一种基于差分的改进STAM算法

李志祥 王寅龙 李 婷 王希武 林克成
(军械工程学院计算机工程系 石家庄 050003)

摘 要 STAM算法是硬件实现非线性函数的一种有效方法,它用查找表和加法运算实现了精确的非线性函数近似,并利用函数的对称性极大地缩减了查找表的尺寸,但算法中第一项系数占用的查找表仍然较大。针对此问题,提出了一种基于差分的改进方法,并在FPGA上实现了双曲正切S形函数。实验结果显示,算法经过改进可以缩减17%~30%的存储空间。

关键词 STAM算法,非线性函数近似,双曲正切,S形函数,FPGA
中图法分类号 TP346 **文献标识码** A

Improved STAM Algorithm Based on Differential Method

LI Zhi-xiang WANG Yin-long LI Ting WANG Xi-wu LIN Ke-cheng
(Department of Computer Engineering, Ordnance Engineering College, Shijiazhuang 050003, China)

Abstract STAM algorithm is an efficient approach to implement non-linear function approximation with hardware, in which lookup tables and addition are involved to approximate non-linear function accurately, and size of table could be reduced remarkably due to symmetry of function. Nevertheless the first coefficient still uses relatively big table. An improved method based on differential was brought up aiming at relieve the problem, and with the algorithm hyperbolic tangent sigmoid function was implemented on FPGA. Experimental results show that 17% to 30% of memory was saved after improvement.

Keywords STAM algorithm, Non-linear function approximation, Hyperbolic tangent, Sigmoid function, FPGA

1 引言

实时信号处理、人工神经网络、图形图像处理等领域常需要高速的非线性函数计算[1-3]。随着超大规模集成电路技术的发展,非线性函数的硬件实现与传统基于软件的插值逼近方法相比更具速度和精度的优势。围绕非线性函数的硬件实现方法研究很多,如查找表[4]、分段线性近似[2]、CORDIC(Coordinate Rotation Digital Computer)[5,6]等,但其硬件实现各有缺点,如查找表方法会消耗大量的硬件存储资源[7];分段线性近似算法虽极大地减少了查找表的大小,但其精度难以提高[8];而CORDIC方法又存在计算速度慢的问题。

1999年,Michael J. Schulte和James E. Stine提出了一种STAM(Symmetric Table Lookup Addition Method)算法[9],其对非线性函数进行泰勒级数展开后保留前二项,用查找表和加法运算实现了精确的非线性函数近似,并利用函数的对称性极大地缩减了查找表的尺寸。但STAM算法中第一项系数占用的查找表仍然较大,本文针对此问题提出了一种基于差分的改进方法,并在FPGA上实现了人工神经网络中一种常用的激活函数——双曲正切函数。实验结果显示,算法经过改进可以进一步缩减17%~30%的存储空间。

2 STAM算法

为实现对非线性函数 $f(x)$ 的近似计算,首先把 n 位输入 x 分为 $m+1$ 个部分 $x_0, x_1, \cdots, x_m$,其长度分别为 $n_0, n_1, \cdots, n_m$,如图1所示。

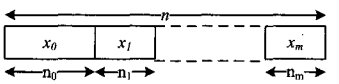

图1 将 n 位输入划分为 $m+1$ 个部分

STAM算法的基本思想是将 $f(x)$ 近似为若干项之和,其中每一项只与 x_0 和 x_i 相关,即:

$$f(x)\approx\sum_{i=1}^{m}a_{i-1}(x_0,x_i) \tag{1}$$

为简化表达式,定义 $x_{j:k}=\sum_{i=j}^{k}x_i$, $n_{j:k}=\sum_{i=j}^{k}n_i$,如 $x_{2:4}=x_2+x_3+x_4$,假定 $0\leqslant x\leqslant 1-2^{-n}$,那么:

$$\begin{aligned}&0\leqslant x_0\leqslant 1-2^{-n_0}\\&0\leqslant x_i\leqslant 2^{-n_{0:(i-1)}}-2^{-n_{0:i}}\ (1\leqslant i\leqslant m)\end{aligned} \tag{2}$$

定义 δ_i 为 x_i 的最大值和最小值的中间值,即:

$$\delta_i=2^{-n_{0:i-1}-1}-2^{-n_{0:i}-1}\ (1\leqslant i\leqslant m) \tag{3}$$

$f(x)$ 在 $x_{2:m}=\delta_{2:m}$ 点进行二阶泰勒级数展开为:

到稿日期:2012-02-23 返修日期:2012-06-01

李志祥(1981-),男,硕士,讲师,主要研究方向为人工神经网络及数字系统设计,E-mail:zhxlii@gmail.com;王寅龙(1976-),男,硕士,讲师,主要研究方向为数字图像处理技术;李 婷(1983-),女,硕士,讲师,主要研究方向为数字图像处理技术;王希武(1966-),男,副教授,主要研究方向为神经网络及战场态势感知;林克成(1971-),男,讲师,主要研究方向为信息安全及数字系统设计。

$$f(x)\approx f(x_0+x_1+\delta_{2,m})+f'(x_0+x_1+\delta_{2,m})(x_{2,m}-\delta_{2,m}) \tag{4}$$

式中，$\delta_{2,m}=\sum_{i=2}^{m}\delta_i=2^{-n_0-n_1-1}-2^{-n-1}$ 为使第二项与 x_1 无关，用 δ_1 代替第二项中的 x_1，并将 $x_{2,m}$ 和 $\delta_{2,m}$ 展开：

$$f(x)\approx f(x_0+x_1+\delta_{2,m})+\sum_{i=2}^{m}f'(x_0+\delta_1+\delta_{2,m})(x_i-\delta_i) \tag{5}$$

与等式(1)比较，就可以确定系数为：

$$\begin{aligned}a_0(x_0+x_1)&=f(x_0+x_1+\delta_{2,m})\\ a_{i-1}(x_0+x_i)&=f'(x_0+\delta_1+\delta_{2,m})(x_i-\delta_i)(2\leqslant i\leqslant m)\end{aligned} \tag{6}$$

这样，$f(x)$ 的近似问题就转变为 m 个查表运算和 1 个加法运算，即把 x_0+x_1 作为第 1 个查找表的输入得到 $a_0(x_0,x_1)$，把 x_0 和 x_i 作为第 i 个查找表的输入得到 $a_i(x_0,x_i)$，最后把 m 个值相加，就得到 $f(x)$ 的值。

由于算法中第二个及以后的系数远小于第一个系数的值，因此算法可以有效缩减存储空间。对于第 i 个系数 $a_{i-1}(x_0,x_i)(i\geqslant 2)$，$|x_i-\delta_i|\leqslant 2^{-n_{0,i-1}-1}-2^{-n_{0,i}-1}<2^{-n_{0,i-1}-1}$，其大小就被限定为：

$$|a_{i-1}(x_0,x_i)|<|f'(\xi_1)|2^{-n_{0,i-1}-1} \tag{7}$$

式中，ξ_1 是位于[0,1]之间的点，在这一点使 $f(x)$ 的 i 阶导数的绝对值 $|f^i(x)|$ 取得最大值，a_{i-1} 相对于 a_0 归一化后取对数，得：

$$\log_2|a_{i-1}(x_0,x_i)/f(\xi_0)|<-n_{0,i-1}-1-\log_2(|f(\xi_0)/f'(\xi_1)|) \tag{8}$$

即 a_{i-1} 有 $n_{0,i-1}+1+\log_2(|f(\xi_0)/f'(\xi_1)|)$ 位的值为 0（如果 $a_{i-1}<0$，值为 1），这些 0 或者 1 是不需要存储的，在执行加法之前，可以通过对 a_{i-1} 的特定位的符号扩展得到。

另外，利用系数 $a_{i-1}(x_0,x_i)$ 对于 x_i 的对称性可进一步减小存储空间。当 $2\delta_i-x_i$ 为 x_i 的补码时，$a_{i-1}(x_0,2\delta_i-x_i)$ 也成为 $a_{i-1}(x_0,x_i)$ 的补码，即 $a_{i-1}(x_0,x_i)=-a_{i-1}(x_0,2\delta_2-x_2)$。具体实现时检测 x_i 的最高位，将 x_i 的其余各位与该最高位异或后作为查找表的输入，查找表的输出各位与该最高位相异或后再进行加法运算(见 4.2 节图 2)。第 2 至 m 个查找表分别缩减 50%。

由于只保留泰勒级数前 2 项、在第二项系数中用 δ_i 代替 x_i、用有限精度的位数存储系数等原因，STAM 算法存在一定的近似误差。为了保证算法结果可靠，需将误差控制在精度要求范围内。经分析，总误差需满足如下条件：

$$|f''(\xi_2)|2^{-2n_0-n_1-2}(1+2^{-n_1})+(m-1)\times 2^{-p_f-g-1}+2^{-p_f-1}<2^{-p_f} \tag{9}$$

式中，p_f 为最终计算结果的二进制小数位数，p_f+g 为系数的二进制小数位数。满足关系式的一种方法是选择合适的 n_0，n_1 和 g，使它们满足下面的不等式：

$$\begin{aligned}2n_0+n_1&\geqslant p_f+\log_2(|f''(\xi_2)|)\\ g&\geqslant 2+\log_2(m-1)\end{aligned} \tag{10}$$

第一个不等式影响 n_0 和 n_1 的选择，决定输入数据的划分方式；第二个不等式影响 g，决定系数表的宽度。

3 算法改进

STAM 算法主要靠减小系数 $a_{i-1}(x_0,x_i)(i\geqslant 2)$ 的字长和深度实现存储空间的缩减，而系数 $a_0(x_0,x_1)$ 占用字长为 p_f+g、深度为 $2^{n_0+n_1}$ 的查找表，仍有较大的压缩空间。考虑到非线性函数 $f(x)$ 的可微性，由式(6)可知，系数 $a_0(x_0,x_1)$ 也是可微的，其值离散化以后相邻值之间存在较大的相关性，利用这种相关性可以进一步压缩存储空间。

不妨记 $x_{01}=x_0+x_1$，用 n_0+n_1 位二进制小数表示，其最小步进单位为 $\sigma=2^{-n_0-n_1}$，变量 x_{01} 数字化以后可记为：

$$x_{01}(i)=i\times\sigma,i=0,1,2\cdots 2^{n_0+n_1}-1 \tag{11}$$

对应的系数 a_0 为：

$$a_0(i)=f(i\times\sigma+\delta_{2,m}),i=0,1,2\cdots 2^{n_0+n_1}-1 \tag{12}$$

在第 i 点进行一阶差分为：

$$\Delta a_0=a_0(i+1)-a_0(i)\approx f'(x_0+x_1)\sigma \tag{13}$$

式中，$\sigma=2^{-n_0-n_1}$，$|f'(x_0+x_1)|\leqslant|f'(\xi_1)|$，可得：

$$|\Delta a_0|\leqslant 2^{-n_0-n_1}|f'(\xi_1)| \tag{14}$$

因此，存储 Δa_0 时有 $n_0+n_1+\log_2(|f(\xi_0)/f'(\xi_1)|)$ 位不需要存储，其可通过扩展符号位获得，由此达到压缩存储空间的目的。

具体实现时，每 k 个系数中选取 1 个存储原值，其余系数只存储其与前值之差，即对系数 $a_0(x_0,x_1)$ 进行差分运算后要存储的系数为：

$$\begin{aligned}a_{00}(i/k)&=a_0(i),\text{if } i\%k=0\\ a_{0j}((i-j)/k)&=a_0(i)-a_0(i-1),\text{if } i\%k=j\neq 0\end{aligned} \tag{15}$$

式中，% 为取余数操作。分解后的系数原值由第 1 个查找表存储，差值按照 $i\%k$ 的值分别存储在 $k-1$ 个查找表中，每个查找表的深度均变为原来的 $1/k$，后面 $k-1$ 个查找表的位宽减小 $n_0+n_1+\log_2(|f(\xi_0)/f'(\xi_1)|)$ 位。

4 双曲正切函数的实现

双曲正切函数 $f(x)=(e^x-e^{-x})/(e^x+e^{-x})$ 是人工神经网络中最常见的激活函数之一，不妨以此函数为例来说明改进 STAM 算法在 FPGA 上的具体硬件实现。

4.1 参数计算

首先根据精度要求确定一系列参数。假定要求函数输出的精度为 e，首先根据 e 确定输入、输出的小数位宽，如 $2^{-15}<e<2^{-16}$，取 $p_f=16$ 即可满足精度要求。经对比研究[9]，取 $m=3$ 时空间占用较小，同时系统也不会过于复杂。选择一组 n_0、n_1、n_2、n_3，使其满足式(10)，如取 $n_0=6$、$n_1=4$、$n_2=3$、$n_3=3$，并可算得 $g=3$。由 n_0、n_1、n_2、n_3，可计算式(6)中的 δ_i。为了将最大绝对误差限定在 2^{-p_f}，需要确定输入的取值范围，假定 x 取值范围为[0，$x_{\max}$]，当 $x=x_{\max}$ 时，$1-f(x_{\max})<2^{-p_f}$，解不等式得：$x_{\max}>\ln(2^{p_f+1}-1)/2$。当 $p_f=16$ 时，$x_{\max}>5.9$，则取为 6 满足要求，需 3 个整数位表示，而由 x 取值范围可计算 $|f(\xi_0)|<1$，$|f'(\xi_1)|=1$。

4.2 系数计算与存储

参数确定后，运用 Matlab 根据式(6)分别计算整数为 0～5 时系数 a_0、a_1、a_2 的值，此时计算出的系数值均为 19 位二进制定点小数表示的精确值。接着根据式(8)计算各系数的存储位宽，计算得到，a_1 需保留后 9 位，a_2 保留后 6 位，以此作为查找表的初始化值。根据改进算法，取 $k=2$，由式(14)计算差值位宽为 10 位，将系数 a_0 拆分为存储原值位宽为 19 的表 a_{00} 和存储差值位宽为 10 的表 a_{01}，如图 2 所示。

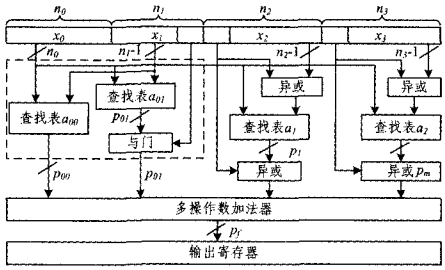

图 2　$k=2$ 时改进 STAM 算法硬件结构图

因 FPGA 中存储元件 RAMB16 的输入信号为 4 位，故保留位数均取 4 的整数倍，即表 a_{00} 位宽取 20 位，表 a_{01} 取 12 位，a_1 取 12 位，a_2 取 8 位。在 Spartan 系列 FPGA 上实现时，20 位的系数 a_{00} 和 12 位的 a_{01} 可以合并成由 1 个 32 位数据存储在 1 个双口 RAM 中，同理，a_1 和 a_2 也可合并存储。因此，4 个查找表实际上只用了 2 个 RAMB16 模块即可实现。

4.3　首项系数表的实现($k=4$)

改进算法中若取 $k=4$，则将首项系数拆分为 4 部分。根据式(15)将 4.2 节中计算出的 a_0 拆分为 a_{00}、a_{01}、a_{02}、a_{03}，其在电路中的具体实现如图 3 所示，其中系数选通信号 a、b、c 由 $x_1[1:0]$确定，真值表如表 1 所列，经推算得 $a=x_1[1]$ OR $x_1[0]$，$b=x_1[1]$，$c=x_1[1]$ AND $x_1[0]$。图 2 虚线框部分用图 3 电路替代就可以实现 $k=4$ 的改进 STAM 算法。

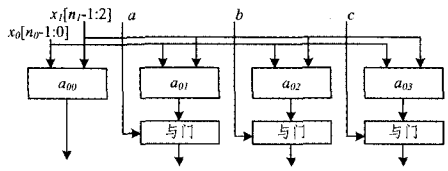

图 3　改进的首项系数表实现框图($k=4$)

表 1　系数选通信号真值表

$x_1[1]$	$x_1[0]$	a	b	c
0	0	0	0	0
0	1	1	0	0
1	0	1	1	0
1	1	1	1	1

5　仿真结果分析

5.1　功能验证

在 Xilinx 公司 Spartan 系列 FPGA(型号为 xc3s500e-4-fg320)上采用改进 STAM 算法($k=2$)实现了双曲正切 S 形函数，使用 Modelsim6.5 仿真的结果片段如图 4 所示，输入(input)和硬件电路的输出(output)均为 16 位二进制定点小数，以十六进制显示，参考值(ref)是用 Matlab 预先算出的函数值的 16 位定点小数表示，误差(error)为输出与参考值之差。经验证，误差总小于 1 个 ulp(即 2^{-16})。

```
# input output  ref   error
# 8069  769f   0769f  00000
# 806a  76a0   076a0  00000
# 806b  76a1   076a1  00000
# 806c  76a1   076a1  00000
# 806d  76a2   076a2  00000
# 806e  76a3   076a3  00000
# 806f  76a4   076a4  00000
# 8070  76a5   076a5  00000
# 8071  76a5   076a5  00000
# 8072  76a6   076a6  00000
# 8073  76a7   076a7  00000
# 8074  76a8   076a8  00000
# 8075  76a8   076a8  00000
# 8076  76a9   076a9  00000
```

图 4　改进 STAM 算法实现 tansig(x)仿真结果片段

5.2　性能分析

采用改进的 STAM 算法对双曲正切函数进行近似，硬件电路占用的存储空间较 STAM 算法显著改善，如表 2 所列；且 k 越大，占用的存储空间越小，但这增加了多操作数加法器的负担，因此需在两者之间取得平衡。为衡量加法器的增加换来的存储空间缩减效率，定义效用比 Q 为：

$$Q=\frac{\text{存储空间缩减比例}}{\text{加法器数}}\times 100 \tag{16}$$

由表 2 中 Q 值可见，综合考量存储空间和加法复杂性，取 $k=2$ 时效用比最高；而当存储空间作为主要考量时，取 $k=4$ 则可更有效地节省存储空间。

表 2　改进 STAM 算法存储空间对比表

算法	表大小	存储位数	缩减比例	加法器数	Q
STAM	$2^{10}\times19+2^9\times9+2^9\times6$	27136	0	2	0
改进(k=2)	$2^9\times19+2^9\times10+2^9\times9+2^9\times6$	22528	17%	3	5.67
改进(k=4)	$2^8\times19+2^8\times10\times3+2^9\times9+2^9\times6$	20224	26%	5	5.2
改进(k=8)	$2^7\times19+2^7\times10\times7+2^9\times9+2^9\times6$	19072	30%	9	3.3

硬件电路的 ISE 综合报告显示算法最大延时小于1.5ns，可以工作在 333MHz 的高频时钟，表明算法效率较高。布局布线后硬件资源占用情况如表 3 所列，可见总体占用硬件资源较少。存储块 RAMB16 是 FPGA 内部珍贵的存储资源，算法中的 4 个查找表经合并处理由 2 个 RAMB16 实现，提高了硬件资源利用效率。

表 3　硬件资源占用表

逻辑资源	占用	可用	占用比
Slice	44	4656	1%
RAMB16	2	20	10%
LUT4	51	9312	1%
FlipFlop	12	9312	0%

结束语　本文讨论了硬件实现非线性函数的 STAM 算法的原理，在算法分析的基础上提出了对首项系数进行差分分解的改进方法，并以双曲正切函数为例，说明了算法实现所需的一系列参数的求解方法以及硬件电路结构。最后，通过分析仿真结果证明，改进后的 STAM 算法功能正确，而且有效地节约了存储器资源，使得最终存储器位数减少了 17%～30%。

参 考 文 献

[1] Pineiro J A, Bruguera J D. High-speed double-precision computation of reciprocal, division, square root, and inverse square root [J]. IEEE Transactions on Computers, 2002, 51(12): 1377-1388

[2] Basterretxea K, Tarela J M, Campo I. Digital design of sigmoid approximator for artificial neural networks[J]. Electronics Letters, 2002, 38(1): 35-37

[3] Hyun-Chul S, Jin-Aeon L, Lee-Sup K. A Minimized Hardware Architecture of Fast Phong Shader Using Taylor Series Approximation in 3D Graphics[C]// International Conference on Computer Design: VLSI in Computers and Processors, 1998(ICCD '98). Austin, Texas, 1998: 286-291

[4] Karl L, Ashkan H N, Roberto M, et al. High Speed VLSI Implementation of the Hyperbolic Tangent Sigmoid Function[C]// Third International Conference on Convergence and Hybrid Information Technology, 2008(ICCIT '08). Busan, 2008: 1070-1073

(下转第 285 页)

示。FFTW estimate 策略下，2 核的加速比为 1.55，4 核的加速比为 2.55。FFTW measure 策略下，2 核的加速比为 1.50，4 核的加速比为 2.75。相比其它主流处理器，龙芯 3A 处理器浮点运算能力相对较强，但是访存能力较慢，然而 FFT 变换过程中多数都是对内存的不连续访问，因此很难得到很高的并行加速比。

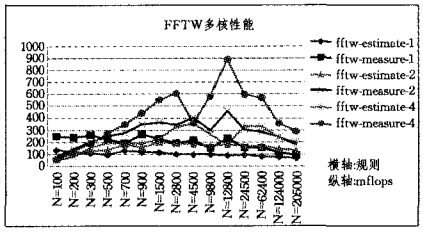

图 12　龙芯 3A-FFTW 多核性能

结束语　本文主要分析了 FFTW、UHFFT 和 SPIRAL 这 3 个软件包的自适应框架，并在龙芯 3A 上进行了性能分析和对比。总结起来，3 个软件包原理相通，主要分为高级别的优化和低级别的优化，也就是算法层和实现层的优化。不同点主要体现在问题规模的表示方式、搜索策略和片段代码这几方面。龙芯 3A 处理器作为国产多核 CPU，在一定程度上能够有效地支持自适应 FFT 软件包的运行，并且也取得了良好的多核加速比。

本文下一步要做的工作为进一步分析 3 个软件包的多核实现特点，并在当前主流多核平台上和龙芯 3A 平台上进行详细的测试分析，以更深入地理解自适应技术的特点，并特别针对龙芯 3A 平台进行算法的调优，将这些技术用在其它软件包中。

参 考 文 献

[1] Cooley J W. The re-discovery of the fast Fourier transform algorithm[J]. Mikrochimica Acta, 1987, 3: 33-45

[2] Cooley J W, Tukey J W. An algorithm for the machine calculation of complex Fourier series[J]. Math. Comp., 1965, 19: 297-301

[3] Frigo M. A fast Fourier transform compiler[C]// Proceedings of the ACM SIGPLAN 1999conference on Programming Language Design and Implementation, PLDI '99. New York, NY, USA: ACMPress, 1999: 169-180

[4] Frigo M, Johnson S G. The design andimplementation of FFTW3[C]// Proceedings of the IEEE Special Issue on Program Generation, Optimization, and Platform Adaptation. 2005: 216-231

[5] Frigo M, Johnson S G. The fastest fourier transform in the west [M]. technical report technical reportMIT-LCS-TR-728, Sep. 1997

[6] Frigo M, Johnson S G. Fftw: An adaptive software architecture for the fft[C]// Proc. ICASSP 3. 1998: 1381-1384

[7] Loan C V. Computational frameworks for the fastFourier transform. Society for Industrial and AppliedMathematics[M]. Philadelphia, PA, USA, 1992

[8] Mirkovic D, Johnsson S L. AutomaticPerformance Tuning in the UHFFT Library[C]// Proceedings of the International Conference onComputational Sciences-Part I, ICCS'01. London, UK: Springer-Verlag, 2001: 71-80

[9] Franchetti F, Voronenko Y, Püschel M. FFT program generationfor shared memory: SMP and multicore[C]// Proc. Supercomputing (SC). 2006

[10] Spiral Web site[OL]. http://www.spiral.net

[11] Xiong J, Johnson J, Johnson R, et al. SPL: A language andcompiler for DSP algorithms[C]// Proc. Programming Language Designand Implementation (PLDI). 2001: 298-308

[12] FFTW Web site[OL]. www.fftw.org

[13] UHFFT Web site[OL]. www2.cs.uh.edu/~ayaz/uhfft

[14] Franchetti F, Voronenko Y, Püschel M. A rewriting system forthe vectorization of signal transforms[C]// Proc. High PerformanceComputing for Computational Science (VECPAR). 2006

[15] Franchetti F, Voronenko Y, Püschel M. Loop merging forsignal transforms[C]// Proc. Programming Language Design andImplementation (PLDI). 2005: 315-326

[16] Johnson J R, Johnson R W, Rodriguez D, et al. Amethodology for designing, modifying, and implementing Fourier transformalgorithms on various architectures[J]. IEEE Trans. Circuits, Systems, and Signal Processing, 1990, 9(4): 449-500

[17] Ali A, Johnsson L, Subhlok J. Scheduling FFT computation onSMP and multicore systems[C]// Proc. Int'l Conf. Supercomputing (ICS). 2007

[18] Temperton C. A Note on Prime Factor FFT Algorithms[J]. Journal of Computational Physics, 1983, 52: 198-204

[19] Duhamel P, Hollmann H. Split Radix FFT Algorithms[J]. Electronic Letters, 1984, 20: 14-16

[20] Singleton R C. An algorithm for computing the mixed radix fast Fourier transform[M]. IEEE Trans. on Audioand Electroacoustics. 1969: 93-103

[21] Rockmore D N. The FFT: An algorithm the whole family can use[J]. Computing in Science & Engineering, 2000, 2(1): 60-64

[22] Brigham E O. The Fast Fourier Transform and Its Applications [M]. Prentice Hall Signal Processing Series, Englewood Cliffs. NJ, 1988

(上接第 254 页)

[5] Boudabous A, Ghozzi F, Kharrat M W, et al. Implementation of hyperbolic functions using CORDIC algorithm[C]// The 16th International Conference on Microelectronics, 2004(ICM 2004). Tunisia, 2004: 738-741

[6] 张华军，赵金，丛喆. CORDIC 在基于 FPGA 的神经网络设计中的应用[J]. 华中科技大学学报：自然科学版，2009，37(6)：36-39

[7] 张智明，张仁杰. 神经网络激活函数及其导数的 FPGA 实现[J]. 现代电子技术，2008(18)：139-142

[8] 夏欣，贾永刚，王素珍. RBF 神经网络中指数函数 e^x 的 FPGA 实现[J]. 微计算机信息，2005，21(7-2)：145-146

[9] Michael J S, James E S. The Symmetric Table Addition Method for Accurate Function Approximation[J]. Journal of VLSI Signal Processing, 1999(11): 1-12