High Speed VLSI Implementation of the Hyperbolic Tangent Sigmoid Function

Karl Leboeuf, Ashkan Hosseinzadeh Namin, Roberto Muscedere, Huapeng Wu, and Majid Ahmadi
Department of Electrical and Computer Engineering, University of Windsor
401 Sunset Avenue, Windsor, Ontario N9B 3P4 Canada
Email: {leboeu3, hossei1, rmusced, hwu, ahmadi}@uwindsor.ca

Abstract

The hyperbolic tangent function is commonly used as the activation function in artificial neural networks. In this work, two different hardware implementations for the hyperbolic tangent function are proposed. Both methods are based on the approximation of the function rather than calculating it, since it has an exponential nature. The first method uses a lookup table to approximate the function, while the second method reduces the size of the table by using range addressable decoding as opposed to the classic decoding scheme. Hardware synthesis results show that proposed methods perform significantly faster, and use less area compared to other similar methods with the same amount of error.

1. Introduction

Artificial neural networks (ANNs) are currently employed for many diverse purposes, ranging from image classification to motor control [2, 6]. Since ANN systems are computationally intensive, they require large execution times in software implementations. Hardware implementations can eliminate this issue. One of the challenges presented when designing a hardware-based ANN system is the implementation of the activation function. There are several different activation functions available including, but not limited to, the sigmoid, hyperbolic tangent, and step functions [2, 6]. An important property of the activation function is that a continuous derivative exists, which is desirable when performing backpropagation-based learning. These functions are used to threshold the output of every artificial neuron; increasing the speed of the activation function will improve the entire system’s performance.

The hyperbolic tangent function is among the most widely used activation functions in ANNs. As it is shown in Fig. 1, this function produces a sigmoid curve, which is a curve having an “S” shape. Its variation is limited outside the range of (−2, 2).

Currently, there are several different approaches for the hardware implementation of the activation function. Piecewise linear approximation (PWL), lookup tables (LUTs), and hybrid methods have been widely used for this purpose [1, 5, 7]. With the use of current hardware synthesizers, LUTs are not only faster, but also occupy less area than piecewise linear approximation methods. In this work, range addressable lookup tables are proposed as a solution that offers advantages compared to simple LUT implementation in terms of speed and area utilization.

Range addressable lookup tables (RALUTs), similar to regular LUTs, are a type of non-volatile, read-only memory. In LUTs, every output corresponds to a unique address, whereas in RALUTs every output corresponds to a range of addresses as shown in Fig. 2. This type of table was originally proposed in [4] to implement highly nonlinear, discontinuous functions, and it will be shown to be suitable for implementing the hyperbolic tangent activation function. Depending on the desired accuracy, ranges of inputs will have the same output, which could be implemented more efficiently using RALUTs rather than a regular LUT.
In this work we initially present a LUT implementation for the hyperbolic tangent function. Hardware implementation results show that a LUT implementation was significantly faster and smaller than a recently published PWL approximation method, while possessing the same level of accuracy. RALUT implementations were presented afterwards, which yielded even further improvements over LUTs in area and speed. Hardware designs were implemented using a digital CMOS $0.18\mu m$ process; the same technology node used by the PWL implementation used in our comparison.

The rest of this paper is organized as follows. Section 2 briefly reviews previous work on hyperbolic tangent function implementation. Section 3 discusses a LUT-based approach, while section 4 examines the RALUT approach to implement the activation function. In section 5, a complexity comparison between several different methods is presented. Finally, section 6 discusses some concluding remarks.

2. A Brief Review of Different Hyperbolic Tangent Function Implementations

Efficient implementation of the activation function is an important part of designing an ANN system in hardware. The activation function is typically unsuitable for direct implementation since it is formed of an infinite exponential series. In practice, approximations of the function are used, as opposed to the function itself.

Currently, there are three main approaches used to approximate and implement the hyperbolic tangent function in hardware; lookup table (LUT) approximation, piece-wise linear (PWL) approximation, and hybrid methods, which are essentially a combination of the former two. Following is a brief overview of each of these methods.

2.1. Piecewise Linear Approximation

Piecewise linear schemes use a series of linear segments to approximate a function [1]. The number and location of these segments are chosen such that error, processing time, and area are minimized. This approach usually requires several clock cycles and the use of multipliers, which are expensive in terms of area. A piecewise linear approximation of the hyperbolic tangent function with five segments is shown in Fig. 3.

![Figure 3. Piecewise Linear Approximation of tanh(x) with Five Segments](image1)

2.2. Lookup Table Approximation

In this method, the function is approximated with a limited number of points [5]. The points are uniformly distributed across the entire input range. There is a direct relation between the number of bits used to represent the address (input) and output, and as such, care must be taken to ensure enough are used to minimize the error. A LUT approximation of the hyperbolic tangent function with eight points (a three bit input representation) is shown in Fig. 4.

![Figure 2. LUT and RALUT Addressing Schemes](image2)

2.3. Hybrid Methods

Hybrid methods use a combination of look-up tables and other hardware to generate the result of a function [7]. They typically take several clock cycles, however they do not employ multipliers, which significantly increases their speed.
3. Lookup Table Implementation of the Hyperbolic Tangent Function

The major advantages of using a LUT is its high operating speed, particularly when compared to PWL approximation which uses multipliers in its design. There are two different ways to implement a lookup table in hardware. The first is to use a ROM. The main drawback of this method is that no further optimization can be done after the exact input/output bit patterns are known.

The second method is to use a logic synthesizer to implement the table as a purely combinational circuit. This works well because the synthesizer performs very well in optimizing away large amounts of logic.

In our implementation, we generated MATLAB code to determine the number of input and output bits, as well as the output bit patterns themselves, for a table with a specified maximum error. For a maximum error of 0.04, 9 bits were used for both the input and output, whereas 10 bits were required to keep the maximum error below 0.02.

Once the input/output characteristics of the table were determined, HDL code employing them was written, and a hardware design was synthesized using Synopsys’ Design Compiler. Virtual Silicon standard library cells for a TSMC CMOS 0.18µm process were used for this design, and synthesis parameters were chosen to maximize operating speed. Hardware implementation results with a maximum error of 0.04 and 0.02 are summarized in the second row of tables 1 and 2 respectively.

4. Range Addressable Lookup Table Implementation of the Hyperbolic Tangent Function

A range addressable lookup table, originally proposed in [4] to accurately approximate non-linear, discontinuous functions, shares many aspects with the classic LUT with a few notable differences. In LUTs, every data point stored by the table corresponds to a unique address. In RALUTs, every data point corresponds to a range of addresses. This alternate addressing approach allows for a large reduction in data points, particularly in situations where the output remains constant over a range. An example of this is the hyperbolic tangent function, where the output changes only slightly outside the range of \((-2, 2)\). Rather than store every individual point, a single point is used to represent an entire range.

To implement the hyperbolic tangent function, we first wrote MATLAB code to select the minimum number of data points, while keeping the maximum error beneath a specified threshold. We were able to represent the activation function with 61 points using 9 bits for the inputs and outputs, with an error below 0.04. Using a 10 bit representation, only 127 were needed to maintain a maximum error below 0.02. The required number of points for these levels of maximum error using classic LUTs were 512 and 1024, respectively. This large reduction in stored values is what drives the RALUT approach to achieve better results than a LUT implementation of the same function.

Both sets of data points were passed on to HDL code, and the designs were synthesized with Synopsys Design Compiler using CMOS 0.18µm technology. Design parameters were chosen to maximize operating speed.
<table>
<thead>
<tr>
<th>Architectures</th>
<th>Max-Error</th>
<th>AVG-Error</th>
<th>Area</th>
<th>Delay</th>
<th>Area × Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheme-1 [3]</td>
<td>0.0430</td>
<td>0.0078</td>
<td>32069.83 $\mu^2$</td>
<td>903 $ns$</td>
<td>$2.895 \times 10^{-9}$</td>
</tr>
<tr>
<td>Proposed-LUT</td>
<td>0.0365</td>
<td>0.0040</td>
<td>9045.94 $\mu^2$</td>
<td>2.15 $ns$</td>
<td>$1.944 \times 10^{-11}$</td>
</tr>
<tr>
<td>Proposed-RALUT</td>
<td>0.0357</td>
<td>0.0089</td>
<td>7090.40 $\mu^2$</td>
<td>1.85 $ns$</td>
<td>$1.311 \times 10^{-11}$</td>
</tr>
</tbody>
</table>

Table 1. Complexity comparison of different implementations for 0.04 maximum error

<table>
<thead>
<tr>
<th>Architectures</th>
<th>Max-Error</th>
<th>AVG-Error</th>
<th>Area</th>
<th>Delay</th>
<th>Area × Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheme-2 [3]</td>
<td>0.0220</td>
<td>0.0041</td>
<td>83559.17 $\mu^2$</td>
<td>1293 $ns$</td>
<td>$1.080 \times 10^{-4}$</td>
</tr>
<tr>
<td>Proposed-LUT</td>
<td>0.0180</td>
<td>0.0020</td>
<td>17864.24 $\mu^2$</td>
<td>2.45 $ns$</td>
<td>$4.376 \times 10^{-11}$</td>
</tr>
<tr>
<td>Proposed-RALUT</td>
<td>0.0178</td>
<td>0.0057</td>
<td>11871.53 $\mu^2$</td>
<td>2.12 $ns$</td>
<td>$2.516 \times 10^{-11}$</td>
</tr>
</tbody>
</table>

Table 2. Complexity comparison of different implementations for 0.02 maximum error

5. Comparison of Different Hardware Implementations

Comparisons of hardware implementations for a maximum error of 0.04 and 0.02 are shown in tables 1 and 2. In table 1, the first row represents results from “Scheme-1”, which is an isosceles triangular approximation of the hyperbolic tangent function. In table 2, the same row shows results from “Scheme-2”, which is a PWL approximation of the hyperbolic tangent function. Both Scheme-1 and Scheme-2 designs were implemented using CMOS 0.18 $\mu$m technology; the same used by our implementations. Also note that all designs accept an input in the range of $(-8, 8)$.

Our proposed RALUT design was able to improve over the LUT implementation in both cases. With a maximum error of 0.04, the RALUT was 13% faster, and occupied 21.6% less area than the classic LUT approach. When the maximum error threshold was reduced to 0.02, the RALUT maintained a speed improvement of 13.5%, and area was further reduced by 33.5% compared to the LUT.

As can be seen from the tables, the LUT and RALUT designs prove to be significantly faster than the work recently presented in [3]. This is largely because our approach uses combinational logic exclusively, allowing results to appear after a single clock cycle, whereas multiple clock cycles are needed by the other designs. The “area × delay” was calculated as a performance metric to compare the overall efficiency of the designs. It is shown in the last column of tables 1 and 2.

6. Conclusion

The hyperbolic tangent function is commonly used as the activation function in artificial neural networks. In this work, two different hardware implementations for this function are proposed. The first method uses a classic LUT to approximate the function, while the second method uses a RALUT to do so. Hardware synthesis results show that proposed methods perform significantly faster, and use less area compared to other similar methods with the same amount of error. On average, the speed was improved by 13%, while area was reduced by 26% when using the second method compared to first in implementing a hyperbolic tangent function. The proposed designs can be used in the hardware implementation of ANNs.

References


