老古开发网
技术文章和新闻

嵌入式系统中FFT算法研究

发布时间:2002年8月5日
来源:单片机与嵌入式系统应用　　作者:华东交通大学 肖宛昂

摘要 首先分析实数FFT算法的推导过程,然后给出一种具体实现FFT算法的C语言程序,可以直接应用于需要FFT运算的单片机或DSP等嵌入式系统中。

关键词 嵌入式系统 FFT算法 单片机 DSP

目前国内有关数字信号处理的教材在讲解快速傅里叶变换（FFT）时,都是以复数FFT为重点,实数FFT算法都是一笔带过,书中给出的具体实现程序多为BASIC或FORTRAN程序并且多数不能真正运行。鉴于目前在许多嵌入式系统中要用到FFT运算,如以DSP为核心的交流采样系统、频谱分析、相关分析等。本人结合自己的实际开发经验,研究了实数的FFT算法并给出具体的C语言函数,读者可以直接应用于自己的系统中。

1　倒位序算法分析

按时间抽取（DIT）的FFT算法通常将原始数据倒位序存储,最后按正常顺序输出结果X（0）,X（1）,...,X（k）,...。假设一开始,数据在数组 float dataR[128]中,我们将下标i表示为（b6b5b4b3b2b1b0）b,倒位序存放就是将原来第i个位置的元素存放到第（b0b1b2b3b4b5b6）b的位置上去.由于C语言的位操作能力很强,可以分别提取出b6、b5、b4、b3、b2、b1、b0,再重新组合成b0、b1、b2、b3、b4、b5、b6,即是倒位序的位置。程序段如下（假设128点FFT）:

```
/* i为原始存放位置,最后得invert_pos为倒位序存放位置 */
    int b0=b1=b2=b3=b4=b5=6=0;
    b0=i&0x01; b1=(i/2)&0x01; b2=(i/4)&0x01;
    b3=(i/8)&0x01; b4=(i/16)&0x01; b5=(i/32)&0x01;
    b6=(i/64)&0x01; /*以上语句提取各比特的0、1值*/
    invert_pos=x0*64+x1*32+x2*16+x3*8+x4*4+x5*2+x6;
```

大家可以对比教科书上的倒位序程序,会发现这种算法充分利用了C语言的位操作能力,非常容易理解而且位操作的速度很快。

2　实数蝶形运算算法的推导

我们首先看一下图1所示的蝶形图。

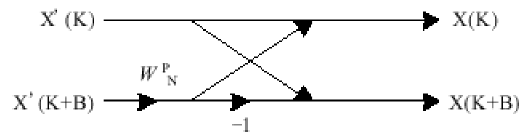

图1

蝶形公式：

X(K) = X’(K) + X’(K+B)W PN ,

X(K+B) = X’(K) - X’(K+B) W PN

其中W PN= cos (2πP/N) - jsin (2πP/N)。

设 X(K+B) = XR(K+B) + jXI(K+B),

X(K) = XR(K) + jXI(K) ,

有：

XR(K)+jXI(K)= XR’(K)+jXI’(K)+[XR’(K+B) + jXI’(K+B)]*[cos (2πP/N) - jsin (2πP/N)];

继续分解得到下列两式：

XR(K)= XR’(K)+ XR’(K+B) cos (2πP/N) + XI’(K+B) sin (2πP/N) (1)

XI(K)= XI’(K)-XR’(K+B) sin (2πP/N) +XI’(K+B)cos (2πP/N) (2)

需要注意的是：XR(K)、XR’(K)的存储位置相同,所以经过（1)、(2）后,该位置上的值已经改变,而下面求X(K+B)要用到X’(K),因此在编程时要注意保存XR’(K)和XI’(K)到TR和TI两个临时变量中。

同理：XR(K+B)+jXI(K+B)= XR’(K)+jXI’(K)- [XR’(K+B)+jXI’(K+B)] *[cos (2πP/N) -jsin (2πP/N)]继续分解得到下列两式：

XR(K+B)= XR’(K)-XR’(K+B) cos (2πP/N) - XI’(K+B) sin (2πP/N) (3)

XI(K+B)= XI’(K)+ XR’(K+B) sin (2πP/N) - XI’(K+B) cos (2πP/N) (4)

注意：

① 在编程时，式（3)、(4）中的XR’(K)和 XI’(K)分别用TR和TI代替。

② 经过式（3）后，XR(K+B)的值已变化,而式（4）中要用到该位置上的上一级值，所以在执行式（3）前要先将上一级的值XR’(K+B)保存。

③ 在编程时，XR(K)和 XR’(K)，XI(K)和 XI’(K)使用同一个变量。

通过以上分析,我们只要将式（1)、(2)、(3)、(4）转换成C语言语句即可。要注意变量的中间保存,详见以下程序段。

```
/* 蝶形运算程序段 ,dataR[]存放实数部分,dataI[]存放虚部*/
/* cos、sin函数做成表格,直接查表加快运算速度 */
TR=dataR[k]; TI=dataI[k]; temp=dataR[k+b];/*保存变量,供后面语句使用*/
dataR[k]=dataR[k]+dataR[k+b]*cos_tab[p]+dataI[k+b]*sin_tab[p];
```

```
dataI[k]=dataI[k]-dataR[k+b]*sin_tab[p]+dataI[k+b]*cos_tab[p];
dataR[k+b]=TR-dataR[k+b]*cos_tab[p]-dataI[k+b]*sin_tab[p];
dataI[k+b]=TI+temp*sin_tab[p]-dataI[k+b]*cos_tab[p];
```

3 DIT FFT 算法的基本思想分析

我们知道N点FFT运算可以分成LOGN2 级,每一级都有N/2个碟形。DIT FFT的基本思想是用3层循环完成全部运算(N点FFT)。

第一层循环：由于N=2m需要m级计算,第一层循环对运算的级数进行控制。

第二层循环：由于第L级有2L-1个蝶形因子（乘数）,第二层循环根据乘数进行控制,保证对于每一个蝶形因子第三层循环要执行一次,这样,第三层循环在第二层循环控制下,每一级要进行2L-1次循环计算。

第三层循环：由于第L级共有N/2L个群,并且同一级内不同群的乘数分布相同,当第二层循环确定某一乘数后,第三层循环要将本级中每个群中具有这一乘数的蝶形计算一次,即第三层循环每执行完一次要进行N/2L个碟形计算。

可以得出结论：在每一级中,第三层循环完成N/2L个碟形计算;第二层循环使第三层循环进行 2L-1次,因此,第二层循环完成时,共进行2L-1 *N/2L=N/2个碟形计算。实质是：第二、第三层循环完成了第L级的计算。

几个要注意的数据：

① 在第L级中,每个碟形的两个输入端相距b=2L-1个点。
② 同一乘数对应着相邻间隔为2L个点的N/2L个碟形。
③ 第L级的2L-1个碟形因子WPN 中的P,可表示为p = j*2m-L,其中j = 0,1,2,...,(2L-1-1)。

以上对嵌入式系统中的FFT算法进行了分析与研究。读者可以将其算法直接应用到自己的系统中,欢迎来信共同讨论。(Email:xiaowanang@163.net)

附128点DIT FFT函数：

```
/* 采样来的数据放在dataR[ ]数组中,运算前dataI[ ]数组初始化为0 */
void FFT(float dataR[],float dataI[])
    {int x0,x1,x2,x3,x4,x5,x6;
    int L,j,k,b,p;
    float TR,TI,temp;
    /********** following code invert sequence ************/
    for(i=0;i<128;i++)
        { x0=x1=x2=x3=x4=x5=x6=0;
        x0=i&0x01;              x1=(i/2)&0x01;             x2=(i/4)&0x01;
```

```
x3=(i/8)&0x01;x4=(i/16)&0x01; x5=(i/32)&0x01; x6=(i/64)&0x01;
        xx=x0*64+x1*32+x2*16+x3*8+x4*4+x5*2+x6;
        dataI[xx]=dataR[i];
        }
    for(i=0;i<128;i++)
        { dataR[i]=dataI[i]; dataI[i]=0; }
    /************** following code FFT *******************/
    for(L=1;L<=7;L++) { /* for(1) */
        b=1; i=L-1;
        while(i>0)
            {b=b*2; i--;} /* b= 2^(L-1) */
        for(j=0;j<=b-1;j++) /* for (2) */
        { p=1; i=7-L;
            while(i>0) /* p=pow(2,7-L)*j; */
            {p=p*2; i--;}
            p=p*j;
            for(k=j;k<128;k=k+2*b) /* for (3) */
        { TR=dataR[k]; TI=dataI[k]; temp=dataR[k+b];
            dataR[k]=dataR[k]+dataR[k+b]*cos_tab[p]+dataI[k+b]*sin_tab[p];
            dataI[k]=dataI[k]-dataR[k+b]*sin_tab[p]+dataI[k+b]*cos_tab[p];
            dataR[k+b]=TR-dataR[k+b]*cos_tab[p]-dataI[k+b]*sin_tab[p];
            dataI[k+b]=TI+temp*sin_tab[p]-dataI[k+b]*cos_tab[p];
        } /* END for (3) */
    } /* END for (2) */
} /* END for (1) */
for(i=0;i<32;i++){ /* 只需要32次以下的谐波进行分析 */
    w[i]=sqrt(dataR[i]*dataR[i]+dataI[i]*dataI[i]);
    w[i]=w[i]/64;}
    w[0]=w[0]/2;
} /* END FFT */
```