Exploring Hardware Activation Function Design: CORDIC Architecture in Diverse Floating Formats

***Abstract*—With the increase in demand for neural networks (NNs) on edge devices, there is a need for cost-effective and hardware-efficient architectures. Activation functions induce non-linearity in the much-needed NN space and hence are essential to formulate real-world scenarios to an artificial model. In this work, three different architectural styles for *Softmax* and a single architecture each for *Tanh* and *Sigmoid* implementations using an improved CORDIC (COordinate-Rotation-DIgital-Computer) algorithm in diverse floating-point data-formats are explored. The work pays dual attention; firstly focusing on different architectural possibilities to implement hardware efficient activation functions, and secondly investigating the impact of utilizing different floating-point data formats on hardware parameters and error metrics. Different data formats are investigated in this work including the state-of-the-art (SOTA) Floating-Point 16 (FP16), Floating-Point 32 (FP32), Brain Floating-Point-BFloat16 (BF16), POSIT and TensorFloat32 (TF32). The area utilization, critical path delay, and power costs are presented for both FPGA (Zynq 7000 Zedboard) and ASIC (Cadence 45 nm library) synthesis. Error metrics are generated by comparing hardware simulated results with actual Floating-Point 64 bit (FP64) values. In conclusion, a benchmarking analysis that covers the best, nominal, and worst data formatted designs for the CORDIC implementation of activation functions was presented.**

***Index Terms*—Neural networks, CORDIC, Floating Point, POSIT, BFloat, TensorFloat, Data Representation, Activation Function, Tanh, Sigmoid, Softmax**

I. Introduction

Neural networks have gained huge popularity owing to its wide range of applications and its precision in decision-making. Their applications are found in various fields showcasing its impact in multiple domains such as medicine [1], transportation [2], [3], object detection [4], signal processing [5], and others. Neural networks establish a robust model through rigorous learning from the input-output relationships for the labelled training dataset, enabling them to draw inferences to a new input data. Typically, a neural network is made up of an input layer, few hidden layers, and an output layer.

Activation functions (AF) also referred to as transfer functions, are applied at the end of hidden layers to determine whether a particular neuron needs to be activated. They introduce non-linearity in the neural network space so that complex real-life problems are neatly handled and mapped appropriately for the supplied inputs. In real-life scenarios, mapping input data directly to an outcome remains a challenge. Instead, it is more suitable to employ non-linear functions to capture the decision-making process. Hence activation functions form a critical component in the design of neural network. Hardware implementation of neural network requires careful and tight realization of activation function. A slip in result is likely to upset the whole neural network co-processor design. Therefore, it is crucial to design a performance efficient hardware accelerator design with minimal loss in precision if possible. The state-of-the-art (SOTA) methods for implementing these non-linear activation functions include LUT-based [6]–[10], Piece-wise linear approximations [11]–[14], and CORDIC-based approaches [15]–[18]. Some of the limitations in these methods are: The LUT-methods struggle with high memory usage and limited precision for wide ranges [6], [8]. The Piece-wise methods require power-intensive multipliers [12]. To overcome many of these limitations, CORDIC method is used predominantly to realize activation functions. While a few previous CORDIC based methods often sacrifice significant PPA (power, performance and area) gains with poor hardware architecture designs [16], the other CORDIC based works fail to leverage the hardware architectural design benefits for different activation functions to achieve similar accuracy results with better PPA gains [17]. Hence, an implementation of CORDIC algorithm considering the different hardware architecture design approaches for different activation functions is needed.

Despite the choice of functions to be implemented and the algorithm to follow, it is important to choose data-format representation. Since the memory for the computing machinery is limited, a finite number of bits are available for representing real numbers. While achieving infinite precision is impractical, it is essential to strike a balance that ensures sufficient accuracy without the need for excessive hardware resources. To suit multiple application driven design requirements, different data representation formats are explored. The state-of-the-art (SOTA) data-formats are IEEE 754 Single Precision Format, which is most widely used for designing hardware systems. But in machine learning applications, where a moderate decrease in precision does not affect the overall inference model accuracy, there is a demand for employing hardware efficient formats. These data-formats utilize fewer bits than the standard 32-bit floating point representation, making a trade-off between precision and data size. Data formats such as BF16, TF32 and POSIT have proven to be easy on hardware and offer adequate precision and range making them ideal for AI applications. However a comprehensive study on different data-formats for activation functions implemented on hardware is missing in the literature, and hence this paper fills the gap by presenting hardware characteristics and error metrics associated for CORDIC implementation of activation functions in all possible data-formats. Besides, the CORDIC implementation with different data-formats including POSIT, TF32,

and BF16 is investigated for realizing activation functions possibly for the first time, as per the authors' knowledge. This exploration effort is expected to aid designers to choose an optimal combination of required activation function with the most efficient data-format representation based on applications' demand, without losing on precision.

We showcase the following in this work:

1) Implementation of three major activation functions including Sigmoid, hyperbolic-tangent, and Softmax in hardware using CORDIC algorithm both in ASIC and FPGA.
2) Three different styles of hardware architectures for the Softmax function realization.
3) Design space exploration of the implemented activation functions in CORDIC algorithm using different floating point representations.
4) Bench-marking the results of the different above said implementations in the perspective of hardware (PPA) gains and the corresponding error-metrics.
5) A Final analysis on best, optimal and worst choices for required activation function on the lines of their hardware and error metrics.

All the designs are made freely available in [19] for further usage to the researchers and designers' community.

II. Related Information

A. Data Formats

1) Half-Precision Floating Point: Half-precision FP (FP16) is a 16-bit floating point format with 1 sign bit, 5 exponent bits and 10 mantissa bits.

2) IEEE 754 Standard - Single Precision Floating Point: Single-precision floating point is the IEEE 32-bit floating point standard (FP32). It consists of 1 sign bit, 8 exponent bits and 23 mantissa bits [20]. It covers wide range of values and also offers higher precision.

3) BFloat16: BF16 is a data format used in Google's TPU to improve hardware efficiency with minimal loss [21]. It consists of 1 sign bit, 8 exponent bits and 7 mantissa bits. The reduced number of mantissa bits helps in reducing the hardware resources when compared to floating point. This data format is mostly used in Machine Learning applications.

4) POSIT: POSIT is a type of universal number system (UNUM) designed to overcome the limitations in floating point format that primarily includes multiple exceptional numbers, and limited accuracy. POSIT system is hardware-friendly since it avoids exceptional cases including sub-normal numbers, not a number, and +/- zero [22]. As opposed to the regular floating point number, with sign, exponent and mantissa fields, this format has an extra field called 'regime'. The regime field helps in achieving tapered accuracy. The bit length of each of these fields depends completely on the regime whose bit width is flexible. Within the context of a POSIT data-format, given total bit-width, and fixed exponent size, the regime's size will differ, depending on the arrangement of bits. It is determined by the length of a consecutive string of either continuous zeros or ones following the sign bit. The next 'es' number of bits correspond to the exponent part and the remaining is assigned to the mantissa component. In this work, POSIT format is represented as *POSIT(n,es)* where n is the total number of bits including sign, regime, exponent and mantissa and es is the bit width assigned for the exponent field. For example, *POSIT(12,2)* is a 12-bit POSIT number with a maximum of 2 bits of exponent.

5) TensorFloat32: TF32 is a new math mode engineered for Nvidia A100 GPUs to run AI workloads [23]. It consists of 19 bits in total with 1 sign bit, 8 exponent bits (same as that of FP32) and 10 mantissa bits (same as that of FP16). This hybrid combination enables the data-format to achieve the range same as that of FP32 due to similar exponent bit-width. Although the precision is compromised due to reduced mantissa bit-width, the method is widely utilized for AI operations due to its massive hardware benefits while offering comparable accuracy.

TABLE I: Range and Precision of Multiple Data Formats investigated in this work.

Data Format	Range	Max Precision
FP16	-3.05 $\times 10^{-5}$ to 6.55 $\times 10^{4}$	2^{-10}
FP32	-1.17 $\times 10^{-38}$ to 3.4 $\times 10^{38}$	2^{-23}
BF16	-1.17 $\times 10^{-38}$ to 3.4 $\times 10^{38}$	2^{-7}
POSIT(10,1)	-1.52 $\times 10^{-5}$ to 6.55 $\times 10^{4}$	2^{-6}
POSIT(12,1)	-9.53 $\times 10^{-7}$ to 1.04 $\times 10^{6}$	2^{-8}
POSIT(12,2)	-9.09 $\times 10^{-13}$ to 1.09 $\times 10^{12}$	2^{-7}
POSIT(12,3)	-8.27 $\times 10^{-25}$ to 1.2 $\times 10^{24}$	2^{-7}
TF32	-1.17 $\times 10^{-38}$ to 3.4 $\times 10^{38}$	2^{-10}

B. CORDIC Architecture

CORDIC algorithms are popularly employed to compute Trigonometric, Hyperbolic, Exponential, Division operations, and other basic non-linear mathematical functions, which are inherently difficult to realize using traditional logical synthesis methods. It works on the principle of iteratively shifting the phase angle such that the co-ordinates at the end of all iterations furnish the functional output for the given input [15]. Conventionally, only positive iterations are considered [24], [25], but negative iterations are expected to improve the accuracy [18], hence both iterations are considered in this work. A total of 19 iterations that includes iterations on negative side ranging from -5 to 0, and iteration on positive side ranging from 1 to 13 is considered. The algorithm is defined as per the Equation 1, where d_i represents the direction of rotation.

$$\begin{cases} x_{i+1} = \begin{cases} x_i + y_i d_i (1 - 2^{i-2}), & \rightarrow \text{if } i \leq 0 \\ x_i + y_i d_i (2^{-i}), & \rightarrow \text{otherwise} \end{cases} \\ y_{i+1} = \begin{cases} y_i + x_i d_i (1 - 2^{i-2}), & \rightarrow \text{if } i \leq 0 \\ y_i + x_i d_i (2^{-i}), & \rightarrow \text{otherwise} \end{cases} \end{cases} \tag{1}$$

1) CORDIC Exponential Unit (CEU): CORDIC algorithm employs hyperbolic rotation mode to evaluate the exponential e^z function. Initial values of x, y, and z are supplied with 1, 0, and input data respectively. At the end of 19 iterations which includes 6 on negative side, and 13 on the positive side, x, and y renders to $sinh(x)$, and $cosh(x)$ computational value

respectively, and subsequently generating e^z function. Inputs x and y are updated according to Equation 1 and z is updated as per Equation 2 until z→0.

$$\begin{cases} z_{i+1} = z_i - d_i R_i \\ R_i = \begin{cases} \tanh^{-1}(1-2^{i-2}), & \rightarrow \text{if } i \leq 0 \\ \tanh^{-1}(2^{-i}), & \rightarrow \text{otherwise} \end{cases} \\ d_i = \begin{cases} -1, & \rightarrow \text{if } z_i < 0 \\ +1, & \rightarrow \text{otherwise} \end{cases} \end{cases} \tag{2}$$

2) CORDIC Division Unit (CDU): To perform division, CORDIC architecture is implemented in linear vector mode. The initial values of x, y, and z are supplied with #1 operand, #2 operand, and 0 respectively. Inputs x is continuously iterated to x_{i+1} = x_i, y is updated according to Equation 1 and z is updated as per Equation 3, such that at the end of 19 iterations, we achieve z as $\frac{y}{x}$, until y→0, Note that R_i is stored in LUT.

$$\begin{cases} z_{i+1} = z_i - d_i R_i \\ R_i = \begin{cases} (1-2^{i-2}), & \rightarrow \text{if } i \leq 0 \\ (2^{-i}), & \rightarrow \text{otherwise} \end{cases} \\ d_i = \begin{cases} -1, & \rightarrow \text{if } y_i < 0 \\ +1, & \rightarrow \text{otherwise} \end{cases} \end{cases} \tag{3}$$

III. Proposed Tanh, Sigmoid, and Softmax Functions

A. Tanh Design

Figure 1 depicts the proposed method of computing *Tanh(x)* which is comprised of two steps that includes computing hyperbolic functions followed by the division operation. The same is also expressed in the Equation 4. In the first step, hyperbolic configuration of the CORDIC algorithm was configured to compute the said functions: *sinh(x)* and *cosh(x)* for a given input *x*. In the subsequent step, the linear mode of the CORDIC algorithm was configured to perform division operation of *sinh(x)* and *cosh(x)*, and achieve the final output: *tanh(x)*.

$$tanh(x) = \frac{sinh(x)}{cosh(x)} \tag{4}$$

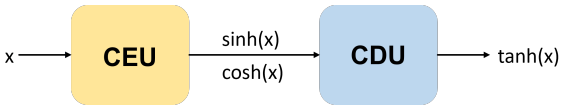


Fig. 1: Block Diagram representing CORDIC architectured implementation of *Tanh* function.

B. Sigmoid Design

Figure 2 depicts the proposed method of computing *Sigmoid(x)* function which is comprised of three steps. It can be observed from the Equation 5 that realization of exponential function and a division operation is required to implement the Sigmoid function. In the first step, the hyperbolic configuration of the CORDIC architecture was employed to compute the exponential function (e^x). In the second step, an adder unit(AU) that is specific to every data representation format under investigation, was employed to compute $1 + e^x$. In the last step, the linear mode of the CORDIC architecture was configured to perform division operations and compute the final output: $\frac{e^x}{1+e^x}$.

$$Sigmoid(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} \tag{5}$$

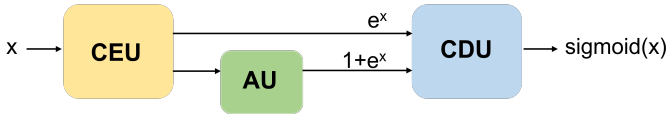


Fig. 2: Block Diagram showing CORDIC method for implementing Sigmoid Function.

C. Softmax Design

For implementing *Softmax* function, three diverse data-flow methods in the form of Serial, Parallel, and Hybrid architectures based on the CORDIC algorithm are realized. Figure 3 depicts the Serial architecture design for the implementation of the *Softmax* function. The Serial architecture enables ultra-low power ASIC implementation of *Softmax* function, with an intention to achieve a reduced footprint design with the trade-off in the number of clock cycles utilized. In this work, the *Softmax* function is implemented as an activation function for a class 10 CNN architecture design, as defined in the Equation 6, where i varies from 1 to 10. The input x_i (10 inputs for class 10) is supplied to the CEU to compute the corresponding exponential output for each input. These exponential outputs are later fed to the AU to generate summation results by making use of the feedback latch, and the exponential outputs are also stored in the FIFO unit buffer for final division operation as shown in Figure 3. The exponential summation output from the AU and the FIFO output is supplied to the CDU, to perform the division operation and compute the final class 10 probabilities for completing the *Softmax(z_i)* operation.

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{10} e^{x_i}} \tag{6}$$

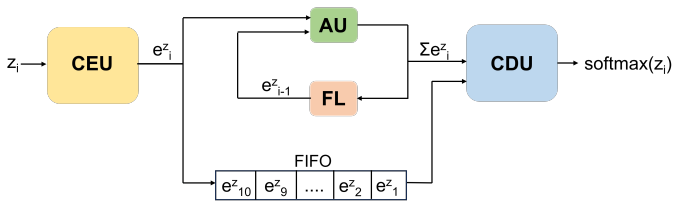


Fig. 3: Block Diagram representing Serial architecture design for realizing CORDIC based Softmax Function.

Figure 4 presents a CORDIC-based parallel architectural design for implementing *Softmax* function. This architecture focuses on FPGA implementation of the *Softmax* function with a high-performance design. All the ten inputs x_i (10 inputs for

class 10) are supplied in parallel at the same time to compute the corresponding exponential output for each input. These exponential outputs are later time multiplexed and supplied to the exponential-sum computing unit, inorder to generate the summation result by making use of the feedback latch. The same exponential outputs are supplied to the register unit to make use of the same at the final division operation as shown in Figure 4. The exponential summation output and all the 10 elements in the register unit are fed as inputs in parallel to perform the division operation and compute the final class 10 probabilities thereby completing the *softmax(z_i)* implementation. The proposed architectural design suffers from area utilized, and power consumed considering the multiple usage of hardware units to achieve accelerated results.

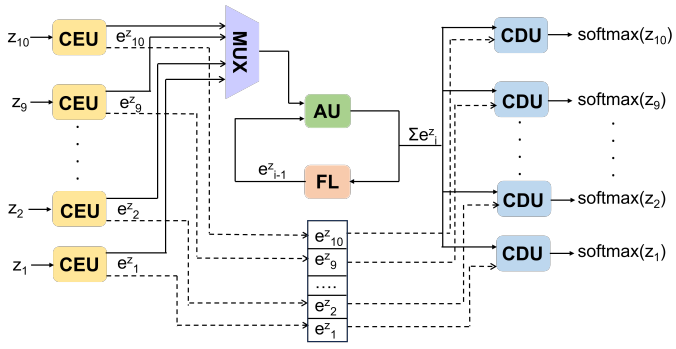


Fig. 4: Block Diagram showing CORDIC-based Parallel architecture design for realizing Softmax function.

Although the parallel architecture provides a very high-performance design, it is not practical to realize such architecture in silicon through ASIC implementations due to the following reasons: i) Large number of input and output ports are required to furnish inputs and extract the outputs in parallel, which occupies a large silicon footprint and hence is not practically viable for ASIC-based silicon implementation, and ii) Unacceptable computational delay is expected at the adder unit for generating exponential summation in silicon, which is otherwise accelerated in FPGA by employing adder chains. To overcome these challenges and implement a high-performance design for silicon implementation, a hybrid architecture is proposed. Figure 5 depicts the hybrid architecture design for implementing *Softmax* function. This architecture focuses on ASIC implementation of the *Softmax* function showcasing high-performance design with a trade-off in footprint and power cost. The ten inputs z_i (10 inputs for class 10) are directed with clocked offsets among them to compute the corresponding exponential output for each input. These exponential outputs that are generated at offsets are supplied in sequence through a multiplexer to compute the summation of the exponential output using the feedback latch. The exponential outputs are stored in the FIFO unit which are further utilized for the final division operation. The exponential summation output and the FIFO output are supplied as inputs in pipeline mode to perform the division operation and compute the final class 10 probabilities to complete *softmax(z_i)* operation.

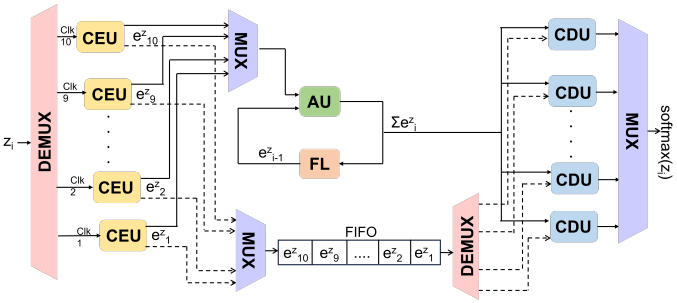


Fig. 5: Block Diagram of CORDIC-based Hybrid architecture design for implementing Softmax function.

IV. Results and Discussion

In this section, the hardware metrics along with the corresponding error metrics for all the activation functions are discussed. An additional exploration of the implemented designs in the following data-formats: BFloat16(BF16), POSIT(10,1), POSIT(12, 1), POSIT (12,2), POSIT(12,3), TensorFloat32 (TF32), FP16 (Half Precision IEEE 754 Standard) and FP32 (Single precision IEEE 754 Standard) are presented in detail.

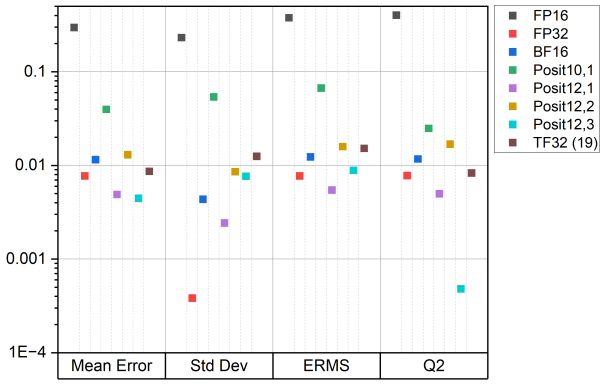
A. Methodology

For hardware metrics, both FPGA and ASIC results are evaluated. The FPGA synthesis is performed using AMD Xilinx Zedboard Zynq-7000 (xc7z020clg484-1) on Xilinx AMD Vivado tool, and ASIC synthesis was performed on Cadence 45 nm gpdk and Skywater 130nm technology node libraries, using Cadence Genus tools. The 45 nm gpdk library result plots are presented in this paper, whereas the 130 nm results are posted in [19] in the interest of shortage of page length for this manuscript. Besides, the trend of results for 130 nm is similar to 45 nm ASIC results. For error metrics, the input is considered in the range of [0,1]. To compute the error values, actual FP64 (Double precision IEEE 754 Standard) outputs were considered for the mentioned range of inputs. Various error metrics including Mean error, standard deviation, Upper Quartile Range (Q1), Median (Q2), Lower Quartile Range(Q3), Inter Quartile Range, Minimum and Maximum value of errors, Error Rate and Number of Error Bits are computed for all the proposed data-formatted activation function implementations. Plots for Mean error, Standard deviation, ERMS and Q2, in logarithmic scale, as depicted by Figure 6, are presented for a clear bench-marking. Please note that the values of error metrics in tables are signed, but for comparisons and graphs, the absolute values of the same are considered. Due to limited manuscript space availability, all the mentioned error-metrics are presented in [19].

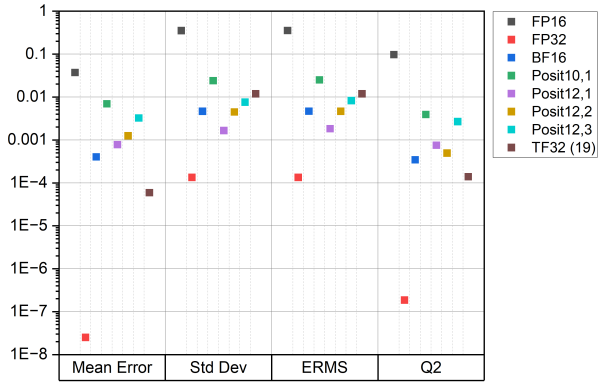
B. Hyperbolic Tangent function

Tanh function is implemented for FP16, FP32, BF16, POSIT(12,1) and POSIT(12,2), TF32 data formats. *Tanh* is reported to not converge for POSIT(10,1) and POSIT(12,3),

(a)



(b)



(c)

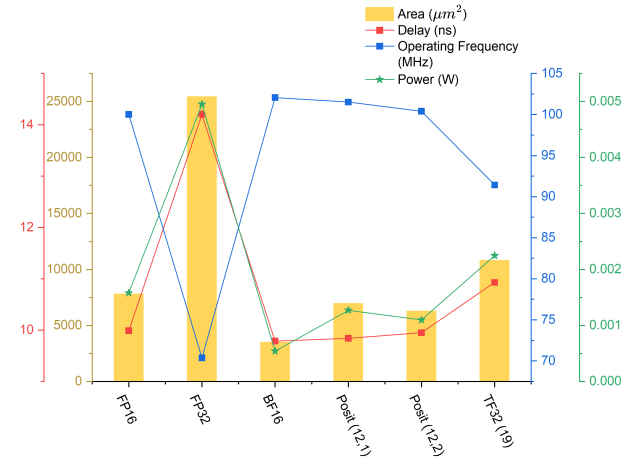
Fig. 6: Error metrics of (a) *Tanh*, (b) *Sigmoid*, and (c) *Softmax* activation functions implemented using CORDIC technique.

besides conceding high error rates hence these data-formatted designs are not included in the comparison study.

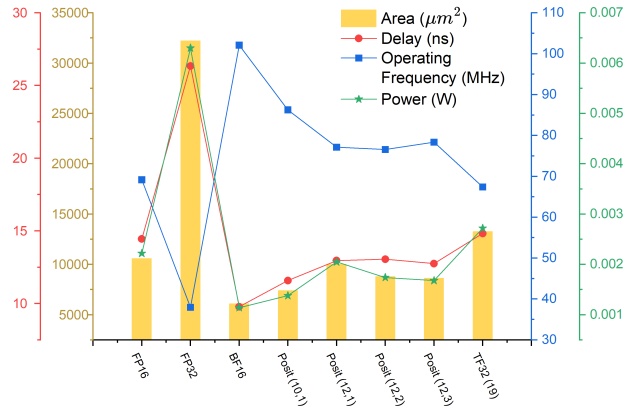
1) Error metrics: All the data formats show substantial improvement in error than that of FP16. Although FP32 data format with the highest number of bits possesses the best error metrics (highest number of mantissa bits), it was included just to benchmark and evaluate other data formats that avail a lesser number of bits, while we focus on the next best SOTA available data formats with respect to FP16. The error metrics are shown in Figure 6 (a). To state a few of them, POSIT(12,2) formatted design presents the least mean error which is 96.49% lesser, and TF32 formatted design shows the least standard deviation which is 98.89% lesser, than FP16 formatted function design respectively.

TABLE II: FPGA synthesized Hardware Metrics of CORDIC-based *Tanh* function implementation in different data formats.

Hardware Metrics	FP16	FP32	BF16	POSIT (12,1)	POSIT (12,2)	TF32 (19)
LUTs	1530	4298	1513	1773	1689	1990
Delay (ns)	33.979	56.553	37.808	37.151	37.269	42.212
Power (W)	0.124	0.13	0.082	0.083	0.083	0.118
Operating Frequency (MHz)	29.31	17.64	26.34	26.80	26.72	23.61



(a)



(b)

Fig. 7: ASIC flow derived hardware characterized results for the realized (a) *Tanh*, and (b) *Sigmoid* function.

2) PPA metrics: The ASIC synthesized results are presented in Figure 7(a) and FPGA results are listed in Table II. In the ASIC characterized results, BF16 shows maximum power savings of 65.82% with respect to FP16 data format, among all data formats under study. BF16 formatted *Tanh* realized design shows maximum operating frequency of 102.07 MHz, and other POSIT formatted designs succeed very closely, among all the formats. In terms of footprint occupied on silicon, the best results are demonstrated by BF16 with 55.11% savings over FP16. BF16 is also characterized with least delay, showcasing an improvement of 2.01% over FP16 formatted

TABLE III: FPGA synthesized Hardware Metrics of CORDIC based *Sigmoid* function implementation in different data formats.

Hardware Metric	FP16	FP32	BF16	POSIT (10,1)	POSIT (12,1)	POSIT (12,2)	POSIT (12,3)	TF32 (19)
LUTs	1853	5101	1856	1708	2153	2031	1997	2436
Delay (ns)	60.642	109.511	44.498	49.614	56.517	54.548	55.837	61.707
Power (W)	0.118	0.121	0.083	0.079	0.081	0.081	0.080	0.116
Operating Frequency (MHz)	16.45	9.12	22.39	20.09	17.64	18.28	17.86	16.16

design. POSIT formats also show delay improvements over FP16 formatted designs. On the FPGA synthesis front, power savings of 33.87% for BF16 and POSIT was achieved over FP16 data formatted design. Besides, BF16 shows 1.11% savings on hardware resource utilization when compared to FP16 data-format. Although achieving power and resource utilization gains for BF16, and POSIT formatted designs, the performance metric for FP16 still remains the best among all the designs. POSIT(12,1) formatted design delay is stretched by only 9.34% with respect to FP16 formatted design.

C. *Sigmoid function*

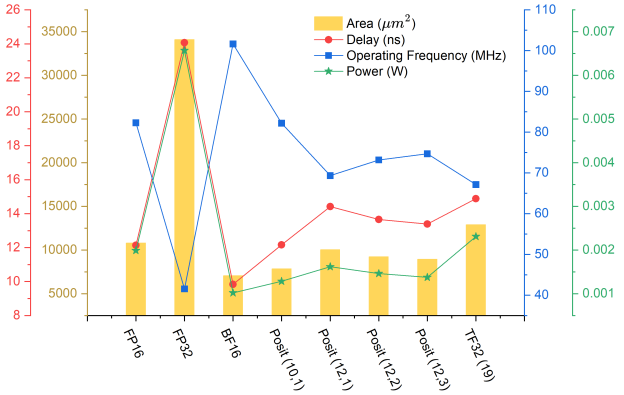
Sigmoid function is realized in FP16, FP32, BF16, POSIT(10,1), POSIT(12,1), POSIT(12,2), POSIT(12,3) and TF32 data formats.

1) Error Metrics: Figure 6 (b) shows the complete error metrics for *Sigmoid* function implementation with each of the data formats. POSIT(12,3) and POSIT(12,1) presents 98.50% and 98.35% reduction in Mean Error in comparison to FP16 formatted design. On the other hand, POSIT(12,1) and BF16 suggests 98.95% and 98.12% lesser standard deviation, when compared to FP16.

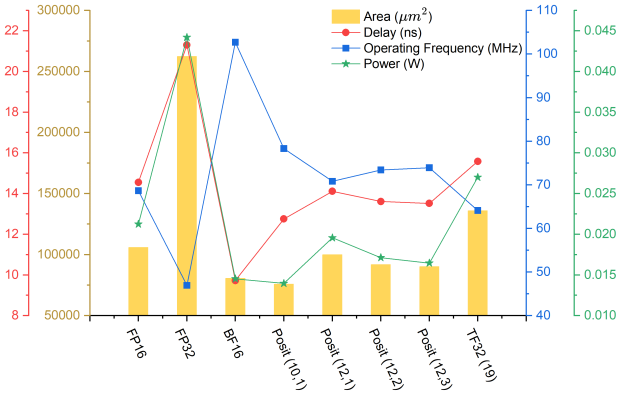
2) PPA metrics: Table III shows the FPGA synthesis results and Figure 7(b) depicts the ASIC results for 45nm technology node library. On hardware metrics, POSIT formatted design shows several benefits over others. Firstly, POSIT(10,1), and POSIT(12,3) sees power savings by 33.05% and 32.20% in FPGA synthesized designs respectively. Other POSIT methods also closely follow the above two. Among all the data formatted designs, BF16 presents the highest Operating Frequency which is 36.11% and 47.60% more than that of FP16 formatted designs, in FPGA flow and on ASIC flow on silicon respectively. BF16 and POSIT(10,1) formatted design offers power savings of 48.69% and 37.74% respectively over FP16

TABLE IV: FPGA synthesized Hardware Metrics of CORDIC based *Softmax* function in Serial, Parallel and Hybrid configurations in different data formats.

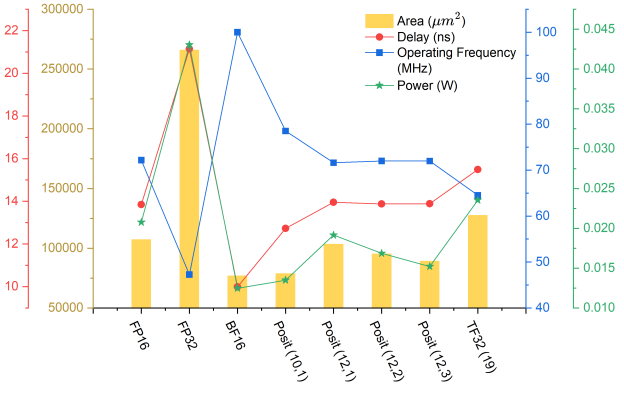
Hardware Parameter	FP16	FP32	BF16	POSIT (10,1)	POSIT (12,1)	POSIT (12,2)	POSIT (12,3)	TF32 (19)
Serial								
LUTs	1137	3217	1263	1127	1404	1324	1295	1259
Delay (ns)	43.982	91.938	45.091	50.855	56.275	56.893	56.364	44.512
Power (W)	0.115	0.115	0.077	0.076	0.076	0.076	0.076	0.112
Operating Frequency (MHz)	22.52	10.83	21.96	19.53	17.66	17.47	17.63	22.29
Clock Cycles	370							
Parallel								
LUTs	16959	47123	17344	16061	20196	19307	19078	22478
Delay (ns)	50.490	87.099	45.756	50.465	55.632	55.191	56.881	61.707
Power (W)	0.261	0.306	0.195	0.164	0.180	0.177	0.159	0.224
Operating Frequency (MHz)	19.75	11.46	21.78	19.75	17.92	18.06	17.53	16.16
Clock Cycles	32							
Pipeline								
LUTs	48875	47277	17425	16774	21119	19849	19419	22568
Delay (ns)	85.156	87.099	45.756	52.89	57.342	55.945	58.237	61.707
Power (W)	0.278	0.273	0.172	0.130	0.150	0.141	0.138	0.200
Operating Frequency (MHz)	11.72	11.20	21.78	18.85	17.39	17.82	17.12	16.16
Clock Cycles	62							



(a)



(b)



(c)

Fig. 8: ASIC flow derived hardware characterized results for *Softmax* function implemented in (a) Serial (b) Parallel (c) Hybrid configurations.

TABLE V: Evaluation of hardware metrics using ASIC synthesis and error metrics across different data formatted design choices for the analyzed activation function, including the best, standard, and worst designs.

Functions	Metrics	Best	Nominal	Worst
Tanh	**Error**	*POSIT(12,2)*	*TF32* < *POSIT(12,1)* < BF16	FP16
	Area	*BF16*	*POSIT(12,2)* < *POSIT(12,1)* < FP16	TF32
	Delay	*BF16*	*POSIT(12,1)* < *POSIT(12,2)* < FP16	TF32
	Power	*BF16*	*POSIT(12,2)* < *POSIT(12,1)* < FP16	TF32
	Operating Frequency	*BF16*	*POSIT(12,1)* > *POSIT(12,2)* > FP16	TF32
Sigmoid	**Error**	*TF32* < *POSIT(12,1)*	*POSIT(12,3)* < BF16 < POSIT(12,2) < POSIT(10,1)	FP16
	Area	*BF16*	*POSIT(10,1)* < *POSIT(12,3)* < POSIT(12,2) < POSIT(12,1) < FP16	TF32
	Delay	*BF16*	*POSIT(10,1)* < *POSIT(12,3)* < POSIT(12,1) < POSIT(12,2)	FP16 < TF32
	Power	*BF16*	*POSIT(10,1)* < *POSIT(12,3)* < POSIT(12,2) < POSIT(12,1)	FP16 < TF32
	Operating Frequency	*BF16*	*POSIT(10,1)* > *POSIT(12,3)* > POSIT(12,1) > POSIT(12,2)	FP16 > TF32
Softmax Serial	**Error**	*TF32*	*BF16* < *POSIT(12,1)* < POSIT(12,2) < POSIT(12,3) < POSIT(10,1)	FP16
	Area	*BF16* <*POSIT(10,1)*	*POSIT(12,3)* < POSIT(12,2) < POSIT(12,1) < FP16	TF32
	Delay	*BF16*	*FP16* < *POSIT(10,1)* < POSIT(12,3) < POSIT(12,2)	POSIT(12,1) < TF32
	Power	*BF16* < *POSIT(10,1)*	*POSIT(12,3)* < POSIT(12,2) < POSIT(12,1) < FP16	TF32
	Operating Frequency	*BF16*	*FP16* > *POSIT(10,1)* > POSIT(12,3) > POSIT(12,2)	POSIT(12,1) > TF32
Softmax Parallel	**Error**	*TF32*	*BF16* < *POSIT(12,1)* < POSIT(12,2) < POSIT(12,3) < POSIT(10,1)	FP16
	Area	*POSIT(10,1)*	*BF16* < *POSIT(12,3)* < POSIT(12,2) < POSIT(12,1)	FP16 < TF32
	Delay	*BF16*	*POSIT(10,1)* < *POSIT(12,3)* < POSIT(12,2) < POSIT(12,1) < FP16	TF32
	Power	*POSIT(10,1)*	*BF16* < *POSIT(12,3)* < POSIT(12,2) < POSIT(12,1)	FP16 < TF32
	Operating Frequency	*BF16*	*POSIT(10,1)* > *POSIT(12,3)* > POSIT(12,2) > POSIT(12,1) > FP16	TF32
Softmax Hybrid	**Error**	*TF32*	*BF16* < *POSIT(12,1)* < POSIT(12,2) < POSIT(12,3) < POSIT(10,1)	FP16
	Area	*BF16* < *POSIT(10,1)*	*POSIT(12,3)* < POSIT(12,2) < POSIT(12,1) < FP16	FP16 < TF32
	Delay	*BF16*	POSIT(10,1) < *FP16* < POSIT(12,2) < POSIT(12,3) < POSIT(12,1)	TF32
	Power	*BF16* < *POSIT(10,1)*	*POSIT(12,3)* < POSIT(12,2) < POSIT(12,1) < FP16	FP16 < TF32
	Operating Frequency	*BF16*	*POSIT(10,1)* > *FP16* > POSIT(12,2) > POSIT(12,3) > POSIT(12,1)	TF32

formatted designs. Best performance metric is offered by the BF16 formatted design, followed by POSIT(10,1) in both FPGA and ASIC flow. The delay cut of 26.62% and 32.27% is achieved for BF16 design, and delay improvement of 18.18% and 19.85% is achieved for POSIT(10,1) design for FPGA and ASIC flow respectively over FP16 formatted design. The silicon footprint for BF16 design is conserved by 42.39% over FP16 design. POSIT formated designs also shows compact designs over FP16 designs. Note that FP16 designs offers the most compact LUT driven FPGA design, however among other data formatted designs, POSIT(10,1) offers the least and shows LUTs usage slightly more than FP16 by 7.82%.

D. Softmax Function

All the three different implementations of *Softmax* function namely Serial, Parallel, and Hybrid, are designed to optimize hardware parameters.

All the three architecturally different designs for realizing *Softmax* functions are implemented in diverse formats: FP16, FP32, BF16, POSIT(10,1), POSIT(12,1), POSIT(12,2), POSIT(12,3) and TF32.

1) Error metrics: Figure 6 (c) shows the variation of error metrics for *Softmax* function implementation across different data formats. TF32 presents a remarkable reduction in mean error by 98.84%, followed by BF16's 98.92% when compared to FP16 design. POSIT(12,1) and POSIT(12,2) presents a standard deviation drop by 99.53% and 98.73% when compared to FP16, respectively. These radical improvements are attributed to the less convergent FP16 designs resulting in huge deviation in the output values.

2) PPA Metrics: Table IV presents the FPGA synthesized results and the ASIC flow characterized results for 45 nm technology library are plotted in the Figure 8, for all the architecture styles. For *Softmax* hybrid implementation, BF16 and POSIT(10,1) presents power savings by 39.90% and 35.06% in comparison with the corresponding hybrid implementation of FP16 formatted design. For *Softmax* parallel implementation, BF16 offers 9.37% and 33.16% improvement in delay for FPGA and ASIC flow designs, over the corresponding parallel configuration of FP16 designs. For the realized *Softmax* function in serial configuration, the most compact footprint is presented by BF16 corresponding design. BF16 and POSIT(10,1) shows footprint savings of 34.62% and 27.39% respectively in comparison to FP16 data formatted design.

E. Overall Analysis

Table V summarizes the evaluation of data formats for all the three activation functions over error, and hardware parameters. A trend of best, worst and the nominal choices of data formatted designs for realizing activation functions across the parameters of interest are listed. In most of the parameters across the activation functions realized, only one design falls in best and worst categeories. In few cases, the designs that fall close to the best or worst design are also listed in the same categeory. The other categeory labelled as nominal includes all the designs which does not fall in the best and worst categeories and are listed in the order of preferences for the specified parameter. These three categeory and the order of preferences helps in selecting the most pareto-optimal data formatted CORDIC architecture designs for the

desired activation functions. From Table V, it is observed that BF16 offers the least footprint, best delay, least power cost and highest operating frequency for most cases. Hence, BF16 is the best choice when hardware gains are given priority. On the other hand, TF32 has the least mean error in most cases but compensates with higher hardware metrics. Therefore, TF32 is best for applications demanding high accuracy. However, for a pareto-optimal designs that fall in the trade-off line between accuracy and hardware design on silicon, POSIT formatted design fits the best among all other data formats for all the three activation functions listed. Similarly, Similarly, POSIT designs with both error metrics and hardware utilization results falling within the acceptable range, is considered the most optimal FPGA designs for all the activation functions studied.

V. Conclusion

This paper presents an extensive study on the hardware and error characteristics of activation functions including *Tanh*, *Sigmoid*, and *Softmax* using both positive and negative iterations adopted CORDIC architecture for diverse data formats. It was evidently found that there exists trade-off between the hardware metrics - delay, footprint occupied, power, and frequency of operation, versus error computed metrics. The hardware results for both FPGA (Zynq 7000 Zedboard) and ASIC (Cadence 45 nm technology node) were presented for a fair comparison. On a general note, it was observed that for extremely low cost and high performance, BF16 stands out, and for high accuracy outputs, TF32 is a preferred design solution for realizing all the three hardware activation functions. For applications seeking a pareto-optimal design solution, POSIT is an optimal choice, since it has lesser resource consumption than TF32 and lower error values than BF16. It is also noticed that the error metric pattern varies with each of the activation functions. For instance, when mean error is considered, POSIT format works best for *Tanh* and *Sigmoid*, and TensorFloat32 format favours the *Softmax* function. These diverse comparisons of data formatted activation functions are useful for optimal design of hardware accelerators for efficient CNN runs. This effort is a step towards designing hardware efficient and acceptable accuracy loss for modern day AI workloads. All the designs are made freely available in [19] for further usage to the researchers and designers' community.

References

[1] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng, and M. Chen, "Medical image classification with convolutional neural network," in *2014 13th International Conference on Control Automation Robotics Vision (ICARCV)*, 2014, pp. 844–848.

[2] Y. Li, C. Wu, and T. Yoshinaga, "Vehicle speed prediction with convolutional neural networks for its," in *2020 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, 2020, pp. 41–46.

[3] H. Le, M. Nguyen, W. Q. Yan, and S. Lo, "Training a convolutional neural network for transportation sign detection using synthetic dataset," in *2021 36th International Conference on Image and Vision Computing New Zealand (IVCNZ)*, 2021, pp. 1–6.

[4] Q. Liu, X. Lu, Z. He, C. Zhang, and W.-S. Chen, "Deep convolutional neural networks for thermal infrared object tracking," *Knowledge-Based Systems*, vol. 134, pp. 189–198, 2017.

[5] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro, "Convolutional neural network architectures for signals supported on graphs," *IEEE Transactions on Signal Processing*, vol. 67, no. 4, pp. 1034–1049, 2019.

[6] A. U. F. Piazza and M. Zenobi, "Neural networks with digital lut' activation functions," *Proceedings of 1993 International Joint Conference on Neural Networks*, 1993.

[7] K. K. M. Revathi Pogiri, Samit Ari, "Design and fpga implementation of the lut based sigmoid function for dnn applications," *2022 IEEE International Symposium on Smart Electronic Systems (iSES)*, 2022.

[8] D. Larkin, A. Kinane, V. Muresan, and N. O'Connor, "An efficient hardware architecture for a neural network activation function generator," in *Proceedings of the Third International Conference on Advances in Neural Networks - Volume Part III*, ser. ISNN'06. Berlin, Heidelberg: Springer-Verlag, 2006, p. 1319–1327.

[9] Y. Xie, A. N. Joseph Raj, Z. Hu, S. Huang, Z. Fan, and M. Joler, "A twofold lookup table architecture for efficient approximation of activation functions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 12, pp. 2540–2550, 2020.

[10] H. C. Prashanth and M. Rao, "Somalib: Library of exact and approximate activation functions for hardware-efficient neural network accelerators," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*, 2022, pp. 746–753.

[11] S. Saranya and B. Elango, "Implementation of pwl and lut based approximation for hyperbolic tangent activation function in vlsi," in *2014 International Conference on Communication and Signal Processing*, 2014, pp. 1778–1782.

[12] H. Dong, M. Wang, Y. Luo, M. Zheng, M. An, Y. Ha, and H. Pan, "Plac: Piecewise linear approximation computation for all nonlinear unary functions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 9, pp. 2014–2027, 2020.

[13] V.-T. Nguyen, T.-K. Luong, H. Le Duc, and V.-P. Hoang, "An efficient hardware implementation of activation functions using stochastic computing for deep neural networks," in *2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, 2018, pp. 233–236.

[14] N. R, R. V, and M. Rao, "Accelerated piece-wise-linear implementation of floating-point power function," in *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2022, pp. 1–4.

[15] A. Boudabous, G. Fahmi, M. Kharrat, and N. Masmoudi, "Implementation of hyperbolic functions using cordic algorithm," 01 2005, pp. 738 – 741.

[16] A. Kagalkar and S. Raghuram, "Cordic based implementation of the softmax activation function," *2020 24th International Symposium on VLSI Design and Test (VDAT)*, pp. 1–4, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221717276

[17] T. D. Nguyen, D. H. Kim, J. S. Yang, and S. Y. Park, "High-speed asic implementation of tanh activation function based on the cordic algorithm," in *2021 36th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*, 2021, pp. 1–3.

[18] H. Chen, L. Jiang, Y. Luo, Z. Lu, Y. Fu, L. Li, and Z. Yu, "A cordic-based architecture with adjustable precision and flexible scalability to implement sigmoid and tanh functions," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[19] https://sites.google.com/view/activation-functions-using-cor/home/asic-130nm-tech-node-results.

[20] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[21] https://cloud.google.com/tpu/docs/bfloat16/.

[22] Gustafson and Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomput. Front. Innov.: Int. J.*, vol. 4, no. 2, p. 71–86, jun 2017. [Online]. Available: https://doi.org/10.14529/jsfi170206

[23] https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/.

[24] Y. Cao, W. Xiao, J. Jia, D. Wu, and W. Zhou, "Cordic-based softmax acceleration method of convolution neural network on fpga," in *2020 IEEE International Conference on Artificial Intelligence and Information Systems (ICAIIS)*, 2020, pp. 66–70.

[25] R. S, A. S. Bharadwaj, D. S K, M. S. Khadabadi, and A. Jayaprakash, "Digital implementation of the softmax activation function and the inverse softmax function," in *2022 4th International Conference on Circuits, Control, Communication and Computing (I4C)*, 2022, pp. 64–67.