摘　要

本文重点研究了 softmax 函数的实现算法、实现结构和验证方法。选择固定数据格式和分段逼近算法作为超越函数的实现算法基础。对分段逼近算法进行了深入研究，提出了分段二次逼近函数算法，并验证了其合理性与正确性。在实现函数过程中，硬件结构计算使用的是十六位浮点型数据，做乘除法的时候需要调用 IP 核。

本文对硬件实现 softmax 函数的每个模块进行模拟仿真验证，对所实现函数加速器按照自底向上的方法分层次进行了充分的功能验证。平均误差 2.3460e-05，最大误差 4.3830e-05。

关键字： softmax；verilog；分段二次逼近；二次插值法

ABSTRACT

In this paper, we focus on the implementation of the Softmax function algorithm, the implementation of the structure and verification methods. The fixed data format and piecewise approximation algorithm are chosen as the basis for the implementation of transcendental function. The piecewise approximation algorithm is studied in depth, and a piecewise two approximation function algorithm is proposed, and its rationality and correctness are verified. In the process of implementing the function, the hardware structure is computed using 16-bit floating-point data, and the IP core needs to be invoked when multiplication and division is done.

In this paper, the simulation of each module of the Softmax function is validated, and the function accelerator is fully validated according to the Bottom-up method. Average error is 2.3460e-05, and maximum error is 4.3830e-05.

Key words: softmax; verilog; piecewise two approximation; quadratic interpolation method

目 录

第一章 绪论 1

1.1 研究背景及意义 1
1.2 Softmax 函数 1
1.3 研究发展现状 2
1.4 本课题的主要工作和内容安排 2

第二章 softmax 函数的实现算法研究与软件验证 4

2.1 浮点数介绍 4
2.1.1 浮点表示法 4
2.1.2 浮点数加减法 4
2.1.3 浮点数乘法 5
2.2 基于查找表的多项式逼近 6
2.2.1 分段线性逼近 6
2.2.2 二次逼近 8
2.3 分段二次逼近算法的研究 9
2.3.1 最佳等距分段法 9
2.3.2 二次插值算法 10
2.4 算法实现结果对比 12
2.5 算法的 Matlab 仿真 12
2.6 基于 Python 语言验证 softmax 函数单元 14
2.7 本章小结 15

第三章 softmax 单元的硬件实现 16

3.1 系统整体介绍 16
3.2 参数存储模块 16
3.3 寻址模块 17
3.4 计算模块 18
3.5 本章小结 21

第四章 设计验证和误差分析 22

4.1 系统模块验证 22
4.1.1 寻址模块验证 22

4.1.2 取值模块验证.....22
4.1.3 计算 y 值模块验证.....23
4.1.4 计算最终函数值模块验证.....24
4.3 系统整体验证 25
4.4 误差分析 25
4.5 本章小结 27
结束语.....35
致谢.....36
参考文献.....37

第一章　绪论

1.1　研究背景及意义

神经网络[1]是一种模仿人类神经系统工作机制的信息处理技术。它是基于错误传播的前向网络。它是由多个神经元相互连接而成的网络结构。根据输入信息调整内部数学映射模型，最终具有较强的映射能力，可以解决各种实际应用。

神经网络的发展非常曲折。它诞生于 20 世纪 40 年代，但在 20 世纪 70 年代后逐渐遇到瓶颈。但幸运的是，许多学者并没有放弃。 20 世纪 80 年代以后，随着 Hopfield 反馈网络[2]和自组网络[2]的提出，神经网络再次迎来了蓬勃发展的春天[1]。作为一门新兴学科，它在算法，理论和应用方面有很大的发展空间。世界各地都开展了许多关于人工神经网络的研究活动。因此，大量与神经网络相关的工程应用和专用芯片正在涌现。随着大规模集成电路，半导体器件和人造神经计算机的发展，人造神经网络呈现出蓬勃发展的景象。

有许多类型的神经网络，其中最常见的是 softmax 单元。它广泛应用于函数逼近，模式分类，聚类，数据压缩，回归拟合等领域[2]。大部分各种应用都基于软件实现。然而，随着电子技术的不断发展，软件实施缓慢已成为阻碍神经网络进一步发展的原因之一。为了解决这个问题，有必要研究一种实现 softmax 单元的硬件设计方法。

1.2　Softmax 函数

在数学中，尤其是概率论和相关领域中，Softmax 函数，或称归一化指数函数，是逻辑函数的一种推广。它能将一个含任意实数的 K 维的向量 z 的“压缩”到另一个 K 维实向量 σ(z) 中，使得每一个元素的范围都在(0,1)之间，并且所有元素的和为 1。

X 为一个数组，X_i 表示 X 中的第 i 个元素，这个元素对应的 softmax 值为

$$softmax(X)_i = \frac{exp(X_i)}{\sum_j exp(X_j)} \qquad (1\text{-}1)$$

10 分类的 softmax 函数图像如图 1.1 所示：

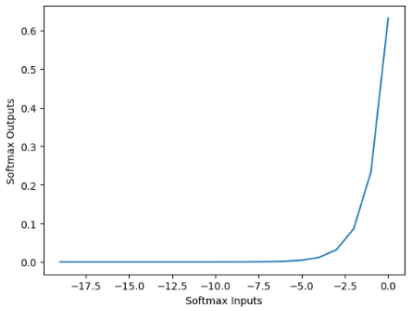

图 1.1　10 分类的 softmax 函数

1.3　研究发展现状

Softmax 可以计算与标注样本的差距，在计算上非常非常的方便，在算法、理论、应用等方面都有很大的发展空间值得我们去研究和探索。许多和人工神经网络相关的科研活动，在全世界各地进行得如火如荼。其中，关于神经网络硬件加速的课题早已成为学术界的热门。

针对这个课题，国内外许多学者对神经网络的硬件实现开展了许多科研立项。S. L. Pinjare 选择了误差反向传播算法在 FPGA 上实现图片压缩和解压缩[3]。研究表明，神经网络芯片可以实现在线训练并且具备图片压缩解压缩的功能，最大压缩率为 7%。Chaitra.P 为了处理诸如大规模并行，容错，自学习，适应性和计算复杂度等问题,研发了硬件实现的 BP 人工神经网络智能系统[4]。使用 Verilog 语言实现 ANN 和反向传播训练算法，利用 ModelsimSE 6.3F Simulator 进行仿真验证 Verilog 程序的正确性。在这些反向传播算法中，设计人员提出了不同的算法来训练神经网络，为本论文提供了很好的指导。除此之外，李利歌等人研究了一种如何在 FPGA 上实现神经网络的硬件技术 [5]。结果显示，该硬件网络可以提升网络 68%的学习速度。

总的来说，随着神经网络的迅猛发展，硬件实现神经网络的技术将在越来越多的行业中发挥重要作用，应用前景更加广阔。

1.4　本课题的主要工作和内容安排

本文主要内容是基于神经网络实现硬件实现 softmax 单元。抛砖迎玉，介绍了神经网络硬件实现的方法。其主要工作如下：

（1） 研究了神经网络的神经元及网络模型，简明扼要地阐述了硬件实现神经网络的具体意义。

（2） 设计编写 python 语言程序，完成软件上对 softmax 单元算法的验证。详细介绍 softmax 单元设计的基本方法。

（3） 根据算法思想和系统要求，设计算法硬件电路系统。在 Modelsim 上进行功能仿真，验证其逻辑正确性。

（4） 根据详细的实验结果分析，对设计工作进行了总结，并对今后的学习和研究提出了建议。

本文包括七个章节，安排如下：

第一章， 撰写绪论，提出本论文的主要研究问题和目的，国内外发展现状及对深度神经网络进行研究，归纳总结深度神经网络中 softnax 单元设计的基本要点。给系统设计积累理论指导。

第二章， softmax 函数的实现算法研究，通过二次逼近拟合 softmax 函数，并研究了分段二次逼近算法的方法和仿真验证，利用 python 语言编写程序，验证系统设计的合理性。

第三章， 利用 verilog 语言设计电路系统，并且通过 Modelsim 进行功能仿真。

第四章， 设计系统验证方案，分析误差。

第二章 softmax 函数的实现算法研究与软件验证

2.1 浮点数介绍

常见的浮点数是在 IEE75 标准下制定的一种浮点数格式[9]。通常，浮点数的表示如式 2-1 所示。

$$N=S*R^{j} \tag{2-1}$$

式中，S 为尾数（正负亦可）J 为阶码（正负亦可），R 是基数。在计算中，基数可取 2、4、8 或 16 等。

2.1.1 浮点表示法

IEE754 标准定义了两种浮点数，一种为单精度，另一种为双精度浮点形式。在本文中，需要用 Verilog 语言实现单精度 16 位浮点数的算法，图 2.1 所示，一个 16 位单精度浮点数组成是有三个部分： 第 16 位是符号位 s（ Sign），0 表示正数，1 表示负数。第 15~10 位是阶码 e（ Exponent）。第 9~0 位是 10 位尾数 f（ Fraction）。

s(1)	e(5)	f(10)

图 2.1 单精度浮点数的格式

IEE754 标准还定义了浮点数的的浮点格式，一种为规格化浮点数，另一种非规格化浮点数。如果 $0< e < 255$, 单精度浮点数的值如式 2-2 所示：

$$（-1）^{s}\times 2^{e-127}\times 1.f \tag{2-2}$$

此时， 这个数就表示为规格化的浮点数。需要注意的是，在计算规格化的公式中有一项为 *1.f*， 1 是隐藏位，他不存在于 16 位数据中的任何一位。阶码的表示都是用移码，偏移值为 127。在表示非规格化的浮点数时，它的格式如下：若 e=0，非规格化的单精度 浮点数的值如式 2-3 所示：

$$（-1）^{s}\times 2^{e-127}\times 0.f \tag{2-3}$$

若 *e=0* 且 *f=0*，浮点数的数值就为+0 或者-0。

若 *e=255*， *f=0*，这个浮点数就会表示成+∞或者-∞。

若 *e=255*， *f≠0*，那么，这个浮点数则不是一个数

2.1.2 浮点数加减法

浮点的加减法运算原理是先要对数据进行预处理，保证两组数据的运算的正确性。在使用浮点数做减法时，需要用大数减小数，所以先判断绝对值的大 小。

结果阶码暂定成为阶码较大的数，因为在对结果进行规格化处理之后，阶码可能会发生改变。在做实际运算的情况下，需要判断结果的符号位，是根据两个数的符号位 以及运算符号判别。预处理也应考虑两数是否为 NAN 或无穷大（ INF），用以 判断结果。最后则是进行移位处理，用以保证阶码相等的情况下进行运算。在右移的情况下，我们要根据 IEE754 标准规定进行运算以保障运算精度。

IEE754 标准规定在用浮点数做加减法运算时，需要在尾数的末位再加 3 位的附加位，从左至右分别为 G（ Guard） R（ Round） S（ Sticky）。在计算时，绝 对值更大的数的 GRS 为 0。在对绝对值比较小的数进行右移的情况下，若 R 位后有任何一位为 1，则设置 S 为 1。

第二步进行运算。在运算时，需要根据实际情况来进行运算，选择加法还 是减法。计算式应考虑 GRS 的计算，并且考虑到尾数运算结果可能是与那里一个尾数两倍时，在最高位还需加 1 位。

最后就要对运算的结果进行规格化处理。根据 IEE754 标准可以得知， 对运算后的结果需要经过两个步骤进行处理。第一步先进行舍入处理，即对预先 留下的 GRS 进行处理。 IEE754 规定了如下所示的 4 种舍入方式情况：

（1） 就近舍入：共有三种情况，第一：如果 GRS>100,则要对尾数的结果 1；第二：如果 GRS<100，即尾数不进行操作；第三： GRS=100，这时就要看尾数的最后一位，如果最后一位数为 1，就对尾数加 1，否则不加。

（2） 向下舍入：若结果为负，且 GRS≠000，尾数加 1。

（3） 向上舍入：若结果为正，且 GRS≠000，尾数加 1。

（4） 向 0 舍入：直接舍弃 GRS。

经过舍入处理之后可能结果会使得阶码加 1 ，或者阶码也可能直接全 1，所以此时应该对结果进行上溢处理。 IEE754 标准规定如下：

（1） 就近舍入：上溢结果设置成无穷大。

（2） 向 0 舍入：上溢结果设置成最大的规格化数。

（3） 向下舍入：把负数的上溢结果设置为无穷大，正数的上溢结果置为绝对值最大的正规格化数。

（4） 向上舍入：把正数的上溢结果设置为无穷大，负数的上溢结果置为绝对值最大的负规格化数。

2.1.3 浮点数乘法

在对浮点数进行乘法运算的情况下，可以判断乘法运算会比加减法运算简

单许多许多，这是因为，两组数据不需要判断数据的大小，对阶码只需相加就 可。做乘法运算，就是对尾数相乘，并且提前判定好符号位的正负值，最终再 执行与加法运算相似的规格化运算即可，这样乘法运算的结果就出现了。 所以在实际运算过程中， 加法器的运算也是类似的，同样，最开始为预处 理，第二步为乘法运算，第三步为规划化处理。下面为浮点乘法的运算格式。 设浮点数 a = $\{s_a, e_a, f_a\}$,浮点数 b= $\{s_b, e_b, f_b\}$,浮点数 c= $\{s_c, e_c, f_c\}$= a × b。

a，b 都为规格化数，则 c 的绝对值如公式 2-4 所示：

$$|c|=|a|=|b|=2^{(e_a+e_b-127)} \times (1.f_a \times f_b) \tag{2-4}$$

可以得知 $1.0 \ll （ 1.f_a \times 1.f_b） \leq 4.0$。若结果$< 2.0$，则 $e_c = e_a + e_b - 127, 1.f_c = 1.f_a \times 1.f_b$；否则$e_c = e_a + e_b - 127 + 1, 1.f_c = 1.f_a \times 1.f_b >> 1$。由于规格化数的阶码 e 需要满足$1 \leq e \leq 254$,而$-124 \leq （ e_a + e_b - 127 + 1） \leq 382$，所以$e_c$的结果有可能会超过浮点数的表达形式。若 $1 \leq e_c \leq 254$， 则 c 是一个规格化的浮点数；若$e_c > 254$，计算结果则用无穷大表示（ $e_c = 255$， $f_c = 0$）；若 $e_c < 1$ 时 ，若 $|c| \geq （ 2^{-127} \times 0.00000000000000000000001 = 2^{-149}$），可用非规格化数说明，否则用 0 说明。

2.2 基于查找表的多项式逼近

随着集成电路技术的不断发展，硬件更方便实施大规模查找表。但是，直接查找表算法仍然无法完成高精度的操作。因此，直接查找表方法必须与其他方法结合使用，以减少硬件资源的消耗。基于查找表的多项式近似方法[6]是更有效的算法之一，它不需要过多的查找表来浪费资源，也不需要与直接多项式近似值一样多的迭代。可以达到精度要求的操作数量，这种折衷方法取决于速度和面积。

根据加权查找表或加权运算，将其分为三种算法：基于操作的算法，基于查找表的算法和中间算法。主要的操作算法是使用加法和乘法的循环来完成多项式近似，并且速度仍然相对较慢。基于查找表的算法非常简单，通常只需要部分添加，包括部分数组方法和二进制表方法。但是，基于查找表的算法仍然无法完成高精度的操作，否则搜索表面积将变得非常大。最后的中间算法衡量查找表的大小和操作次数，从而在执行时间和硬件开销之间取得平衡。该算法根据不同的顺序可以分为分段线性近似和二次近似两种方法。由于过多的乘法和加法操作，高阶分段三次多项式近似算法并不常见。

下面介绍三种不同的算法，包括分段线性近似和二次近似算法。

2.2.1 分段线性逼近

分段线性逼近算法[7]的原理是将非线性函数的特征曲线按照一定的方法分成几个部分，并用一条直线近似地逼近各部分的特征曲线部分。该方法是直接查表法和简单多项式近似法的结合。分段线性逼近方法将低阶多项式与较小的查找表结合起来，具有较少的资源消耗和较快的速度。它已被广泛应用于数据压缩[9]，非线性模型到线性模型转换[10]和图形图像处理[11]等传感网络领域。它已成为计算资源有限条件下的超越功能的一个理想的选择。算法原理如图 2.2 所示。

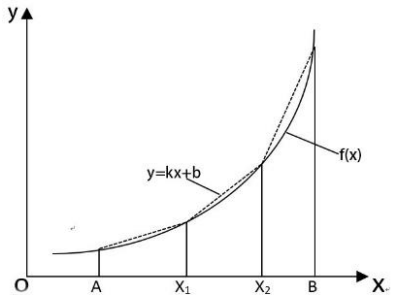

图 2.2 分段线性逼近示意图

分段线性逼近算法是根据一定的分段方法，如面积法、等分法等，将曲线 f(x) 所在的一段区间[A,B]划分为若干段，在每一段里面通过一个线性函数来进行逼近。如下式（2-5）。

$$y(x) = k * x + b \quad x \in [A, B] \tag{2-5}$$

其中， k 表示线性函数的斜率， b 表示线性函数的偏移量。其中，在每一段逼近区间[x_1,x_2]里面，区间的两个端点需要满足：

$$f(A)=y_A \ ; f(B)=y_B \ ; f(C)=y_C \tag{2-6}$$

此时，联立即可求解第 i 段逼近直线 $y(x)=k_i*x+b_i$ 中的 k_i 和 b_i 的值。

分段线性逼近算法最重要的部分是获得函数区间的分割，即分割点数，点数，分段大小选择。分割越多，间隔越小，精度越高，但是相应的占用查找表会更大并且消耗更多的资源;而较少分割的方法可以使用较小的查找表，但是可以获得逼近函数。错误会更大。不同的分割方法是在两者之间进行折衷以获得更好的结果。通过某种方法得到[A,B]上超越函数曲线的分割和各分段的近似系数 k_i，b_i 后，即可完成超越函数的分段线性逼近。

该算法主要分为三个步骤，具体如图 2.3 所示。

步骤一. 按照一定的方法划分分割间隔，得到所有的分割点;

步骤二. 在每个分段间隔中，根据两个端点的值查找线段 f(x)=ax+b，存储在查找表中以供使用;

步骤三. 使用硬件查找表，读取公式中每个片段的系数，并通过相乘和相加

来计算每个片段的系数。通过乘法和加法计算每段函数的近似值，从而实现超越函数的分段线性逼近。

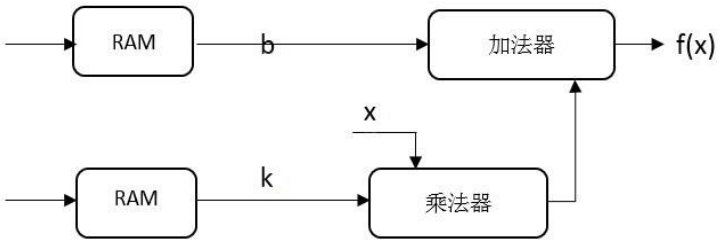

图 2.3 分段线性逼近法实现结构

2.2.2 二次逼近

与分段线性近似类似，分段二次近似算法[6]也是将区间分段后，对各区间用二次函数逼近。如图 2.4 所示。

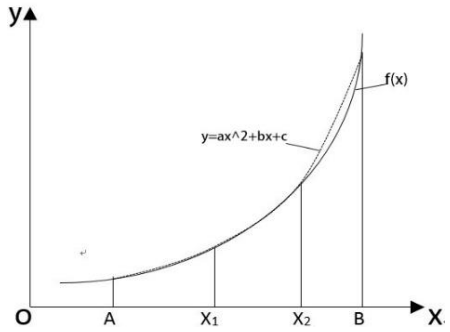

图 2.4 二次逼近示意图

常见的方法包括最大极小二次插值算法和线性规划的方法对参数进行约束，每段都用二次多项式近似，其基本形式为：

$$y = ax^2 + bx + c \quad x \in [A, B] \tag{2-7}$$

可以看出，二次逼近要完成一次平方运算，两次乘法运算和三次加法运算。假如有更快速的平方运算模块则可提高计算速度，又由于平方运算带来的资源和面积的消耗不可避免。所以综合上面我们把公式（2-7）还可变换为公式（2-8），这是通过乘法分配率得到：

$$y = (ax + b)x + c \quad x \in [A, B] \tag{2-8}$$

由此可见，公式（2-8）只需 2 次乘法累加运算，对乘法器加法器的要求较

高。这两个近似类似于分段线性近似算法，它们都被近似为近似区域中的几个近似段。在每一段中，近似由不同的线性方程或两个方程来执行。实现结构如图 2.5。

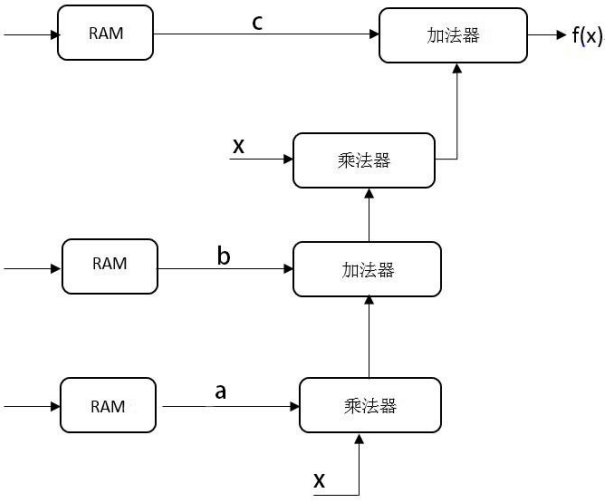

图 2.5 二次逼近实现结构

2.3 分段二次逼近算法的研究

2.3.1 最佳等距分段法

为了减少查找表，提高计算精度，节约资源，我们采用最佳等距分段法[8]来操作分段。分段思想是利用正、负误差带，同时满足等距分割的基本要求来进行分段。以下详细介绍基本原理。

如图 2.6 所示，f(x)为要逼近的函数曲线，若给定存在误差ε(x)，则从图中可看出

$$f_+(x)=f(x)+\varepsilon(x) \quad f_-(x)=f(x)-\varepsilon(x) \tag{2-9}$$

其中 f_+为给定误差的上限函数，f_-为给定误差的下限函数，x_1 为曲线斜率变化较大的一点，x_{11} 和 x_{22} 是以点（x_1,$f_-(x_1)$）做射线后与曲线相切的点。

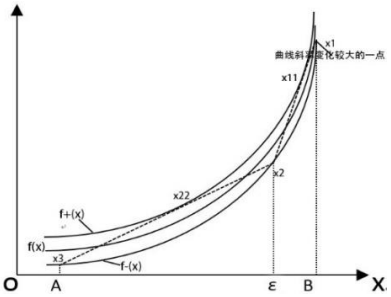

图 2.6 查找最小间距的原理图

在区间中以端点为起点，做以（x_{11},$f_+(x_{11})$）为切点的切线延长与 $f_-(x)$交与

（x_2,f-(x_2)）,再以（x_2,f-(x_2)）为起点，依次循环。显然可得：

$$\begin{cases} f_+(x_{11}) = \dot{f}_+(x_{11})(x_{11} - x_1) + f_-(x_1) \\ f_-(x_2) = \dot{f}_+(x_{11})(x_2 - x_{11}) + f_-(x_{11}) \end{cases} \tag{2-10}$$

式（2-10）中$\dot{f}_+(x_{11})$为 $f_+(x)$曲线在 x_{11} 的导数。

若令ε(x)=C（常数），亦可得以下式（2-11）

$$\begin{cases} f(x_{11}) - f(x_1) + 2C = \dot{f}(x_{11})(x_{11} - x_1) \\ f(x_{11}) - f(x_1) - 2C = \dot{f}(x_{11})(x_2 - x_{11}) \end{cases} \tag{2-11}$$

联立可求出 x_2.

以此类推可求出 x_3,x_4,x_5…,x_n。求出相邻两点的距离，选出最小的间距值，以该最小间距值为间距。从斜率变化最大的点开始进行等间距分段，求出新的区间端点值（逼近基点）

2.3.2 二次插值算法

二次插值法[12]即抛物线法，本文主要用其二次插值函数进行目标函数的逼近拟合。

由数值分析知，连接几个已知点所形成的曲线成为这些点的插值曲线，其对应函数成为插值函数。初始区间是由函数值所呈“两端大中间小”的相邻三个点所确定的闭区间，若这三点分别为 a，b 和 c，且 a<c<b，对应的函数值用 f_a，f_b，f_c 表示。记 x_1=a，x_2=b，x_3=c，y_1=f_a，y_2=f_b，y_3=f_c。于是在坐标平面得到（x_1，y_1），（x_2，y_2），（x_3，y_3）三个坐标点。过这三点可以画出一条二次曲线，设这个函数为

$$p(x)=a*x^2+b*x+c \tag{2-12}$$

将这三点及函数值分别带入式（2-12），得到以式（2-13）

$$\begin{cases} y_1 = ax_1^2 + bx_1 + c \\ y_2 = ax_2^2 + bx_2 + c \\ y_3 = ax_3^2 + bx_3 + c \end{cases} \tag{2-13}$$

联立求解可得系数 a，b 和 c，由于二次函数对称轴的横坐标（记为 x_p）为式（2-14）

$$x_p = -\frac{b}{2a} \tag{2-14}$$

综上所述

$$x_p = \frac{1}{2}\frac{(x_2^2-x_3^2)f_1+(x_3^2-x_1^2)f_2+(x_1^2-x_2^2)f_3}{(x_2-x_3)f_1+(x_3-x_1)f_2+(x_1-x_2)f_3} \tag{2-15}$$

由上式（2-15）求出的 x_p 为插值函数的极小点，也是原目标函数的近似极小点。以此点作为下一次缩小区间的中间插入点，如图 2.7 所示。

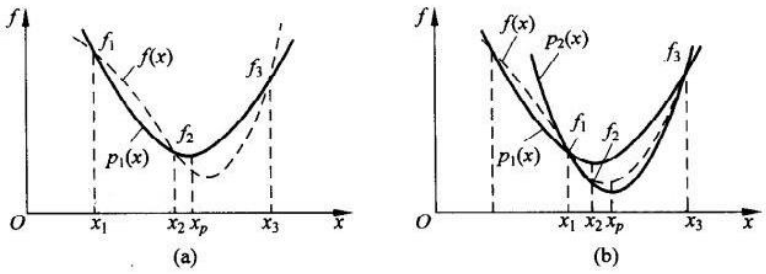

图 2.7 二次插值法的逼近过程

二次插值法以两个中间插入点间的距离足够小作为收敛原则，即当式（2-16）

$$\left|x_2 - x_p\right| \le \varepsilon \qquad (2\text{-}16)$$

成立时，x_p 与 x_2 中的较小者作为极小点

二次插值法的程序框图如图 2.8 所示。

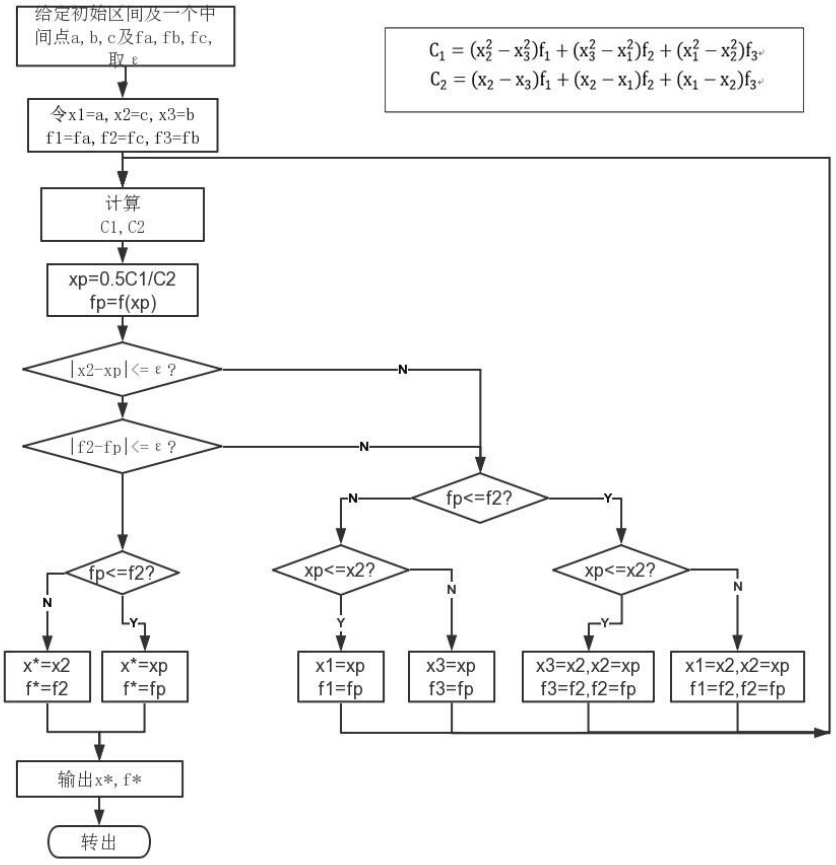

图 2.8 二次插值法的流程框图

2.4 算法实现结果对比

为了突出本文所述算法的优越性，我们可以做大量的分析对比。假设以$y = e^x$为例，对其使用不同的算法逼近，分别计算其误差，并做分析比对。

我们分别用 y=kx+b 线性逼近，$y = ax^2 + bx + c$二次逼近及泰勒展开分别对$y = e^x$函数逼近，如图 2.9 所示。

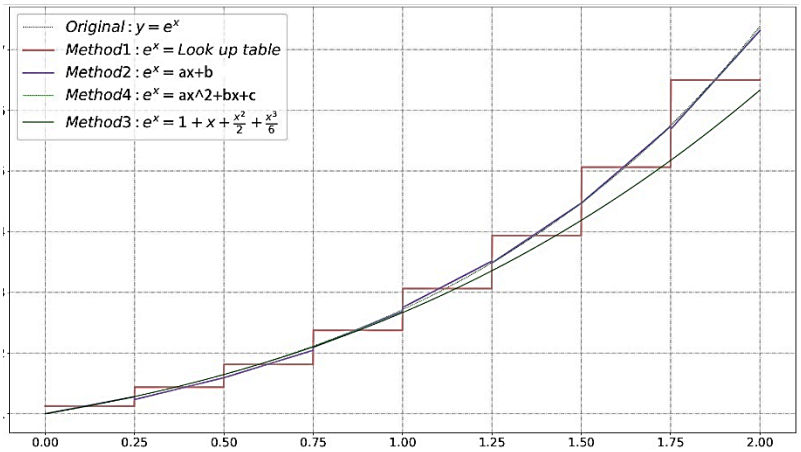

图 2.9 不同算法实现e^x对比图

可以明显看出，泰勒展开的误差明显比其他两个大很多，所以泰勒展开算法是不能应用的。我们对也上面的图进行量化（在其他条件相同情况下），如表 2.1

表 2.1 不同算法实现e^x分段逼近结果图

算法	精度
线性逼近	0.07
二次逼近	1.2e-5
泰勒展开	0.3

由图 2.9 与表 2.1 的实验结果可以得出结论：

1）与其他的算法相比，本文的算法的实现效率更高。另外，在保证一定精度的前提下，本文的实现方法更加节省硬件资源；

2）对比表 2.1 可以看出：与的逼近结果相比，分段数目相同的情况下， 本文的算法可以大大降低逼近误差。

2.5 算法的 Matlab 仿真

为了验证分段二次逼近算法的逼近效率，本文使用指数函数来检验在（0，1）范围内的各区间二次曲线的系数近似值，并采用 MATLAB 仿真软件模拟误差区间。如表 2.2。

表 2.2 指数函数各区间系数

$f(x)=e^x$	a	b	c
$[0,\frac{2}{2^6})$	$\frac{91}{64}$	$\frac{1}{2^{13}}$	0
$[\frac{2}{2^6},\frac{4}{2^6})$	$\frac{11}{8}$	$\frac{3}{2^{11}}$	$\frac{7}{2^{14}}$
$[,\frac{4}{2^6},\frac{8}{2^6})$	$\frac{169}{128}$	$\frac{43}{2^{13}}$	$\frac{27}{2^{14}}$
$[\frac{8}{2^6},\frac{12}{2^6})$	$\frac{5}{4}$	$\frac{57}{2^{12}}$	$\frac{47}{2^{13}}$
$[\frac{12}{2^6},\frac{16}{2^6})$	$\frac{19}{16}$	$\frac{13}{2^9}$	$\frac{109}{2^{13}}$
$[\frac{16}{2^6},\frac{20}{2^6})$	$\frac{9}{8}$	$\frac{335}{2^{13}}$	$\frac{26}{2^{10}}$
$[\frac{20}{2^6},\frac{24}{2^6})$	$\frac{69}{64}$	$\frac{227}{2^{12}}$	$\frac{46}{2^{10}}$
$[\frac{24}{2^6},\frac{28}{2^6})$	$\frac{33}{32}$	$\frac{149}{2^{11}}$	$\frac{89}{2^{10}}$
$[\frac{28}{2^6},\frac{32}{2^6})$	$\frac{63}{64}$	$\frac{381}{2^{12}}$	$\frac{123}{2^{10}}$
$[\frac{32}{2^6},\frac{36}{2^6})$	$\frac{15}{16}$	$\frac{447}{2^{12}}$	$\frac{167}{2^{10}}$
$[\frac{36}{2^6},\frac{40}{2^6})$	$\frac{29}{32}$	—	$\frac{244}{2^{10}}$
$[\frac{40}{2^6},\frac{44}{2^6})$	$\frac{223}{256}$	—	$\frac{281}{2^{10}}$
$[\frac{44}{2^6},\frac{48}{2^6})$	$\frac{27}{32}$	$\frac{179}{2^{10}}$	—
$[\frac{48}{2^6},\frac{54}{2^6})$	$\frac{103}{128}$	$\frac{209}{2^6}$	—

通过在 matlab 来进行测试，计算分段结果的绝对误差，得到图 2.10

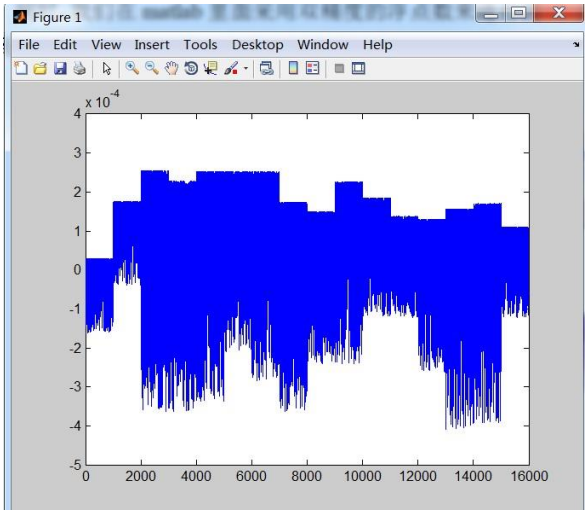

图 2.10 二次逼近算法误差

分段区间划分为 16 段，分段二次逼近算法最终逼近误差范围为 [-0.060%,0.046%]。

实验结果表明：分段二次逼近算法可以实现一个在不同取值范围内分段逼近区间的较优逼近，能够有效的降低分段区间的数量，提高精度，减少查找表资源消耗。

2.6 基于 Python 语言验证 softmax 函数单元

softmax 函数是将任意 n 维的实值向量转换为取值范围在(0,1)之间的 n 维实值向量，并且总和为 1。如：向量 softmax[1.0, 2.0, 3.0]转换为[0.09003057, 0.24472847, 0.66524096]。

其有三个性质：

1) softmax 是单调递增函数；

2) 将原始输入映射到(0,1)区间，并且总和总为 1，常用于概率计算；

3) softmax(x) = softmax(x+c)，这个性质用于保证数值的稳定性。

经过系统的运算得到图 2.11 的数据，在表 2.3 中详细列出。

```
Type "help", "copyright", "credits" or "license" for more information.

Imported NumPy 1.8.1, SciPy 0.14.0, Matplotlib 1.3.1
+ guidata 1.6.1, guiqwt 2.3.2
Type "scientific" for more details.
>>> runfile('C:/Users/Administrator/.spyder2/.temp.py', wdir=r'C:/Users/Administrator/.spyd
er2')
[ 0.00534383  0.0048353   0.00721342  0.01189292  0.01960811  0.0323283
  0.05330036  0.08787743  0.14488539  0.23887562  0.39383932]
>>>
```

图 2.11 输出的数据

表 2.3 输入与输出数据

输入	输出
0.1	0.0048353
0.2	0.00534383
0.5	0.00721342
1.0	0.01189292
1.5	0.01960811
2.0	0.0323283
2.5	0.05330036
3.0	0.08787743
3.5	0.14488539
4.0	0.23887562
4.5	0.39383932

实验结果是 10 分类最终得到的 softamx 的数据，也为下一章的硬件实现对比做好准备。

2.7 本章小结

本章分别介绍了分段二次逼近算法的具体实现和基于 python 语言完成 softmax 函数的设计。首先，该算法在降低查找表资源消耗[16]的同时，误差明显较小，我们通过 MATLAB 对其进行了仿真，证实了实验结果；然后，基于 python 语言提供的标准求解函数，对本文设计的 softmax 的正确性进行了验证。

第三章　softmax 单元的硬件实现

3.1 系统整体介绍

根据前面的研究，及参照文献[20]，要使用硬件去实现 softmax 函数首先必须硬件实现 exp(x)。实现 exp(x)的设计可以分为五个阶段：

1) 参数储存阶段

2) 寻址阶段

3) 查找表找出系数的值阶段

4) 得到 y 的值，

5) 最后计算出最终结果。

在硬件实现之前，我们做了很多关于阈值和精度对此算法的实验，发现把 softmax 的输入压缩到一定的阈值，比如(-10,10)，这样对误差的影响很小，所以本文的数据均在（-10，10）区间内。另外，硬件结构计算使用的是十六位浮点型数据，做乘除法的时候只需要调用 IP 核。

设计是采用自顶向下的方式来实现，首先定义出整个系统所要实现的功能，然后划分各级流水线，确定每一级流水线要执行的功能，将这些流水线级化为一个个功能模块，在功能模块中再根据细致的任务划分将其分为几个子模块，如乘法器、加法器等。

系统设计主要考虑 RAM 的存储，查找表取值，数学运算的实现等问题。基于这些问题，设计系统时一共划分为参数储存模块、取值模块、运算模块。系统模块结构如图 3.1 所示。

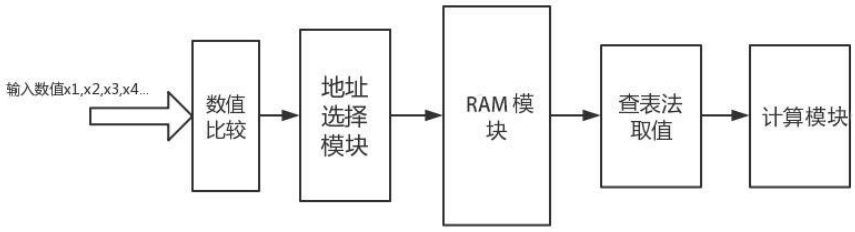

图 3.1 系统模块结构图

3.2 参数存储模块

此部分是把前文中所求的各系数存入 RAM 中，然后通过输入数据查找表求出各系数值。

RAM 模块需要一个写控制信号 wr[3:0]和时钟信号 clk（下降沿触发）以及要写入的数据输入端口 data_in[31:0]。除此之外还需要操作存储器的地址输入端口 adrr[5:0]和数据输出端口 data_out[31:0]。

具体电路如图 3.2 所示：

图 3.2 RAM 模块端口图

wr 端口接上 control 信号，用来控制写空能。每个阶段地址选择器会选择出对应的地址，统一赋给 select_q_16[5:0]和 select_q_4[5:0]两个信号中。则只要把所有存储器的 adrr 端口和这两个信号关联即可。

3.3 寻址模块

下面是此模块的总体结构图，如图 3.3。

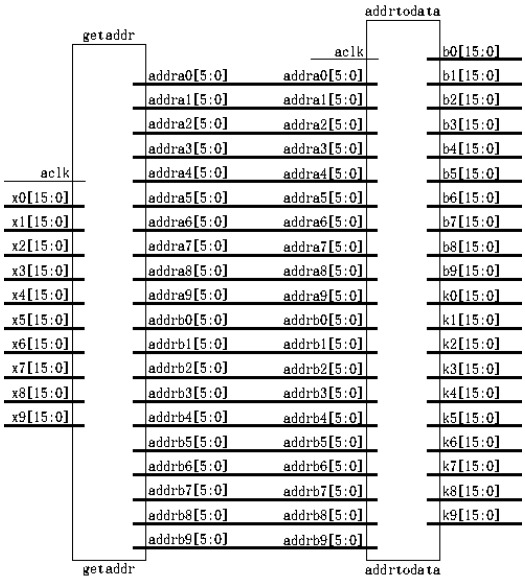

图 3.3 寻址模块结构图

这部分是把输入的 x0 到 x9 进行大小范围的判定，找到系数相应的地址，然

后查找表的方式得出系数的具体值以计算出 y 得值。

具体的电路结构图如下图 3.4。

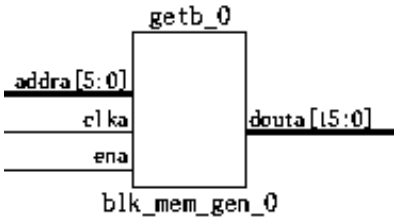

图 3.4 得出系数值的电路结构

端口定义如表 3.1 所示：

表 3.1 寻址模块端口定义

信号名	I/O	位宽	描述
clka	I	1	系统时钟
addra	I	5	输入
ena	I	1	输入
douta	O	15	输出信号控制器模块计数结果

3.4 计算模块

运算主要包括浮点数乘法，浮点数加法和 softmax 函数（除法）的运算。

下面是此模块的总体结构图，如图 3.5。

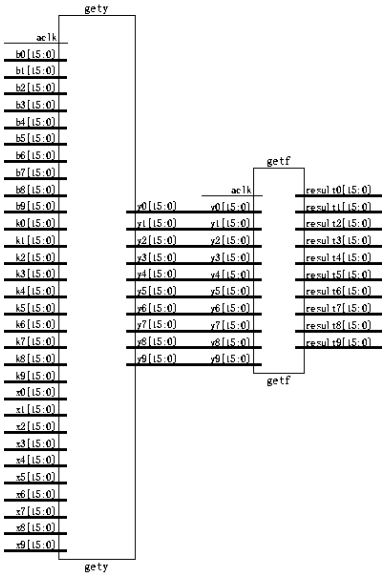

图 3.5 计算模块总体结构图

主要分为以下几个部分实现：

1） 根据二次逼近的原理，$y = ax^2 + bx + c$改为$y = (ax + b)x + c$，使用两次乘法器和一次加法器来计算出 y 的值。电路图 4.6，结构图 4.7。

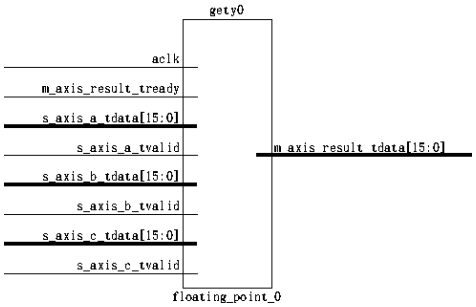

图 3.6 计算 y 值的电路

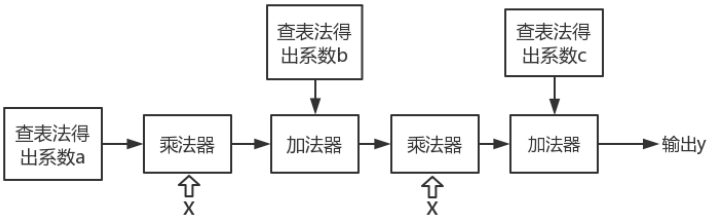

图 3.7 计算 y 值的电路结构图

端口定义如下表 3.2 所示

表 3.2 计算 y 值的端口定义

信号名	I/O	位宽	描述
clk	I	1	系统时钟
s_axis_a_date	I	15	系数 a 输入
s_axis_b_date	I	15	系数 b 输入
s_axis_c_date	I	15	系数 c 输入
m_axis_result_date	O	15	输出 y 的值

2） 为了减小误差，在计算$\mathrm{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$的分母时，对这几个数进行了迭代累加，如式（3-1）为例：

$$\begin{cases} f0 = y0 + y1 \\ f1 = y2 + y3 \\ f2 = y4 + y5 \\ f3 = y6 + y7 \\ f4 = y8 + y9 \end{cases} \rightarrow \begin{cases} f5 = f0 + f1 \\ f6 = f2 + f3 \end{cases} \rightarrow f7=f5+f6 \rightarrow f=f4+f7 \qquad (3\text{-}1)$$

以此得出 softmax 函数的分母。电路图 3.8 所示。

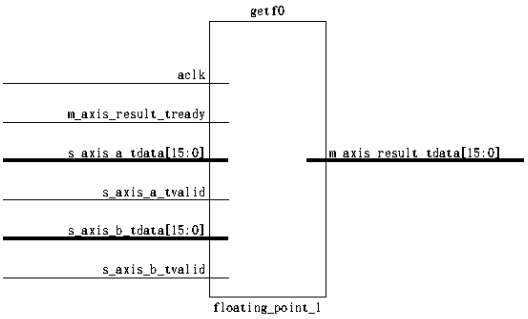

图 3.8 计算 f 值的电路结构图

端口定义如下表 3.3 所示

表 3.3 计算 f 值的端口定义

信号名	I/O	位宽	描述
clk	I	1	系统时钟
m_axis_result_tready	I	15	输入 y
s_axis_a_date	I	15	输入
s_axis_b_date	I	15	输入
m_axis_result_date	O	15	输出 f 的值

3) 此时就可以得出当 x 输入某个数时所要求的 softmax 函数得出的值，即做一次除法，如下式（3-2），电路图 3.9 所示，结构图 3.10。

$$x0 \rightarrow \frac{yo}{f} \qquad (3\text{-}2)$$

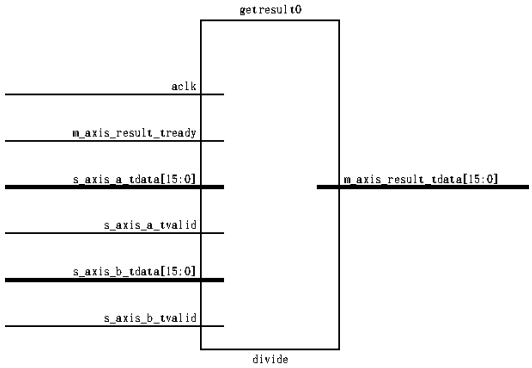

图 3.9 计算 softmax 函数的电路

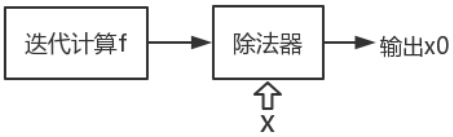

图 3.10 计算 softmax 函数的电路结构图

端口定义如下表 3.4 所示

表 3.4 计算 softmax 函数值的端口定义

信号名	I/O	位宽	描述
clk	I	1	系统时钟
m_axis_result_tready	I	15	输入 y
s_axis_a_tdate	I	15	输入 f
s_axis_b_tdate	I	15	
m_axis_result_date	O	15	输出最终结果

3.5 本章小结

本章主要详细讲述了 softmax 单元的硬件实现。主要分成总体设计和详细设计两大部分。完成了 softmax 函数的硬件实现和系统各个模块的硬件设计。为下一章的实验验证提供测试基础。

第四章　设计验证和误差分析

4.1 系统模块验证

我们把每个单独的任务 module 都生成一个 HDL 文件。对于每个独立的功能模块我们都需要对其进行一次单独的仿真，编写相应的 Testbench，对每个模块的功能进行验证。

以下各模块均采用 vivado2015.4 进行模拟仿真。

4.1.1 寻址模块验证

vivado 运行结果如图 4.1 所示。

图 4.1 是第一个阶段的地址选择结果图。由图可见，罗列了输入的 10 个 x 的数据，并在时钟脉冲下得到系数的地址。这一阶段结束后，将地址给下个阶段取值模块进行查找表得出系数的值。

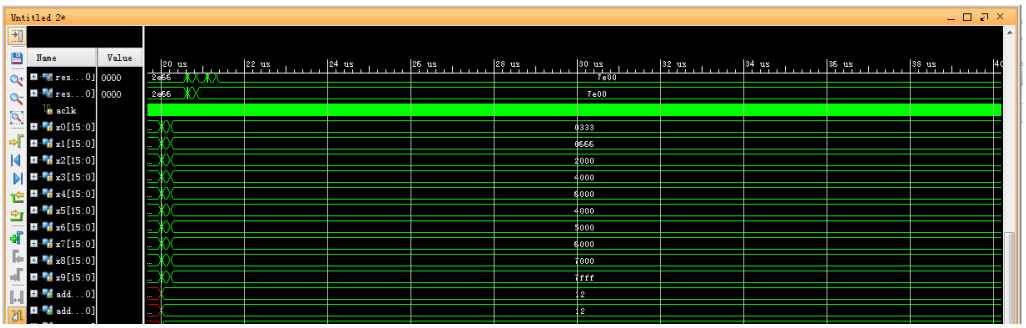

图 4.1 寻址模块仿真结果图

4.1.2 取值模块验证

vivado 运行结果如图 4.2 所示。

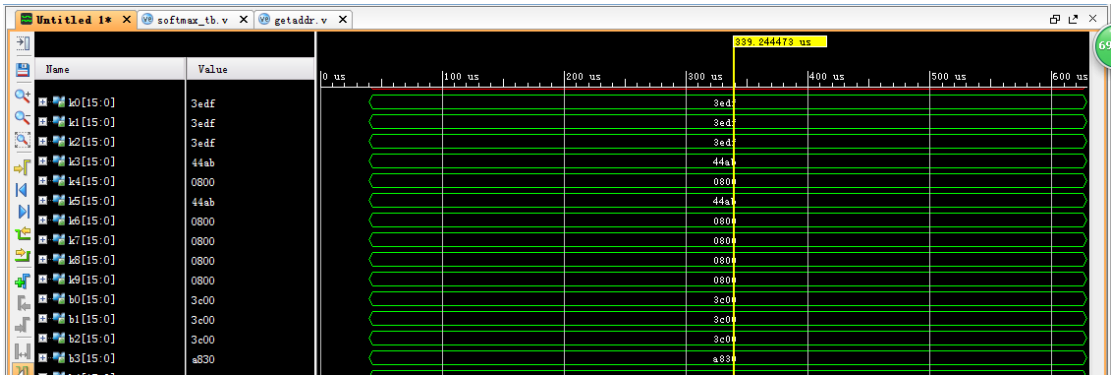

图 4.2 取值模块仿真结果图

在输入 10 组数时，通过查找表取出各系数，如图中的系数

a1,a2,a3…,b1,b2,b3…,c1,c2,c3…

4.1.3 计算 y 值模块验证

vivado 运行结果如图 4.3 所示。

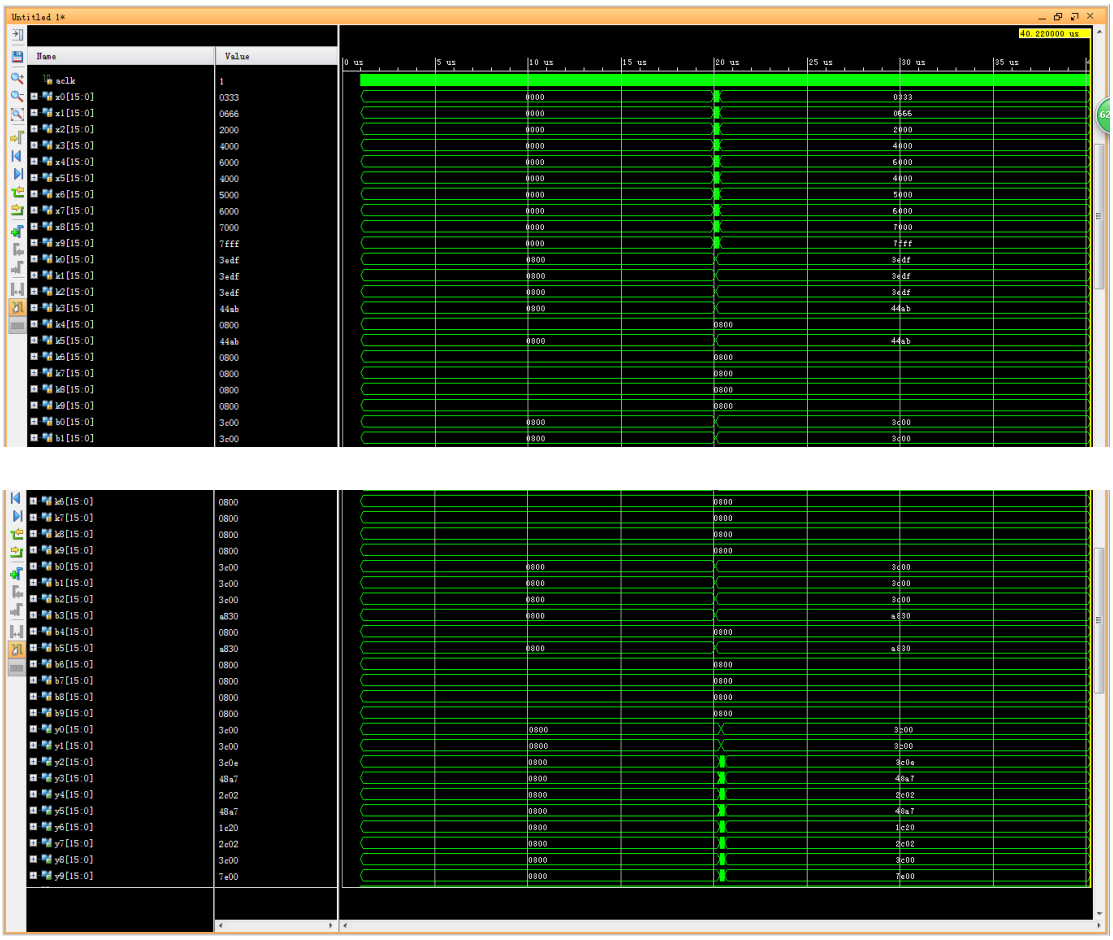

图 4.3 计算 y 值模块仿真结果图

结果看到的数据是十六进制，经过转换为十进制，如表 4.1。

表 4.1 y 值数据

序号	输入 x	输出 y
0	0.1	15360
1	0.2	15360
2	0.5	15374
3	1.0	18599
4	1.5	11266

5	2.0	18599
6	2.5	7200
7	3.0	11266
8	3.5	15360
9	4.0	32256
10	4.5	19565

4.1.4 计算最终函数值模块验证

vivado 运行结果如图 4.4 所示。

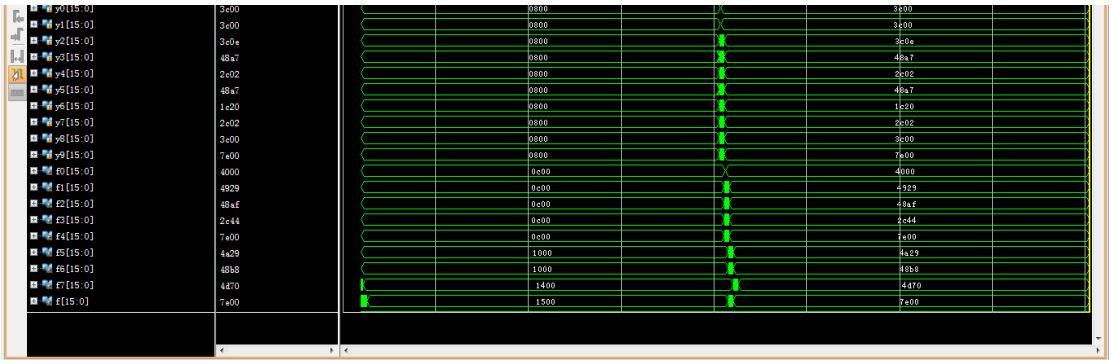

图 4.4 计算最终函数值仿真结果图

对所得结果进行处理后，最终得出如下数据，如表 4.2 所示。

表 4.2 输出 f(x)最终结果

序号	输入 x	输出 f(x)
0	0.1	0.0048
1	0.2	0.0053
2	0.5	0.0072
3	1.0	0.0119
4	1.5	0.0196
5	2.0	0.0323
6	2.5	0.0533
7	3.0	0.0879

8	3.5	0.1449
9	4.0	0.2389
10	4.5	0.3938

4.2 系统整体验证

因为系统设计过于复杂，并且很多地方没有考虑到资源的有限性，所以不能上板子验证系统设计合理性。但是，可以通过软件进行验证系统逻辑合理性。具体方案就是利用 c 语言程序读取记录着输出层存储器数值的文本。通过和软件所得的数据做对比，验证系统合理性。

由上文中已经介绍过，精度对神经网络训练精度的实验缩小 softmax 的输入压缩到一定的阈值，比如(-10,10)发现对训练正确率的影响很小，另外，硬件结构计算使用的是十六位浮点型数据，做乘除法的时候只需要调用 IP 核，这会造成一定的误差，所以，还需对其误差做分析。

系统整体的 vivado 仿真结果如图 4.5 所示。

图 4.5 总体仿真结果图

4.3 误差分析

由上文中已经介绍过，通过缩小 softmax 的输入，并压缩到一定的阈值，比如(-10,10)，发现对误差的影响很小，我们对其做误差分析。

用 matlab 软件对用软件所得数据和硬件所得数据做对比分析。图 4.6 是误差对比图，为了更直观的表现其误差，可对其放大，图 4.7 是误差的放大图。

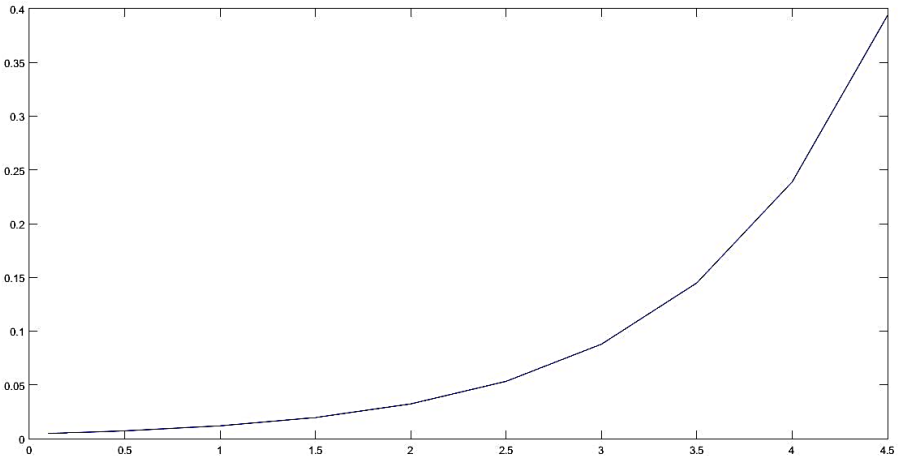

图 4.6 误差对比图

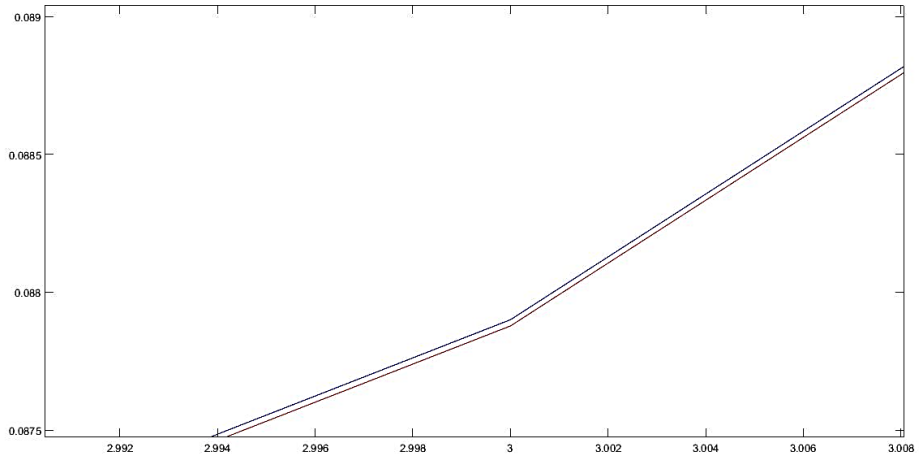

图 4.7 误差对比放大图

对于输入的每组数据都需要选定一定的范围， 除过正切函数，每种运算都在(-10,10)之间，每组向量的取值范围都在(-10,10)之间，具有普遍性。图 4.8 是 matlab 计算的误差，其中，平均误差为 2.3460e-05，最大误差 4.3830e-05。

图 4.8 matlab 计算的误差

上图中已给出最大误差以及平均误差，最大误差为绝对误差，而平均误差指相对误差。绝对误差越小，表示计算值与真值越接近，准确度也越高。相对误差指 “计算的绝对误差和真值之比”，相对误差越小，表示计算越精确。因此平均误差高的代表测量结果偏离真值较大。

4.4 本章小结

本章主要对各个模块进行验证，并对最终所得数据进行误差分析。

结束语

本文的研究工作主要集中在 softmax 函数算法的研究和硬件实现上。对国内外的研究动态和超越函数进行了研究。对算法进行分析和说明，最后选择分段近似算法的实现算法。在对算法进行连续性研究的基础上，提出了超越函数计算的分段二次逼近算法的存在性。提出了一种分段二次逼近方法，以减小误差，解决资源浪费问题。

本文通过对区间进行划分，采用二次插值算法，并通过 MATLAB 仿真验证了该算法的有效性。在此基础上，对各个功能模块进行了划分，确定了整个算法的设计流程。并最终模拟仿真实现。

本文所采用的超越函数实现算法为分段二次逼近超越函数算法， 其每段分段区间采用的是二次函数方程进行逼近，其虽然实现简单，但是逼近精度却不是很高，未来有机会可以采取更高阶的多项式拟合与查找表结合算法，本文仅仅是一次学习与尝试。超越函数的研究一直是业界的一个研究热点，其与数学领域一直存在较深的联系。未来选取更优的分段方法、更高阶的多项式，甚至辅以更优的查找表计算方式，超越函数必将获得更快的计算方法，满足人们更多的需求。

致谢

首先，衷心感谢指导我完成毕业设计的导师杜慧敏教授和常立博老师。在完成毕业设计这段时间，常立博老师引导着我在正确的方向上逐渐地靠近研究最终所要达到的结果，以严厉、严谨的治学态度时时鞭策于我，也许目前我还无法做到这样严谨的治学态度，但这至少会成为我今后学习生活的一个目标，激励着我逐渐前进。同时也要感谢在毕业设计完成过程中，帮助我们解决各种难题，并为我们提供各种参考资料的杜教授。杜教授教学严格，认真负责，让我受益匪浅。值此论文完成之际，我要向杜教授和常立博老师表示最诚挚的感谢。谢谢！

其次，感谢我的家人，感谢他们多年来各个方面的照顾，特别是对我个人的信任，对我学业的支持。感谢你们！

然后，在同学们的帮助下，当我的毕业设计遇到困难时，你的帮助给了我很多指导和启发，还有很多兄弟姐妹，谢谢你们对我的支持和建议。我还要感谢我的母校和学校领导为学校提供的培养、交流平台和学习环境，使我能在四年里收获很多，有很多的理解，这样我就可以顺利毕业后今天顺利完成。学习四年。谢谢您。

最后，感谢各位评审老师在百忙之中抽出时间来审阅我的论文，我深感荣幸；同时感谢所有参加了我的毕业答辩评审的老师和同学，谢谢！

参考文献

[1] 吴岸城. 神经网络与深度学习[M]. 电子工业出版社. 2016

[2] Samir Palnitkar. Verilog HDL 数字设计与综合（第二版）[M].夏宇闻.等译.电子工业出版社，2012.

[3] 朱梦尧, 高留闯, 郑佳,等. 基于对数分段线性逼近的图形处理方法: CN, CN 103268307 A[P]. 2013.

[4] 牛涛. 初等函数运算器的设计研究[D]. 上海：浙江大学， 2013.

[5] Nicolas Brisebarre,Sylvain ChevillardAn Efficient Method for Evaluating Polynomial and Rational Function. Application-Specific Systems, Architectures and Processors,July.2008: 233-238

[6] 牛涛,沈海斌. 基于分段二次插值的初等函数逼近低成本设计[J]. 计算机工程,39(8):285-291.

[7] Veidinger L. On the numerical determination of the best approximations in the Chebyshev sense[J]. Numerische Mathematik, 1960, 2(1):99-105.

[8] 王少军，张启荣，彭宇， 彭喜元. 超越函数 FPGA 计算的最佳等距分段线性逼近方法[J]. 仪器仪表学报， 2014,35(6):1210-1216.

[9] 张建明，林亚平，傅明等. 传感网络中误差有界的分段逼近数据压缩算法[J]. 软件学报，2011,22(9):2149−2165.

[10] 江钊. 分段线性逼近法在梯级水电站优化调度中的应用[D]. 华中科技大学：水利工程,2012:4-5.

[11] Byeong-GyuNam, Hyejung Kim, Hoi-Jun Yoo. Powerand Area-Efficient Unified Computationof Vector and Elementary Functionsfor Handheld 3D Graphics Systems [J].IEEE transactions on computers,April2008,vol.57 No.4:490-504.

[12] 李俊文.基于 MATLAB 的二次插值法的优化设计[J].广东技术师范学院学报，2013,7

[13] Chaitra.P. Hardware implementation of Artificial Neural Networks using Back Propagation Algorithm on FPGA [C]. University of Mosul, College of Engineering, Mosul, Iraq, August 2015.

[14] James G.Eldredge. A Hardware Implementation of the Backpropagation Algorithm Using Reconfigurable FPGAs [C].Brighan Young University Provo,UT 84602, Sep 2014.

[15] 钱玉多. 基于 FPGA 的神经网络硬件实现研究[D].华中科技大学,2012.

[16] 牛涛,沈海斌. 基于分段二次插值的初等函数逼近低成本设计[J]. 计算机工程,39(8):285-291.

[17] 李炜，黄心汉. 一种改进的模糊似然函数算法[J]. 信号处理,2005,3

[18] 杜慧敏,黄小康,田征. 改进的超越函数分段线性逼近方法[M]. 计算机应用,2016,7

[19] 江钊. 分段线性逼近法在梯级水电站优化调度中的应用[D]. 华中科技大学：水利工程，2012:4-5

[20] 杜慧敏, 沙亮, 张彦芳,等. 浮点超越函数设计与实现 [J]. 西安邮电大学学报, 2015, 20(2):16-20.

[21] Rafid Ahmed Khalil. Hardware Implementation of Backpropagation Neural Networks on Field programmable Gate Array (FPGA) [C]. International Journal of Computer Applications (0975 – 8887), Volume 52– No.6, Sep. 2007.

[22] S. L. Pinjare. Implementation of Neural Network Back Propagation Training Algorithm on FPGA [C]. International Journal of Computer Applications (0975 – 8887), Volume 52– No.6, August 2015.

[23] 林钰凯. 高性能并行乘法器关键技术研究[D]. 西安电子科技大学, 2010.

[24] 张航,陈向东. 基于 FPGA 的电子鼻中 Sigmoid 函数分区间线性逼近实现[J]. 计算机应用,2014,(S2):352-356.