Design Tangent Hyperbolic on Digital Hardware – FPGA

Contents

Introduction

The quick development of artificial intelligence in the most of industries shows its dramatic important. Specifically, neural networks (NNs) play a vital role in many fields, such as computer vision, natural language processing, medicine and automatic driving [1]. ANNs have been designed to mimic the functions of the human brain that learn from experiences and adapt accordingly to the situation. Like the human brain has a multi-tiered structures which are containing billions of neurons arranged in a hierarchy layer, ANN also has a network of neurons that are interconnected to each other via axons.

The NNs are implemented on the combination of (Central Processing Unit) CPU and graphics processing unit (GPU) conventionally, which results in high costs and power consumption, although its precision can be increased by a substituted proper solution. Recently, scientists have found field-programmable gate array (FPGA) with low-cost and power architectures for NNs implementation [1].

There is simple perceptron on the figure 1, and each input like x1, x2, …, xn would be multiplied to their own weights then activation function will be applied in order to convert the linear input signals of a node into non-linear output signals to facilitate the learning of high order polynomials that go beyond one degree for deep networks. The output will be compared to expected result and the difference will be the absolute error.

The next process will be backward propagation which includes inverse direction from output towards input and compute differentiation of activation function for optimal point due to optimal point leads to have minimum error which is the fundamental desired in the NNs. Each of come and back as forward and backward propagation has been named as epoch, number of epochs nominates total iteration in NNs.

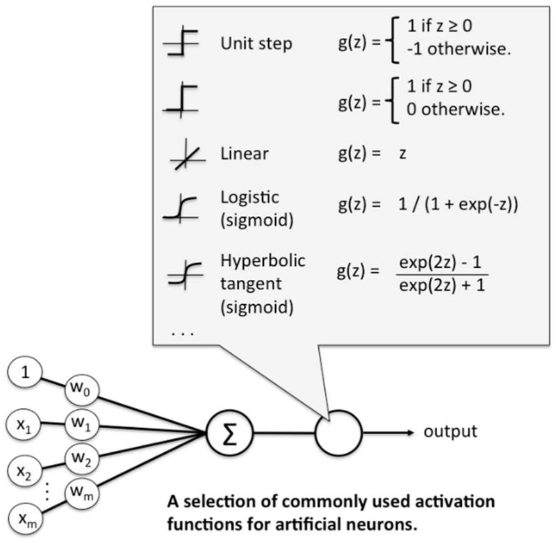

Unit step
g(z) = { 1 if z ≥ 0; -1 otherwise.
g(z) = { 1 if z ≥ 0; 0 otherwise.
Linear
g(z) = z
Logistic (sigmoid)
g(z) = 1 / (1 + exp(-z))
Hyperbolic tangent (sigmoid)
g(z) = (exp(2z) - 1) / (exp(2z) + 1)
...
1
w0
x1
w1
x2
w2
xm
wm
Σ
output
A selection of commonly used activation functions for artificial neurons.

Figure 1: Neural Network and Neurons, Weights, Activation Functions

Activation Function

The requirement in neural network is to have a better predict and higher precision in the results. So, the nonlinear functions have more rounding and bending which can categorize clusters more precisely. In the comparison between linear and nonlinear functions, it is obvious that nonlinear is more accurate to predict and has better boundary decision line for categorizing between two different classes.

In the figure 2, the left picture shows linear function which could not separate one of red circle while by the aid of nonlinear function at the right picture all of red circle and blue triangle could be determined perfectly. Activation functions as exponential or tangency function would be applied on the NNs through forward and backward propagation in order to find better wight which leads to have less error.

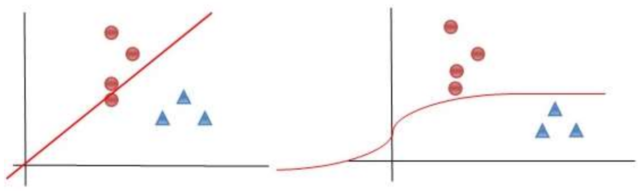
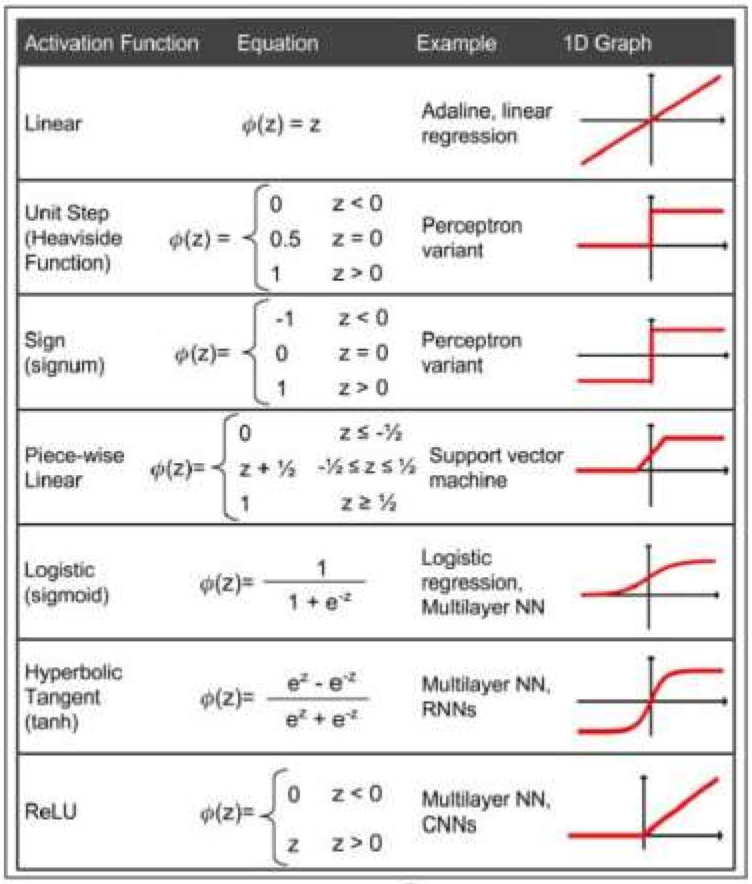


Figure 2: Linear and Nonlinear Functions.

These sort of activation functions must be differentiable because of backward propagation needs to compute the better weights and answer and it needs to calculate minimum optimal point in this process. The list of activation function has been listed on the figure 3 such as sigmoid, tangent hyperbolic (Tanh), SoftMax and Rectified Linear Unit (ReLU), each of these activation functions has their own cons and pros.



Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z)= \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z)= \begin{cases} 0 & z \le -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \le z \le \frac{1}{2} \\ 1 & z \ge \frac{1}{2} \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z)= \frac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z)= \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z)= \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

Figure 3: List of Activation Functions [2].

Sigmoid

The sigmoid function is a non-linear activation function in feedforward neural networks. It is a differentiable real function and has a specific degree of smoothness. The sigmoid function appears in the output layer of the NNs models and is used for predicting probability-based outputs. The sigmoid function is represented as:

$$f(x) = \left(\frac{1}{1 + exp^{-x}}\right) \tag{1}$$

Gradient saturation is problem in sigmoid which means that after some epochs, the learning happens fast, but at some points the value of the linear part will be far from the center of the sigmoid and it saturates such as figure 4. Indeed, gradient saturation takes too much time to update the weights because the value of gradient is small.

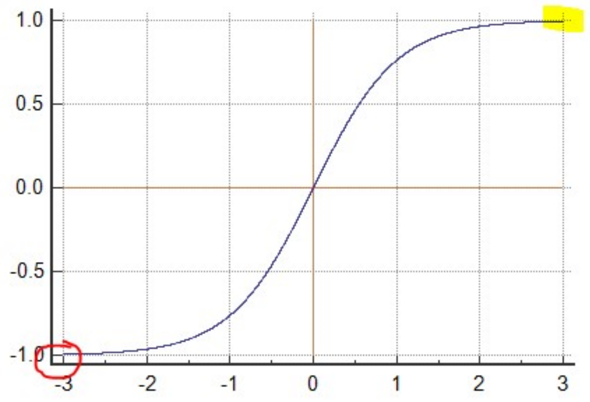


Figure 4: Gradient Saturation.

Another drawback in sigmoid function is slow convergence, sharp damp gradients during backpropagation from within deeper hidden layers to the input layers, and non-zero centered output that causes the gradient updates to propagate in varying directions.

Tangent Hyperbolic

The hyperbolic tangent function is a smoother, zero-centered function having a range between -1 to 1. As a result, the output of the tanh function is represented by:

$$f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2}$$

The tanh function delivers better training performance for multilayer neural networks rather than sigmoid function. The biggest advantage of the tanh function is that it produces a zero-centered output, for supporting the backpropagation process. The tanh function has been mostly used in recurrent neural networks (RNN) for natural language processing (NLP) and speech recognition (SR).

The problem of Tanh is that it cannot solve the vanishing gradient problem. The gradient will be increased exponentially when NNs in propagate the model until they eventually explode. Also, the Tanh function can only obtain a gradient of 1 when the input value is 0 (x is zero). As a result, the function can produce some dead neurons during the computation process.

Rectified Linear Unit (ReLU)

ReLU is a fast-learning activation function which gives the best performance and final results, due to its own generalization and as in this thesis mentioned, generalization is the uppermost and

elegant feature of active functions in order to reduce time consumption and enhance the precision. ReLU is a nearly linear function that retains the properties of linear models, which makes them easy to optimize with gradient-descent methods. ReLU guarantees faster computation – it does not compute exponentials and divisions, thereby boosting the overall computation speed
The ReLU function performs a threshold operation on each input element where all values less than zero are set to zero. Thus, the ReLU is represented as:

$$f(x) = \max(0, x) = \begin{cases} x_i & if\ \ x_i \geq 0 \\ 0 & if\ \ x_i < 0 \end{cases} \qquad (3)$$

Hardware Digital Implementation

Artificial Neural network is popular computing model among the rest one because of its precisions and abilities to work with big data. ANNs are applied wide fields from in instrumentation and measurements and bio-informatics to digital signal processing.

There are many operations in ANNs, as it was mentioned above in the activation function part, such as multiplications and additions specifically in deep learning which has many layers in hidden layers. Also there are multiple activation functions in each layer in order to find the better weight which leads to the best precisions.

All of mentioned operations like multiplications, additions and exponential, nonlinear functions in activations cause cost with the high time consumption.

The ANNs are implemented in dedicated hardware such as Field Programmable Gate Array (FPGA) or Application Specific

Integrated Circuits (ASIC) due to cost-effective and real time results [3]. Although computing directly on FPGA or ASIC with nonlinear activation functions with conventional arithmetic circuits involve large number of multiplication and addition gates which creates problems.

There are multiple approaches to prevent this dilemma and evaluate hardware implementation. The approaches in this thesis are Lookup-Table, Range Addressable Lookup-Table (RALUT), Piece Wise Linear (PWL), Power of Two, Cordic.

Field Programmable Gate Array (FPGA)

FPGAs are integrated circuits which have the abilities to be programmed by customers after sale. FPGA configurations need Hardware Description Language (HDL). FPGAs are arrays of programmed logical blocks which are hierarchically connected in order to be adjusted in new design later.

Those logical blocks can be configured for complicated tasks or simple logical gate such as AND/OR. In the most FPGAs, logical blocks include memory elements which are likely simple flip flops or more complete memory block.

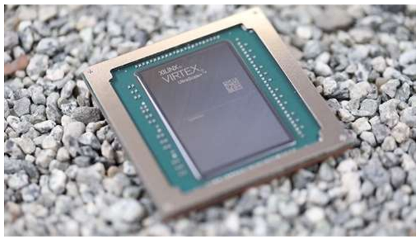


Figure 5: FPGA Board.

The fundamental difference between FPGAs and Micro Controllers is that in FPGAs all processing are done parallel while in Micro Controllers all of tasks must be sequentially which makes problem when processors encounter big computations.

Indeed, FPGAs are something between electronic circuit and a Micro Controller. Therefore, FPGAs have the capabilities to perform the tasks which are hardly done by other processors such as CPU.

FPGA Advantages

a. **High Speed**
   FPGAs perform Real-Time (in the Nano seconds range) computation directly from electronic programmable elements while CPUs has convention processing way which takes time too much and is costly.

b. **Parallel Processing**
   There are many inputs in FPGAs which allows to receive many inputs and have their own arithmetic computation independently. So, there is no queue for inputs to wait for the other operations to be completed.

c. **Flexibility**
   The FPGA tasks depends on designer completely and there is no common point with CPUs which are prepared such as electronic Printed Circuit Board (PCB) that has

predefined and fixed map. There is possibility to change FPGAs in the any state.

d. **Cost and Product Development Time**
The change in FPGAs is without hardware manipulation and it causes rapid development while in the case of building board from scratch needs more time for instance if in the meanwhile verification had wrong result so designer has to change the sketch and build the board again and it will take a lot of time.

FPGA Disadvantage

a. **Single Purpose**
CPUs are General Purpose and operating system control them as they prefer. The CPU task is different time by time and event their own frequency which determines CPU speed and also their utilization which defines CPU usage size are in changing. While FPGAs always perform single task and in the case of changing task, the board design for gates should be corrected.

Although it is not a weak point and single purpose can be strong point in some conditions, and this feature make FPGAs similar to ASICs. The main difference between FPGAs and ASICs is that ASICs are designed for specific task and after that there is no chance to change anything while its performance is too much high but also expensive choice.

b. **Complicated Design**
   FPGA design would be done by one of the languages such as Verilog or VHDL. These languages are different from ordinary programming language and have complex concept to understand. They are Hardware Description Language.

c. **Expensive Board**
   FPGAs are expensive rather than the other micro controllers for example Xilinx Sparten-6 is about 20 times more than Discovery STM-32.

Application Specific Integrated Circuits (ASIC)

Hardware Description Language (HDL)

FPGA design would be done by one of the languages such as Verilog or VHDL. They are Hardware Description Language. The main difference between these languages and the other ones is HDLs can define propagation delay and signal strength.

While Verilog and VHDL are very similar to each other but Verilog is mostly for chip design and VHDL is dedicated for FPGA boards. FPGAs are set of logical gates which will work together as designers specify for them. The more precise definition for FPGAs is that they have three elements as Lookup-Table, Flip Flop and Routing Max.

Lookup-Table

Retrieving/Storing values directly from/to memory is the fastest way which reduce dramatically time consumption. Look up tables are verification tables which determine operation for each element of FPGAs. Lookup Tables are multiple inputs and one output.

There is a memory inside each lookup table where stores relation between inputs and output. The number of inputs are determined by the digit which comes with LUT6 means there are 6 inputs. For instance, the below figure is LUT2 which has 2 inputs. The memory size for lookup table can be whatever and here with two inputs and one output there is AND gate as follow:

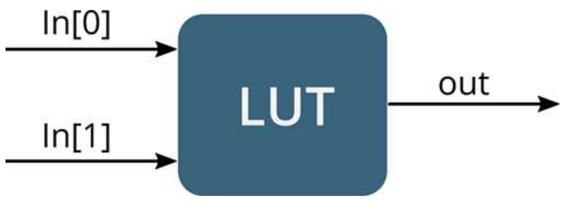
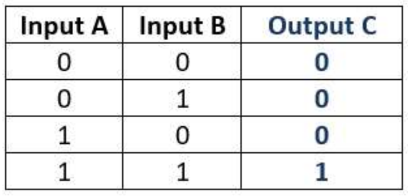


Figure 6: FPGA Lookup Table, LUT2.



Input A	Input B	Output C
0	0	0
0	1	0
1	0	0
1	1	1

Figure 6: Lookup Verification Table.

Flip Flop

The output of each lookup can be connected to flip flops. It is necessary for building arithmetic units such as addition and multiplications. Slice includes multiple lookup tables and flip flops.

Routing Matrix

The most important part is to specify the relationship between slices. There are two slices in the one Complex Logic Block (CLB) in FPGAs. Each CLB is connected to Switch Matrix which has a duty to connect CLBs to the other part of circuit. This switch can connect every input/output of this unit to the rest of other input/output of units through the General Routing Matrix (GRM). GRM connect all of the board together and has hub role in FPGAs.

The main part of GRM is wire and Multiplexers. Multiplexer is an electronic unit with multiple inputs and one output and in the specific second, the special input is connected to output. Multiplexer is for addressing tasks and due to this, multiplexer is using for connecting among CLBs. Multiplexer definitions store inside RAM and because of that, there is an external micro controller besides FPGAs.

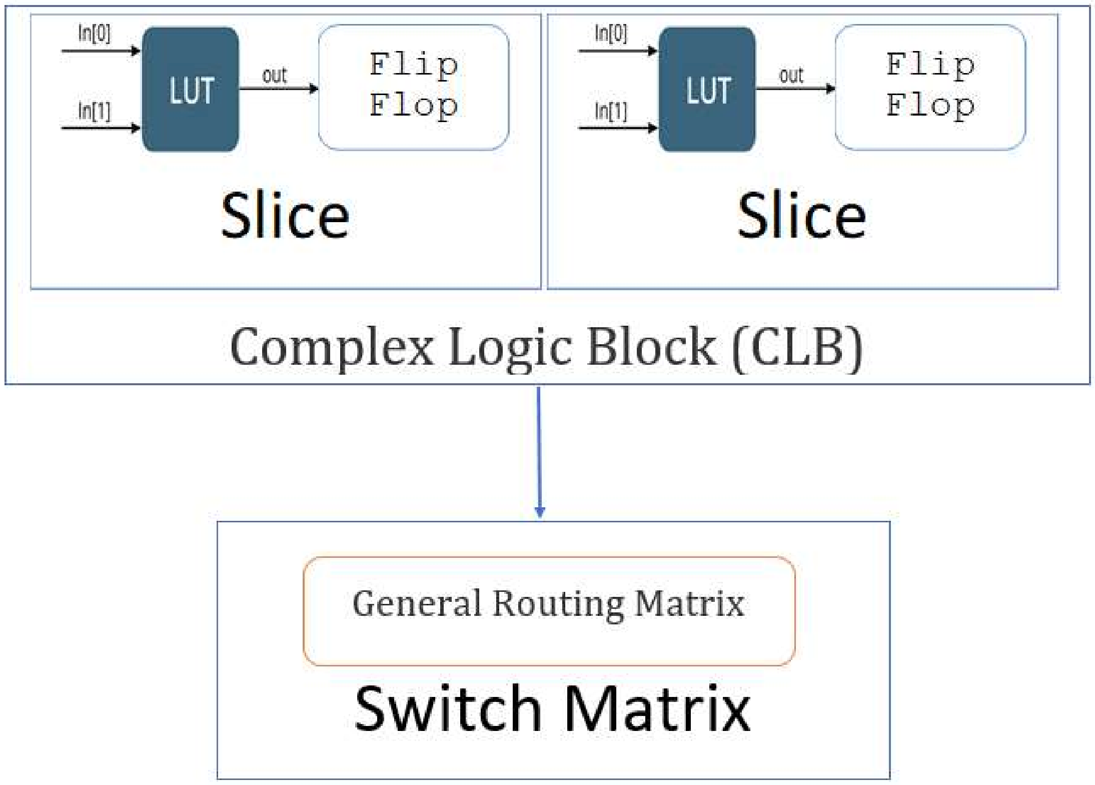


Figure 7: FPGA Internal Board Architecture.

Range Addressable Lookup-Table (RALUT)

Range addressable lookup tables (RALUTs) are similar to lookup tables, with a few notable exceptions. Most importantly, each output in a RALUT corresponds to a range of inputs [12]. This restriction of one output for multiple inputs effectively reduces the number of values that must be stored in the table, while it enhances precision at the same time.

The feature that makes RALUT as elegant solution is that it contains a set of entries addressed to the same output and allows the development of smaller tables without any loss of precision. Moreover, it can reduce hardware resource.

This can be highly efficient in situations where the output does not change over certain input ranges. A classic lookup table would possess a unique (and relatively constant) output for every address in the specific range, whereas a range addressable lookup table stores a single output.

Literature

An Optimized Lookup-Table for the Evaluation of Sigmoid Function for Artificial Neural Networks [3]

The proposed efficient hardware implementation solution in this article is to cutoff redundant cost due to large number of arithmetic circuits by the aid of combination using of piecewise linear (PWL) and Lookup-Table. The best practice was Lookup-Table structure for evaluation of hyperbolic tangent which stores 15 words, 1 Multiplexer, a pair of selective-sign converters with 15 XOR, 15 AND, 13 OR gates, plus one range-decoder with 83 AND/OR gates and 1 NOT gate.

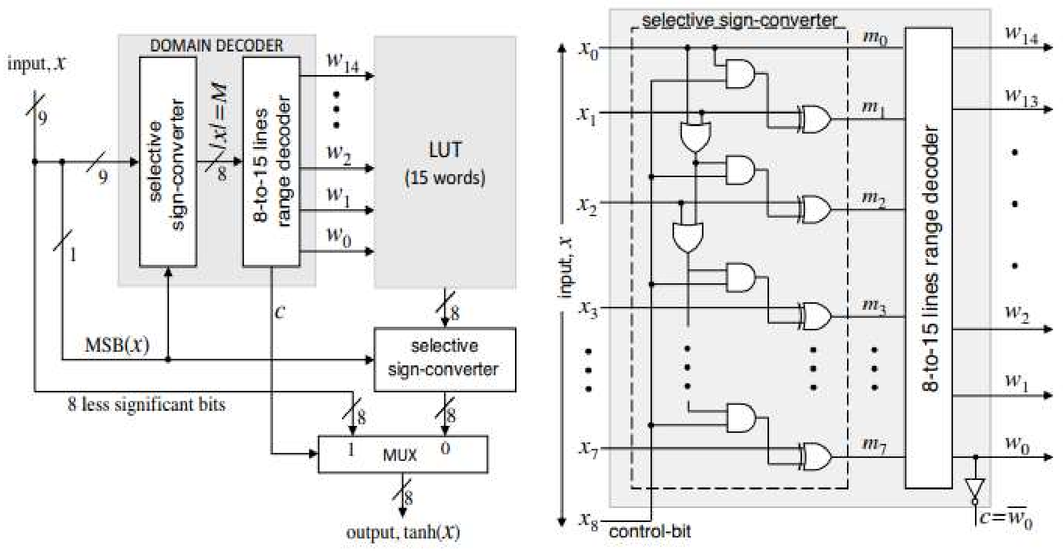


Figure 8: Left: Piecewise Linear and Lookup-Table Solution for Hyperbolic Tangent Sigmoid, Right: Conventional Logical Circuits. [3]

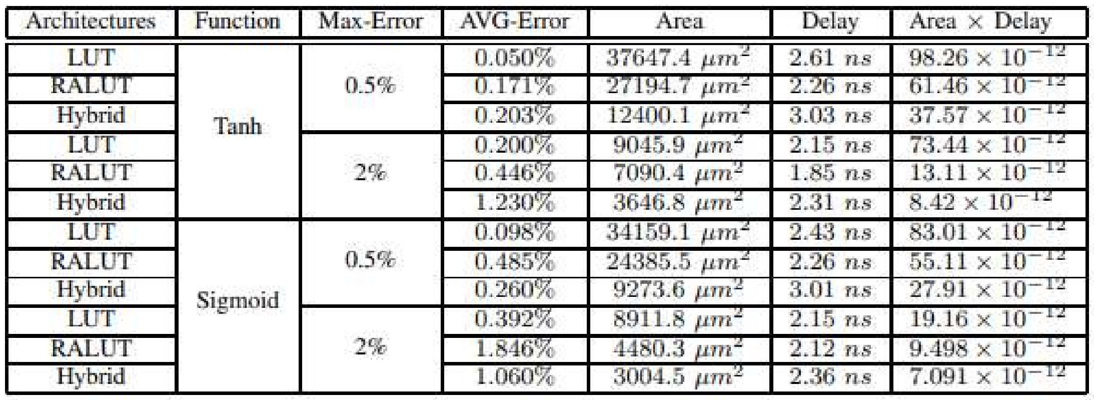
Kumar Meher proposed a novel idea which is an optimized scheme for lookup table implementation of hyperbolic tangent function. There is a basic symmetry property of hyperbolic tangent to have large sub-domains of values where each sub-domain needs to store only one word in the lookup table. Moreover, they used another solution to reduce lookup table size which was linear behavior of hyperbolic tangent for small values.

Eventually the result was less lookup table size in the comparison to the same scenarios for specific limit of accuracy. The last but not least was to design a simple combinational circuit for selective sign-conversion for more reduction of hardware complexity.

Artificial Neural Networks Activation Function HDL Coder [4]

In this article, a new MATLAB toolbox that improves the hardware implementation of artificial neural networks activation function is suggested. This toolbox generates synthesizable platform independent HDL code that approximates the hyperbolic tangent or the sigmoid functions in hardware according to user needs. The hybrid of multiple methods is used here to reduce resources while keep or even enhance precision.

Three different approximation methods are programmed in the toolbox which are piece wise linear function, lookup table and range addressable lookup table, avoid the use of hardware multipliers for high performance. Different speed/area tradeoff designs are easily realized using the SigTan HDL Coder.



Architectures	Function	Max-Error	AVG-Error	Area	Delay	Area × Delay
LUT	Tanh	0.5%	0.050%	37647.4 μm^2	2.61 ns	98.26×10^{-12}
RALUT			0.171%	27194.7 μm^2	2.26 ns	61.46×10^{-12}
Hybrid			0.203%	12400.1 μm^2	3.03 ns	37.57×10^{-12}
LUT		2%	0.200%	9045.9 μm^2	2.15 ns	73.44×10^{-12}
RALUT			0.446%	7090.4 μm^2	1.85 ns	13.11×10^{-12}
Hybrid			1.230%	3646.8 μm^2	2.31 ns	8.42×10^{-12}
LUT	Sigmoid	0.5%	0.098%	34159.1 μm^2	2.43 ns	83.01×10^{-12}
RALUT			0.485%	24385.5 μm^2	2.26 ns	55.11×10^{-12}
Hybrid			0.260%	9273.6 μm^2	3.01 ns	27.91×10^{-12}
LUT		2%	0.392%	8911.8 μm^2	2.15 ns	19.16×10^{-12}
RALUT			1.846%	4480.3 μm^2	2.12 ns	9.498×10^{-12}
Hybrid			1.060%	3004.5 μm^2	2.36 ns	7.091×10^{-12}

Table 1: Complexity of Implementations

The above table shows different area-delay trade-offs that can be obtained using the different approximation errors. Lookup table approximations (LUT and RALUT) present faster designs compared to the hybrid solutions at the cost of increased area utilization. On the other hand, range addressable lookup table implementations are even faster than the regular lookup tables and it is consuming less area.

RALUT implementations become less efficient compared to the lookup table when input range becomes smaller and gets closer to the origin. The hybrid approach is the best trade-off for area and delay.

Methodology

MATLAB to FPGA using HDL Coder

There is a capability in HDL Coder Properties that generates synthesizable VHDL/Verilog code directly from MATLAB and some of the key features of this process in MATLAB is presented in below workflow.

Introduction to HDL Code Generation from MATLAB

FPGAs supplies a good compromise between general purpose processors (GPPs) and Application Specific Integrated Circuits (ASICs). GPPs are fully programmable but are less efficient in terms of power and performance; ASICs implement dedicated functionality and present the best power and performance, while need huge expensive design validation and implementation.

FPGAs are also used for prototyping in ASIC workflows for hardware verification and early software development. Because of performance improvement when running high-throughput, high-performance applications, algorithm designers are mostly using FPGAs to prototype and validate their innovations instead of using conventional processors.

However, many of the algorithms are implemented in MATLAB because of the simplicity in programming model and rich analysis and visualization capabilities. For the most developers mastering the FPGA design workflow is a challenge. Unlike software algorithm development, hardware development requires them to think parallel. While there are many obstacles such as learning the VHDL or Verilog language, mastering IDEs from FPGA vendors such as Xilinx, and understanding difficult concepts like "delay balancing".

There is simple workflow as below figure which describes an easy path from MATLAB code to HDL code for FPGAs. It is possible to automatically generate HDL code from MATLAB algorithm, implement the HDL code on an FPGA, and using MATLAB to verify HDL code.

MATLAB to Hardware Workflow

The process of translating MATLAB designs to hardware consists of the following steps:

1. Modeling algorithm in MATLAB.

2. Generating HDL code – It is automatically created HDL code for FPGA prototyping.
3. Verify HDL code - reuse MATLAB test bench to verify the generated HDL code.
4. Create and verify FPGA prototype - implement and verify design on FPGAs.

There are some challenges in converting MATLAB code to HDL code. MATLAB code is procedural and can be highly abstract; it can use floating-point data. Complex loops can be inferred from matrix operations and toolbox functions.

Implementing MATLAB code in hardware involves

- Converting floating-point in to fixed-point in MATLAB code with the optimized bit widths suitable for efficient hardware generation.
- Identifying and mapping procedural constructs to parallel area, speed optimization hardware operations.
- Introducing the concept of time by adding clocks and clock rates to schedule the operations in hardware.
- Creating resource-shared architectures to implement expensive operators like multipliers and for-loop bodies.
- Mapping large persistent arrays to block RAM in hardware

HDL Coder simplifies the above tasks though the function approximation.

Example MATLAB Algorithm

Lookup table MATLAB function implements hyperbolic tangent. This algorithm, implemented in MATLAB, enhances hardware implementation for activation function by using lookup tables.

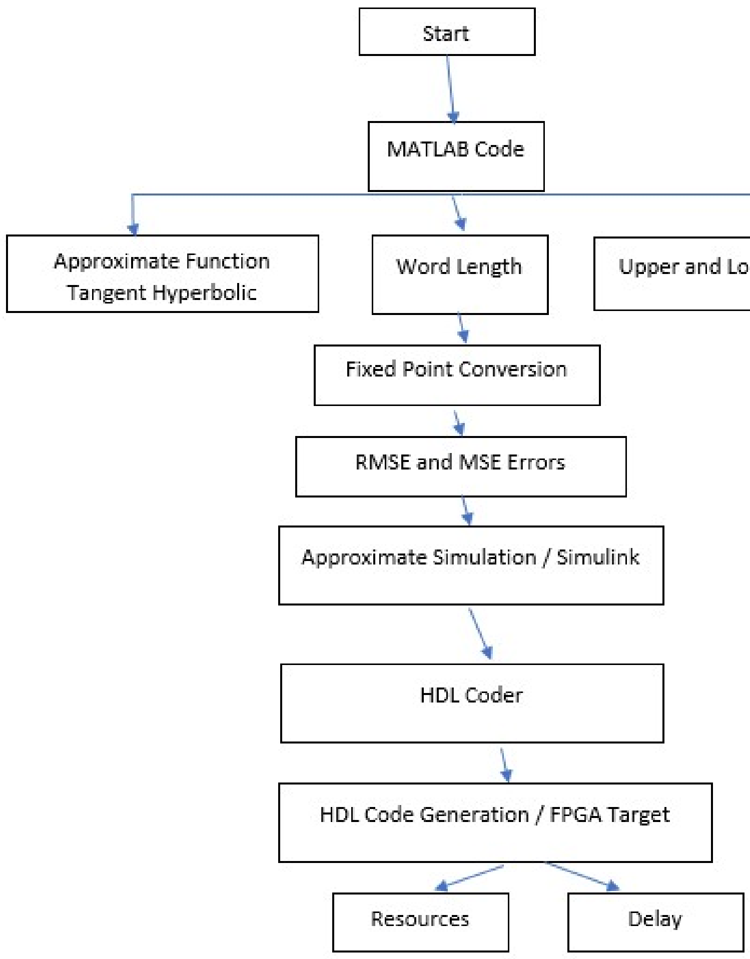


Figure 9: MATLAB Code to HDL Code Conversion Workflow.

Pseudo code

```
Pseudo code for Tangent Hyperbolic Lookup Table

1. Function Determination.

      a. FunctionApproximation.Problem = Tangent Hyperbolic

2. Initialization of fixed-point data sample.

      b. Bound = (-4, 4)
      c. Word-Length = [4, 8, 14, 16]
      d. BreakpointSpecification = [EvenSpacing, ExplicitValues,
         EvenPow2Spacing]

3. Problem Solving.

4. Problem Comparing

5. Problem Approximating.
```

Process

Breakpoint Specification

Spacing of breakpoint data, specified as one of the following values.

ExplicitValues:

Lookup table breakpoints are specified explicitly. Breakpoints can be closer together for some input ranges and farther apart in others.

EvenSpacing:

Lookup table breakpoints are evenly spaced throughout.

EvenPow2Spacing:

Lookup table breakpoints use power-of-two spacing. This breakpoint specification boasts the fastest execution speed because a bit shift can replace the position search.

On Curve Table Values

Whether to constrain table values to the quantized output of the function being approximated

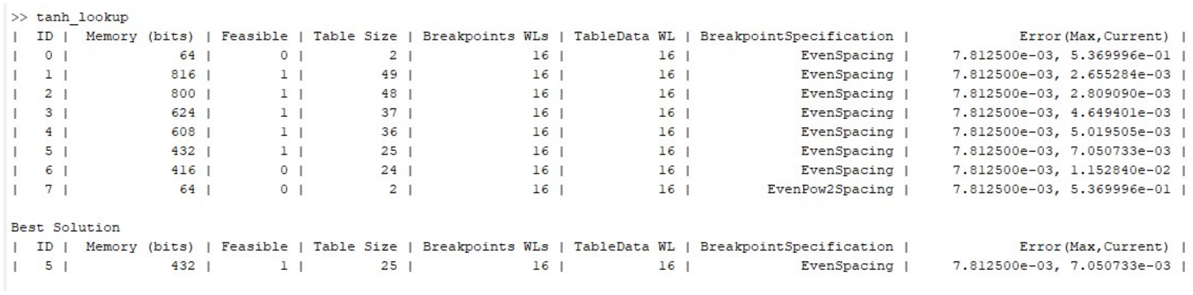
false or 0 (default) | true or 1

Whether to constrain table values to the quantized output of the function being approximated, specified as a numeric or logical 1 (true) or 0 (false). By setting this property to 0 (false) and allowing off-curve table values, you may be able to reduce the memory of the lookup table while maintaining the same error tolerances, or maintain the same memory while reducing the error tolerances.

Result and Discussion

The final result from this part of thesis which have been done Lookup table and RALUT is that the RALUT has a better performance either in resources or precisions, while lookup table uses more resources. Also RALU delay time is on Xilinx Vivado is about 0.03 and lookup table is 8.574 Nano seconds. It presents that how much hardware implementation has effect on the result and eventually using hybrid of these approaches leads us to the best area/delay tradeoff results.

Lookup VS RALUT



```
>> tanh_lookup
|  ID |  Memory (bits) | Feasible | Table Size | Breakpoints WLs | TableData WL | BreakpointSpecification |          Error(Max,Current) |
|   0 |             64 |        0 |          2 |              16 |           16 |             EvenSpacing |    7.812500e-03, 5.369996e-01 |
|   1 |            816 |        1 |         49 |              16 |           16 |             EvenSpacing |    7.812500e-03, 2.655284e-03 |
|   2 |            800 |        1 |         48 |              16 |           16 |             EvenSpacing |    7.812500e-03, 2.809090e-03 |
|   3 |            624 |        1 |         37 |              16 |           16 |             EvenSpacing |    7.812500e-03, 4.649401e-03 |
|   4 |            608 |        1 |         36 |              16 |           16 |             EvenSpacing |    7.812500e-03, 5.019505e-03 |
|   5 |            432 |        1 |         25 |              16 |           16 |             EvenSpacing |    7.812500e-03, 7.050733e-03 |
|   6 |            416 |        0 |         24 |              16 |           16 |             EvenSpacing |    7.812500e-03, 1.152840e-02 |
|   7 |             64 |        0 |          2 |              16 |           16 |         EvenPow2Spacing |    7.812500e-03, 5.369996e-01 |

Best Solution
|  ID |  Memory (bits) | Feasible | Table Size | Breakpoints WLs | TableData WL | BreakpointSpecification |          Error(Max,Current) |
|   5 |            432 |        1 |         25 |              16 |           16 |             EvenSpacing |    7.812500e-03, 7.050733e-03 |
```

Figure 10: Lookup Table

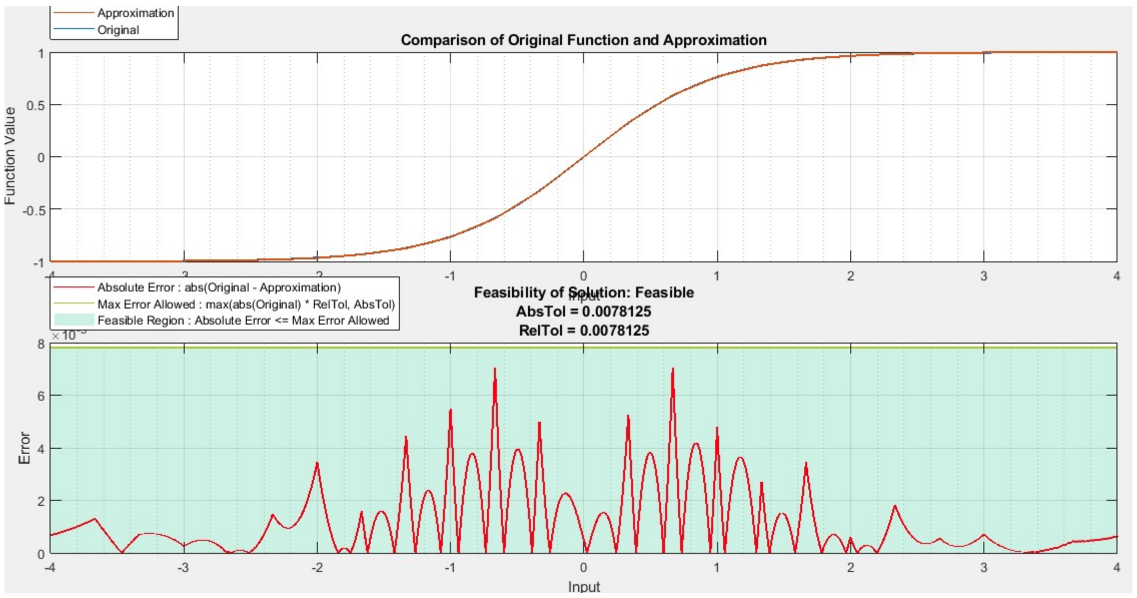
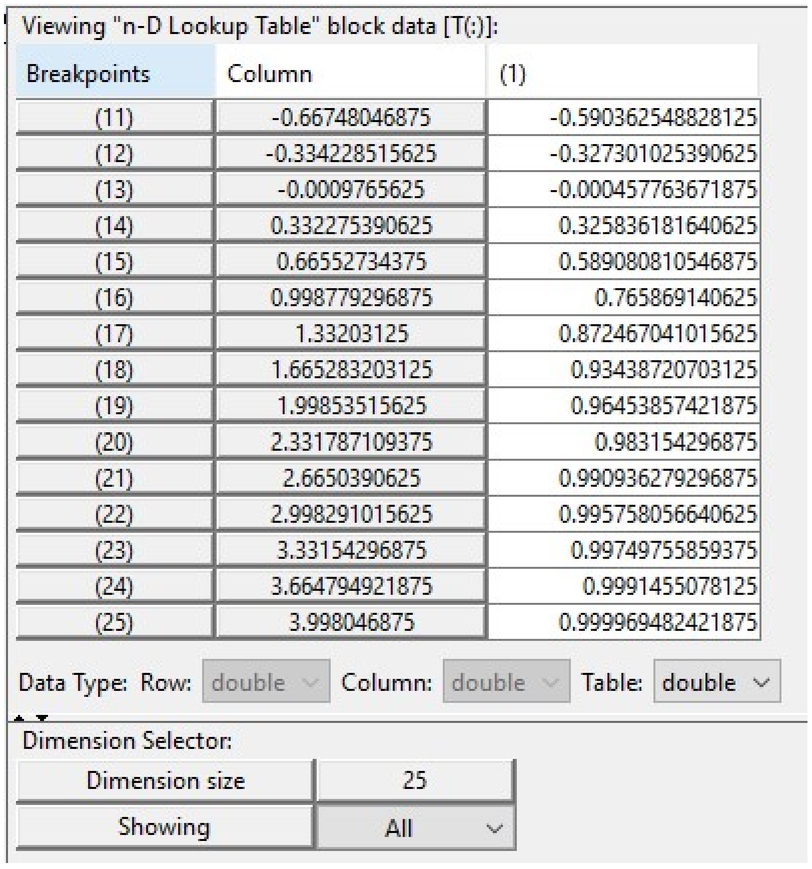


Figure 11: Lookup Table Function.



Viewing "n-D Lookup Table" block data [T(:)]:

Breakpoints	Column	(1)
(11)	-0.66748046875	-0.590362548828125
(12)	-0.334228515625	-0.327301025390625
(13)	-0.0009765625	-0.000457763671875
(14)	0.332275390625	0.325836181640625
(15)	0.66552734375	0.589080810546875
(16)	0.998779296875	0.765869140625
(17)	1.33203125	0.872467041015625
(18)	1.665283203125	0.93438720703125
(19)	1.99853515625	0.96453857421875
(20)	2.331787109375	0.983154296875
(21)	2.6650390625	0.990936279296875
(22)	2.998291015625	0.995758056640625
(23)	3.33154296875	0.99749755859375
(24)	3.664794921875	0.9991455078125
(25)	3.998046875	0.999694824221875

Data Type: Row: double Column: double Table: double

Dimension Selector:

Dimension size	25
Showing	All

Figure 12: Break Points.

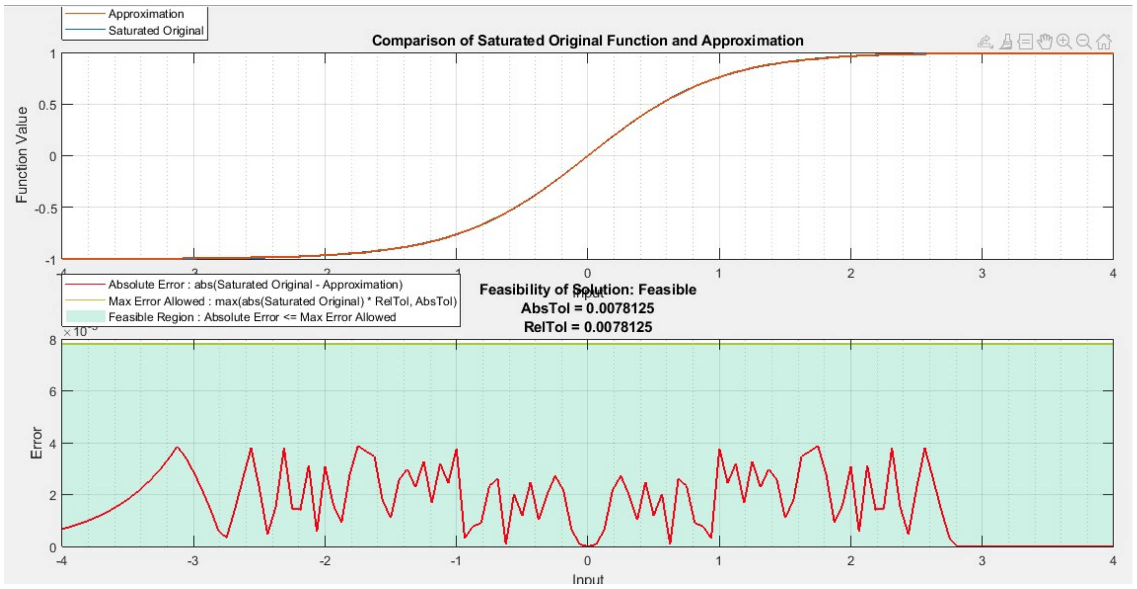
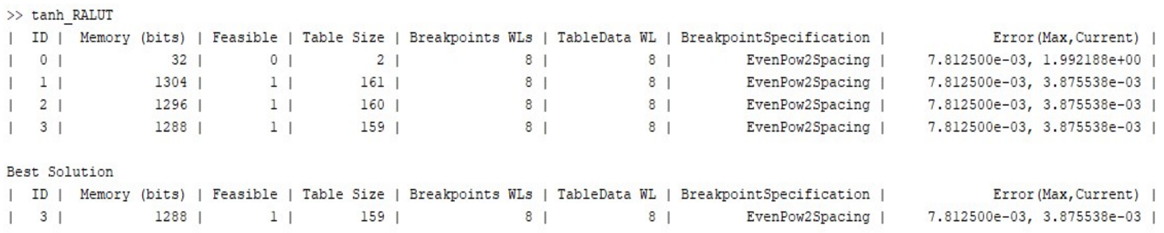


Figure 13: Range Addressable Function.



```
>> tanh_RALUT
|  ID |  Memory (bits) | Feasible | Table Size | Breakpoints WLs | TableData WL | BreakpointSpecification |           Error(Max,Current) |
|   0 |             32 |        0 |          2 |               8 |            8 |          EvenPow2Spacing |    7.812500e-03, 1.992188e+00 |
|   1 |           1304 |        1 |        161 |               8 |            8 |          EvenPow2Spacing |    7.812500e-03, 3.875538e-03 |
|   2 |           1296 |        1 |        160 |               8 |            8 |          EvenPow2Spacing |    7.812500e-03, 3.875538e-03 |
|   3 |           1288 |        1 |        159 |               8 |            8 |          EvenPow2Spacing |    7.812500e-03, 3.875538e-03 |

Best Solution
|  ID |  Memory (bits) | Feasible | Table Size | Breakpoints WLs | TableData WL | BreakpointSpecification |           Error(Max,Current) |
|   3 |           1288 |        1 |        159 |               8 |            8 |          EvenPow2Spacing |    7.812500e-03, 3.875538e-03 |
```

Figure 14: RALUT has less output.

Xilinx Vivado VS Altera Quartus II

FPGA	Word Length	Delay	Multipliers	Adders/Subtractors	Multiplexers	I/O
Xilinx Vivado	16	8.574	2	6	15	32
Altera Quartus II	16	6.042	2	6	15	32
Xilinx Vivado	14	8.574	2	6	18	32
Altera Quartus II	14	6.042	2	6	18	32
Xilinx Vivado	8	8.574	1	5	7	32
Altera Quartus II	8	6.042	1	5	7	32
Xilinx Vivado	4	8.574	2	6	16	8
Altera Quartus II	4	6.042	2	6	16	8

Conclusion

In this part of the thesis, a popular MATLAB solution as Approximation Function that improves the hardware implementation of artificial neural networks is proposed. The function finally generates synthesizable platform independent HDL code that approximates the hyperbolic tangent functions in hardware according to user needs (word length, points, breakpoints).

Two different approximation methods are programmed in this function, both of them (Lookup and RALUT) use less of hardware multipliers for high performance. Different speed/area tradeoff designs are easily realized using the HDL Coder.

Reference

[1] 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," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 12, pp. 2540-2550, Dec. 2020, doi: 10.1109/TVLSI.2020.3015391.

[2] D. Sharma, H. Koyuncu, "Deep learning approach to analyse, detect and classify COVID-19 patients," in NTMSCI 9, Special Issue, No. 1, 1-12, July 2021.

[3] P. K. Meher. "An Optimized Lookup-Table for the Evaluation of Sigmoid Function for Artificial Neural Networks",

[4] A. H. Namin, K. Leboeuf, H. Wu, M. Ahmadi, "Artificial Neural Networks Activation Function HDL Coder".