UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Faculty of Engineering, Built Environment and Information Technology

DEPARTMENT OF ELECTRONIC ELECTRIC AND COMPUTER ENGINEERING (EECE)

EES 424

PRACTICAL 2: DESIGN AND IMPLEMENT IN SIMULATION OF A VHDL BASED FUNCTION CALCULATION PROTOTYPE

Name and Surname	Student Number	Signature	% Contribution
M. H. Khan	14346011		100

By signing this assignment we confirm that we have read and are aware of the University of Pretoria's policy on academic dishonesty and plagiarism and we declare that the work submitted in this assignment is our own as delimited by the mentioned policies. We explicitly declare that no parts of this assignment have been copied from current or previous students' work or any other sources (including the internet), whether copyrighted or not. We understand that we will be subjected to disciplinary actions should it be found that the work we submit here does not comply with the said policies.

September 9, 2020

CONTENTS

LIST OF FIGURES

I. Implementation of the research model

The research model that has been implemented as a piecewise sigmoid activation function this function has been fully described in [1]. The base research model is made up of splitting a sigmoid activation function into seven equal parts from a range of -8 to 8 on the x-axis range and 0 to 1 y-axis. This function is described by the following equation:

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{1}$$

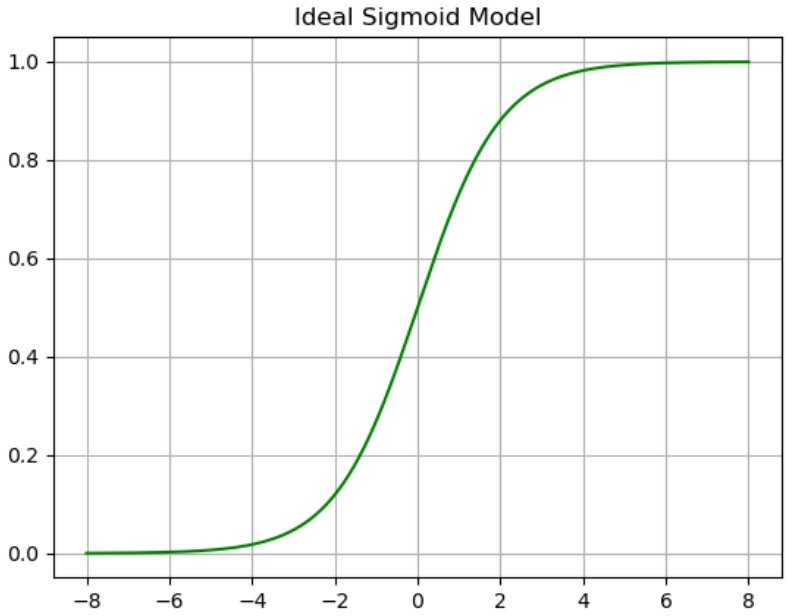

Fig. 1: Ideal sigmoid generated with eq.1

It was noticed that working with floating point values can cause significant amount of trouble. Therefore, all the values in the system were then scaled to very large integers by means of multiplying both axis by 2^{16}. Doing this eliminates the need for keeping account for the position of the decimal point when doing calculations. Once all that data has been captured the scaling values can be removed.

Model 1: Base research model.
Model 2: Improvement upon base research model.

A. Generation of input data

The input data ranges from $-524288 = 2^{16} * (-8)$ to $524288 = 2^{16} * (8)$ this results in a total of 1048576 data points that are split in to seven segments as described in in the table below.

TABLE I: Break points of the A LAW approximation[2]

x	-8.0	-4.0	-2.0	-1.0	1.0	2.0	4.0	8.0
y	0.0	0.0625	0.12	0.25	0.75	0.87	0.937	1.0

By taking a each value in the above table and scaling by 2^{16} will result in the data type that will be suitable for the model that has been implemented in VHDL. A variable x is initialised to -524288 and starts to count up to 524288.

B. State machine diagram of the implemented model

Figure.3 below represents the finite state machine diagram of the model that has been implemented in VHDL.

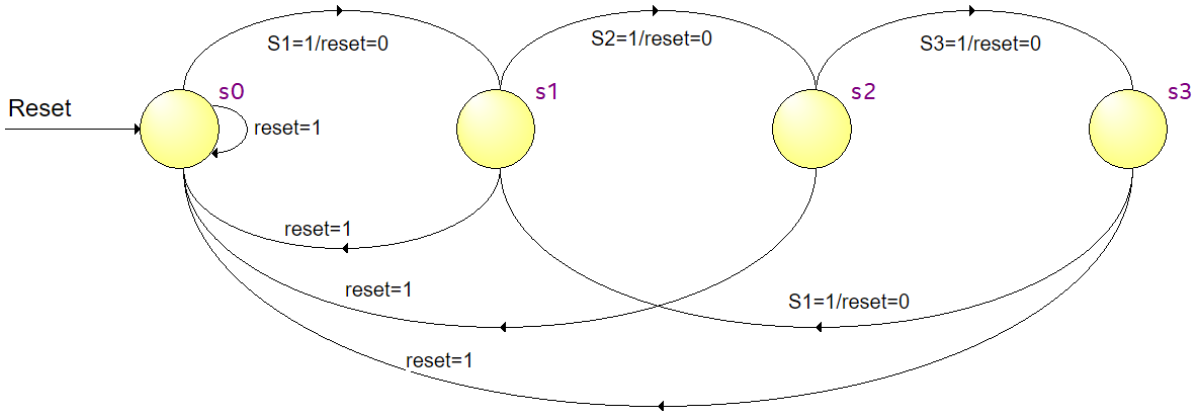

Fig. 2: Finite state machine model

The model consists of four states. These sates are as follows:

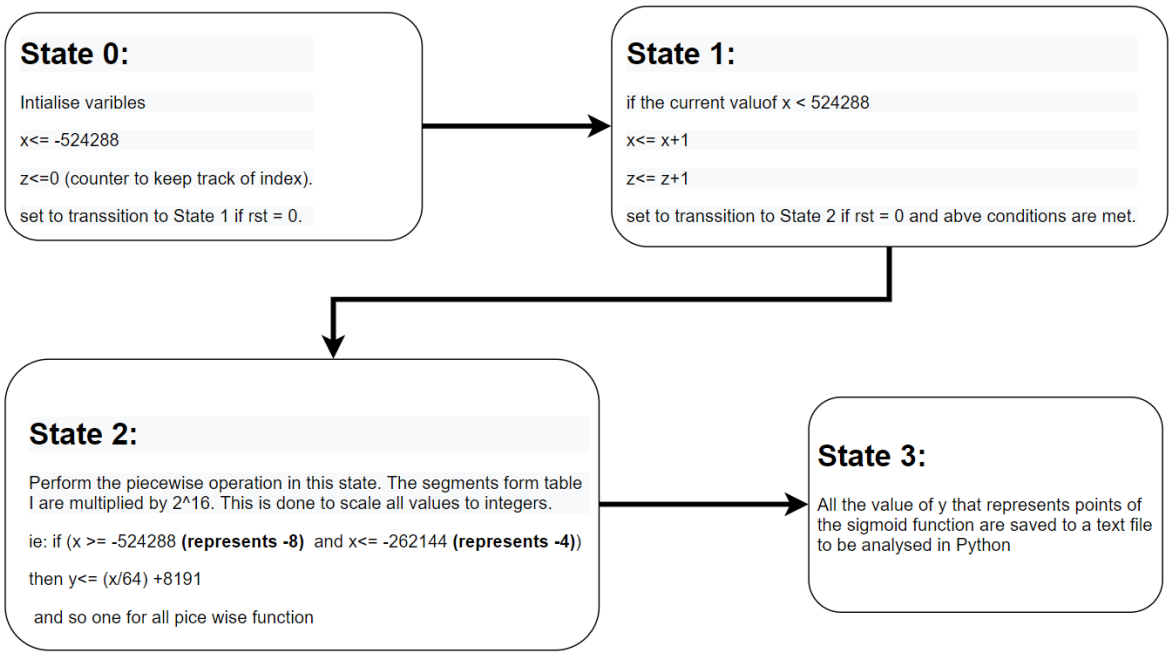

Fig. 3: Code flow diagram

The figure that follows is a plot in python of the data that has been saved to the text file in state 3 of the VHDL model. The plot below has a degree of resemblance to the result that has been obtained by [3].

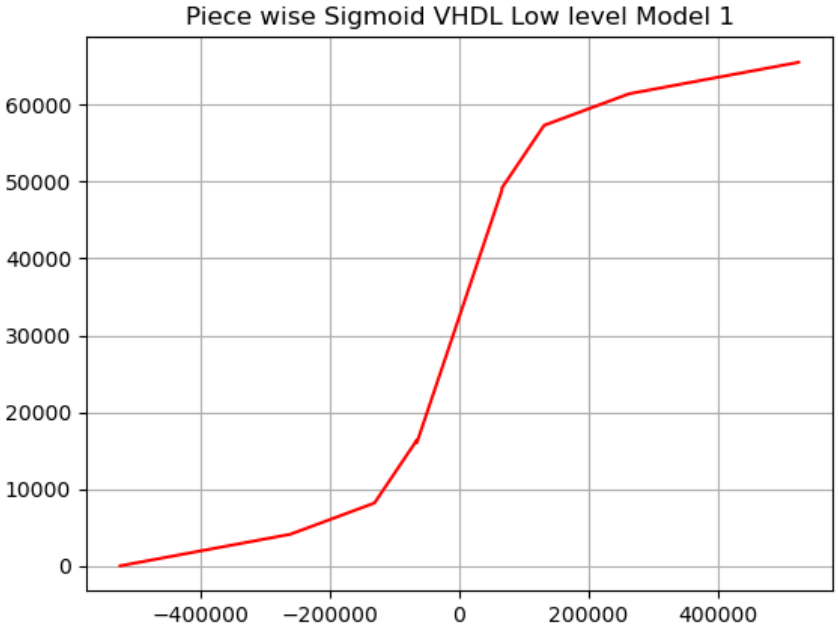

Fig. 4: Low level VHDL model 1.

In this figure the shape of the sigmoid is recognisable. However, this model can be improved to more closely resemble the sigmoid in fig.1. The section that follows will investigate this improved model implementation.

C. Implementation of the improved model.

The model was improved upon by increasing the number of piecewise function Steps from seven to fifteen. In doing this the line segments can be more closely represented by the gradient of the ideal sigmoid. Hence, the piecewise function looks a lot closer to that of the ideal sigmoid that was presented in fig.1.

State 2 modified model:

Perform the piecewise operation in this state. The segments form table I are multiplied by 2^16. This is done to scale all values to integers.

ie: if (x >= -524288 **(represents -8)** and x<= -457852**(represents -7)**)

then y<= (x/2048) + 256

and so one for all pice wise functions but there are now 15 picewise function

Fig. 5: State 2 of the improved model

The core functionality of the code remains the same. There are still four states present however, there is a modification made to state 2. The piecewise functions are increased seen in the above fig.5. increasing these piecewise functions allowed for the improvement in the model seen in fig.6.

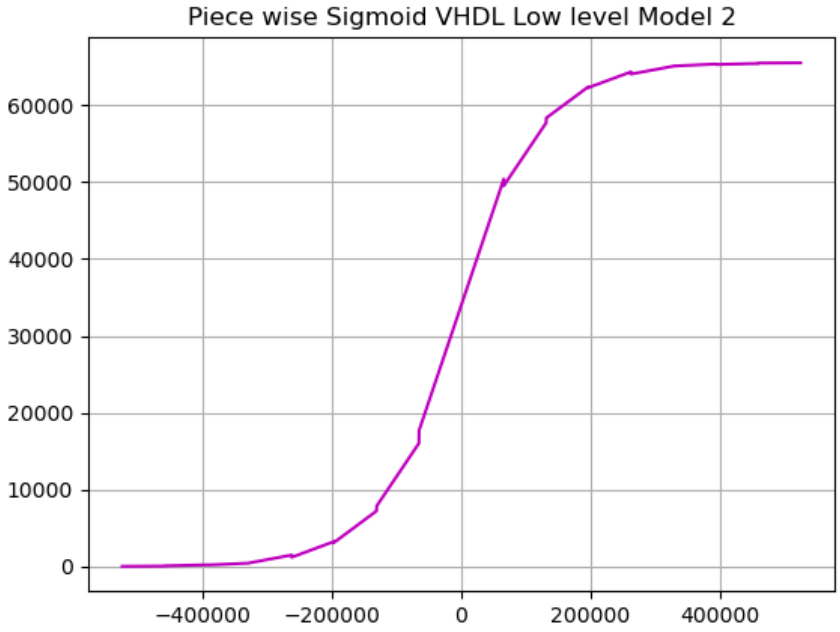

Fig. 6: Low level improved VHDL model.

In the above figure it can be noticed that the new model more closely represents the ideal sigmoid as presented in fig.1. This will result in an improvement in the overall error between the ideal sigmoid and the model. This will be further discussed in the verification methodology section.

II. Verification Methodology

A. *Informal verification strategy*

The informal verification strategy was implemented by taking the resultant value of the model at a position [i] for both the Python and the VHDL low-level models.

TABLE II: Tabulated 10 points of both implemented models

i	Python Model 1	VHDL Model 1	Python Model 2 (improved)	VHDL Model 2 (improved)
158350	2474	2474	268	268
158351	2474	2474	268	268
158352	2474	2474	268	268
158353	2474	2474	268	268
158354	2474	2474	268	268
158355	2474	2474	268	268
158356	2474	2474	268	268
158357	2474	2474	268	268
158358	2474	2474	268	268
158359	2474	2474	268	268
158360	2474	2474	268	268

For the above tabulated data points it can been seen that there is zero error between the Python model and the VHDL model for both model 1 and model 2. From informal verification standpoint one can make the assumption that the models in VHDL are identical to the Python counterpart. Hence, for the visual inspection the Python model was plotted against the VHDL data that was saved in the text file from modelSIM. In the figure below this data is plotted on the same axis so as to make a visual confirmation that the two functions are indeed identical to one another.

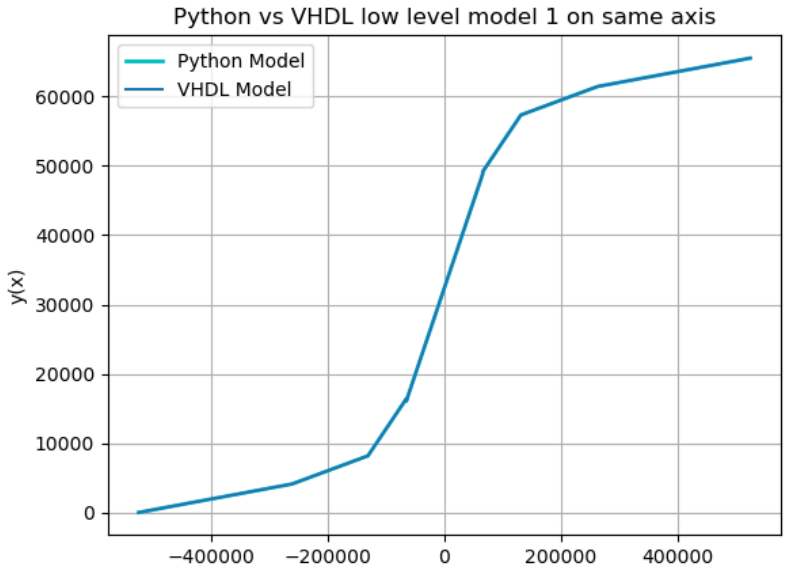

Fig. 7: Python vs VHDL for model 1

The same strategy was applied to the modified model which is depicted in the figure below. It can be seen at the improved model not only closely resembles the ideal sigmoid but also achieved zero error meaning the Python model and the VHDL model are indeed identical.

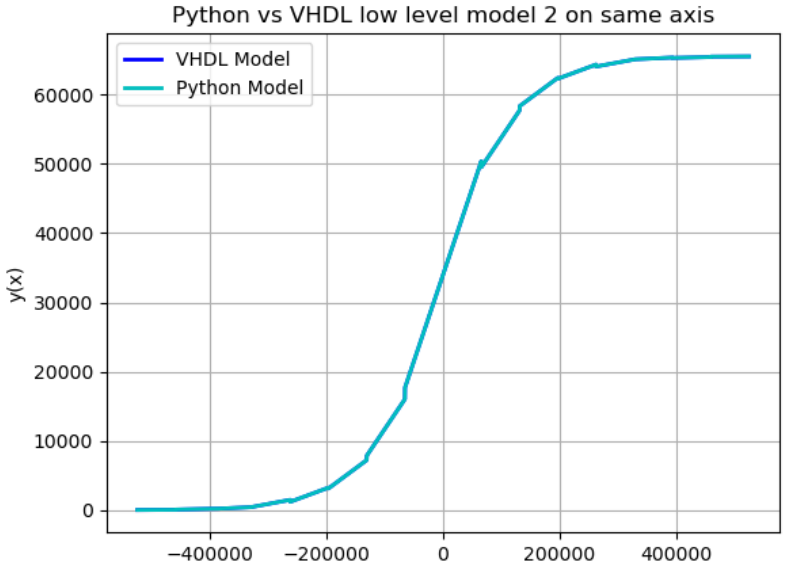

Fig. 8: Python vs VHDL for model 2

B. Formal verification strategy

The flow diagram below describes the formal verification process that is done automatically within python. The Python low-level model data is loaded as well as opening the text file in which the VHDL model was written to. Both models are stored into arrays. A loop is then run to test whether the index of array 1 value is equal to the index of array 2 to at the same point. The difference between the two arrays at the identical index is then plotted. This plot should be a flat line at zero for the entire data range for the for the models to be considered identical and appropriate.

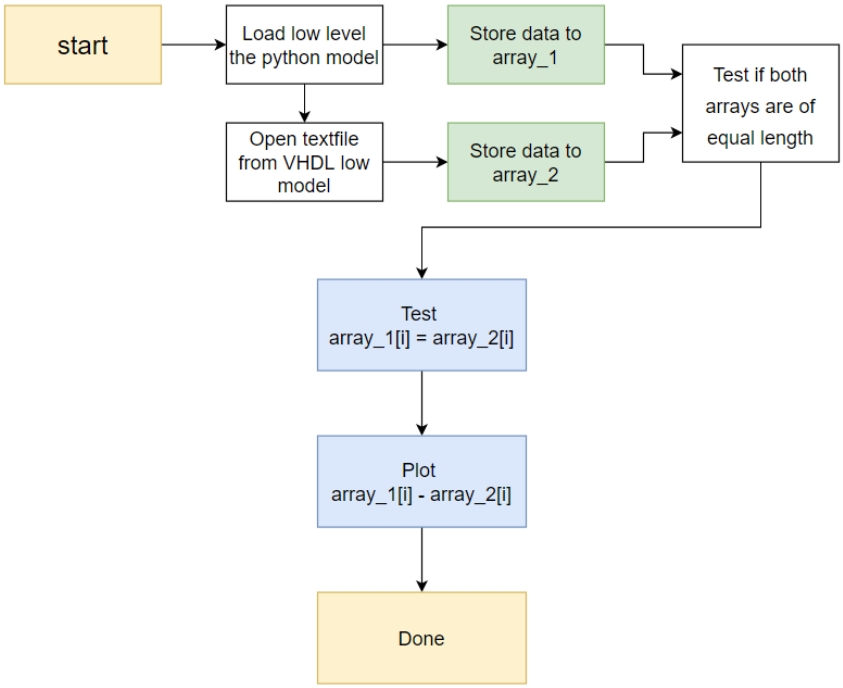

Fig. 9: Formal verification strategy

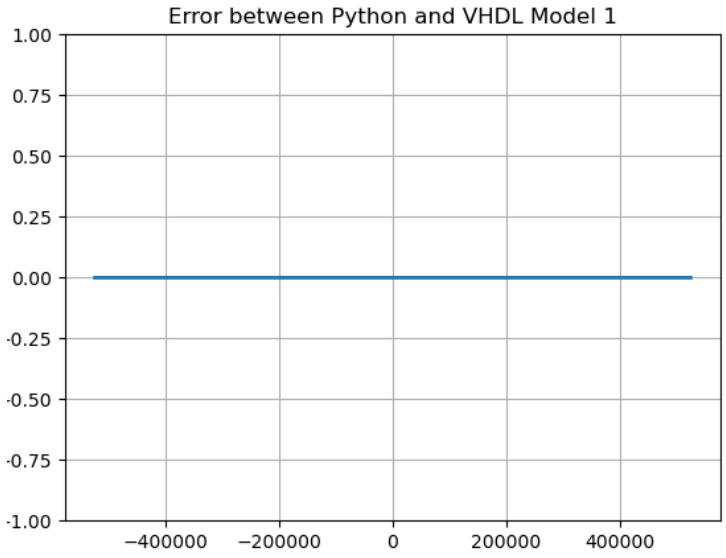

Fig. 10: Difference between VHDL model 1 and Python model 1

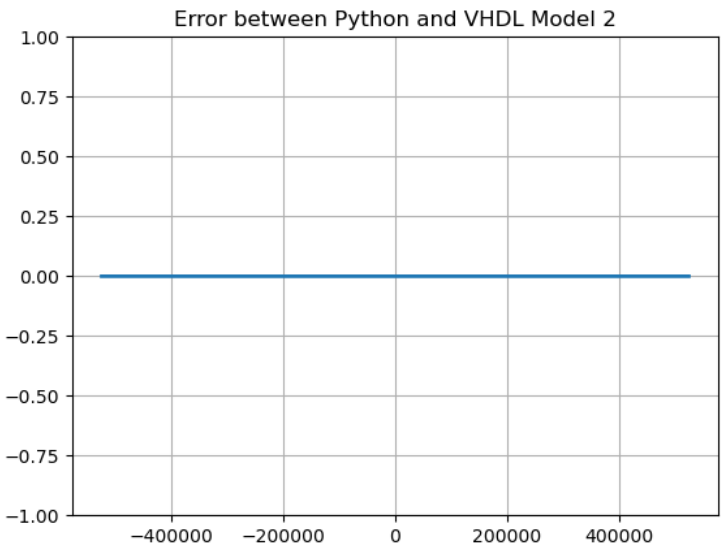

Fig. 11: Difference between VHDL model 2 and Python model 2

III. Improvements over base model

In order to test the overall improvement between the base model and improved model. The area between the ideal sigmoid and the base model as well as the ideal sigmoid and the modified model where recorded. The results of this exercise are presented in this section.

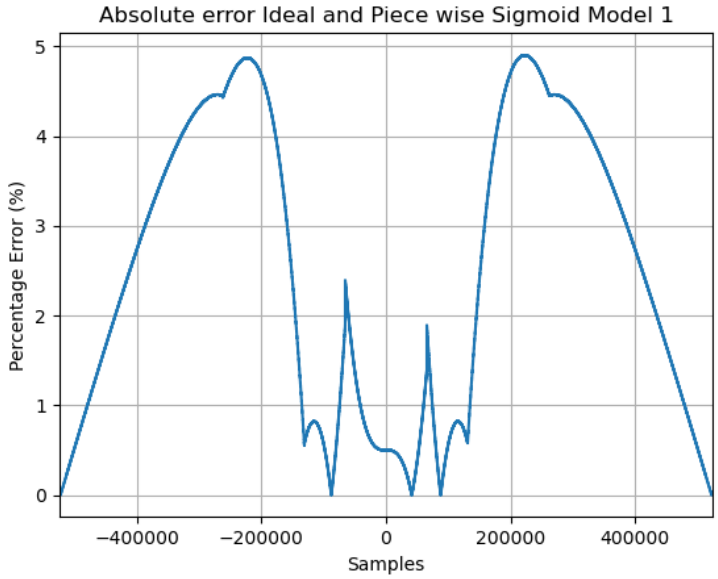

Fig. 12: Absolute error between ideal and the piece wise sigmoid model 1

The absolute error in model 1 was found to be around 4.85% seen in fig.12. This is in line with what was recorded by the research paper published by [2].

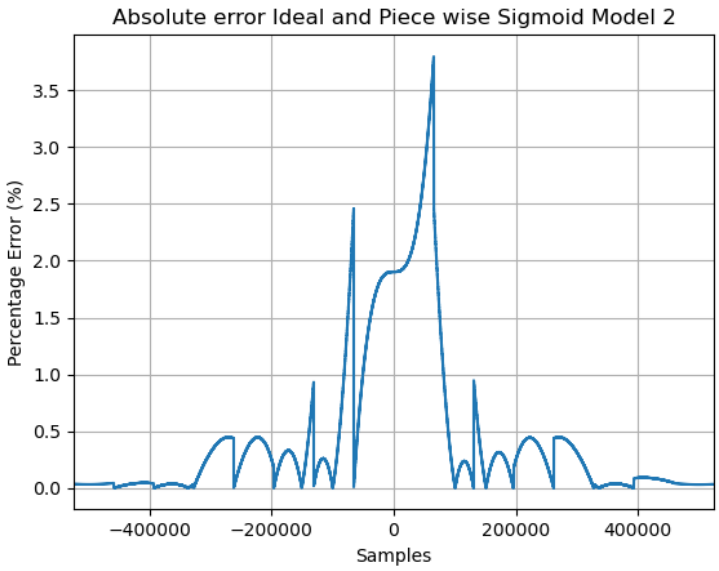

Fig. 13: Absolute error between ideal and the piece wise sigmoid model 2

The absolute error of the improved model was lower to the point of the absolute error in the improve model being 3.58% as presented in fig.13. This absolute error was noticed to be around the zero point of the sigmoid the piecewise functions gradients can be improved to reduce this error.

IV. Conclusion

In conclusion both models were successfully implemented in VHDL an overall bit error between the Python model and the VHDL model for both the base model as well as the modified model will zero. furthermore with the modified model there was an improvement in absolute error of approximately 1.27%. Taking these factors into account it can be concluded that the system was successfully implemented in VHDL and was successfully and thoroughly verified within Python using both informal as well as formal automated verification processes.

REFERENCES

[1] M. Khan, "Research proposal- design of vhdl based function calculation prototype." University of Pretoria, 2020.

[2] A. Tisan, S. Oniga, D. Mic, and B. Attila, "Digital implementation of the sigmoid function for fpga circuits," *ACTA TECHNICA NAPOCENSIS Electronics and Telecommunications*, vol. 50, 01 2009.

[3] D. J. Myers and R. A. Hutchinson, "Efficient implementation of piecewise linear activation function for digital vlsi neural networks," *Electronics Letters*, vol. 25, no. 24, 1989.

APPENDIX A
SOFTWARE TOOL UTILISED

A. *Spyder IDE*

The Spyder Python it was crucial in the development of both low-level models of the sigmoid function . The IDE provides extensive feature as well as ease of use of variable explorer to see values that are being loaded into the variables that are created. This makes debugging and troubleshooting a lot more user friendly as well as reducing downtime looking for bugs. The figures below are images of the console printing values of the implemented models for a small value range.

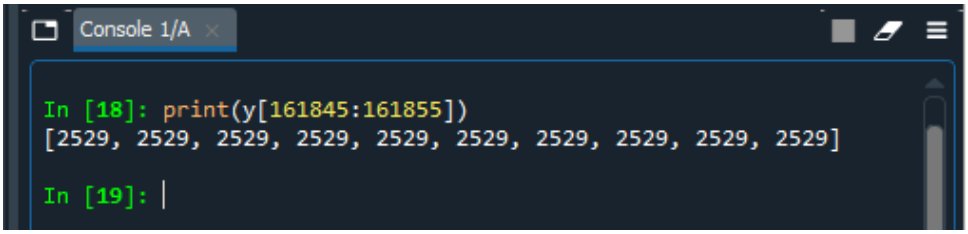

Fig. 14: Spyder console for model 1

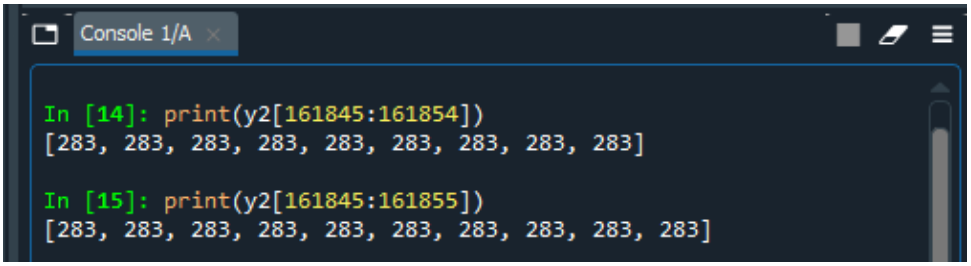

Fig. 15: Spyder console for model 2

B. *ModelSim*

ModelSim is also another integral part in the success of the implementation of the sigmoid function. ModelSim allows the user to run VHDL system test match models within its test environment. In this test environment the user is able to choose different clock frequencies as well as manually assign values to all the variables and signals that have been created in the VHDL Script. The user can print a list of values that are stored at in the signal or variable or the user can plot the waveform of the signals and variables. These data sets can be either represented as binary, hex, decimal or various other data types that are available in the software suite.

The figures that are presented below are the data values from the range of 161845 to the range 161855. The same range is presented in the figures above which are the two Python console figures. It can be noted from that figure that the models in Python and the models in vhdl all have the exact same value for the same data range.

Fig. 16: ModelSim signal list for model 1

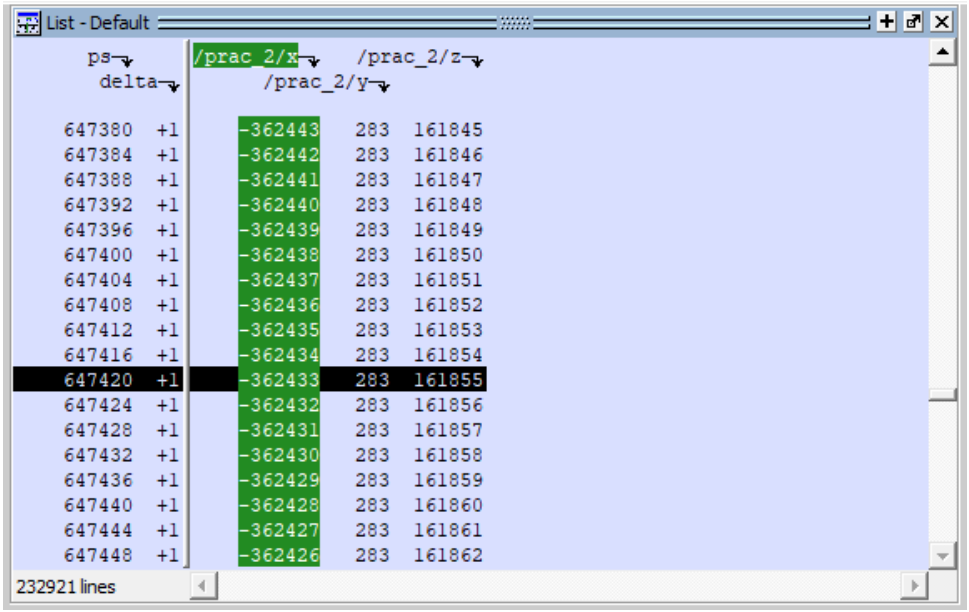

Fig. 17: ModelSim signal list for model 2

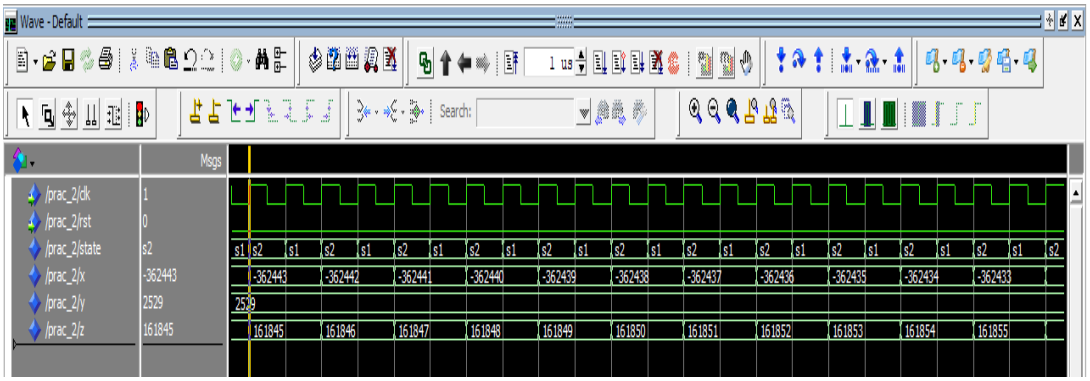

Fig. 18: ModelSim waveform for model 1

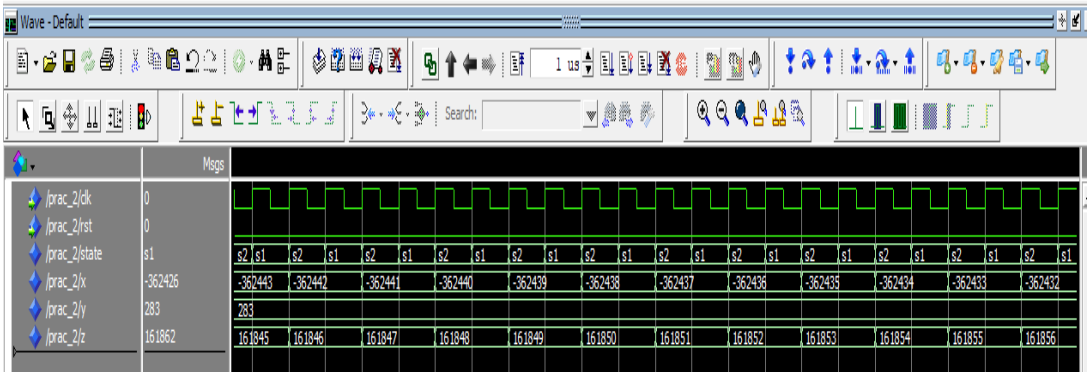

Fig. 19: ModelSim waveform for model 2