed if the following iterations are repeated : 4, 13, 40,.....k,(3k+1). We do not deal with the exact math behind this as it is beyond the scope of this project.

III IMPLEMENTATION OF THE VARIOUS TRIGONOMETRIC FUNCTIONS USING CORDIC

3.1 Sin and Cos Functions Generation using CORDIC

- Both sin and cos functions are implemented using *Circular Rotations* in *Rotation Mode*.

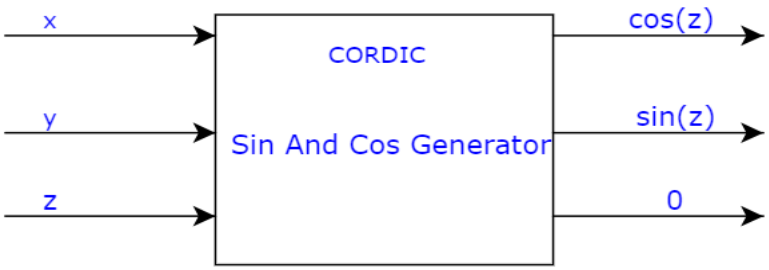


Figure 1: Block Diagram Of Sin and Cos generation

- x is initialized with value 0.6073, y with 0 and z holds the angle value for which we need to find sin and cos values. 10 iterations have been carried out in this project. After 10 iterations, x converges to cos(z) , y converges to sin(z) and z converges to 0.
- To implement this, we use 10 iterations in a combinational always block [i.e, always@(*)]. When the 10 iterations are over, the final result is updated in a separate register. This final updated happens in a clocked always block [i.e, always@(posedge clk)].

3.1.1 Resource Utilization of Sin and Cos

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFGCTRL (32)
N cordic_sin_cos_top	359	82	115	359	19	2
fp (cordic_sin_cos_dac)	344	47	106	344	0	0
pmod (pmodDAC)	15	35	12	15	0	0

Figure 2: Resource Utilization of Sin And Cos Generator Block

3.1.2 Post Implementation Timing Results of Sin and Cos Generator Block

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.117 ns	Worst Hold Slack (WHS):	0.155 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	178	Total Number of Endpoints:	178	Total Number of Endpoints:	84

All user specified timing constraints are met.

Figure 3: Post Implementation Timing Results of Sin And Cos Generator Block

- Although we get a positive slack of 2.117ns with the 100MHz clock signal, max frequency of operation is limited to 30MHz as we are using PMOD DA2 DAC to view the waveforms on oscilloscope.

3.1.3 Viewing Sin and Cosine Waveforms on the Oscilloscope using PMOD DA2 DAC

- The sin and cos values generated were also viewed on the oscilloscope using PMOD DA2 DAC. The DAC can operate at a maximum frequency of 30MHz. So we generate the sin and cosine values at a rate that is much slower than 30MHz.
- For this purpose, we use the original clock of 100MHz to generate a slower clock signal of 3.33MHz. Any value less than 30MHz would work.
- The user provides a 12 bit input using the slide switches present on the FPGA. As the slide switch input given by the user increases, the frequency of the waveform increases and as we move to higher frequencies, distortion also increases. This is because of the limits imposed by the conversion time of the DAC, which is 10μs.
- While sending the values to the DAC, the sine and the cosine values that are generated also need to be up-shifted by a certain amount as the DAC can handle values only between 0 and 4095.

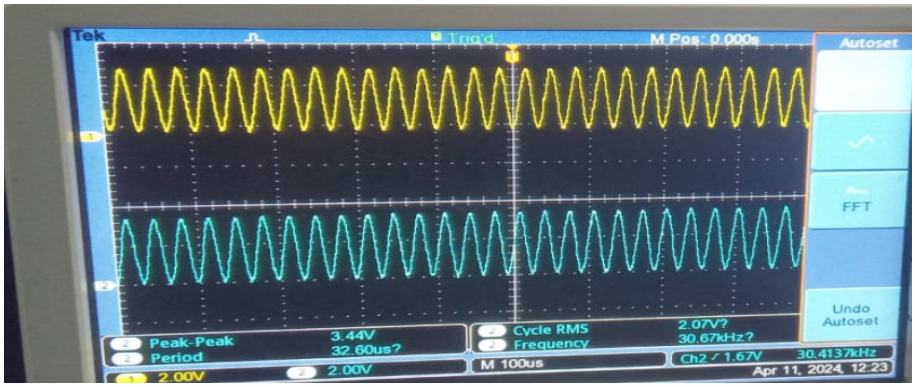


Figure 4: Sin and Cos Waveforms observed on Oscilloscope

3.2 *Tan Function Generation Using CORDIC*

- While it is not possible to directly generate tan waveforms, we perform division through CORDIC. We take the results of sin and cos that were generated through CORDIC and pass them to the CORDIC divider block to get tan waveform. CORDIC divider works in *Linear Rotations* in *Vectoring Mode*. In this method, the z of the divider block converges to tan of the given angle. y of the divider block converges to 0 and the value that x of the divider block converges to is of no consequence to us.

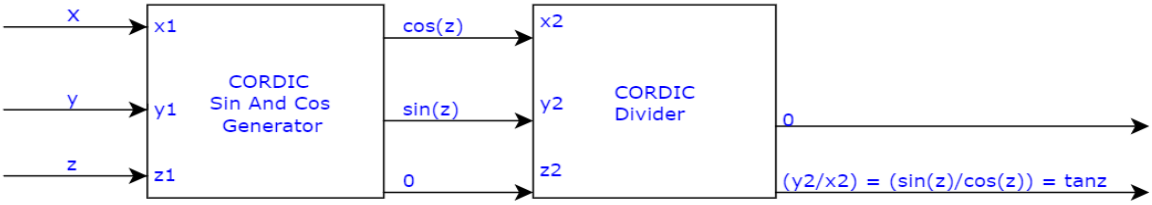


Figure 5: Block Diagram of Tan Generation

3.2.1 Limitations Of CORDIC Division

- The iterative expression for z component of the CORDIC divider is given as :

$$z^{(i+1)} = z^{(i)} - d_i 2^{(-i)} \tag{7}$$

- Here at each iteration d_i can be either +1 or -1. The max value that z can take in the first iteration is 1. The max value in the second iteration is 1 + $\frac{1}{2}$. The max value in the third step is 1 + $\frac{1}{2}$ + $\frac{1}{4}$. This is a geometric progression that converges to 2 after infinite iterations. Hence the max positive value that CORDIC division gives is +2 and the max negative value that CORDIC division gives is -2. Because of this, the tan waveform that is obtained through CORDIC also lies in the range of [-2,2].

3.2.2 Resource Utilization Of Tan Generator Block

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFGCTRL (32)
cordic_tan_dac_top	469	127	146	469	19	3
ecsd (cordic_tan_dac)	454	93	138	454	0	0
pmod (pmodDAC)	15	34	10	15	0	0

Figure 6: Resource Utilization Of Tan Generator Block

3.2.3 Post Implementation Timing Results of Tan Generator Block

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.249 ns	Worst Hold Slack (WHS):	0.056 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	265	Total Number of Endpoints:	265	Total Number of Endpoints:	130

All user specified timing constraints are met.

Figure 7: Post Implementation Timing Results Of Tan Generator Block

- Max operating frequency of the Tan Generator Block is 30MHz. This limitation is imposed by the PMOD DA2 DAC that is used to display the tan waveform on the Oscilloscope.

3.2.4 Expected Tan Waveform Vs Post Implementation Simulation Tan Waveform

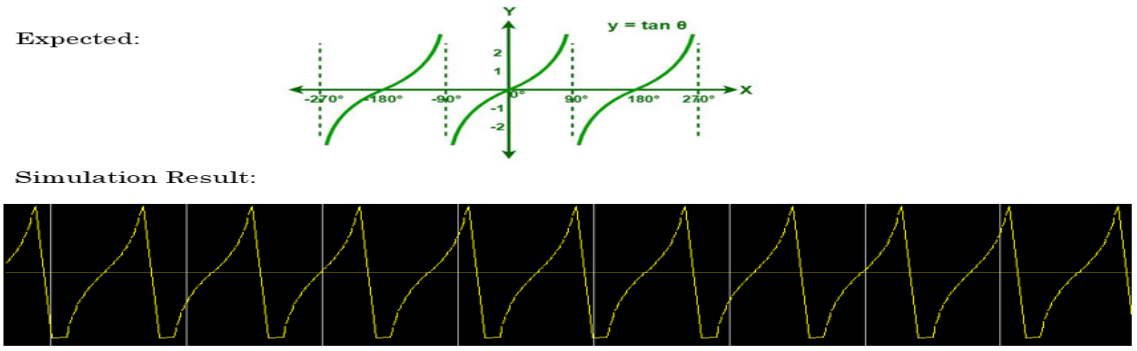


Figure 8: Comparison of Expected Vs Post Implementation Simulation Waveform of Tan

3.2.5 Viewing Tan Waveforms On Oscilloscope Using PMOD DA2 DAC

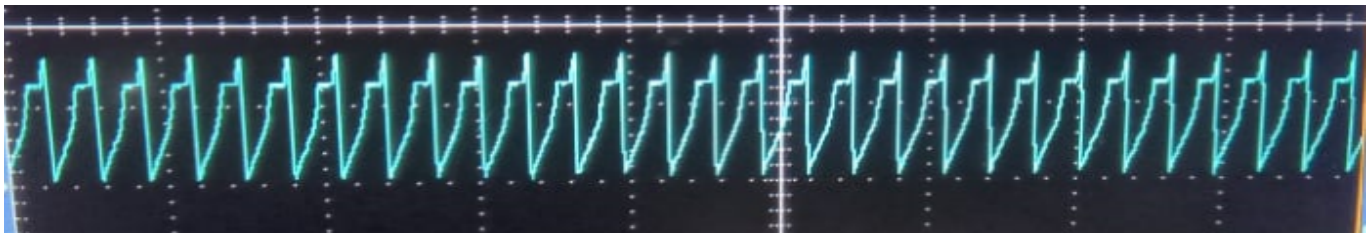


Figure 9: Tan Waveform On The Oscilloscope

- To send the Tan values to the DAC, an offset of 1024 is added to all the values generated in the program to get it within the range that can be handled by the DAC. When this is done, a small portion on the top half of the tan waveform is being clipped and becomes flat. This part needs further investigation.

3.3 Tanh Inverse Function Generation Using CORDIC

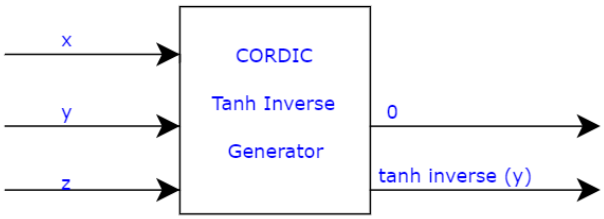


Figure 10: Block Diagram Of Tanh Inverse Generator

- CORDIC $tanh^{-1}(y)$ block operates in *Vectoring Mode* and uses *Hyperbolic Rotations*. To get $tanh^{-1}(y)$, we initialize x with 1 and z with 0. After 10 iterations (Number of iterations is designers' choice), y converges to 0, z converges to $tanh^{-1}(y)$ and the value that x converges is of no consequence to us.

3.3.1 Resource Utilization Of Tanh Inverse Generator

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFGCTRL (32)
N cordic_tanh_inverse	376	39	110	376	42	1

Figure 11: Resource Utilization of tanh inverse generator

3.3.2 Post Implementation Timing Results Of Tanh Inverse Generator

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.107 ns	Worst Hold Slack (WHS):	0.051 ns	Worst Pulse Width Slack (WPWS):	15.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	78	Total Number of Endpoints:	78	Total Number of Endpoints:	40

All user specified timing constraints are met.

Figure 12: Post Implementation Timing Results Of Tanh Inverse Generator

- The tanh inverse block was operated with clock period of 32ns. Hence the maximum operating frequency is given by:

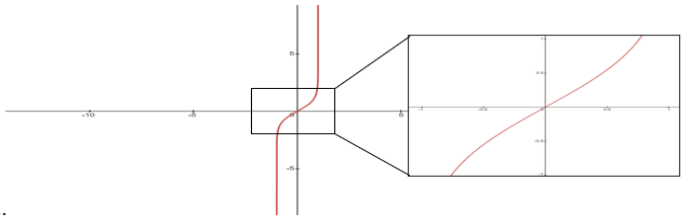
$$f_{max} = \frac{1}{32n - 1.107n} = \frac{1}{30.893n} = 32.369MHz. \tag{8}$$

3.3.3 Limitations Of Tanh Inverse Generation Using CORDIC

- $tanh^{-1}(y)$ is not a bounded function. It tends to +∞ as y tends to 1 and tends to -∞ as y tends to -1.
- Hence $tanh^{-1}(y)$ cannot be accurately captured by CORDIC for all possible values of y. Beyond $|y| > 1$, it gets difficult to capture the values of $tanh^{-1}(y)$ accurately due to fixed number of iterations used in CORDIC and also due to limitations imposed by fixed point representation.
- However, CORDIC algorithm gives values of $tanh^{-1}(y)$ with decent accuracy when y lies in the range of [-1,1].

3.3.4 Expected Vs Post Implementation Waveforms of Tanh Inverse

Expected:



Simulation Result:

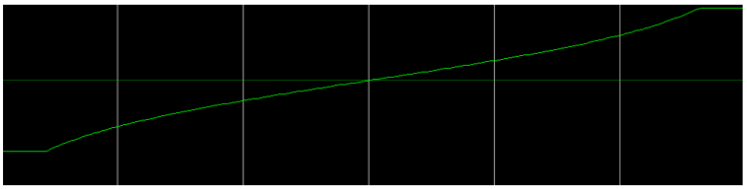


Figure 13: Expected Waveform Vs Post Implementation Waveform of $tanh^{-1}(y)$

3.4 Tan Inverse Function Generation Using CORDIC

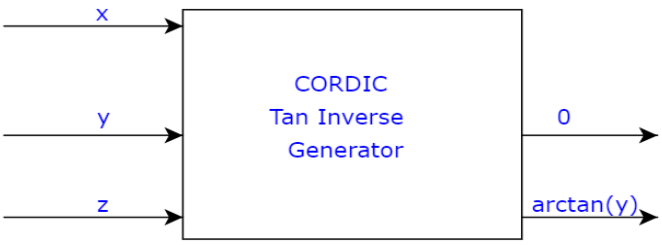


Figure 14: Block Diagram of $tan^{-1}(y)$ Generator

- $tan^{-1}(y)$ is obtained through *Circular Rotations* in *Vectoring Mode* of CORDIC. y value converges to 0 and the z value converges to $tan^{-1}(y)$. For this, x must be initialized to 1 and z must be initialized to 0.

3.4.1 Resource Utilization Of Tan Inverse Generator Block

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFGCTRL (32)
N cordic_tan_inverse	413	40	130	413	42	1

Figure 15: Resource Utilization of $tan^{-1}(y)$ Generator

3.4.2 Post Implementation Timing Results Of Tan Inverse Generator

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.079 ns	Worst Hold Slack (WHS):	0.510 ns	Worst Pulse Width Slack (WPWS):	14.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	59	Total Number of Endpoints:	59	Total Number of Endpoints:	41

All user specified timing constraints are met.

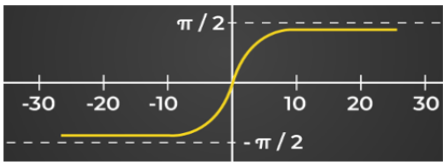
Figure 16: Post Implementation of $tan^{-1}(y)$ Generator

- T_{clk} of the $tan^{-1}(y)$ generator block is 30ns. The max operating frequency of $tan^{-1}(y)$ generator is:

$$f_{max} = 1/(30n - 0.079n) = 33.421\ MHz \tag{9}$$

3.4.3 Expected Vs Post Implementation Waveforms of Tan Inverse

Expected:



Simulation Result:

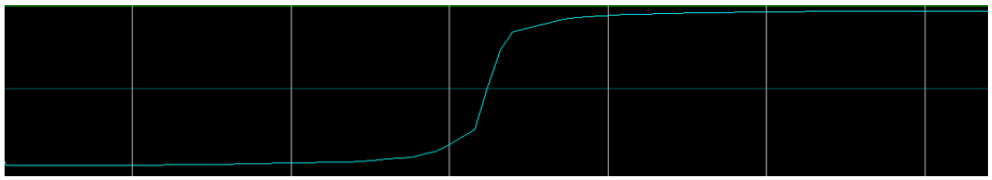


Figure 17: Expected Vs Post Implementation Simulation Waveforms Of Tan Inverse Function

- $tan^{-1}(y)$ function is generation very accurately by CORDIC directly through *Circular Rotations* in *Vectoring Mode* as it is a bounded function.

3.5 Tanh Function Generation Using CORDIC

- While it is not possible to generate tanh(z) function directly through CORDIC, we generate sinh(z) and cosh(z) directly through CORDIC and then pass the results to CORDIC Divider block to get tanh(z) function.

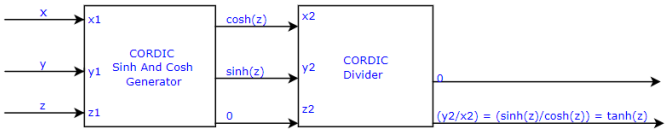


Figure 18: Block Diagram Of Tanh Generator

- While generating sinh(z) and cosh(z), x is initialized to 1.2075, y is initialized to 0. z holds the angle value in radians initially. At the end of the fixed number of iterations, x converges to cosh(z) , y converges to sinh(z) and the z component converges to in the *sinh and cosh* generator block.

3.5.1 Limitations Of Sinh And Cosh Function Generation Using CORDIC

- As was the case with $tanh^{-1}(y)$ generation, it is not possible to compute sinh(z) and cosh(z) for all values of z accurately using CORDIC. This is due to the fact that sinh(z) and cosh(z) are unbounded functions. When we deal with unbounded functions, accuracy is limited by the fixed number of iterations the algorithm employs and also by the fixed point representation used by the algorithm.
- Through simulation results, it was observed that sinh(z) and cosh(z) were obtained with good accuracy when z was within the range [-1,1].
- This is not a big problem as our main focus is on generating tanh(z). Because tanh(z) is *approximately* 1 for z greater than 1. So we use the values of the sinh(z) and cosh(z) from the first CORDIC stage and pass it on to the divider block to get tanh(z)

values with good accuracy for z in the range of [-1,1]. Beyond [-1,1] , sinh(z) and cosh(z) values from the first CORDIC stage saturate to a fixed value. When these saturated values are passed to the CORDIC Divider block, we get the result to be 1 (approximately). So we get a reasonably good approximation of the tanh(z) waveform.

3.5.2 Resource Utilization Of Tanh Function Generation

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFGCTRL (32)
N cordic_tanh	624	99	176	624	42	1
tanh1 (cordic_sinh_cosh)	430	40	114	430	0	0
tanh2 (cordic_division)	194	59	64	194	0	0

Figure 19: Resource Utilization Of Tanh Generator

3.5.3 Post Implementation Timing Results Of Tanh Generator

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.881 ns	Worst Hold Slack (WHS):	0.147 ns	Worst Pulse Width Slack (WPWS):	14.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	197	Total Number of Endpoints:	197	Total Number of Endpoints:	100

All user specified timing constraints are met.

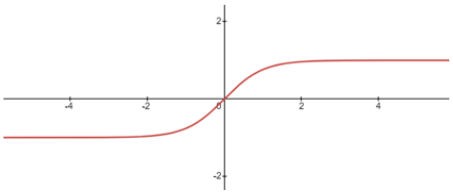
Figure 20: Post Implementation Timing Results Of Tanh Generator

- T_{clk} of the tanh generator block is 30ns. Max Frequency of Operation is given by:

$$f_{max} = \frac{1}{30n - 1.881n} = \frac{1}{28.119n} = 35.563MHz. \tag{10}$$

3.5.4 Expected Vs Post Implementation Simulation Waveform of Tanh Function

Expected:



Result:

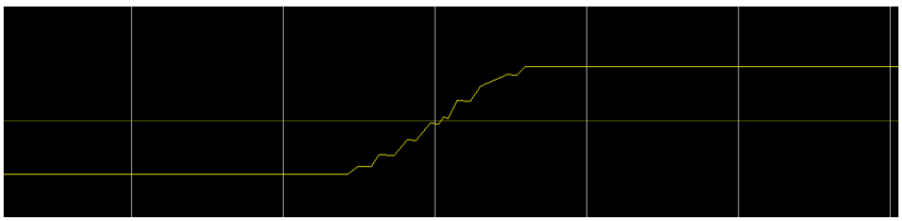


Figure 21: Expected Vs Post Implementation Waveform of Tanh Fucntion

- Here, the coarse nature of the tanh(z) waveform for z lying in the range of [-1,1] is attributed to the fact that inputs given to the module via testbench are not spaced very close to each other. Hence the obtained waveform is not as smooth as the expected waveform.

IV DESIGN STRATEGIES EMPLOYED

- In the generation of the various trigonometric functions, 10 iterations are run in a combinational always block (*always@(*)*) and the final result that we obtain after 10 iterations are updated in a register in a clocked always block(*always@(posedge clk)*). Asynchornous Reset has been employed in all the designs.
- For the generation of sin(z), cos(z) and tan(z) functions, final result after 10 iterations was updated by a slower clock signal of frequency 3.33MHz as these values had to be sent to the PMOD DA2 DAC. This 3.33 MHz is arbitrary and any value less than 30MHz would do. The final result after of $tan^{-}1(y)$ and $tanh(z)$ are updated using 30ns clock and that of $tanh^{-1}(y)$ function is updated using 32ns clock. These specifications satisfy all the timing constraints.

V IMPLEMENTATION CHALLENGES

- For the implementation of sin(z) and cos(z), a slower clock signal *clk2* was used as enable signal inside clocked always block of the original 100MHz clock signal (*always@(posedge clk)*). Because of this, timing constraints were met only when T_{clk} was set to 23ns. Later clk2 was not used as an enable signal and the final result was updated inside clocked always block of clk2 (*always@(posedge clk2)*), timing constraints were met for T_{clk} of 10ns with a positive slack.
- Since clk2 was generated using counters and the original clk of 100MHz, we used an asynchronous reset to support all the reset activities. This is because, when *rst* signal was active, the counter used to generate clk2 signal was also reset and there were no more edges in clk2 signal. Hence asynchronous reset has been employed in all the designs
- For the functions $tanh(z)$, $tan^{-1}(y)$, $tanh^{-1}(y)$, the inputs had to be given manually in the test bench and each design had its own fixed point representation (i.e, different number of bits before and after the binary point). Giving inputs manually was an arduous task. Hence a python script was programmed to generate input test vectors with the necessary precision. This accelerated the testing process.

VI RESULTS

- The functions sin(z), cos(z), tan(z) were visualized on the oscilloscope.
- The functions tanh(z), $tan^{-1}(y)$, $tanh^{-1}(y)$, were verified through simulations in Xilinx Vivado.
- Timing Analysis was carried out for all the designs. Power reports are included in the final submission and not added in the report to maintain brevity of the report.

VII CONCLUSIONS AND POSSIBLE IMPROVEMENTS

- It was found that for functions that are bounded, CORDIC gave very good results.
- For functions that are unbounded, CORDIC gave good results up to a certain point beyond which it was not possible to capture the results accurately due to fixed number of iterations and fixed point representation in the algorithm.
- For functions involving division, results were good as long as the expected value lies in the range of [-2,2]. Beyond this range, the values saturate at 2. This was the case in tan(z) function. Fortunately, this was not fatal for the implementation of tanh(z) function because tanh(z) function takes values only within [-1,1].
- 10 iterations were used in this project for the generation of the various types of trigonometric functions. Number of iterations is the designer's choice. When more iterations are used, it is worthwhile to use *pipeline* the design for faster results. Due to time constraints, the designs in this project are un-pipelined.
- To conclude, when we want to implement trigonometric functions whose results are bounded and do not need division, CORDIC is a hardware friendly option. Because not a single * operator was used in the implementation of all the designs. However, when functions increase rapidly or if division computations are involved, one has to look for alternate options if accuracy is of high priority as CORDIC fails to provide good accuracy in these cases.

VIII ACKNOWLEDGEMENTS

- Thanks to the instructor of the course Professor Debayan Das, for his guidance over the course of the mini project.
- Thanks to DESE seniors Sachin Thomas, Arimardan, Varsha, Vibhore Jain, Swapnankit and classmate Elizabeth Kuruvilla for their inputs.

IX REFERENCES

- Article on CORDIC in ”allaboutcircuits.com” : https://www.allaboutcircuits.com/technical-articles/an-introduction-to-the-cordic-algorithm/
- NPTEL video:” CORDIC Algorithm ” by Prof Nitin Chandrachoodan , IITM. Lecture Series : *Mapping Signal Processing Algorithms to Architecture*.
- NPTEL video: ” CORDIC Architecture ” Prof. Indranil Hatai . Lecture Series: *Architectural Design of Digital integrated circuits*.