2002

Approximation of the sigmoid function and its derivative using a minimax approach

Jason Schlessman
Lehigh University

Schlessman, Jason

Approximation of the Sigmoid Function and Its Derivative Using a Minimax Approach

January 2003

APPROXIMATION OF THE SIGMOID FUNCTION AND ITS DERIVATIVE USING A MINIMAX APPROACH

by

Jason Schlessman

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Computer Engineering

Lehigh University

August 2002

This thesis is accepted in partial fulfillment of the requirements for the degree of Master of Science.

Aug. 26, 2002
(Date)

Prof. Michael J. Schulte
(Thesis Advisor)

Prof. Meghanad D. Wagh
(Co-Advisor)

Prof. Donald Bolle
(Chairperson of Department)

Acknowledgments

I would like to thank the following people regarding this thesis:

Michael Schulte, my thesis advisor, for his constant support and motivation as well as his willingness to work with me and include me in his research group.

Meghanad Wagh, my undergraduate advisor, for his inspirational teaching and compassion regarding my academic progress.

Terry Boult, my professor and independent research advisor, for his guidance throughout my academic career.

George Walters, a fellow graduate student, for providing his time, patience, and VHDL generation program.

Sharon and Warren Schlessman, my parents, for their endless support of my education at Lehigh.

Jamie Schlessman, my sister, for never giving up on me.

Megg Mass, Jim Weber, and Steve Haworth, for helping me move towards my academic career.

Shanna Terry, for her love, support, and inspiration throughout all facets of my life.

Table of Contents

List of Tables

List of Figures

Abstract

Accurate function approximation is essential for a number of applications, including signal processing, multimedia, and neural networks. The sigmoid function and its derivative are particularly important to neural network applications. Typically, the sigmoid function is used as a learning function and a threshold determinant for training neural networks. The sigmoid's derivative is used as an activation function for neural networks. As demands on neural network speed and accuracy increase, methods for effective digital approximation of the sigmoid function is increasingly important. It is with this in mind that an overview of previously employed design choices for sigmoid apprximation are presented, as well as a discussion of the mathematical properties of the sigmoid function and its derivative. In addition, a novel minimax approach to approximating the sigmoid and its derivative is presented. This approach was developed with reduction of maximum error and simplicity of approximation as improtant design criteria. Designs for the sigmoid and its derivative are shown for seven to eleven fractional bits of accuracy. The designs were implemented in VHDL and synthesized to an FPGA device and an ASIC library. For both technologies, the designs increase consistently in both area and delay with increased numbers of accurate bits.

Chapter 1

Introduction to Function Approximation

This chapter presents an introduction to some underlying concepts of approximating functions using digital hardware. It begins with a motivation for function approximation. This is followed by some basic design blocks used for implementations. Next, details of three classes of function approximation techniques are presented, with examples of each technique. The chapter concludes with an outline of the remaining chapters within this thesis.

1.1 Overview of Function Approximation

Function approximation using digital hardware involves the development of a digital system, which upon receiving an input value x outputs a value $y \approx f(x)$. This area of digital system research holds potential benefits based on the application domain. The distinction of potential benefit lies in the optimization of parameters crucial to the application. One application requiring function approximation is signal processing, including multimedia. In many cases, speed is often the optimization criterion. Scientific computing is another application domain, which employs approximations with more emphasis on the accuracy of the approximation result. Additionally, in applications involving mobile devices, area and power consumption are important factors to consider. With these optimization considerations in mind, the question then remains, how should this issue of approximating necessary function values be approached in digital hardware?

1.2 Design Foundations

An initial solution to the function approximation design problem is to treat the function approximation as a direct mapping. That is, have a specific output value directly associated with each input value. This approach is conceived in digital hardware with a table lookup. A memory device or a portion of memory, typically ROM or RAM, is devoted to containing output values relative to input addresses. In

cases involving a small number of possible inputs, this approach is the ideal one. An advantage being that assuming a direct mapping within the lookup table, function accuracy is optimal. Also, these tables are optimized for speed, as memory units tend to have low delay. Needs for larger input and output word sizes, however, are present both from a push towards single and double precision word sizes and the sensitivity of specific applications and their algorithms. For example, some computer graphics applications rely upon input word sizes of 24 and 32 bits and mandate corresponding accuracy. For each bit the input word length is increased, the size of the lookup table increases to twice its former size. As input lengths are increased past a size of 12 bits, table sizes become very large. As considerations of device size are increasingly essential in many applications, lookup tables alone are clearly insufficient in solving all approximation needs. Based on these reasons, another design component, in addition to lookup tables, is necessary.

Because of inadequacies inherent with lookup tables, arithmetic devices serve as the sometimes needed additional design component in digital function approximation. Examples of these devices include multipliers and adders. These devices often scale in a linear fashion, as an increase in input width mandates a directly proportional increase in device size. However, these devices suffer a drawback in speed [2]. Regardless of optimizations employed, these devices and the algorithms that use them tend to be slower than lookup tables. Additionally, there is potential for implicit error dependent upon the approximation method employed in designing

the arithmetic devices. By use of both lookup tables and arithmetic devices, a number of approaches exist for resolving the conflicting design constraints demanded by function approximation.

1.3 Iterative Approaches

One method of performing the approximation is by use of iterative algorithms. That is, methods which repeat themselves a specified number of cycles. One such algorithm, which is widely used, is the CORDIC algorithm. The CORDIC algorithm is a method for determining the phase and magnitude of a two-dimensional vector [3]. It is based on the following equations, which are solved iteratively to reach a desired level of precision:

$$X_{i+1} = X_i - ms_iY_i2^{-i} \tag{1.1}$$

$$Y_{i+1} = Y_i + s_iX_i2^{-i} \tag{1.2}$$

$$Z_{i+1} = Z_i - s_i\frac{1}{\sqrt{m}}\tan^{-1}(\sqrt{m}2^{-S(m,i)}) \tag{1.3}$$

where s_i is the direction of rotation, $S(m,i)$ is the shift sequence, and m is a coordinate system defining parameter such that

$$m = \begin{cases} 1 & \text{for circular coordinates} \\ -1 & \text{for hyperbolic coordinates} \\ 0 & \text{for linear coordinates} \end{cases}$$

The CORDIC algorithm has two operational modes, rotational and vectoring, which can be used to approximate elementary mathematical function values. In rotational mode, each successive Z_i is brought closer to 0, leaving the X_n and Y_n as the function values of interest, assuming n iterations. In vectoring mode, Y_i is decreased iteratively closer to 0, leaving the X_n and Z_n as the desired values. By appropriate arrangement of input values, this approach can be used to attain approximations of trigonometric, hyperbolic, and arithmetic functions. Consequently, it has seen wide use in approximation unit designs.

One advantage of CORDIC and other iterative methods is their relative simplicity, lending them well to a digital implementation. The CORDIC algorithm, for example, typically uses only an adder for calculation and memory for storing arc tangent values. Another advantage of this algorithm is that it allows for scalable output accuracy; for a desired level of accuracy, the method is applied iteratively until the level is achieved. This, however, leads to a major drawback of CORDIC and many other iterative approaches. Due to their iterative nature, for a given iteration i, the $(i-1)^{th}$ iteration result is needed. This dramatically reduces the overall speed if high accuracy is needed.

1.4 Piecewise Approaches

Another class of approximation techniques is known as piecewise approximations [4]. Within this class, for a given function $f(x)$, an interval $[a, b]$ is selected over which $f(x)$ is to be approximated. Within $[a, b]$, n subintervals $[a_i, b_i]$ are chosen along with an approximation function $f_i(x)$ for each subinterval. The $f_i(x)$'s are chosen such that they approximate the portion of $f(x)$ within $[a_i, b_i]$ more accurately than one single approximation function over the entire interval. The overall $f_i(x)$'s can be viewed as a group of functions over $[a, b]$ such that:

$$f(x) \approx \begin{cases} f_0(x): & a_0 < x < b_0 \\ \vdots & \vdots \\ f_n(x): & a_n < x < b_n \end{cases}$$

Piecewise approximations are further classified by the type of functions used in the approximation. For example, piecewise linear approximations use functions of the form $f(x) = c_0 + c_1 x$, piecewise quadratic approximations use functions of the form $f(x) = c_0 + c_1 x + c_2 x^2$, and so on. Naturally, choosing a higher degree of polynomial functions leads to a higher level of accuracy in the approximation. Typically, piecewise methods are implemented in digital hardware using multipliers, adders, and memory. For low precisions, these implementations are often either linear or quadratic approximations. On the other hand, piecewise polynomial approximations

with high degree polynomials are popular for high precision. Compared to iterative approaches, they tend to require fewer cycles to perform the approximations and can be more easily pipelined.

One widely used polynomial technique involves the use of Taylor series approximations [4]. The Taylor series approximates a function $f(x)$ in the following manner:

$$f(x) \approx \sum_{i=0}^{n} \frac{f^i(x_0)(x - x_0)}{i!} \tag{1.4}$$

where x_0 is a point at which the function's value is known. This approach is typically implemented using table lookups for the coefficients, $\frac{f^i(x_0)}{i!}$, a multiplier, and an adder. With this approach, a variable level of accuracy can be attained depending on the approximation intervals chosen and the number of terms in the approximation. One drawback to this technique is the lookup table size or number of terms in the approximation increases greatly depending on the level of accuracy chosen.

1.5 Organization of Thesis

This thesis examines techniques for approximating the sigmoid function and its derivative. This examination culminates in the exploration of a novel minimax technique. Chapter 2 presents sigmoid function theory, as well as some previous research related to sigmoid function approximation, using the techniques described within this first chapter. Advantages and disadvantages of these techniques relevant to the

sigmoid function are also presented. Chapter 3 focuses on the proposed minimax approximation method for the sigmoid and its derivative, describing minimax function approximation and its application to the sigmoid. Additionally, this chapter contains a presentation of the associated hardware used with this approach. Chapter 4 presents area and delay estimates for synthesized implementations of the approach described in Chapter 3. Finally, Chapter 5 concludes the thesis by discussing advantages and disadvantages of the proposed sigmoid approximation approach. The chapter also offers some future research directions based on the material presented in this thesis.

Chapter 2

The Sigmoid Function, its Derivative, and Previous Research

The sigmoid function and its derivative are used in a number of applications, including neural networks. Typically, within a neural network, the sigmoid is used as a learning function for training the neural network, while its derivative is used as a network activation function, specifying the point at which the network should switch to a true state. A number of the methods specified in the previous chapter have been applied to the approximation of the sigmoid function and its derivative with varying levels of success. In this chapter, the mathematical nature of the sigmoid function and its derivative is presented. This chapter also includes a discussion of those techniques for the sigmoid utilized in previous research with resulting

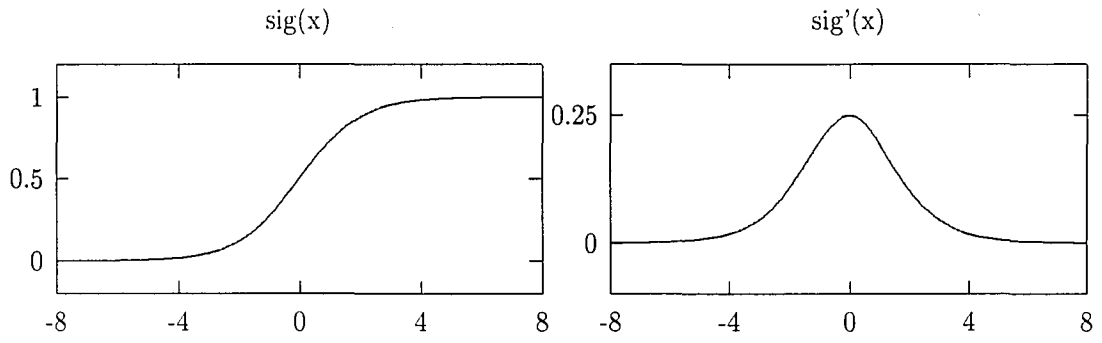

Figure 2.1: Plots of the Sigmoid Function and its Derivative

advantages and disadvantages found in their implementation. These techniques include the CORDIC algorithm, pseudo and standard piecewise approximations, and symmetric table methods.

2.1 The Sigmoid Function and its Derivative

The sigmoid function is defined as:

$$sig(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

with corresponding derivative:

$$sig'(x) = \frac{e^{-x}}{(1+e^{-x})^2} \tag{2.2}$$

As can be seen in Figure 2.1, these functions have the following limits:

$$\lim_{x \to \infty} sig(x) = 1 \tag{2.3}$$

$$\lim_{x \to -\infty} sig(x) = 0 \tag{2.4}$$

and

$$\lim_{x \to \pm\infty} sig'(x) = 0 \tag{2.5}$$

Additionally, the functions have the following symmetry properties:

$$sig(-x) = 1 - sig(x) \tag{2.6}$$

and

$$sig'(-x) = -sig'(x) \tag{2.7}$$

These properties can be used in optimizing the approximation techniques. Based on these defined limits and given a desired level of accuracy, it is feasible to place a bound on the inputs. That is, a threshold can be defined such that all inputs past the threshold cause the limit value to be output. Typically, this is used to define an interval of interest, (a, b), dependent upon the desired accuracy. This interval is applicable, since the functions are within a relatively small distance from the limit values past this interval. Additionally, the symmetric properties of the

sigmoid and its derivative allow for potential benefits and optimizations. In the case of both the sigmoid and its derivative, the inherent symmetric properties make it possible to only require computation of output values for non-negative inputs. For either function, a small amount of combinational logic for detecting a negative input and converting the output appropriately is required. With these points in mind, a number of the previously used approaches for approximating the sigmoid and its derivative are presented.

2.2 CORDIC Approximations

The CORDIC algorithm, as described in Section 1.3, has been used in practical applications of function approximation, including trigonometric, hyperbolic, reciprocal, and square root approximations. This algorithm, however, suffers from the speed drawbacks mentioned in Section 1.3, as well as practical issues that are particular to the sigmoid function. Due to the nature of the CORDIC algorithm, at least two distinct CORDIC operations are required to approximate the sigmoid. Initially, the e^{-x} term is calculated, followed by the reciprocal of $1 + e^{-x}$. The calculation of the sigmoid's derivative adds further complexity, since the reciprocal of $(1 + e^{-x})^2$ is required. Because of this, conventional CORDIC algorithms tend to not be as efficient as other successful approaches to sigmoid approximation.

One potential improvement to the latency inherent with a CORDIC approach is

to use two CORDIC functional units; one for computing e^{-x} and a second for computing the reciprocal of $1+e^{-x}$. Pipelining potentially allows for effective throughput increases. The amount of hardware needed for this sort of implementation, however, is quite complex, considering the amount of steering logic and staging necessary. For these reasons, although the CORDIC algorithm shows a high degree of success for other functions such as sines and cosines, the CORDIC algorithm is not well suited for approximation of the sigmoid function or its derivative.

As an example of previous research in this area, a pipelined CORDIC implementation of a neural network cell, or neuron, is presented in [5]. The technique proposed within that work represents the performance inefficiencies expected when using the CORDIC algorithm for sigmoid approximations. Two CORDIC units, one for exponential calculation and one for division, are pipelined while a ROM-based controller synchronizes the CORDIC units. This approach is somewhat complex with respect to other methods, utilizing a large amount of area while yielding only eight bits of accuracy for the presented CORDIC implementation.

In [6], a method is presented for sigmoid approximations using a combination of standard CORDIC and a modified CORDIC, known as the flat CORDIC algorithm. With this method, two units, one for exponential and one for division, are again employed for the function calculation. The overall performance gain is due to the use of the flat CORDIC algorithm, yielding a smaller device area and lower latency. In this case, larger levels of accuracy are achieved, however, a dramatic increase in

device area occurs with levels of accuracy greater than twelve bits.

2.3 Piecewise Approximations

Piecewise approximation methods, discussed in Section 1.4, have been applied to the sigmoid function and its derivative with varying degrees of success. This success can be attributed to the sigmoid's applicability to linear approximation methods. As discussed in Section 2.2, it is difficult to compute the sigmoid function directly. Since the sigmoid and its derivative are fairly consistent, particularly as input values extend away from the origin, piecewise approximations over wide intervals can be utilized, which reduces device latency and area. One disadvantage to these approaches involves the nature of the hardware typically used in these implementations. A piecewise approximation typically requires selection logic to determine the interval in which the input value lies, using either ROM lookup tables for function values or specialized hardware to calculate the approximation coefficients. Most techniques also use multipliers and adders for calculating the final approximation value. An additional difficulty with these methods can occur when using ROM tables for coefficient values. These tables can grow to unruly sizes, depending on the desired accuracy. Taking these considerations into account, however, useful results can be attained using piecewise methods for sigmoid approximation.

An early use of a piecewise approximation to the sigmoid is presented in [7].

In this paper, the input interval is split into thirteen equally-sized intervals with an approximation function for each interval. These functions then represent the sigmoid function and are realizable with conventional digital hardware. In this paper, simplification of the approximation functions allows for a reduction in overall latency and area. Rather than calculating the functions with multipliers and adders or storing their values in a lookup table, the input values are manipulated via inverters and NAND gates to approximate the sigmoid. Although this negates any dependence upon arithmetic devices, it leads to less accurate approximations. In some cases, this approach yields less than four bits of accuracy. Additionally, the authors offer no provision for an implementation of the sigmoid's derivative.

A more thoroughly presented approach based on piecewise approximations is found in [1]. The authors propose to split an entire domain of inputs within $(-8, 8)$ into segments with corresponding outputs. Both seven and fifteen-segment piecewise approximations are proposed, with the fifteen segments offering greater accuracy at the expense of area and potentially delay. The outputs are specified using shifted versions of the inputs, with variations of the bits shifted into the result, depending upon the desired function.

The method is presented in Table 2.1, for the proposed seven-segment piecewise approximations to the sigmoid. As an example, consider an input of 1.25_{10}, or 0001.010000000000_2. Using this approach, a sigmoid approximation value of 0.110010000000000_2 is output.

Input Range	Binary Input	Binary Output	Output Range
$[a, b)$	xxxx.xxxxxxxxxx	x.xxxxxxxxxxxxxx	$[a, b)$
$[0, 1)$	0000.lkjihgfedcba	0.10lkjihgfedcba0	$[0.5, 0.75)$
$[1, 2)$	0001.lkjihgfedcba	0.110lkjihgfedcba	$[0.75, 0.875)$
$[2, 4)$	001m.lkjihgfedcba	0.1110mlkjihgfedc	$[0.875, 0.9375)$
$[4, 8)$	01nm.lkjihgfedcba	0.1111nmlkjihgfed	$[0.9375, 1)$

Table 2.1: Piecewise Function Approximation Using the Seven-Segment Approach From [1]

The authors report results which are favorable in terms of area and delay, however, little consideration is given to accuracy. The approach outlined in the paper yields an accuracy of seven bits in the fifteen-segment case, and five bits in the seven-segment case. Removing the lookup tables typically used in piecewise approximations significantly reduces accuracy. Additionally, no mention is made of necessary selection logic. Since no implicit selection is available through table addressing, additional multiplexors and combinational logic are required to select the appropriate outputs. Depending upon the desired accuracy, the necessary selection logic could significantly increase the area.

A purer form of piecewise linear approximations for the sigmoid is discussed in [8] and [9]. A generalization of this approach to function approximation is presented in [10]. In this collection of work, three distinct design alternatives using first-order piecewise approximations are described by the authors with respect to relative advantages regarding accuracy, delay, and device area. These alternatives are all based upon a separation of possible inputs into segments. A function, $H(x)$ is defined for each of the segments. $H(x)$ is then further defined by the following

equation:

$$H(x) = Ax + B \tag{2.8}$$

where A and B are linear coefficients determined prior to device implementation. Three first-order methods are considered by the authors for relative advantages regarding accuracy, delay, and device area.

The most direct of the three first-order approximations uses a multiplier, an adder, and a lookup table. The segment in which a given input lies is determined in order to acquire the appropriate multiplicative value, A, by which the input is to be multiplied. This product is then added to the B value associated with the input's segment. One potential disadvantage to this approach is that the computational complexity of its implementation could potentially lead to large area and delay.

The second of these approaches reduces the delay of the first approach. It replaces the multiplication portion of the approximation with an integer power of two multiplication. The Segment's A coefficient is replaced with a value A', where $A' = 2^{-n}$ and n is an integer. The value of A' is chosen such that it is the nearest power of two to the original A value. In addition, B is calculated based on A' rather than A, making it distinct from the previous technique. The following equation indicates the changes made to Equation 2.8 using this technique:

$$H(x) = 2^{-n}x + B \tag{2.9}$$

The powers of two multiplication is implemented by shifting the input value an

appropriate number of places. This reduces the overall computational complexity of the device, reducing device latency and area. This technique has the potential disadvantage of a reduction in accuracy, due to A being approximated by the nearest integer power of two. Also, although the multiplication unit is effectively eliminated, a right-shifter is now required as well as a table lookup.

The third first-order approach takes the most aggressive stance on delay reduction. This approach eliminates the adder used with the first two approaches. This is accomplished by determining the linear coefficient A' in the same manner as the second technique. The B coefficient is chosen with respect to possible u values within a given segment, such that no changing bits of u overlap with nonzero bits of B. The changing bits of u are those bits within a segment which distinguish inputs from one another. For example, using the author's internal number system, the segment $(-2, -1]$ has values of the following form: $1001.xxxxxxxxxx$ where x represents a changing bit of the input value. While it would appear that a large number of changing bits need be considered in this example, the function's A' value will effectively shift a number of the changing bits out of the product considered when determining the B value. In this example, the A' value used is 2^{-3}, eliminating three of the changing bits. B is then chosen such that its seven least significant bits are zero. This method requires a sophisticated system of logic for coefficient determination, as well as shifting units and lookup tables. While it effectively reduces computational complexity, it relies heavily upon lookup tables, which as discussed

in Chapter 1, can lead to large device area.

These designs yield estimated area and delay which scale appropriately with the expectations discussed in this section. Latency was effectively reduced with each successive first-order design approach. Regarding accuracy, the authors report this in terms of maximum error of their sigmoid generator devices. For all of the first-order approximation alternatives described, the maximum error reported implies a guaranteed accuracy of at most six bits. Additionally, a second-order approximation technique is described which also guarantees at most six bits. Again, as with other work discussed in this chapter, the issues of area and latency are well addressed, however, accuracy is still limited.

2.4 Symmetric Table Approximations

Symmetric table methods employ a combination of arithmetic devices and lookup tables, similar to the previously discussed piecewise approximations. These methods take advantage of mathematical symmetry properties of desired functions in order to reduce the overall area and latency. Symmetric table methods have seen high levels of success in reducing these factors for a number of functions, including the sigmoid function and its derivative. Since these methods typically base themselves upon an effective method for function approximation, such as Taylor polynomials, they are applicable to sigmoid approximations. Additionally, since the sigmoid function has

inverse symmetry about the y-axis, and its derivative has symmetry about the y-axis, both functions hold potential for reaping the benefits available with symmetric table methods.

One of these approaches, which offers flexibility in design considerations, is the symmetric table addition method (STAM) for function approximation, presented in [11]. This method provides a compromise between the issues of area and delay versus accuracy. A Taylor approximation, as discussed in Section 1.4, is used as a starting point for the function's approximation. The input x is divided into $m+1$ partitions, which are then utilized as inputs to m lookup tables containing coefficient values. The partitions are addressed within the lookup table such that each lookup table provides a function value $a_i(x_0, x_{i+1})$. Based on this addressing, each table receives the high-order partition, x_0 along with a lower-order partition x_{i+1}. All partitions x_2 to $xm-1$ are conditionally inverted based on their most significant bit before being passed to the lookup table. The outputs of the lookup tables are then conditionally inverted and the result is passed to a multi-operand adder.

By utilizing this method, there is no use of multipliers, thereby lowering the overall complexity of the design. The lookup tables are minimized by taking advantage of inherent binary symmetry found within differentiable functions, as well as examining leading and trailing bit similarities. By effectively providing functional symmetry, the inverters allow for the lookup table sizes to be reduced to half their

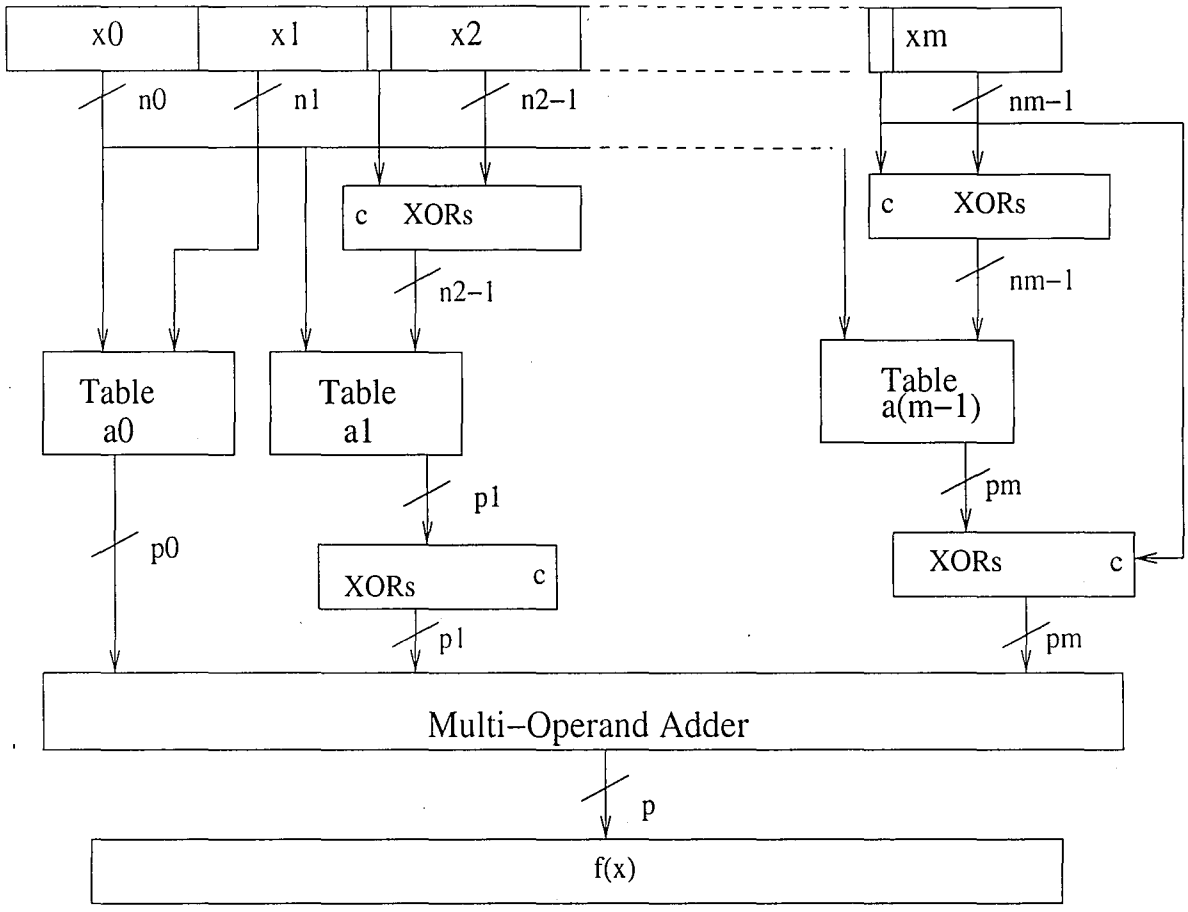

Figure 2.2: STAM Block Diagram

original size. This is further enhanced in situations where all entries have identical trailing or leading bits, as these bits need not be stored in the lookup tables. The STAM method yields decreased table size with minimal increase in delay with subsequent addition of lookup tables. Once past five tables, the reduction in individual table sizes tends to be outweighed by the area consumption of multiple tables. At this point, the hardware complexity tends to outweigh the table size reduction benefits.

In Figure 2.2, an example of a sigmoid approximation unit utilizing the STAM

method is shown. As can be seen in this figure, the input argument x is split into $m + 1$ partitions with corresponding lengths n_i. All partitions beyond the second partition are inverted based on their most significant bit. The partition values are fed to a multi-operand adder to attain the approximation to $f(x)$.

The symmetric table addition method was used in the approximation of the sigmoid and its derivative, as described in [12]. That paper explores five configurations for sigmoid approximation using the STAM method, with two to six lookup tables. For 13, 16, and 24-bit operand devices, the authors give results in terms of memory usage and reduction possible through an increased number of tables. In each of these cases, a decrease in memory occurs for each successive table addition up to five tables. Error values are also given for 16-bit operands. The configurations yield a full 16 bits of accuracy. These results are clearly more favorable than others listed in terms of potential accuracy, though they have large memory requirements.

Chapter 3

Proposed Technique

Given the research previously conducted for approximating the sigmoid and its derivative in digital hardware, the question remains: What is an area and delay efficient implementation, potentially not dependent upon lookup tables, which also allows for accurate results? The technique proposed in this chapter, that of a partitioned linear minimax approximation, attempts to answer this question. In this chapter, a mathematical basis for the minimax approximation technique is presented, followed by a proposed approach for its use with the sigmoid and its derivative. The chapter concludes with a hardware design and a discussion of an implementation for approximating the sigmoid and its derivative using the proposed approach.

3.1 Minimax Function Approximation

While approximation techniques such as Taylor polynomials can be useful for a number of applications, methods exist that can improve accuracy, while lowering overall computational complexity. One such method is the minimax polynomial approach to function approximation. The minimax approach focuses on minimizing the maximum error across a given input segment, as opposed to a Taylor polynomial approximation, which minimizes the error at the point of expansion. A first-order minimax polynomial is defined as the following:

$$m(x) = c_0 + c_1 x \approx f(x) \tag{3.1}$$

For the purposes of reducing error, the approximation error is considered as a function $E(x)$ such that:

$$E(x) = f(x) - (c_0 + c_1 x) \tag{3.2}$$

To reduce the average error over the interval $[a, b]$, the errors at the endpoints are considered, and the desired approximation function is chosen such that its endpoint errors are equal. Thus:

$$f(a) - (c_0 + c_1 a) = f(b) - (c_0 + c_1 b) \tag{3.3}$$

This equation is then solved for c_1 in terms of a and b. Next, the point at which the error is maximum, x_{max}, is determined by setting the derivative of $E(x)$ to zero.

Thus:

$$E'(x_{max}) = f'(x_{max}) - c_1 = 0 \tag{3.4}$$

This equation is then solved for x_{max} in terms of c_1, which can be further expressed in terms of a and b. Given c_1 and x_{max}, c_0 can now be determined based on the linear minimax polynomial requirement that the error at the endpoints be equal to the negation of the error at x_{max}. Thus, the following equation is solved for c_0 in terms of a and b:

$$E(a) = -E(x_{max}) \tag{3.5}$$

As an example of an improvement gained by using minimax polynomials, consider the approximation of $f(x) = e^x$ over the interval $[-1, 1]$ using first-order minimax and Taylor polynomials. The Taylor polynomial for this approximation is $1+x$, while the minimax equivalent is $1.2643 + 1.1752x$. Plots of the original function, along with the Taylor and minimax polynomials, are shown in Figure 3.1. As noted previously, the approximation error is minimal for the Taylor polynomial around the point of expansion, while the minimax polynomial maintains a more consistent error across the interval. For this example, the respective maximum errors for the Taylor and minimax polynomials are 0.718 and 0.279, giving an improvement of over 60%.

With this motivation, an example of the solution of a minimax polynomial is presented. Consider the function $f(x) = 1/x$ over the interval $[a, b]$. The following

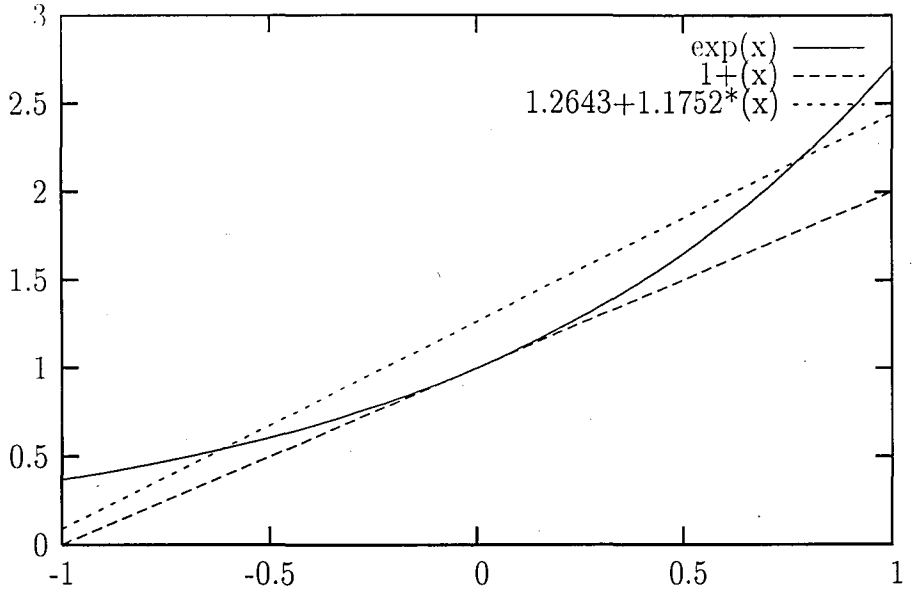

Figure 3.1: Comparison of First-order Taylor and Minimax Polynomials

is true with respect to $E(x)$:

$$E(a) = \frac{1}{a} - (c_0 + c_1 a) \tag{3.6}$$

$$E(b) = \frac{1}{b} - (c_0 + c_1 b) \tag{3.7}$$

setting $E(a)$ equal to $E(b)$ gives:

$$\frac{1}{a} - (c_0 + c_1 a) = \frac{1}{b} - (c_0 + c_1 b) \tag{3.8}$$

solving for c_1 gives:

$$c_1 = -\frac{1}{ab} \tag{3.9}$$

Now, x_{max} is determined with respect to a and b by setting the derivative of the error at x_{max} to zero:

$$E'(x_{max}) = -\frac{1}{x_{max}^2} - c_1 = 0 \tag{3.10}$$

and solving for x_{max}, which gives:

$$x_{max} = \sqrt{ab} \tag{3.11}$$

Next, c_0 is determined by setting $E(a)$ equal to $-E(x_{max})$ and solving for c_0. Thus:

$$\frac{1}{a} - (c_0 - \frac{1}{ab}a) = -(\frac{1}{\sqrt{ab}} - (c_0 - \frac{1}{ab}\sqrt{ab})) \tag{3.12}$$

$$\frac{1}{a} - c_0 + \frac{1}{b} = -\frac{1}{\sqrt{ab}} + c_0 - \frac{1}{\sqrt{ab}} \tag{3.13}$$

$$2c_0 = \frac{1}{a} + \frac{1}{b} + \frac{2}{\sqrt{ab}} \tag{3.14}$$

$$c_0 = \frac{b + a + 2\sqrt{ab}}{2ab} \tag{3.15}$$

giving a closed form minimax approximation of $f(x) = 1/x$ over the interval $[a, b]$.

3.2 Minimax Approach to the Sigmoid

The first step in the desired approach to performing a minimax approximation of the sigmoid and its derivative involves generation of coefficients c_0 and c_1. As shown in the previous section, a closed form solution to the minimax approximation of a function is particularly desirable since it facilitates coefficient calculation. Determining a closed form solution for the sigmoid minimax approximation begins with

the following derivation for c_1. Setting the errors at the interval endpoints equal to one another:

$$E(a) = \frac{1}{1+e^{-a}} - (c_0 + c_1 a) \tag{3.16}$$

$$E(b) = \frac{1}{1+e^{-b}} - (c_0 + c_1 b) \tag{3.17}$$

$$\frac{1}{1+e^{-a}} - (c_0 + c_1 a) = \frac{1}{1+e^{-b}} - (c_0 + c_1 b) \tag{3.18}$$

Solving for $c1$ gives:

$$c_1(b-a) = \frac{1}{1+e^{-b}} - \frac{1}{1+e^{-a}} \tag{3.19}$$

$$c_1 = \frac{e^{-a} - e^{-b}}{(1+e^{-b})(1+e^{-a})(b-a)} \tag{3.20}$$

Thus, a closed form solution for c_1 is attained. A closed form solution for c_0 of the minimax approximation of the sigmoid derivative function can similarly be determined. The complexity of this solution is such that its closed form is not presented in this thesis. Since the purpose of this thesis centers around hardware implementations of the sigmoid and its derivative, the overall coefficient calculation was performed by the Maple software package. This package provides minimax coefficient solutions with its numapprox() library. It should be noted that Maple makes use of the Remez algorithm for its determining of minimax coefficients, which differs

from the approach presented in Section 3.1. This algorithm is employed by Maple, as it is more applicable to minimax polynomials of degrees greater than one.

Regarding the error of these approximations, since a solution for $E(x_{max})$ is not provided, an upper bound for minimax approximation error is considered based on maximum error values for a slightly less accurate but similar approximation class, Chebyshev series approximations. The maximum error of a Chebyshev series first-order approximation on an interval $[a, b)$ is defined in [13] as:

$$E_n(x) = \left(\frac{b-a}{4}\right)^2 \cdot |f''(\xi)| \tag{3.21}$$

where ξ is the point on $[a, b)$ where f'' has its maximum value. In Figure 3.2, a plot of the sigmoid's second derivative is shown. Based on Equation 3.21, the error is dependent upon the size of a given interval as well as the maximum value of f'' within that interval. Due to this dependence, smaller intervals are chosen to reduce error, and special attention is given to points at which the function's second derivative takes relatively large magnitudes. A list of maximum error values calculated by Maple for first-order minimax approximations over a number of intervals are given in Table 3.1 for the sigmoid and Table 3.2 for its derivative. An explanation of the method of selecting the intervals in these tables follows.

Once the coefficients are determined, the next step in the design approach is interval selection. In selecting the intervals to be used with the sigmoid and its derivative using a minimax approximation, considerations are given to maintaining the lower computational complexity of a first-order polynomial approximation, while

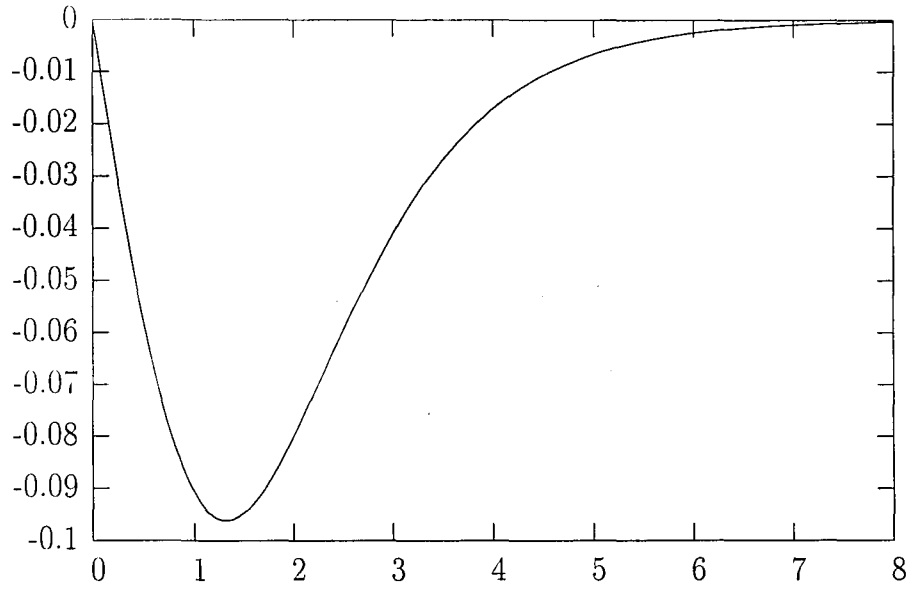

Figure 3.2: Plot of the Second Derivative of the Sigmoid

Interval	Maximum Error
$[0, 8)$	.134251
$[0, 4)$	.070646
$[0, 2)$	.020442
$[2, 4)$	.010768
$[1, 2)$	.005824
$[2, 3)$	.003740
$[4, 8)$	.003656
$[0, 1)$	.003534
$[3. 4)$	.001706

Table 3.1: Sample Maximum Minimax Error Values for the Sigmoid

Interval	Maximum Error
$[0, 8)$	.057549
$[0, 4)$	.032273
$[0, 2)$	.011247
$[2, 4)$	.008136
$[0, 1)$	.005895
$[4, 8)$	.003561
$[2, 3)$	.002491
$[3, 4)$	.001489
$[1, 2)$	.000977

Table 3.2: Sample Maximum Minimax Error Values for the Sigmoid's Derivative

attempting to lower the error across the entire input interval. For the purposes of this thesis, the input interval $(-8, 8)$ was selected due to its applicability to a number of neural network applications. Based on the symmetry properties of the sigmoid and its derivative, the approximation is made on $[0, 8)$ and symmetry properties are used to obtain approximations on $(-8, 0)$.

To accomplish the desired error reduction, a less conventional approach is taken to interval selection. With some approximation methods, coefficients are chosen to be powers of two to use shifting in the implementation rather than multiplication. Additionally, many polynomial approximation approaches utilize equally-sized intervals. This is to provide consistency and potentially permit hardware reduction through inherent properties such as extensive leading or trailing bits. Also, using equally-sized intervals allows for table lookups for coefficient selection. Using the table lookups, an implicit addressing is gained since each successive address represents an interval.

For the purposes of this thesis, intervals are selected based on their error-reducing ability rather than their consistency. Starting with the entire interval over which the function is to be approximated, the interval is split in half to two subintervals. Next, the accuracy of the subintervals is determined. These values are compared to 2^{-k} where k is the desired number of bits of accuracy. Those intervals limiting the overall accuracy of the design are split in half until the desired level of accuracy is attained.These smaller intervals are chosen such that they represents equal halves of the original interval. Once all intervals satisfy the accuracy requirements, the interval set is complete. This method is illustrated in Figure 3.3. While the method potentially eliminates the use of equal-sized intervals, it allows for an increase in overall accuracy for a given number of intervals.

As an example of selecting the necessary intervals for a given accuracy, consider the case of seven bits of accuracy for the sigmoid function. Using the interval $[0, 8)$ as a starting point, the first step is to determine the accuracy of this interval. A minimax approximation of the function over this interval provides much less than seven bits of accuracy, so the interval is split into $[0, 4)$ and $[4, 8)$. Referring to Table 3.1, $[0, 4)$ provides less than seven bits of accuracy, and it is divided into $[0, 2)$ and $[2, 4)$. The accuracy of $[0, 2)$ is checked, again providing less than the desired accuracy, so it is split into $[0, 1)$ and $[1, 2)$. $[0, 1)$ provides the desired accuracy, so it is not split further. Similarly, $[1, 2)$, $[2, 4)$ and $[4, 8)$ all provide sufficient accuracy for this function and are also not divided further. At this point, the interval set

Accurate Bits	#Intervals	Intervals
7	5	$[0,1)$ $[1,2)$ $[2,3)$ $[3,4)$ $[4,8)$
8	6	$[0,1)$ $[1,\frac{3}{2})$ $[\frac{3}{2},2)$ $[2,3)$ $[3,4)$ $[4,8)$
9	10	$[0,\frac{1}{2})$ $[\frac{1}{2},1)$ $[1,\frac{5}{4})$ $[\frac{5}{4},\frac{3}{2})$ $[\frac{3}{2},2)$ $[2,\frac{5}{2})$ $[\frac{5}{2},3)$ $[3,4)$ $[4,6)$ $[6,8)$
10	15	$[0,\frac{1}{2})$ $[\frac{1}{2},\frac{3}{4})$ $[\frac{3}{4},1)$ $[1,\frac{5}{4})$ $[\frac{5}{4},\frac{3}{2})$ $[\frac{3}{2},\frac{7}{4})$ $[\frac{7}{4},2)$ $[2,\frac{9}{4})$ $[\frac{9}{4},\frac{5}{2})$ $[\frac{5}{2},3)$ $[3,\frac{7}{2})$ $[\frac{7}{2},4)$ $[4,5)$ $[5,6)$ $[6,8)$
11	19	$[0,\frac{1}{4})$ $[\frac{1}{4},\frac{1}{2})$ $[\frac{1}{2},\frac{3}{4})$ $[\frac{3}{4},1)$ $[1,\frac{5}{4})$ $[\frac{5}{4},\frac{3}{2})$ $[\frac{3}{2},\frac{7}{4})$ $[\frac{7}{4},2)$ $[2,\frac{9}{4})$ $[\frac{9}{4},\frac{5}{2})$ $[\frac{5}{2},\frac{11}{4})$ $[\frac{11}{4},3)$ $[3,\frac{13}{4})$ $[\frac{13}{4},\frac{7}{2})$ $[\frac{7}{2},4)$ $[4,\frac{9}{2})$ $[\frac{9}{2},5)$ $[5,6)$ $[6,8)$

Table 3.3: Sigmoid Interval Configurations

for the desired accuracy has been attained. The results of this example can be seen in Table 3.3, along with the cases of eight, nine, ten, and eleven bits. Intervals for the derivative of the sigmoid are shown in Table 3.4. It should be noted that the derivative of the sigmoid requires fewer intervals overall. This is due to the less drastic changes in slope throughout the function. Error values, as calculated by Maple, for these intervals are shown in Tables 3.5 and 3.6 for the sigmoid, and Tables 3.7 and 3.8 for the sigmoid's derivative.

With the intervals in place, the appropriate c_1 for each interval can be determined using the closed-form shown in Equation 3.20. As stated previously, however, Maple was used to calculate both c_0 and c_1 for each interval. The decimal values for these coefficients are shown in Tables 3.9 and 3.10 for the sigmoid and Tables 3.11 and 3.12 for the derivative of the sigmoid.

Finally, considerations are made regarding necessary data widths for the devices.

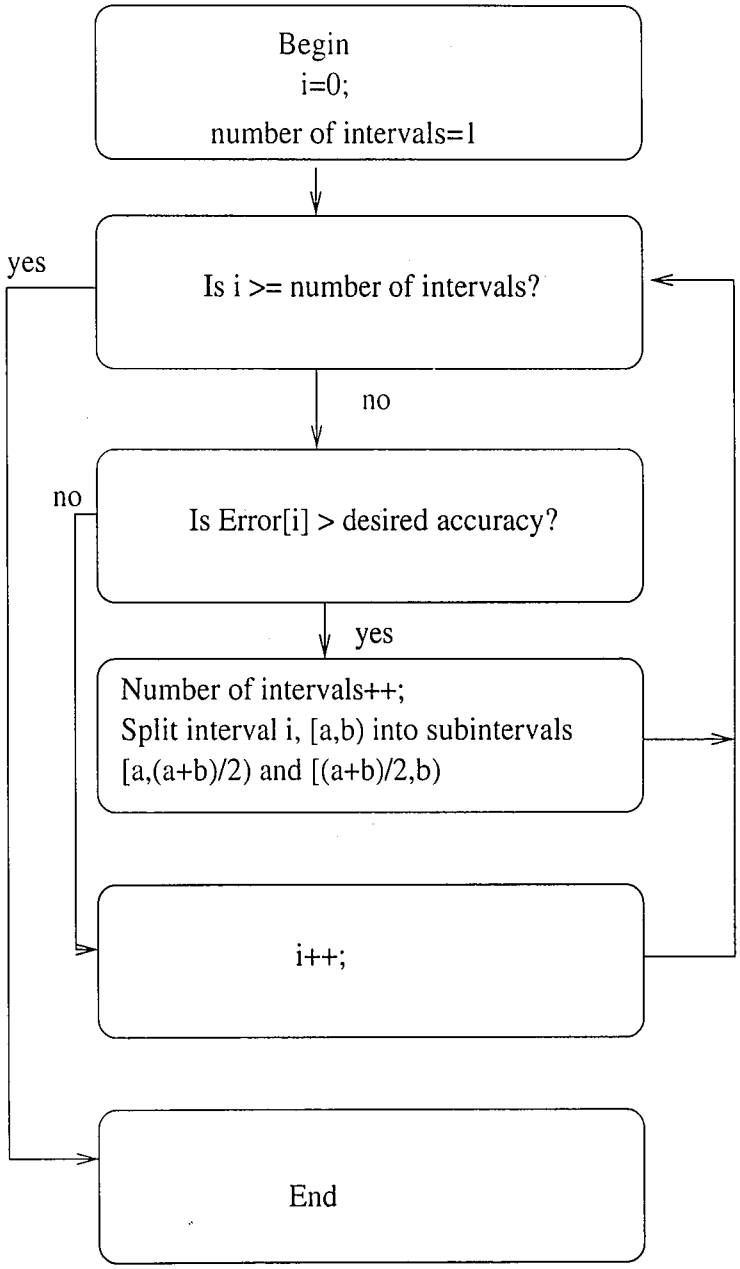

Figure 3.3: Interval Determination Method

Accurate Bits	#Intervals	Intervals
7	5	$[0,1)$ $[1,2)$ $[2,3)$ $[3,4)$ $[4,8)$
8	6	$[0,\frac{1}{2})$ $[\frac{1}{2},1)$ $[1,2)$ $[2,3)$ $[3,4)$ $[4,8)$
9	8	$[0,\frac{1}{2})$ $[\frac{1}{2},1)$ $[1,2)$ $[2,\frac{5}{2})$ $[\frac{5}{2},3)$ $[3,4)$ $[4,6)$ $[6,8)$
10	13	$[0,\frac{1}{4})$ $[\frac{1}{4},\frac{1}{2})$ $[\frac{1}{2},\frac{3}{4})$ $[\frac{3}{4},1)$ $[1,\frac{3}{2})$ $[\frac{3}{2},2)$ $[2,\frac{5}{2})$ $[\frac{5}{2},3)$ $[3,\frac{7}{2})$ $[\frac{7}{2},4)$ $[4,5)$ $[5,6)$ $[6,8)$
11	16	$[0,\frac{1}{4})$ $[\frac{1}{4},\frac{1}{2})$ $[\frac{1}{2},\frac{3}{4})$ $[\frac{3}{4},1)$ $[1,\frac{3}{2})$ $[\frac{3}{2},2)$ $[2,\frac{9}{4})$ $[\frac{9}{4},\frac{5}{2})$ $[\frac{5}{2},\frac{11}{4})$ $[\frac{11}{4},3)$ $[3,\frac{7}{2})$ $[\frac{7}{2},4)$ $[4,\frac{9}{2})$ $[\frac{9}{2},5)$ $[5,6)$ $[6,8)$

Table 3.4: Sigmoid Derivative Interval Configurations

Accurate Bits	Interval	Minimax Error
7	$[0,1)$	.003534
	$[1,2)$	.005824
	$[2,3)$	.003740
	$[3,4)$	.001706
	$[4,8)$	.003656
8	$[0,1)$	.003534
	$[1,\frac{3}{2})$	.001491
	$[\frac{3}{2},2)$	.001383
	$[2,3)$	.003740
	$[3,4)$	.001706
	$[4,8)$	.003656
9	$[0,\frac{1}{2})$	.000489
	$[\frac{1}{2},1)$	.001211
	$[1,\frac{5}{4})$	.000367
	$[\frac{5}{4},\frac{3}{2})$	.000374
	$[\frac{3}{2},2)$	.001383
	$[2,\frac{5}{2})$	.001091
	$[\frac{5}{2},3)$	.000778
	$[3,4)$	.001706
	$[4,6)$	.001815
	$[6.8)$	.000253

Table 3.5: Sigmoid Maximum Errors for 7-9 Accurate Bits

Accurate Bits	Interval	Minimax Error
10	$[0, \frac{1}{2})$	.000489
	$[\frac{1}{2}, \frac{3}{4})$	.000268
	$[\frac{3}{4}, 1)$	.000333
	$[1, \frac{5}{4})$	.000367
	$[\frac{5}{4}, \frac{3}{2})$	.000374
	$[\frac{3}{2}, \frac{7}{4})$	.000359
	$[\frac{7}{4}, 2)$	.000330
	$[2, \frac{9}{4})$	.000292
	$[\frac{9}{4}, \frac{5}{2})$	.000252
	$[\frac{5}{2}, 3)$	.000778
	$[3, \frac{7}{2})$	.000521
	$[\frac{7}{2}, 4)$	.000336
	$[4, 5)$	.000680
	$[5, 6)$	.000258
	$[6, 8)$	.000253
11	$[0, \frac{1}{4})$	.000062
	$[\frac{1}{4}, \frac{1}{2})$	.000174
	$[\frac{1}{2}, \frac{3}{4})$	.000268
	$[\frac{3}{4}, 1)$	.000333
	$[1, \frac{5}{4})$	.000367
	$[\frac{5}{4}, \frac{3}{2})$	.000374
	$[\frac{3}{2}, \frac{7}{4})$	.000359
	$[\frac{7}{4}, 2)$	.000330
	$[2, \frac{9}{4})$	.000292
	$[\frac{9}{4}, \frac{5}{2})$	.000252
	$[\frac{5}{2}, \frac{11}{4})$	.000212
	$[\frac{11}{4}, 3)$	.000176
	$[3, \frac{13}{4})$	.000144
	$[\frac{13}{4}, \frac{7}{2})$	.000116
	$[\frac{7}{2}, 4)$	.000336
	$[4, \frac{9}{2})$	.000211
	$[\frac{9}{2}, 5)$	.000131
	$[5, 6)$	.000258
	$[6, 8)$	.000253

Table 3.6: Sigmoid Maximum Errors for 10-11 Accurate Bits

Accurate Bits	Interval	Minimax Error
7	$[0,1)$	.005895
	$[1,2)$	.000977
	$[2,3)$	.002491
	$[3,4)$	.001489
	$[4,8)$	.003561
8	$[0,\frac{1}{2})$	.001816
	$[\frac{1}{2},1)$	.001050
	$[1,2)$	.000977
	$[2,3)$	.002491
	$[3,4)$	.001489
	$[4,8)$	.003561
9	$[0,\frac{1}{2})$	.001816
	$[\frac{1}{2},1)$	.001050
	$[1,2)$	.000977
	$[2,\frac{5}{2})$	.000645
	$[\frac{5}{2},3)$	.000582
	$[3,4)$	.001489
	$[4,6)$	.001750
	$[6,8)$	.000252

Table 3.7: Sigmoid Derivative Maximum Errors for 7-9 Accurate Bits

Accurate Bits	Interval	Minimax Error
10	$[0, \frac{1}{4})$	.000479
	$[\frac{1}{4}, \frac{1}{2})$	.000422
	$[\frac{1}{2}, \frac{3}{4})$	.000321
	$[\frac{3}{4}, 1)$	.000200
	$[1, \frac{3}{2})$	.000141
	$[\frac{3}{2}, 2)$	.000472
	$[2, \frac{5}{2})$	.000645
	$[\frac{5}{2}, 3)$	.000582
	$[3, \frac{7}{2})$	.000440
	$[\frac{7}{2}, 4)$	.000304
	$[4, 5)$	.000648
	$[5, 6)$	.000253
	$[6, 8)$	.000252
11	$[0, \frac{1}{4})$	.000479
	$[\frac{1}{2}, \frac{3}{4})$	.000321
	$[\frac{3}{4}, 1)$	.000200
	$[1, \frac{3}{2})$	.000141
	$[\frac{3}{2}, 2)$	.000472
	$[2, \frac{9}{4})$	.000159
	$[\frac{9}{4}, \frac{5}{2})$	.000161
	$[\frac{5}{2}, \frac{11}{4})$	.000152
	$[\frac{11}{4}, 3)$	.000137
	$[3, \frac{7}{2})$	.000440
	$[\frac{7}{2}, 4)$	.000304
	$[4, \frac{9}{2})$	.000199
	$[\frac{9}{2}, 5)$	.000126
	$[5, 6)$	.000253
	$[6, 8)$	.000252

Table 3.8: Sigmoid Derivative Maximum Errors for 10-11 Accurate Bits

Accurate Bits	Interval	c_0	c_1
7	$[0,1)$	0.503526	0.231058
	$[1,2)$	0.587144	0.149738
	$[2,3)$	0.740982	0.071777
	$[3,4)$	0.865960	0.029439
	$[4,8)$	0.968019	0.004412
8	$[0,1)$	0.503526	0.231058
	$[1,\frac{3}{2})$	0.559518	0.173031
	$[\frac{3}{2},2)$	0.629290	0.126445
	$[2,3)$	0.740982	0.071777
	$[3,4)$	0.865960	0.029439
	$[4,8)$	0.968019	0.004412
9	$[0,\frac{1}{2})$	0.500484	0.244918
	$[\frac{1}{2},1)$	0.515071	0.217198
	$[1,\frac{5}{4})$	0.546461	0.184965
	$[\frac{5}{4},\frac{3}{2})$	0.576301	0.161098
	$[\frac{3}{2},2)$	0.629290	0.126445
	$[2,\frac{5}{2})$	0.708509	0.086689
	$[\frac{5}{2},3)$	0.782758	0.056864
	$[3,4)$	0.865960	0.029439
	$[4,6)$	0.952796	0.007756
	$[6,8)$	0.991368	0.001068

Table 3.9: Sigmoid Coefficients for 7-9 Accurate Bits

Accurate Bits	Interval	c_0	c_1
10	$[0, \frac{1}{2})$	0.500484	0.244918
	$[\frac{1}{2}, \frac{3}{4})$	0.509288	0.226877
	$[\frac{3}{4}, 1)$	0.523872	0.207519
	$[1, \frac{5}{4})$	0.546461	0.184965
	$[\frac{5}{4}, \frac{3}{2})$	0.576301	0.161098
	$[\frac{3}{2}, \frac{7}{4})$	0.611664	0.137513
	$[\frac{7}{4}, 2)$	0.650373	0.115377
	$[2, \frac{9}{4})$	0.690262	0.095413
	$[\frac{9}{4}, \frac{5}{2})$	0.729481	0.077965
	$[\frac{5}{2}, 3)$	0.782758	0.056864
	$[3, \frac{7}{2})$	0.844413	0.036227
	$[\frac{7}{2}, 4)$	0.891742	0.022652
	$[4, 5)$	0.937520	0.011293
	$[5, 6)$	0.972463	0.004220
	$[6, 8)$	0.991368	0.001068
11	$[0, \frac{1}{4})$	0.500062	0.248706
	$[\frac{1}{4}, \frac{1}{2})$	0.502068	0.241131
	$[\frac{1}{2}, \frac{3}{4})$	0.509288	0.226877
	$[\frac{3}{4}, 1)$	0.523872	0.207519
	$[1, \frac{5}{4})$	0.546461	0.184965
	$[\frac{5}{4}, \frac{3}{2})$	0.576301	0.161098
	$[\frac{3}{2}, \frac{7}{4})$	0.611664	0.137513
	$[\frac{7}{4}, 2)$	0.650373	0.115377
	$[2, \frac{9}{4})$	0.690262	0.095413
	$[\frac{9}{4}, \frac{5}{2})$	0.729481	0.077965
	$[\frac{5}{2}, \frac{11}{4})$	0.766639	0.063086
	$[\frac{11}{4}, 3)$	0.800821	0.050643
	$[3, \frac{13}{4})$	0.831530	0.040395
	$[\frac{13}{4}, \frac{7}{2})$	0.858599	0.032058
	$[\frac{7}{2}, 4)$	0.891742	0.022652
	$[4, \frac{9}{2})$	0.926231	0.013998
	$[\frac{9}{2}, 5)$	0.950497	0.008588
	$[5, 6)$	0.972463	0.004220
	$[6, 8)$	0.991368	0.001068

Table 3.10: Sigmoid Coefficients for 10-11 Accurate Bits

Accurate Bits	Interval	c_0	c_1
7	$[0, 1)$	0.255890	-0.053388
	$[1, 2)$	0.287292	-0.091626
	$[2, 3)$	0.222136	-0.059816
	$[3, 4)$	0.126229	-0.027513
	$[4, 8)$	0.031428	-0.004331
8	$[0, \frac{1}{2})$	0.251816	-0.029992
	$[\frac{1}{2}, 1)$	0.274444	-0.076783
	$[1, 2)$	0.287292	-0.091626
	$[2, 3)$	0.222136	-0.059816
	$[3, 4)$	0.126229	-0.027513
	$[4, 8)$	0.031428	-0.004331
9	$[0, \frac{1}{2})$	0.251816	-0.029992
	$[\frac{1}{2}, 1)$	0.274444	-0.076783
	$[1, 2)$	0.287292	-0.091626
	$[2, \frac{5}{2})$	0.243907	-0.069779
	$[\frac{5}{2}, 3)$	0.194156	-0.049854
	$[3, 4)$	0.126229	-0.027513
	$[4, 6)$	0.046308	-0.007598
	$[6, 8)$	0.008608	-0.001065

Table 3.11: Sigmoid Derivative Coefficients for 7-9 Accurate Bits

Accurate Bits	Interval	c_0	c_1
10	$[0, \frac{1}{4})$	0.250479	-0.015463
	$[\frac{1}{4}, \frac{1}{2})$	0.257686	-0.044521
	$[\frac{1}{2}, \frac{3}{4})$	0.269542	-0.068434
	$[\frac{3}{4}, 1)$	0.281944	-0.085132
	$[1, \frac{3}{2})$	0.291686	-0.094933
	$[\frac{3}{2}, 2)$	0.281132	-0.088305
	$[2, \frac{5}{2})$	0.243907	-0.069779
	$[\frac{5}{2}, 3)$	0.194156	-0.049854
	$[3, \frac{7}{2})$	0.145077	-0.033447
	$[\frac{7}{2}, 4)$	0.103680	-0.021580
	$[4, 5)$	0.061073	-0.011014
	$[5, 6)$	0.020320	-0.002868
	$[6, 8)$	0.008608	-0.001065
11	$[0, \frac{1}{4})$	0.250479	-0.015463
	$[\frac{1}{4}, \frac{1}{2})$	0.257686	-0.044521
	$[\frac{1}{2}, \frac{3}{4})$	0.269542	-0.068434
	$[\frac{3}{4}, 1)$	0.281944	-0.085132
	$[1, \frac{3}{2})$	0.291686	-0.094933
	$[\frac{3}{2}, 2)$	0.281132	-0.088305
	$[2, \frac{9}{4})$	0.254719	-0.074942
	$[\frac{9}{4}, \frac{5}{2})$	0.231484	-0.064616
	$[\frac{5}{2}, \frac{11}{4})$	0.206225	-0.054509
	$[\frac{11}{4}, 3)$	0.180634	-0.045198
	$[3, \frac{7}{2})$	0.145077	-0.033447
	$[\frac{7}{2}, 4)$	0.103680	-0.021580
	$[4, \frac{9}{2})$	0.071834	-0.013592
	$[\frac{9}{2}, 5)$	0.048702	-0.008436
	$[5, 6)$	0.020320	-0.002868
	$[6, 8)$	0.008608	-0.001065

Table 3.12: Sigmoid Derivative Coefficients for 10-11 Accurate Bits

As the multiplication unit will be the limiting factor in accuracy, focus is placed on the widths necessary for x and c_1. For these inputs, for k bits of accuracy desired, there must be $k+3$ bits of the inputs provided. Consider the case of x. The desired accuracy can be expressed as 2^{-k}. Thus, the fractional bits of x must yield a result less than or equal to this value, so:

$$2^{-f_x} \leq 2^{-k} \tag{3.22}$$

$$f_x \geq k \tag{3.23}$$

adding the three integer bits to x gives a total of $k+3$ bits. Similarly, c_1 requires $k+2$ bits, however, to ensure accuracy and provide consistency in multiply-accumulate unit design, it is provided as $k+3$ bits. Furthermore, c_0 is provided at an equal width for consistency.

3.3 Hardware Design

With the intervals and associated coefficients determined, the final step in the proposed minimax sigmoid implementation is to develop a digital hardware implementation of the design. Based on the methodology discussed in Section 3.2, a hardware design is presented.

The implementation for the proposed sigmoid approximation is shown in Figure 3.4. As can be seen in the figure, the approach consists of a number of multiplexors,

signal logic for these multiplexors, XOR gates, and a multiply-accumulate unit. The multiplexors serve as selection logic to determine the appropriate coefficients for the input argument. This is dependent upon the interval in which the input argument lies. The coefficients, discussed in Section 3.2, are hard-wired to the inputs of the multiplexors, eliminating the necessity of registers or ROM table lookups. As $x_2...x_{-2}$ is received, the multiplexor signal logic determines the interval to which the input belongs, allowing for the appropriate coefficient selection. The signal logic then outputs a number of selection signals equal to the number of intervals in the design, shown as m in Figure 3.4. These selection signals are determined by the interval over which the approximations are made. Input sequences that identify these intervals are used as the multiplexor signal inputs. These sequences are shown in Tables 3.13 and 3.14 for the sigmoid and Tables 3.15 and 3.16 for its derivative. Consider the interval $[0, 1)$. This interval is used for any input having all zero integer bits. This is determined by the input sequence $\overline{x}_2\overline{x}_1\overline{x}_0$, as any inputs matching this pattern are within the interval. Using the interval differentiating bit patterns for the multiplexor signals requires use of one-hot multiplexor devices, which typically have less area and delay than a traditional multiplexor. Use of these devices also reduces the necessity for multiplexor signal encoding prior to the multiplexor.

The coefficient values selected are sent to a multiply accumulate unit which calculates $c_0 + c_1x$. This unit is implemented with a tree multiplier as its basis. A row is added to the bottom of the partial product matrix for the addition of c_0 to

the product. This matrix is then reduced using the reduced area technique to form two resultant values. These values are then added using a carry lookahead adder. Additional information regarding the reduction technique employed can be found in [14].

The XOR gates in the device are used to conditionally invert negative inputs. As discussed, the symmetry properties of the sigmoid and its derivative allow for calculation of inputs over the interval $(-8, 0)$ based on inputs over the interval $[0, 8)$. All integer and fractional bits of the input are XOR'd with the input's sign bit, thus providing the one's complement of x for negative inputs. This allows the device to calculate $sig(-x)$ or $sig'(-x)$ as the case may be. Once this calculation is complete, the result is again sent through XOR gates with the input sign bit. This provides $-sig(-x)$ and $-sig'(-x)$. Although by definition $1 - sig(-x)$ is desired, adding a value of 1 to any of the results within the range of $-sig(-x)$ will give a zero in the integer bit. Because of this, the integer bit can be set to zero and the one's complement provides sufficient accuracy.

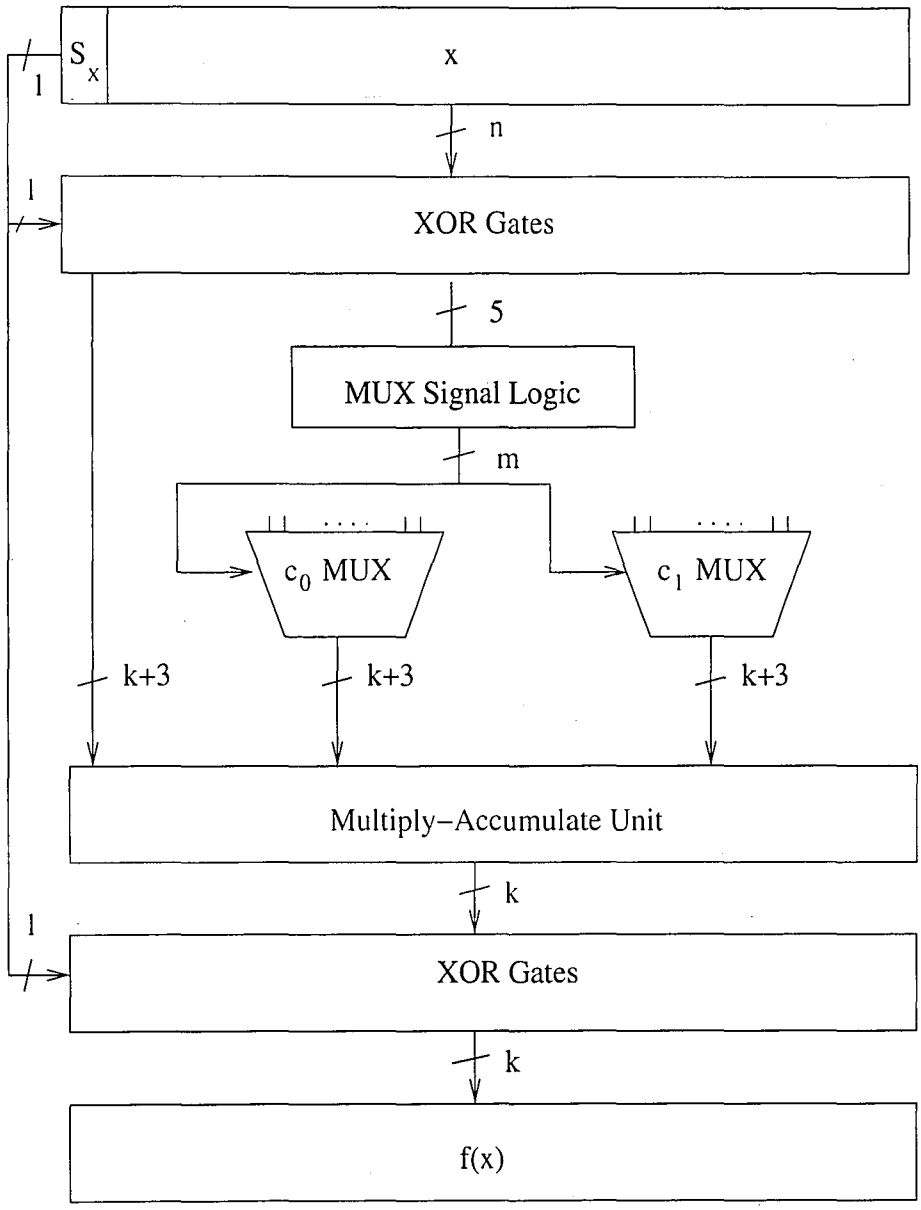

Figure 3.4: Sigmoid Implementation

Accurate Bits	Interval	Differentiating Bits
7	$[0,1)$	$\overline{x}_2\overline{x}_1\overline{x}_0$
	$[1,2)$	$\overline{x}_2\overline{x}_1x_0$
	$[2,3)$	$\overline{x}_2x_1\overline{x}_0$
	$[3,4)$	$\overline{x}_2x_1x_0$
	$[4,8)$	x_2
8	$[0,1)$	$\overline{x}_2\overline{x}_1\overline{x}_0$
	$[1,\frac{3}{2})$	$\overline{x}_2\overline{x}_1x_0\overline{x}_{-1}$
	$[\frac{3}{2},2)$	$\overline{x}_2\overline{x}_1x_0x_{-1}$
	$[2,3)$	$\overline{x}_2x_1\overline{x}_0$
	$[3,4)$	$\overline{x}_2x_1x_0$
	$[4,8)$	x_2
9	$[0,\frac{1}{2})$	$\overline{x}_2\overline{x}_1\overline{x}_0\overline{x}_{-1}$
	$[\frac{1}{2},1)$	$\overline{x}_2\overline{x}_1\overline{x}_0x_{-1}$
	$[1,\frac{5}{4})$	$\overline{x}_2\overline{x}_1x_0\overline{x}_{-1}\overline{x}_{-2}$
	$[\frac{5}{4},\frac{3}{2})$	$\overline{x}_2\overline{x}_1x_0\overline{x}_{-1}x_{-2}$
	$[\frac{3}{2},2)$	$\overline{x}_2\overline{x}_1x_0x_{-1}$
	$[2,\frac{5}{2})$	$\overline{x}_2x_1\overline{x}_0\overline{x}_{-1}$
	$[\frac{5}{2},3)$	$\overline{x}_2x_1\overline{x}_0x_{-1}$
	$[3,4)$	$\overline{x}_2x_1x_0$
	$[4,6)$	$x_2\overline{x}_1$
	$[6,8)$	x_2x_1

Table 3.13: Sigmoid Interval Differentiating Bits for 7-9 Accurate Bits

Accurate Bits	Interval	Differentiating Bits
10	$[0, \frac{1}{2})$	$\overline{x}_2\overline{x}_1\overline{x}_0\overline{x}_{-1}$
	$[\frac{1}{2}, \frac{3}{4})$	$\overline{x}_2\overline{x}_1\overline{x}_0x_{-1}\overline{x}_{-2}$
	$[\frac{3}{4}, 1)$	$\overline{x}_2\overline{x}_1\overline{x}_0x_{-1}x_{-2}$
	$[1, \frac{5}{4})$	$\overline{x}_2\overline{x}_1x_0\overline{x}_{-1}\overline{x}_{-2}$
	$[\frac{5}{4}, \frac{3}{2})$	$\overline{x}_2\overline{x}_1x_0\overline{x}_{-1}x_{-2}$
	$[\frac{3}{2}, \frac{7}{4})$	$\overline{x}_2\overline{x}_1x_0x_{-1}\overline{x}_{-2}$
	$[\frac{7}{4}, 2)$	$\overline{x}_2\overline{x}_1x_0x_{-1}x_{-2}$
	$[2, \frac{9}{4})$	$\overline{x}_2x_1\overline{x}_0\overline{x}_{-1}\overline{x}_{-2}$
	$[\frac{9}{4}, \frac{5}{2})$	$\overline{x}_2x_1\overline{x}_0\overline{x}_{-1}x_{-2}$
	$[\frac{5}{2}, 3)$	$\overline{x}_2x_1\overline{x}_0x_{-1}$
	$[3, \frac{7}{2})$	$\overline{x}_2x_1x_0\overline{x}_{-1}$
	$[\frac{7}{2}, 4)$	$\overline{x}_2x_1x_0x_{-1}$
	$[4, 5)$	$x_2\overline{x}_1\overline{x}_0$
	$[5, 6)$	$x_2\overline{x}_1x_0$
	$[6, 8)$	x_2x_1
11	$[0, \frac{1}{4})$	$\overline{x}_2\overline{x}_1\overline{x}_0\overline{x}_{-1}\overline{x}_{-2}$
	$[\frac{1}{4}, \frac{1}{2})$	$\overline{x}_2\overline{x}_1\overline{x}_0\overline{x}_{-1}x_{-2}$
	$[\frac{1}{2}, \frac{3}{4})$	$\overline{x}_2\overline{x}_1\overline{x}_0x_{-1}\overline{x}_{-2}$
	$[\frac{3}{4}, 1)$	$\overline{x}_2\overline{x}_1\overline{x}_0x_{-1}x_{-2}$
	$[1, \frac{5}{4})$	$\overline{x}_2\overline{x}_1x_0\overline{x}_{-1}\overline{x}_{-2}$
	$[\frac{5}{4}, \frac{3}{2})$	$\overline{x}_2\overline{x}_1x_0\overline{x}_{-1}x_{-2}$
	$[\frac{3}{2}, \frac{7}{4})$	$\overline{x}_2\overline{x}_1x_0x_{-1}\overline{x}_{-2}$
	$[\frac{7}{4}, 2)$	$\overline{x}_2\overline{x}_1x_0x_{-1}x_{-2}$
	$[2, \frac{9}{4})$	$\overline{x}_2x_1\overline{x}_0\overline{x}_{-1}\overline{x}_{-2}$
	$[\frac{9}{4}, \frac{5}{2})$	$\overline{x}_2x_1\overline{x}_0\overline{x}_{-1}x_{-2}$
	$[\frac{5}{2}, \frac{11}{4})$	$\overline{x}_2x_1\overline{x}_0x_{-1}\overline{x}_{-2}$
	$[\frac{11}{4}, 3)$	$\overline{x}_2x_1\overline{x}_0x_{-1}x_{-2}$
	$[3, \frac{13}{4})$	$\overline{x}_2x_1x_0\overline{x}_{-1}\overline{x}_{-2}$
	$[\frac{13}{4}, \frac{7}{2})$	$\overline{x}_2x_1x_0\overline{x}_{-1}x_{-2}$
	$[\frac{7}{2}, 4)$	$\overline{x}_2x_1x_0x_{-1}$
	$[4, \frac{9}{2})$	$x_2\overline{x}_1\overline{x}_0\overline{x}_{-1}$
	$[\frac{9}{2}, 5)$	$x_2\overline{x}_1\overline{x}_0x_{-1}$
	$[5, 6)$	$x_2\overline{x}_1x_0$
	$[6, 8)$	x_2x_1

Table 3.14: Sigmoid Interval Differentiating Bits for 10-11 Accurate Bits

Accurate Bits	Interval	Differentiating Bits
7	$[0,1)$	$\overline{x}_2\overline{x}_1\overline{x}_0$
	$[1,2)$	$\overline{x}_2\overline{x}_1x_0$
	$[2,3)$	$\overline{x}_2x_1\overline{x}_0$
	$[3,4)$	$\overline{x}_2x_1x_0$
	$[4,8)$	x_2
8	$[0,\frac{1}{2})$	$\overline{x}_2\overline{x}_1\overline{x}_0\overline{x}_{-1}$
	$[\frac{1}{2},1)$	$\overline{x}_2\overline{x}_1\overline{x}_0x_{-1}$
	$[1,2)$	$\overline{x}_2\overline{x}_1x_0$
	$[2,3)$	$\overline{x}_2x_1\overline{x}_0$
	$[3,4)$	$\overline{x}_2x_1x_0$
	$[4,8)$	x_2
9	$[0,\frac{1}{2})$	$\overline{x}_2\overline{x}_1\overline{x}_0\overline{x}_{-1}$
	$[\frac{1}{2},1)$	$\overline{x}_2\overline{x}_1\overline{x}_0x_{-1}$
	$[1,2)$	$\overline{x}_2\overline{x}_1x_0$
	$[2,\frac{5}{2})$	$\overline{x}_2x_1\overline{x}_0\overline{x}_{-1}$
	$[\frac{5}{2},3)$	$\overline{x}_2x_1\overline{x}_0x_{-1}$
	$[3,4)$	$\overline{x}_2x_1x_0$
	$[4,6)$	$x_2\overline{x}_1$
	$[6,8)$	x_2x_1

Table 3.15: Sigmoid Derivative Interval Differentiating Bits for 7-9 Accurate Bits

Accurate Bits	Interval	Differentiating Bits
10	$[0, \frac{1}{4})$	$\overline{x}_2\overline{x}_1\overline{x}_0\overline{x}_{-1}\overline{x}_{-2}$
	$[\frac{1}{4}, \frac{1}{2})$	$\overline{x}_2\overline{x}_1\overline{x}_0\overline{x}_{-1}x_{-2}$
	$[\frac{1}{2}, \frac{3}{4})$	$\overline{x}_2\overline{x}_1\overline{x}_0x_{-1}\overline{x}_{-2}$
	$[\frac{3}{4}, 1)$	$\overline{x}_2\overline{x}_1\overline{x}_0x_{-1}x_{-2}$
	$[1, \frac{3}{2})$	$\overline{x}_2\overline{x}_1x_0\overline{x}_{-1}$
	$[\frac{3}{2}, 2)$	$\overline{x}_2\overline{x}_1x_0x_{-1}$
	$[2, \frac{5}{2})$	$\overline{x}_2x_1\overline{x}_0\overline{x}_{-1}$
	$[\frac{5}{2}, 3)$	$\overline{x}_2x_1\overline{x}_0x_{-1}$
	$[3, \frac{7}{2})$	$\overline{x}_2x_1x_0\overline{x}_{-1}$
	$[\frac{7}{2}, 4)$	$\overline{x}_2x_1x_0x_{-1}$
	$[4, 5)$	$x_2\overline{x}_1\overline{x}_0$
	$[5, 6)$	$x_2\overline{x}_1x_0$
	$[6, 8)$	x_2x_1
11	$[0, \frac{1}{4})$	$\overline{x}_2\overline{x}_1\overline{x}_0\overline{x}_{-1}\overline{x}_{-2}$
	$[\frac{1}{4}, \frac{1}{2})$	$\overline{x}_2\overline{x}_1\overline{x}_0\overline{x}_{-1}x_{-2}$
	$[\frac{1}{2}, \frac{3}{4})$	$\overline{x}_2\overline{x}_1\overline{x}_0x_{-1}\overline{x}_{-2}$
	$[\frac{3}{4}, 1)$	$\overline{x}_2\overline{x}_1\overline{x}_0x_{-1}x_{-2}$
	$[1, \frac{3}{2})$	$\overline{x}_2\overline{x}_1x_0\overline{x}_{-1}$
	$[\frac{3}{2}, 2)$	$\overline{x}_2\overline{x}_1x_0x_{-1}$
	$[2, \frac{9}{4})$	$\overline{x}_2x_1\overline{x}_0\overline{x}_{-1}\overline{x}_{-2}$
	$[\frac{9}{4}, \frac{5}{2})$	$\overline{x}_2x_1\overline{x}_0\overline{x}_{-1}x_{-2}$
	$[\frac{5}{2}, 3)$	$\overline{x}_2x_1\overline{x}_0x_{-1}$
	$[3, \frac{7}{2})$	$\overline{x}_2x_1x_0\overline{x}_{-1}$
	$[\frac{7}{2}, 4)$	$\overline{x}_2x_1x_0x_{-1}$
	$[4, \frac{9}{2})$	$x_2\overline{x}_1\overline{x}_0\overline{x}_{-1}$
	$[\frac{9}{2}, 5)$	$x_2\overline{x}_1\overline{x}_0x_{-1}$
	$[5, 6)$	$x_2\overline{x}_1x_0$
	$[6, 8)$	x_2x_1

Table 3.16: Sigmoid Derivative Interval Differentiating Bits for 10-11 Accurate Bits

Chapter 4

Results

In order to test the validity of the proposed approach to sigmoid approximation, discussed in Chapter 3, the designs have been implemented and synthesized for the configurations discussed in Section 3.2. The results of these implementations are presented in this chapter. The chapter begins with a discussion of the manner in which the designs have been implemented, followed by a presentation of the area and delay results for the design synthesis.

4.1 Implementation and Synthesis

Modular, structural-level VHDL descriptions of the ten configurations presented in Chapter 3 have been prepared. Modules for the arithmetic components of the designs were generated automatically using existing VHDL generation Java code

provided through the FGS program.

The designs were simulated for behavior confirmation and then synthesized for FPGA devices using Altera's Quartus software. This software package uses Altera FPGA devices as its target technologies, and claims to provide extensive logic-fitting algorithms for optimal layouts. The software provides synthesis data for device usage in terms of logic cells, esb bits, and pins, as these are typically of importance to FPGA designs. The logic cells implement gates while the esb bits implement memory.

The designs were also synthesized to ASIC technology using the Leonardo Spectrum synthesis package. Leonardo's provided SCL05u ASIC library was used as the target technology for synthesis. The area results provided by this package are in terms of equivalent gates, as per standard ASIC design terminology.

4.2 Area and Delay Estimates

As stated, the devices were synthesized to Leonardo's SCL05u ASIC library. The synthesis results for the sigmoid implementations are shown in 4.1 and the sigmoid derivative results are shown in 4.2. Other than a slight anomaly in delay between the ten and eleven bit sigmoid cases, the devices scale regularly in terms of both area and delay as the number of accurate bits increases. One potential reason for the mentioned anomaly is the irregular size of the multiply-accumulate unit and its

Accurate Bits	Area (Gates)	Delay (ns)
7	2268	16.01
8	2560	16.79
9	3323	19.18
10	3756	19.87
11	4392	19.54

Table 4.1: Sigmoid ASIC Area and Delay Estimates

Accurate Bits	Area (Gates)	Delay (ns)
7	2063	15.66
8	2356	16.56
9	3075	19.49
10	3514	19.70
11	4114	20.06

Table 4.2: Sigmoid Derivative ASIC Area and Delay Estimates

associated carry lookahead adder. Eliminating this anomaly was not explored as the difference in delay between the ten and eleven bit cases was considered negligible.

Regarding the FPGA synthesis, each of the devices was synthesized to an Altera E20k30ETC144-1 FPGA. The synthesis results for the cases synthesized are presented in Table 4.3 for the sigmoid and Table 4.4 for the sigmoid's derivative. Area is given in terms of logic cells as this was the only considerable area result given. No esb usage was reported, since neither registers nor lookup tables were in use. As can be seen in the tables, both delay and area increase with increases in desired accuracy. The largest jump occurs between 10 and 11 bits of accuracy for the sigmoid, as this requires a considerably higher number of intervals than the other cases.

Accurate Bits	Area (Logic Cells)	Delay (ns)
7	351	34.46
8	449	38.54
9	531	40.41
10	603	42.66
11	813	51.86

Table 4.3: Sigmoid FPGA Area and Delay Estimates

Accurate Bits	Area (Logic Cells)	Delay (ns)
7	320	32.60
8	391	32.90
9	518	41.40
10	622	45.84
11	670	46.31

Table 4.4: Sigmoid Derivative FPGA Area and Delay Estimates

Chapter 5

Conclusions

Given the results presented in Chapter 4, a number of conclusions can be made regarding the validity of the proposed approach to sigmoid and sigmoid derivative approximations. In this chapter, potential benefits and drawbacks of the proposed method are discussed, followed by outlets for future research based on the material presented within this thesis.

5.1 Considerations of the Proposed Approach

One important consideration when using the approach proposed in this thesis is its lack of registers and ROM tables. While this potentially offers improvements, it can also lead to implementation issues. Obviously, an improvement offered by this

technique is the lack of dependency upon register logic within one's target implementation technology. Whether an implementation is to be performed on an FPGA or ASIC by varying manufacturers, the device should not perform any differently based on register technology. Also, synthesis to devices not offering extensive memory for lookup tables is a possibility, which could potentially be of benefit. One possible disadvantage, however, is the lack of implicit addressing that lookup tables provide. As discussed in Chapter 3, the technique's implementation calls for one-hot multiplexors to control coefficient selection. By using ROM lookup tables along with evenly-sized intervals, addressing is implicit.

Another consideration involves scalability of this technique. Again, since there is no dependence upon ROM, devices scale linearly or quadratically in terms of area. Delay is found within the multiply-accumulate units and carry-lookahead adders of the device, which could potentially be problematic as input sizes and desired accuracies increase.

Finally, concerning FPGA implementations of these designs, an important point to consider is the amount of area consumption per device module with these designs. For the ten configurations discussed in the previous two chapters, the device which consumes the largest amount of area is the multiply accumulate unit. As stated in Chapter 3, this unit consists of a modified tree multiplier followed by a carry lookahead adder. In many FPGA devices, multipliers and adders are on-board and available for use by other design components. With this in mind, a

design could be implemented using the existing multiplication and addition units, effectively eliminating the largest component of the design, while maintaining the speed and efficiency of the logic gate components.

5.2 Future Research

Using the work presented in this thesis as a starting point, there are a number of areas in which further work could be pursued. First, since the scope of this thesis was to develop a new approach to sigmoid approximation rather than extensive digital synthesis, one possible extension to this work is furthering the FPGA synthesis results. In addition to the Altera FPGA synthesis performed, devices from other manufacturers, such as Xilinx, could be synthesized. ASIC synthesis using other ASIC libraries is also a potential direction. This would potentially provide further proof of the benefits of using this approximation technique for ASIC devices.

With respect to the actual technique discussed, exploration could be made regarding possible advantages of using evenly-sized intervals whenever possible. Also, this technique could be applied to other functions such as logarithmic, exponential, and trigonometric functions. If this research proved effective with these functions as well, a general form for implementing function approximations using the proposed technique might be possible, for any given function meeting criteria such as continuous, differentiable, etc.

Higher-order minimax approximations could be considered for the sigmoid function and its derivative. This thesis focused on first-order approximations as they were simplest to implement. It should be noted, however, that the amount of accuracy available for designs may be limited depending on the function in question. By using higher-order approximations, it may be possible to attain a balance of higher accuracy with moderate hardware complexity.

Finally, software to generate various aspects of this work could be created. One starting point might be the design of the algorithm for interval determination discussed in Chapter 3. Automation of VHDL or Verilog code generation for simulation and synthesis would also be a possible area of future work, potentially working with the existing programs used to generate the carry lookahead adders and multiply-accumulate units discussed in Chapter 3.

As the results and these directions show, this implementation technique of partitioned minimax approximation of the sigmoid and its derivative offers potential benefits for FPGA and ASIC implementations, due to its lack of lookup table and register dependencies, as well as its relatively simple implementation.

Bibliography

[1] P. Murtagh and A. Tsoi, "Implementation issues of sigmoid function and its derivative for VLSI neural networks," in *IEEE Proc.- E*, May 1992. vol.139, no.3.

[2] J. Hennesey and D. Patterson, *Computer Architecture: A Quantitative Approach, Second Edition.* San Francisco, CA: Morgan-Kauffman, 1996.

[3] J. S. Walther, "A unified algorithm for elementary functions," in *Proceedings of Spring. Joint Computing Conference*, pp. 379–385, 1971.

[4] K. Atkinson, *Elementary Numerical Analysis, Second Edition.* New York, NY: Wiley & Sons, 1993.

[5] M. Wedlake and H. Kwok, "A CORDIC implementation of a digital artificial neuron," in *IEEE PACRIM Proceedings*, vol. 2, pp. 798–801, 1997.

[6] T. S. B. Gisuthan and K. Asari, "A high speed flat CORDIC based neuron with multi-level acitvation function for robust pattern recognition," *IEEE*, vol. 11,

pp. 1–12, 2000.

[7] D. Myers and R. Hutchinson, "Efficient implementation of piecewise linear activation function for digital VLSI neural networks," *Electronic Letters*, vol. 25, pp. 1662–1663, 1989.

[8] J. D.-F. Stamatis Vassiliadis and M. Zhang, "High performance with low implementation cost sigmoid generators," in *International Joint Conference on Neural Networks*, pp. 1931–1933, 1993.

[9] S. V. M. Zhang and J. Delgado-Frias, "Sigmoid generators for neural computing using piecewise approximations," *IEEE Transactions on Computers*, vol. 45, September 1996. No. 9.

[10] M. Z. S. Vassiliadis and J. Delgado-Frias, "Elementary function generators for neural-network emulators," *IEEE Transactions on Neural Networks*, vol. 11, pp. 1438–1449, November 2000. No. 6.

[11] J. E. Stine and M. J. Schulte, "The symmetric table addition method for accurate function approximation," *Journal of VLSI Signal Processing*, vol. 11, pp. 1–12, 1999.

[12] J. S. N. Koc-Sahan and M. Schulte, "Symmetric table addition methods for neural network approximations," in *Proceedings of SPIE: Advanced Signal Processing Algorithms, Architectures, and Implementations XI*, pp. 126–133, 2001.

[13] J.H.Mathews, *Numerical Methods for Computer Science, Engineering and Mathematics*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[14] K.C. Bickerstaff, M. J. Schulte, and E. E. Swartzlander, Jr., "Parallel reduced area multipliers," *Journal of VLSI Signal Processing*, vol. 9, pp. 181–192, April 1995.

Vita

Jason Schlessman was born in Reading, Pennsylvania on June 11, 1975 to Warren and Sharon Schlessman. He attended the Pennsylvania State University as an undergraduate with a major in computer engineering from August 1993 to May 1995. During this time Jason was named to the dean's list each semester and received the Vollner-Kleckner award for academic excellence. At this point he took time off to consider his academic career. Upon consideration, Jason attended Lehigh University as an undergraduate with a major in computer engineering beginning in January 1999 and culminating in his earning a Bachelor of Science degree in computer engineering June 2001. During this time he was named to the dean's list each semester, graduated summa cum laude, and served as a grader and teaching assistant for two senior level computer engineering courses. Since June 2001, Jason has pursued a Master of Science degree at Lehigh University as a graduate student and expects to earn this degree August 2002. Jason served as a teaching assistant for a senior level computer engineering course as a graduate student as well.

Jason published four papers in international journals under the auspices of the

Pennsylvania State University as well as two conference papers under the auspices of Lehigh University. Jason served as speaker for one of the latter papers at the 2001 International SPIE Conference. [12]

END OF TITLE