

打印输出

2017年10月21日 8:31

正文

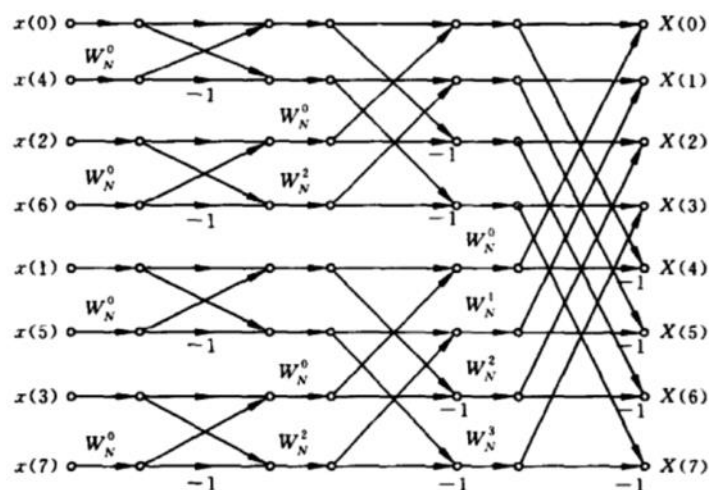
字体大小：大 中 小

MATLAB的一个FFT程序 (2011-08-18 17:11:45)

转载 ▼

标签：教育 分类：DSP数字信号处理

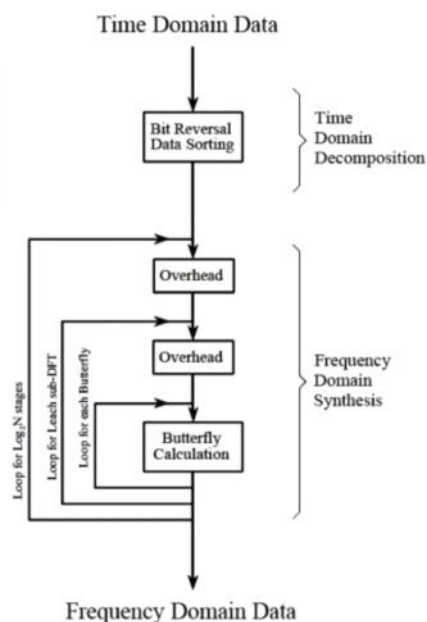
FFT信号流图：



程序实现是这样：

程序流程如下图：

FIGURE 12-7
Flow diagram of the FFT. This is based on three steps: (1) decompose an N point time domain signal into N signals each containing a single point, (2) find the spectrum of each of the N point signals (nothing required), and (3) synthesize the N frequency spectra into a single frequency spectrum.



首先进行位逆转，其实很简单，就是把二进制的位逆转过来：

推荐: Matlab 的位反转程序 [以 0 为起点的快速算法](#) “不合群”的人 [×](#)

[新浪首页](#) [登录](#) [注册](#)

```
function a=bitreverse(Nbit, num)
%Nbit = 4;
%num = 8;
a = 0;
b = bitshift(1,Nbit-1);
for i = 1:Nbit;
if (bitand(num,1) == 1)
a = bitor(a,b);
end
num = bitshift(num,-1);
b = bitshift(b,-1);
end;
```

说明: Nbit是逆转位是几位, num是逆转的数即变量。

三个循环, 第一个循环是进行N阶的FFT运算

第二个循环其实就, 每一阶FFT的时候, 有多少组DFT对象, 拿8点来说, 第一阶的时候, 有4组DFT对象, 到了第二阶, 就有2组, 到了第三, 就是最后一阶, 只有一组。

第三个循环, 其实是在每一组DFT里边, 执行多少次蝶形运算! 8点DIT FFT来说, 第一阶每组有一个蝶形, 第二阶每组有2个, 第三阶每组有4个蝶形。所以很容易得到三者的关系

i, j, k 三者, 反别表示三层循环, 然后得出循环次数的关系

stages = log2(PointNum)

i 从 0到stages - 1 !

j 从 0 到 $2^{stages-i}/2-1$ 其实就是 $2^{stages-i-1}-1$

k 从0 到 2^i-1

旋转因子W的选择:

因为根据8点DIT-FFT图, 从第一阶到最后一阶, 可以总结出一个规律:

都是 N是每组蝶形数据个数, 比如第一阶每组有2个元素, N就是2, 第二阶每组4个元素, N就是4等。然后x往往都是从0开始到N/2 - 1;

根据旋转因子的性质, 其实可以有每阶段每组都是: $W_N^{(2^i-x)}$

蝶形运算设计:

根据信号流图, 得出以下算式:

$$b_0 = a_0 + a_1 W_N^N$$

$$b_1 = a_0 - a_1 W_N^N$$

$$e^{-j\theta} = \cos\theta - j\sin\theta。$$

$$a_0 = \text{Re}(a_0) + j\text{Im}(a_0), \quad a_1 = \text{Re}(a_1) + j\text{Im}(a_1)。$$

$$\text{所以有 } a_1 w_N^m = (\text{Re}(a_1) + j\text{Im}(a_1)) * (\cos\theta - j\sin\theta)。$$

$$a_1 w_N^m = \text{Re}(a_1)\cos\theta - j\text{Re}(a_1)\sin\theta + j\text{Im}(a_1)\cos\theta + \text{Im}(a_1)\sin\theta。$$

所以有：

$$b_0 = \text{Re}(a_0) + \text{Re}(a_1)\cos\theta + \text{Im}(a_1)\sin\theta + j(\text{Im}(a_0) - \text{Re}(a_1)\sin\theta + \text{Im}(a_1)\cos\theta)$$

$$b_1 = \text{Re}(a_0) - \text{Re}(a_1)\cos\theta - \text{Im}(a_1)\sin\theta + j(\text{Im}(a_0) + \text{Re}(a_1)\sin\theta - \text{Im}(a_1)\cos\theta)。$$

从而设：

$$R1 = \text{Re}(a_1)\cos\theta$$

$$R2 = \text{Im}(a_1)\sin\theta$$

$$T1 = \text{Re}(a_1)\sin\theta$$

$$T2 = \text{Im}(a_1)\cos\theta$$

$$\text{有：} \quad b_0 = \text{Re}(a_0) + R1 + R2 + j(\text{Im}(a_0) - T1 + T2)$$

$$b_1 = \text{Re}(a_0) - R1 - R2 + j(\text{Im}(a_0) + T1 - T2)。$$

完成了蝶形运算！

全部的matlab程序有：

```
PointNum = 512;
PointBitNum = 9;

fs = 1024*2;
t = 0:1:PointNum - 1;
%for u = 1:1:PointNum;
sampletab = cos(2*pi*543*t/fs) + cos(2*pi*100*t/fs) + 0.2 + cos(2*pi*857*t/fs) + cos(2*pi*222*t/fs);
%end
zeros(1,PointNum);
sampletabl = sampletab;
index = 0;
for i = 1:PointNum
    k = i - 1
    index = bitreverse(PointBitNum,i - 1)
    sampletab(i) = sampletabl(index + 1);
end
%sampletabl
%sampletab
REX = sampletab;
IMX = zeros(1,PointNum);
i = 0; %T Loop for Log2N stages
j = 0; %T loop for leach sub-DFT
k = 0; %T Loop for each butterfly
stages = log2(PointNum);
for i = 0 : stages - 1
    lenNum = 0;
    for j = 0 : 2^(stages - (i + 1)) - 1
        for k = 0 : 2^i - 1
            R1 = REX(lenNum + 2^i + 1) * cos(2*pi*k*2^(stages - (i + 1))/PointNum);
```

推荐: 十八款村衫文艺搭配 那些不合群的人 文

```

R2 = IMX(lenNum + 2^i + 1) * sin(2*pi*k*2^(stages - (i + 1))/PointNum);
T1 = REX(lenNum + 2^i + 1) * sin(2*pi*k*2^(stages - (i + 1))/PointNum);
T2 = IMX(lenNum + 2^i + 1) * cos(2*pi*k*2^(stages - (i + 1))/PointNum);
REX(lenNum + 2^i + 1) = REX(lenNum + 1) - R1 - R2;
IMX(lenNum + 2^i + 1) = IMX(lenNum + 1) + T1 - T2;
REX(lenNum + 1) = REX(lenNum + 1) + R1 + R2;
IMX(lenNum + 1) = IMX(lenNum + 1) - T1 + T2 ;

lenNum = lenNum + 1;
endNum = lenNum + 2^i;

end
lenNum = endNum;
end
end

```

```

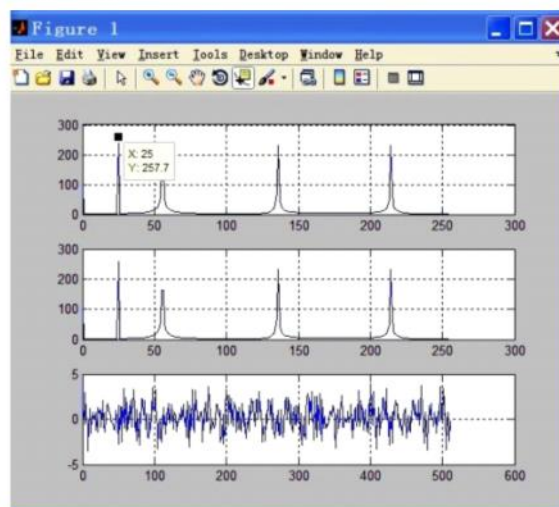
subplot(3,1,1);
fft(sampletab1, PointNum);
x1 = abs(fft(sampletab1, PointNum));
plot([0 : PointNum/2 - 1], x1(1:PointNum/2));
grid on
subplot(3,1,2);
% [REX IMX]
am = sqrt(abs(REX.*REX) + abs(IMX.*IMX));

plot(0:1:PointNum/2 - 1, am(1:PointNum/2));
grid on

subplot(3,1,3);
plot(t, sampletab);
grid on

```

我还做了与MATLAB原来带有的FFT做比较:
画出的图如下:



第一个是MATLAB自带的FFT函数频谱图
第二个是我自己设计的FFT频谱图
第三个是信号的时域波形

思想已经有了, 我以前也改过人家的FFT的程序但是不是很理解, 打算有机会用C语言实现定点FFT, 因为在嵌入式上多数用定点FFT, 相应的C++版本应该也会写。

推荐：十八款衬衫文艺搭配 那些“不合群”的人 ×
下面是网上的一些设计FFT的资料：

[新浪首页](#) [登录](#) [注册](#)

N点基-2 FFT算法的实现方法

从图4我们可以总结出对于点数为 $N=2^L$ 的DFT快速计算方法的流程：

1. 对于输入数据序列进行倒位序变换。

该变换的目的是使输出能够得到 $X(0) \sim X(N-1)$ 的顺序序列，同样以8点DFT为例，该变换将顺序输入序列 $x(0) \sim x(7)$ 变为如图4的 $x(0), x(4), x(2), x(6), x(1), x(5), x(3), x(7)$ 序列。其实现方法是：假设顺序输入序列一次存入 $A(0) \sim A(N-1)$ 的数组元素中，首先我们将数组下标进行二进制化(例：对于点数为8的序列只需要 $\text{LOG}_2(8) = 3$ 位二进制序列表示，序号6就表示为110)。二进制化以后就是将二进制序列进行倒位，倒位的过程就是将原序列从右到左书写一次构成新的序列，例如序号为6的二进制表示为110，倒位后变为了011，即使十进制的3。第三步就是将倒位前和倒位后的序号对应的数据互换。依然以序号6为例，其互换过程如下：

$\text{temp} = A(6); A(6) = A(3); A(3) = \text{temp};$

实际上考虑到执行效率，如果对于每一次输入的数据都需要这个处理过程是非常浪费时间的。我们可以采用指向指针的指针来实现该过程，或者是采用指针数组来实现该过程。

2. 蝶形运算的循环结构。

从图4中我们可以看到对于点数为 $N = 2^L$ 的fft运算，可以分解为L阶蝶形图级联，每一阶蝶形图内又分为M个蝶形组，每个蝶形组内包含K个蝶形。根据这一点我们就可以构造三阶循环来实现蝶形运算。编程过程需要注意旋转因子与蝶形阶数和蝶形分组内的蝶形个数存在关联。

3. 浮点到定点转换需要注意的关键问题

上边的分析都是基于浮点运算来得到的结论，事实上大多数嵌入式系统对浮点运算支持甚微，因此在嵌入式系统中进行离散傅里叶变换一般都应该采用定点方式。对于简单的DFT运算从浮点到定点显得非常容易。根据式(1)，假设输入 $x(n)$ 是经过AD采样的数字序列，AD位数为12位，则输入信号范围为 $0 \sim 4096$ 。为了进行定点运算我们将旋转因子实部虚部同时扩大 2^{12} 倍，取整数部分代表旋转因子。之后，我们可以按照(1)式计算，得到的结果与原结果成比例关系，新的结果比原结果的 2^{12} 倍。但是，对于使用蝶形运算的fft我们不能采用这种简单的放大旋转因子转为整数计算的方式。因为fft是一个非对称迭代过程，假设我们对旋转因子进行了放大，根据蝶形流程图我们可以发现其最终的结果是，不同的输入被放大了不同的倍数，对于第一个输入 $x(0)$ 永远也不会放大。举一个更加形象的例子，还是以图4为例。从图中可以看出右侧的 $X(0)$ 可以直接用下式表示：

$$X(2) = \left[\left(x(0) + W_N^0 x(4) \right) - W_N^0 \left(x(2) + W_N^0 x(6) \right) \right] + W_N^2 \left[\left(x(1) + W_N^0 x(5) \right) - W_N^0 \left(x(3) + W_N^0 x(7) \right) \right]$$

从上式我们可以看到不同输入项所乘的旋转因子个数(注意这里是指数，就算是 w_N^0 ，也被考虑进去了，因为在没有放大时 w_N^0 等于1，放大后所有旋转因子指数模均不为1，因此需要考虑)。这就导致输入不平衡，运算结果不正确。经查阅相关资料，比较妥善的做法是，首先对所有旋转因子都放大 2^Q 倍，Q必须要大于等于L，以保证不同旋转因子的差异化。旋转因子放大，为了保证其模为1，在每一次蝶形运算的乘积运算中我们需要将结果右移Q位来抵消这个放大，从而得到正确的结果。之所以采用放大倍数必须是2的整数次幂的原因也在于此，我们之后可以通过简单的右移位运算将之前的放大抵消，而右移位又代替了除法运算，大大节省了时间。

4. 计算过程中的溢出问题

最后需要注意的一个问题就是计算过程中的溢出问题。在实际应用中，AD虽然有12位的位宽，但是采样得到的信号可能较小，例如可能在 $0 \sim 8$ 之间波动，也就是说实际可能只有3位的情况。这种情况下为了在计算过程中不丢失信息，一般都需要先将输入数据左移P位进行放大处理，数据放大可能会导致溢出，从而使计算错误，而溢出的极限情况是这样：假设我们数据位宽为D位(不包括符号位)，AD采样位数B位，数字放大倍数P位，旋转因此放大倍数Q位，FFT级联运算带来的最大累加倍数L位。我们得到：

$$B + L + P + Q \leq D$$

假设AD位宽12，数据位宽32，符号位1位，因此有效位宽31位，采样点数N，那么我们可以得到 $\log_2(N) + P + Q \leq 19$ ，假设点数128，又 $Q=L$ 可以得到放大倍数 $P \leq 5$ 。

3 0
喜欢 赠金笔

分享：

阅读(1427) | 评论(1) | 收藏(0) | 转载(9) | 喜欢♥ | 打印 | 举报

已投稿到： 排行榜







推荐：十八款衬衫文艺搭配 那些“不合群”的人 ×
前一篇：《dinv into python》开始了解python
后一篇：Dive into python 第4章 自省的威力







[新浪首页](#) [登录](#) [注册](#)







评论 重要提示：警惕虚假中奖信息 [发评论]

ca8801
好，帮了大忙了。不过还是自己编不出来
2015-5-17 22:02 回复(0)

发评论

更多>>





登录名： 密码： 找回密码 注册 ☒ 记住登录状态

☐ 评论并转载此博文

发评论

以上网友发言只代表其个人观点，不代表新浪网的观点或立场。

< 前一篇 后一篇 >
《dinv into python》开始了解python Dive into python 第4章 自省的威力

新浪BLOG意见反馈留言板 不良信息反馈 电话：4006900000 提示音后按1键（按当地市话标准计费） 欢迎批评指正
新浪简介 | About Sina | 广告服务 | 联系我们 | 招聘信息 | 网站律师 | SINA English | 会员注册 | 产品答疑

Copyright © 1996 - 2017 SINA Corporation, All Rights Reserved
新浪公司 版权所有