

用二维坐标下降法实现 $x = b/a$ 的 FPGA 设计

罗 珍¹, 刘海林²

(广东工业大学 1. 信息工程学院; 2. 应用数学学院, 广东 广州 510006)

摘要: 以二维坐标下降法(DCD 算法)为基础,每次成功迭代后以二分之一的步长进行下一次迭代,使得它在解决线性方程时不仅保证了收敛度,并且实现了在 FPGA(现场可编程门阵列)中无需乘法器就可以计算出两个参数的比率,最后通过 Matlab 仿真验证了 DCD 算法在此应用中的可行性及其收敛速度快的特点。

关键词: DCD 算法; 步长; 线性方程; 参数比率; 无需乘法器

中图分类号: TP302.7

文献标志码: A

文章编号: 1007-7162(2011)01-0032-06

在信号处理和通信系统中,自适应滤波、信道估计与均衡、消除回声和干扰等方面都需要在实时中解决线性二乘法问题^[1]。而最小二乘法问题等同于系统中线性方程(也可称为标准方程)的求解^[2-3]。一般的解决方法是利用逆矩阵来求解,然而逆矩阵很难在实时中实现。从数值分析的角度上看,最好的解决方法是避免求解逆矩阵^[3-4],因此那些不需要用到逆矩阵的方法成为首选。不需要用到逆矩阵的方法大致可以分为两类:直接法和迭代法。直接法包括 Cholesky 分解、高斯消去法、QR 分解法,但是这些方法的收敛速度都很慢;迭代法包括最快下降法、共轭梯度法,这些方法有着较快的收敛速度,但是每次迭代都需要用到乘法和除法,并且在实现过程中尤其是在硬件实现中会非常复杂。

DCD 算法以共轭梯度法为基础,常用于解线性方程。目前 DCD 算法已经运用到自适应滤波^[5-6]、放大器线性化^[7]、多用户监测^[8]以及阵列天线集束中^[9],并且已经证实这种算法具有复杂度低的特征。

在信号处理中许多算法需要计算两个参数的比率,但是由于微控制器、ASICs(专用集成电路)、FPGA(现场可编程门阵列)等硬件平台缺少专门的乘法器,因此这些算法用不动点的方法实现很困难。本文以二维坐标下降法(简称 DCD 算法)为基础,每次成功迭代后以二分之一的步长进行下一次迭代,使得它在解决线性方程时不仅保证了收敛度,且实现了在 FPGA 中无需用到乘法器就可以计算出两个参数的比率,最后通过 Matlab 仿真验证了改进后的

DCD 算法的可行性及其收敛速度快的特点。

1 DCD 算法

在最小二乘法问题中,常常需要求解方程的最小值: $J(h) = \|Zh - d\|^2, h \in \mathbb{R}^N, d \in \mathbb{R}^M$, 其中 h 是一个 $N \times 1$ 维向量, Z 是一个 $M \times N$ 矩阵, d 是 $M \times 1$ 维向量。解决此问题等同于解决线性方程: $Ah = b$, 其中, $A = Z^T Z, b = Z^T d$ 。

在以下条件考虑最小二乘法问题:

$$h \in U = \{ (h_1, h_2, \dots, h_N) : |h_i| \leq H, i = 1, 2, \dots, N \}, \quad (1)$$

h_i 是向量 h 中的元素, H 是一个已知的数,且 $H > 0$, 在很多实际应用中这是一种很现实的假设。

当运用坐标下降最优化法来最小化函数 $J(h)$, 在第 k 次迭代的时候分析如下不等式

$$\Delta J(h_k) = J(h_k \pm \alpha e_i) - J(h_k) < 0, \quad (2)$$

其中 α 是步长, e_i 是单位向量,且只有第 i 个元素取 1,其余的元素都取 0, $i = k \bmod(N) + 1$ 。如果不等式 (2) 成立,就视此次迭代已经成功。那么则有 $h_{k+1} = h_k \pm \alpha_k e_i, \alpha_{k+1} = \alpha_k$ 。反之,如果迭代不成功的话, $h_{k+1} = h_k$, 当 $i = N, h_k = h_{k-N+1}$ 时 $\alpha_{k+1} = \lambda \alpha_k (0 < \lambda < 1)$, 否则 $\alpha_{k+1} = \alpha_k$ 。

从不等式 (2) 中可以看到, $\Delta J(h_k)$ 映射出当前值和理想值之间的间隔。如果这个间隔一直在变小,说明迭代成功,否则的话 $\Delta J(h_k) < 0$ 。那么如何选择加减符号呢,如果沿着步长的正方向迭代成功的话就选择加号,如果沿着补偿的负方向迭代成功的话就选择减号^[10]。

收稿日期: 2010-11-15

作者简介: 罗 珍(1985-),女,硕士研究生,主要研究方向为功率放大器的数字基带预失真、自适应算法。

下面选取一个方向来推导整个过程:

$$J(\mathbf{h}) = \mathbf{h}^T \mathbf{A} \mathbf{h} - 2\mathbf{b}^T \mathbf{h} = \sum_{m=1}^N \sum_{n=1}^N a_{mn} h_m h_n - 2 \sum_{m=1}^N b_m h_m, \quad (3)$$

其中 a_{mn} 、 b_m 分别是矩阵 $\mathbf{A}$ 和向量 $\mathbf{b}$ 中的元素,则

$$\begin{aligned} J(\mathbf{h} + \alpha \mathbf{e}_1) &= a_{11}(h_1 + \alpha)(h_1 + \alpha) + \sum_{n=2}^N a_{1n}(h_1 + \alpha)h_n + \\ &\sum_{m=2}^N \left[a_{m1}h_m(h_1 + \alpha) + \sum_{n=2}^N a_{mn}h_m h_n \right] - 2 \sum_{m=1}^N b_m h_m = \\ J(\mathbf{h}) &+ a_{11}\alpha^2 + 2 \sum_{m=1}^N a_{1m}\alpha h_m - 2b_1\alpha, \quad (4) \end{aligned}$$

整理后可得到

$$J(\mathbf{h} + \alpha \mathbf{e}_1) - J(\mathbf{h}) = a_{11}\alpha^2 - 2b_1\alpha + 2 \sum_{m=1}^N a_{1m}\alpha h_m, \quad (5)$$

$$\Delta J(\mathbf{h}_k) = \alpha(\alpha a_{ii} \pm Q_i), \quad (6)$$

其中 $Q_i = -2b_i + 2 \sum_{m=1}^N h_m a_{mi}$.

根据式(2)和(6),可以得到

$$Q_i < -\alpha a_{ii} \text{ 或者 } Q_i > \alpha a_{ii}. \quad (7)$$

只有当迭代成功 h 和 Q 才能得到更新,从而使计算得以简化. 假设 $Q_n^{(k)}$ 是向量 Q 在第 k 次迭代后的元素,如果第 k 次迭代没有成功,则 $h_i^{(k+1)} = h_i^{(k)}$, $Q_n^{(k+1)} = Q_n^{(k)}$, 其中 $n = 1, 2, \dots, N$. 反之,

$$h_i^{(k+1)} = h_i^{(k)} \pm \alpha, \quad (8)$$

$$Q_n^{(k+1)} = Q_n^{(k)} \pm 2\alpha a_{ni}, n = 1, 2, \dots, N. \quad (9)$$

以上推导的算法称之为 DCD(dichotomous coordinate descent) 算法,主要用于信号处理中线性问题的求解,如何选择向量 h 的初始值是 DCD 算法的关键. 从理论上讲, h 可以取任意值(0 除外),考虑到算法的有效性,步长要以 2 倍的速度下降. 在硬件实现的过程中,DCD 算法中出现的乘法除法可以分别用左移或右移来实现,这种免除的特征使得 DCD 算法在硬件实现中可以节省大量的资源. 为了尽可能让 DCD 算法变得有效,在选取初始值的时候要尽量选择 2 的倍数,这样输出变量就能用控制码的形式来编码.

2 DCD 算法的应用

2.1 求解线性方程 $Rx = b$

在信号处理中,诸如参数频谱分析工具、Kalman 状态估算、功率放大器的线性化以及循环最小二乘法滤波器中,都要求解线性问题 $Rx = b$,其中 R 是

已知的 $N \times N$ 对称正矩阵, b 是已知的 N 维实向量,可运用于上一节介绍的 DCD 算法.

$Rx = b$ 可以看成是当 $D(x) = dJ(x)/dx = 2Rx - 2b = 0$ 时求方程 $J(x) = x^T Rx - 2x^T b$ 的最小值. 用 DCD 算法来解决这一问题,首先初始化 x_0 使得 $x_0 = 0$, 则 $J(x_k)$ 下降 $d_k = D(x_k)$ 最终到达零点,然后检验 $J(x_k + he_i)$ 和 $J(x_k - he_i)$ 是否小于 $J(x_k)$, 其中 h 代表步长,最终可以得到:

如果 $d_{i,k} < -hR_{ii}$, 则

$$d_{k+1} = d_k + 2hRe_i, \quad x_{k+1} = x_k + he_i; \quad (10)$$

如果 $d_{i,k} > hR_{ii}$, 则

$$d_{k+1} = d_k - 2hRe_i, \quad x_{k+1} = x_k - he_i. \quad (11)$$

其中 $d_{i,k} = e_i^T d_k$, $R_{ii} = e_i^T R e_i$, 如果满足这些条件之一,则开始比较下一个元素;如果所有的条件都不满足的话, h 被 2 整除以后,重新检验式(10)、(11)中的条件.

综上所述,DCD 算法可用以下语句来描述:

Initialization: $x = 0, r = -2b, h = H, k = 0$

for $m = 1 : M_b$

$h = h/2$

while

Flaga = 0

for $n = 1 : N$

if $|r_n| > h \times R_{n,n}$ then

$x_n = x_n - \text{sign}(r_n) \times h$

$r = r - 2\text{sign}(r_n) \times h \times R(:, n)$

$k = k + 1, \text{Flaga} = 1$

end if

if $k > N_u$, algorithm stops

end of the n -loop

repeat while Flaga = 1

end of the m -loop

这里有几点对 DCD 算法的说明:

(1) 如果迭代不成功,步长 h 取零值;如果迭代成功,步长 h 则以上一次迭代步长值的二分之一继续迭代.

(2) H 代表步长的初始值, M_b 代表方程解参数最终固定小数点后的表示值的最小比特数.

(3) 计算负载量主要取决于迭代成功的次数,参数 N_u 是人为设定的,它是对迭代成功的次数一个限制,也是对计算过程中出现的不确定错误的一个限定.

(4) $h \times R_{n,n}$ 在硬件上可以用左移来表示, $h/2$ 则用右移来表示, 因此输出变量可以用控制码来表示: 例如用小数点前 8 位和小数点后 7 位来给 a 、 b 编码(所有的数字都在 $-128/128 = -1$ 和 $+127/128 \approx +1$ 之间), 如果 $a = 1/128$, x 的最小值取 $-128(b = -128/128)$, 最大值取 $+127(b = +127/128)$, 因此, b/a 的计算结果小数点前至少有 8 位, 小数点后的精确度可以人为设定. 比如可以取小数点前 9 位小数点后 7 位, 但是为了让 DCD 算法达到最高的效率, h 一般取值为 128. 如表 1 所示, $a = 1.357, b = 1.853, h$ 的初始值 $h_{\text{init}} = 16$, 当 e_n 小于 $h_{\text{init}}/2^{M_b-1} \approx 1.19 \times 10^{-7}$ 时算法停止. 所得结果用 28 位表示, 从而得到一系列有关 x_n 的取值; 经过 17 次迭代以后, 设定的精确度已经达到, 也就是说, DCD 算法经历 34 次加减以后就可计算出 b/a 的结果. 图 1 见证了 $(x_n, J(x_n))$ 快速取得最小值的过程; 图 2 则映射了误差函数 e_n 和模型 $e_n = 10^{-(n-2)/2}$ 之间的细微差别. 从图 2 中可以看出, 实线是关于 n 的方程 $e_n = |x_n - b/a|$, 虚线为 $10^{-(n-2)/2}$ 几何序列. 图中显示虚线的近似速率为 $10^{-1/2}$, 也就是说两次迭代需要增加一个字节的精确度. 这两个函数都接近于 $10^{-1/2}$, 说明两次迭代可以提高一个位的精确度.

表 1 每次迭代产生的误差值

n	x_n	$e_n = x_n - b/a $
1	0.000 000	1.365 512E + 000
2	2.000 000	6.344 878E - 001
3	1.000 000	3.655 122E - 001
4	1.500 000	1.344 878E - 001
5	1.250 000	1.155 122E - 001
6	1.375 000	9.487 841E - 003
7	1.359 375	6.137 159E - 003
8	1.367 188	1.675 341E - 003
9	1.365 234	2.777 842E - 004
10	1.365 723	2.104 971E - 004
11	1.365 479	3.364 355E - 005
12	1.365 540	2.739 161E - 005
13	1.365 509	3.125 972E - 006
14	1.365 513	6.887 257E - 007
15	1.365 512	2.649 486E - 007
16	1.365 512	2.118 886E - 007
17	1.365 512	2.653 000E - 008

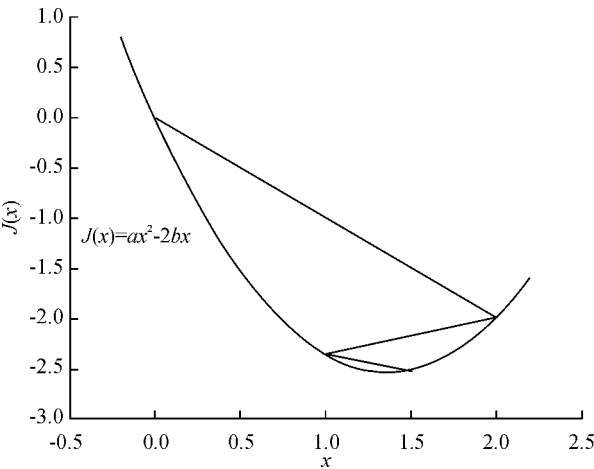


图 1 $J(x_n)$ 取得最小值的过程

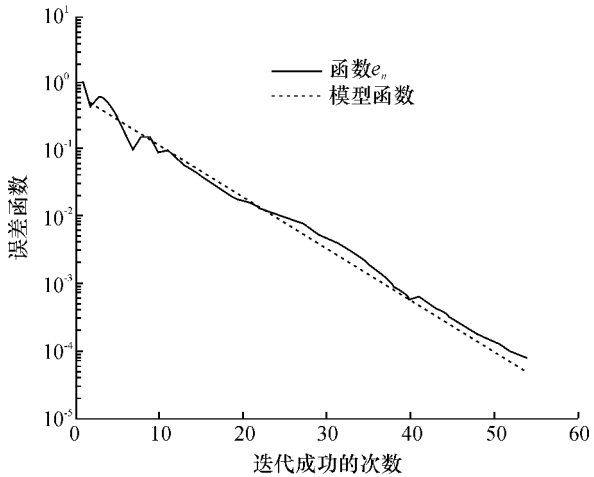


图 2 函数 e_n 与模型函数 $10^{-(n-2)/2}$ 的误差极小

2.2 求解两个参数的比率 $x = b/a$

在信号处理问题中常常需要计算两个虚数除法的速率^[3], 即当 $b = b_r + jb_i, a = a_r + ja_i$ 时求解 $x = b/a$. 根据前一节, 可以推导出

$$Ax = b, \text{ 其中 } A = \begin{pmatrix} a_r & -a_i \\ a_i & a_r \end{pmatrix},$$
$$x = \begin{pmatrix} x_r \\ x_i \end{pmatrix}, b = \begin{pmatrix} b_r \\ b_i \end{pmatrix}. \quad (12)$$

在方程 $Ax = b$ 左右两边同时乘以一个对称矩阵 A^T ,

$$A^T Ax = A^T b,$$
$$\Rightarrow \begin{pmatrix} a_r^2 + a_i^2 & 0 \\ 0 & a_r^2 + a_i^2 \end{pmatrix} x = \begin{pmatrix} a_r b_r + a_i b_i \\ a_r b_i - a_i b_r \end{pmatrix}. \quad (13)$$

用传统的方法来解决此问题需要用到 6 步乘法 和 3 步除法, 下面运用 DCD 算法来解决这个问题. 根据 DCD 算法可知, 在偶数次迭代时, 考虑实部的

迭代:

如果 $b_{r,2k-1} > ha_r$, 则

$$b_{2k} = b_{2k-1} - ha, x_{2k} = x_{2k-1} + h; \quad (14)$$

如果 $b_{r,2k-1} < -ha_r$, 则

$$b_{2k} = b_{2k-1} + ha, x_{2k} = x_{2k-1} - h. \quad (15)$$

在奇数次迭代时,考虑虚部的迭代:

如果 $b_{i,2k} > ha_r$, 则

$$b_{2k+1} = b_{2k} - jha, x_{2k+1} = x_{2k} + jh; \quad (16)$$

如果 $b_{i,2k} < -ha_r$, 则

$$b_{2k+1} = b_{2k} + jha, x_{2k+1} = x_{2k} - jh. \quad (17)$$

由上可见,此算法通过加减达到实部和虚部的迭代,其中 h 的初始值要取 2 的倍数,这样的话计算过程只需要用到加法和左移右移从而达到免乘的目的. 如图 3 所示,当 $a = 0.8 + j3.6, b = 0.909 + j345$, h 的初始值为 8,成功迭代的次数共 63 次,图 3 见证了 $(x_{r,n}, x_{i,n})$ 是如何快速获取最佳值的;从图 4 中误差函数与模型的比较可知,经历 5 次迭代后可以提高所得计算结果一个精确度,从而验证了 DCD 算法收敛速度快度的特点.

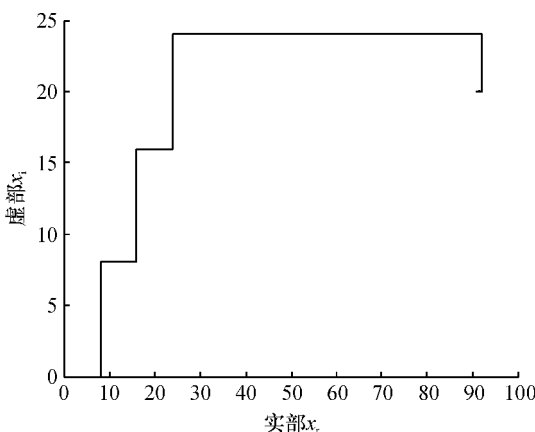


图 3 迭代 63 次后 x 取得最佳值

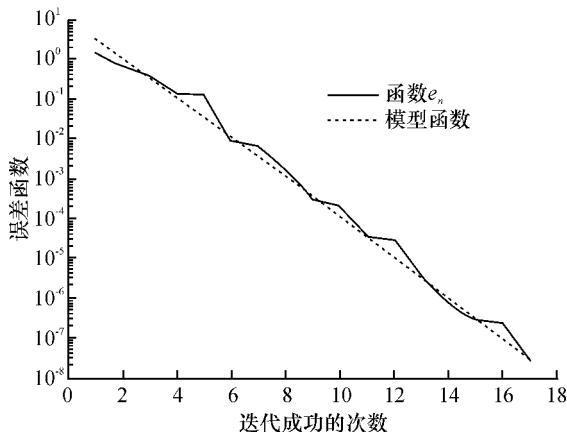


图 4 函数 e_n 与模型函数 $10^{-(n-2)/2}$ 误差极小

3 $x = b/a$ 的 FPGA 实现

3.1 $x = b/a$ 的构造图

在 Xilinx XUP Virtex-II 环境中用 DCD 算法设计 FPGA 的核心^[4],然后用 VHDL 语言描述 FPGA 的核心,最后用 Xilinx ISE 11.1 中的软件包把它合成后下载到目标平台.所有 a 和 b 中的初始值都要用 $[-1, 1)$ 范围内的 16bits Q15 控制码来表示^[5],为了防止计算过程中出现溢出,所有的变量都要以 32bits Q16 控制码的形式储存在 FPGA 的核心中.

如图 5,虚数除法器主要由 1 个控制器和 4 个加减法器组成,整个设计与外部时钟信号是同步的,当核心区接到信号“开始”时系统启动,而“结束”信号则表示整个除法过程结束,“0”表示接收到错误信号,参数 b, a, x 的元素通过不同的出入口进行传输.

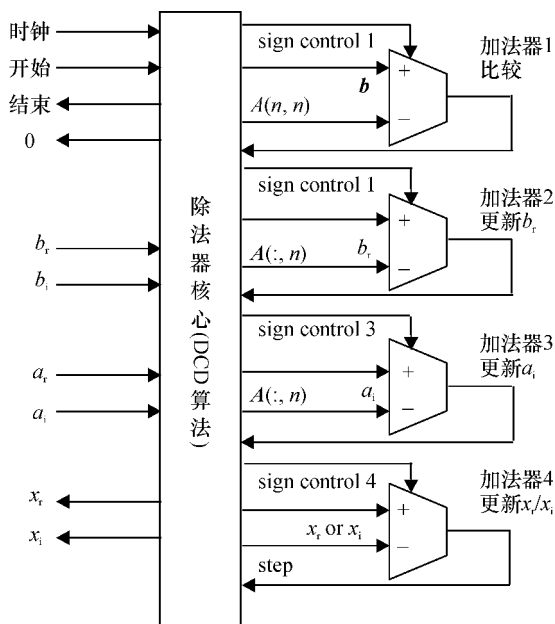


图 5 除法器的构造图

3.2 核心控制器的状态流程及实现过程

状态机械流程图^[6] (state machine flowchart,简称 SM 图) 在软件和硬件设计中非常有用,它有助于更好地理解整个系统的操作过程.除此之外,SM 图不同于普通的流程图,它必须遵循一定的规则,当这些规则被遵循时,SM 图与状态图是等效的,从而能够直接用于硬件实现.本节介绍了 DCD 算法作为核心控制器是如何在整个除法构造器中起着核心控制作用的.

如图 6 所示,可以用 SM 图来展现图 5 的整个操作流程,SM 图协调着整个系统的运行:

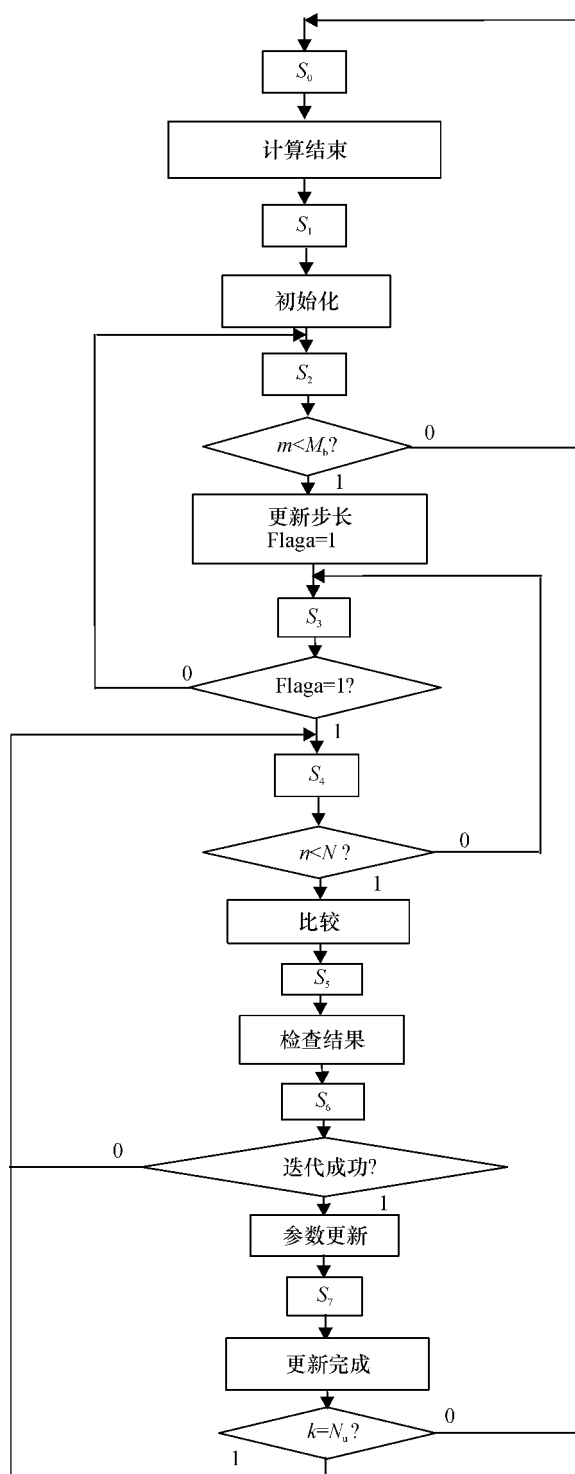


图6 核心控制器的状态流程图

State 0(S_0) : 开始

状态 S_0 开始计算以下参数:

$$Z = \begin{pmatrix} d_r & -d_i \\ d_i & d_r \end{pmatrix}, d = \begin{pmatrix} r_r \\ r_i \end{pmatrix},$$

$$R = Z^*Z, b = Z^*d, r = -2b.$$

State 1(S_1) : 初始化

状态 S_1 开始对步长 “step” 和参数 k 初始化:

“step = 2H”, $k = 0$.

State 2(S_2) : m -循环控制以及步长更新

如果 $m < M_b$ 为真, 步长将更新为 $\text{step} = \text{step}/2$, 并且设置 $\text{Flag} = 1$. 如果条件为假, 那么意味着 m -循环已经完成, 整个计算处于成功状态.

State 3(S_3) : While 语句循环控制

如果 $\text{Flag} = 1$ 为真, 重新比较所有的向量以后再执行下一步; 如果 $\text{Flag} = 0$, 则返回到 S_2 更新步长.

State 4(S_4) : n -循环控制

如果 $n < N$, 比较 $|r_n|$ 和 $\text{step} \times R(n, n)$, 当 $|r_n| > \text{step} \times R(n, n)$ 时继续执行下一步, 否则重新返回并检查 Flag 的值.

State 5(S_5) : 比较

S_5 将完成比较环节, 接着检查所得结果.

State 6(S_6) : 判断迭代是否成功

如果迭代成功了, 对向量进行以下更新:

$$q(n) = q(n) - \text{sign}(r_n) \times \text{step}, r = r - 2\text{sign}(r_n) \times \text{step} \times R(n, n).$$

如果迭代不成功, 返回 S_4 比较下一个元素.

State 7(S_7) : 判断是否达到最大迭代数

S_7 将完成整个更新, 如果此时已经达到了所设定的最大迭代数, 完成此次迭代以后程序将终止; 否则返回 S_4 , 比较下一个元素.

4 结论

文中主要描述了 DCD 算法及其应用, 采用二分之一步长在 FPGA 中以移位方式来实现乘法器操作, 实现了在 FPGA 中无需用到乘法器就可以计算出两个参数的比率, 并通过 Matlab 仿真验证了 DCD 算法在此应用中的可行性, 并具有收敛速度快、简便等特点; 当位增大时, 计算所得结果的精确度会增高, 此算法的收敛方向跟逆矩阵算法所得出的结论大致相同. 针对这一特征, 可以把 DCD 算法应用于高功率放大器的数字基带预失真中去, 以达到消除功率放大器的非线性效应的目的.

参考文献:

- [1] Haykin S Adaptive Filter Theory [M]. Englewood Cliffs, 2001, 122(6): 8-12.
- [2] Sayed A H, Kailath. Recursive least-squares adaptive filters [M]. Los Angeles: LLC Press, 1999.
- [3] Doherty J F. Channel equalization as a regularized inverse

- problem [M]. Pennsylvania: CRC Press, 1998.
- [4] Moon T K, Stirling W C. Mathematical Methods and Algorithms for Signal Processing [J]. Englewood Cliffs, 2000, 33(7): 11-42.
 - [5] Uribe R, Cesear T. Implementing matrix inversions in fixed-point hardware [J]. Xilinx DSP Mag, 2005, 12(1): 32-35.
 - [6] Zakharov V V, Teixeira M. SMI-MVDR beamformer implementations for large antenna array and small sample size [J]. IEEE Trans Circuits Syst, 2008, 55(8): 3317-3327.
 - [7] Zakharov Y, Albu F. Coordinate descent iterations in fast affine projection algorithm [J]. IEEE Signal Process, 2005, 101(12): 353-356.
 - [8] Zakharov Y, White G and Jie Liu. Low complexity RLS algorithms using dichotomous coordinate descent iterations [J]. IEEE Trans. Signal Process, 2008, 99(56): 3150-3161.
 - [9] Muruganathan S D, Sesay A B. A QRD-RLS-based predistortion scheme for high-power amplifier linearization [J]. IEEE Trans. Circuits Syst, 2006, 53(11): 1108-1112.
 - [10] Quan Z, Jie Liu, Zakharov Y. FPGA implementation of DCD based CDMA multiuser detector [J]. Proc Conf. DSP Cardiff, 2007, 21(4): 319-322.
 - [11] Jie Liu, Weaver B, Zakharov Y, et al. An FPGA-based MVDR beamformer using dichotomous coordinate descent iterations [J]. Proc ICC, 2007, 24(28): 2551-2556.
 - [12] Jie Liu, Weaver B, Zakharov Y V. FPGA implementation of multiplication-free complex division [J]. Electronics Letters of the IEEE, 2008, 44(9): 6-8.
 - [13] Kay S M, Marple S L. Spectrum Analysis-A Modern Perspective [J]. Proc of the IEEE, 1981, 69(14): 1380-1419.
 - [14] Xilinx XUP Virtex II P Development System [EB/OL]. <http://www.xilinx.com/univ/xupv2p.html>, 1990-06-18.
 - [15] Bateman and Stephens I P. The DSP handbook: algorithms [J]. Applications and design techniques, 2002, 39(11): 33-36.
 - [16] Jie Liu, Ben Weaver, George White. FPGA Implementation of the DCD algorithm [J]. Digital Signal Processing, 2007, 8(3): 331-334.

FPGA Design of the Dichotomous Coordinate Descent Algorithm for Complex Divider Problems

Luo Zhen¹, Liu Hai-lin²

(1. Faculty of Information Engineering; 2. Faculty of Applied Mathematics,
Guangdong University of Technology, Guangzhou 510006, China)

Abstract: Based on the dichotomous coordinate descent (DCD) algorithm, having the next iteration with the power of half step size after a successful iteration guarantees the convergence speed when it solves the problem with the linear equations. It makes it possible to get the ratio of two numbers without multiplication on FPGA. MATLAB simulation is carried out to verify that the DCD algorithm has higher convergence speed in the application.

Key words: DCD algorithm; step size; linear equations; the ratio of two numbers; divider without multiplier