目录

[实验一 快速傅里叶变换（FFT）算法实验 3](#_Toc515299439)

[一．实验目的 3](#_Toc515299440)

[二．实验设备 3](#_Toc515299441)

[三．基本原理 3](#_Toc515299442)

[四．实验步骤 3](#_Toc515299443)

[五．实验流程图 4](#_Toc515299444)

[六．主要代码内容 5](#_Toc515299445)

[七．运行结果 6](#_Toc515299446)

[实验二 有限冲击响应滤波器（FIR）算法实验 8](#_Toc515299447)

[一．实验目的 8](#_Toc515299448)

[二．实验设备 8](#_Toc515299449)

[三．实验原理 8](#_Toc515299450)

[四．实验步骤 8](#_Toc515299451)

[五．实验流程图 9](#_Toc515299452)

[六．主函数程序代码 9](#_Toc515299453)

[七．各类参数过程及结果输出截图 11](#_Toc515299454)

[八．心得体会 13](#_Toc515299455)

[实验三 无限冲击响应滤波器（IIR）算法实验 14](#_Toc515299456)

[一．实验目的 14](#_Toc515299457)

[二．实验设备 14](#_Toc515299458)

[三．实验原理 14](#_Toc515299459)

[四．实验步骤 14](#_Toc515299460)

[五．实验流程图 15](#_Toc515299461)

[六．主要程序代码 15](#_Toc515299462)

[七．输入数据和输出数据的对比 18](#_Toc515299463)

[八．心得体会 19](#_Toc515299464)

[实验四 双音多频信号的产生与检测 19](#_Toc515299465)

[一、设计任务书 19](#_Toc515299466)

[二、设计内容 20](#_Toc515299467)

[三、设计方案、算法原理说明 20](#_Toc515299468)

[四、程序主要代码 23](#_Toc515299469)

[五. 结果对比图 28](#_Toc515299470)

[六．心得体会 29](#_Toc515299471)

# 实验一 快速傅里叶变换（FFT）算法实验

## 一．实验目的

加深对DFT算法原理和基本性质的理解；熟悉FFT算法原理和FFT子程序的应用；学习用FFT对连续信号和时域信号进行谱分析，了解可能出现的分析误差及其原因，以便在实际中正确应用FFT。

## 二．实验设备

计算机，CCS3.1版软件。

## 三．基本原理

1. 离散傅里叶变换DFT的定义：将时域周期性采样变换成频域周期性离散函数，频域的采样也可以变换成时域的周期性离散函数，将这样的变换称为离散傅里叶变换，简称DFT。
2. FFT是DFT的一种快速算法，将DFT的N2步运算减少为（N/2）log2N步，极大地提高了运算的速度。
3. 旋转因子的变化规律。
4. 蝶形运算规律。
5. 基2FFT运算。

## 四．实验步骤

1. 复习DFT的定义、性质和用DFT做谱分析的有关内容；
2. 复习FFT算法原理及编程思想，并对照DIF-FFT运算流程图和程序框图，了解本实验提供的FFT子程序；
3. 阅读本实验所提供的样例子程序；
4. 运行CSS3.3软件，对样例程序进行跟踪，分析结果；记录必要的数据。

## 五．实验流程图

初始化

输入数组排序

计算第一层中间值

计算层数

计算对应层步长

计算各层中间结果

计算层数=0？

N

Y

计算结果输出

## 六．主要代码内容

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Filename: rfft\_t.c

// Version: 0.01

// Description: test for cfft routine

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <math.h>

#include <tms320.h>

#include <dsplib.h>

#include "test.h"

short i;

void main(void)

{

/\* compute \*/

while(1)

{

cbrev(x,y,NX/2);

rfft(y,NX,1);

for(i=0;i<NX/2;i++)

{

w[i]=y[i]; w[i+32]=y[i+32];

z[i]=(int)sqrt((long)y[2\*i]\*y[2\*i]+(long)y[2\*i+1]\*y[2\*i+1]);

}

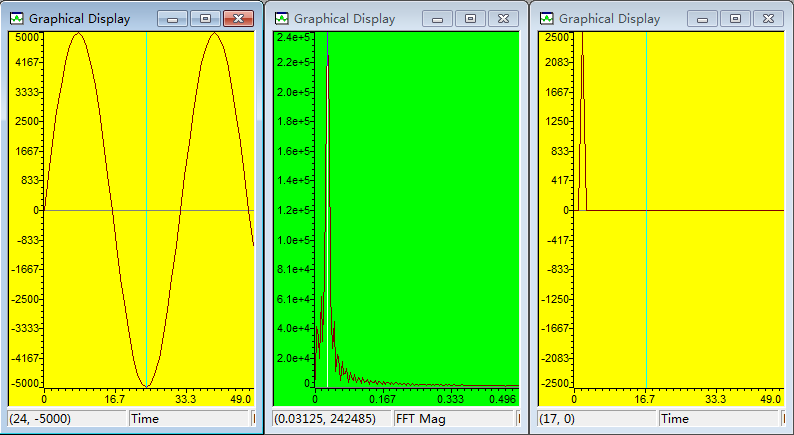
}

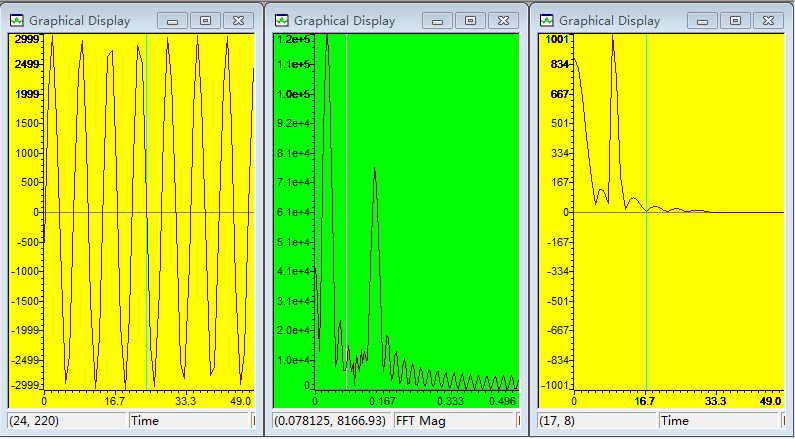
/\* test \*/

}

## 七．运行结果

输入为两个不同频率的正弦波，可见其FFT变换的结果与其频谱的对比。

****

****

可见FFT变换能够快速得出信号的频谱图，并具有较高的准确性。

## 八．心得体会

FFT是快速傅里叶变换的简称。在信号处理的过程中，我们通常需要得到信号的频谱以便进行处理，FFT提供了一个十分简便快捷的方法。

FFT能够减少乘法和加法的运算次数，使运算更加迅速。在实验中，代码主要实现了将输入序列重排序分组，分别将每组进行傅里叶变换，然后将各组数据交叉运算迭代，最后得出结果。

# 实验二 有限冲击响应滤波器（FIR）算法实验

## 一．实验目的

掌握用窗函数法设计FIR数字滤波器的原理和方法； 熟悉线性相位FIR数字滤波器特性；了解各种窗函数对滤波器特性的影响。

## 二．实验设备

计算机、CSS3.3版软件。

## 三．实验原理

有限冲击响应数字滤波器的基础理论； 模拟滤波器原理（巴特沃斯滤波器、切比雪夫滤波器、椭圆滤波器、贝塞尔滤波器）；数字滤波器洗漱的确定方法。

## 四．实验步骤

1.复习如何设计FIR数字滤波；阅读本实验原理，掌握设计步骤；

2.阅读本实验所提供的样例子程序；

3.运行CCS软件，对样例程序进行跟踪，分析结果。

## 五．实验流程图

结束

构建FIR滤波器进行滤波

调用FIR滤波器参数计算子程序各参数

构建滤波器频域特性波形

输入低通滤波器各频率参数

初始化

## 六．主函数程序代码

#include "volume.h"

int inp\_buffer[BUF\_SIZE]; /\* BUF\_SIZE的定义见volume.h \*/

int out\_buffer[BUF\_SIZE];

int \*input;

int \*output;

int volume = 2;

struct PARMS str =

{

2934,9432,213,9432,&str

};

const int BL = 32;

const int B[32] = {

51, -61, 53, 0, -119, 280, -397, 339, 0,

-617, 1338, -1809, 1555, 0, -3996, 19766, 19766, -3996,

0, 1555, -1809, 1338, -617, 0, 339, -397, 280,

-119, 0, 53, -61, 51

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\* NAME: main() \*/

/\* \*/

/\* FUNCTION: Volume Control: Read input buffer \*/

/\* multiply by volume coeficient and write to \*/

/\* output buffer. \*/

/\* \*/

/\* PARAMETERS: none. \*/

/\* \*/

/\* RETURN VALUE: none. \*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

main()

{

int num = BUF\_SIZE;

int a,i;

long int sum;

while(TRUE) /\* loop forever \*/

{

input = &inp\_buffer[0];

for (a =0;a < BUF\_SIZE; a++){

sum = 0;

for (i=0;i < BL;i++){

sum += (inp\_buffer[a-i<0?a-i+BUF\_SIZE:a-i] \* B[i]);

}

output[a] = (int) (sum >> 15);

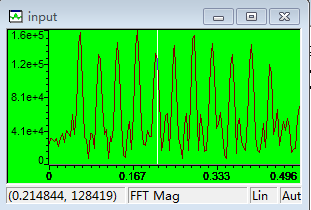
}

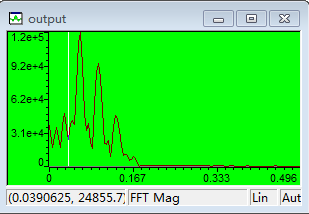
}

}

## 七．各类参数过程及结果输出截图

下面是输入图像频谱以及输出图像频谱。可见实现了低通滤波。我们输入了一个11个正弦波（频率由500Hz到3500Hz，每间隔300Hz一个，共11个）的组合数据序列，使用一个截止频率为1000Hz的低通滤波器进行滤波，可见得到的结果图中只剩下了500，800和1100Hz的少量频率分量。





关于输入序列

我们使用matlab并使用以下代码生成了浮点数格式的数据文件（160个采样点）：

t=0:0.000125:0.02;

y=sin(2\*pi\*500\*t)+sin(2\*pi\*800\*t)+sin(2\*pi\*1100\*t)+sin(2\*pi\*1400\*t)+sin(2\*pi\*1700\*t)+sin(2\*pi\*2000\*t)+sin(2\*pi\*2300\*t)+sin(2\*pi\*2600\*t)+sin(2\*pi\*2900\*t)+sin(2\*pi\*3200\*t)+sin(2\*pi\*3500\*t);plot(t,y)

q=quantizer([16,15]);

t=num2hex(q,y);

fid=fopen('input.dat','w');

fprintf(fid,'1651 1 0 1 0\r\n');

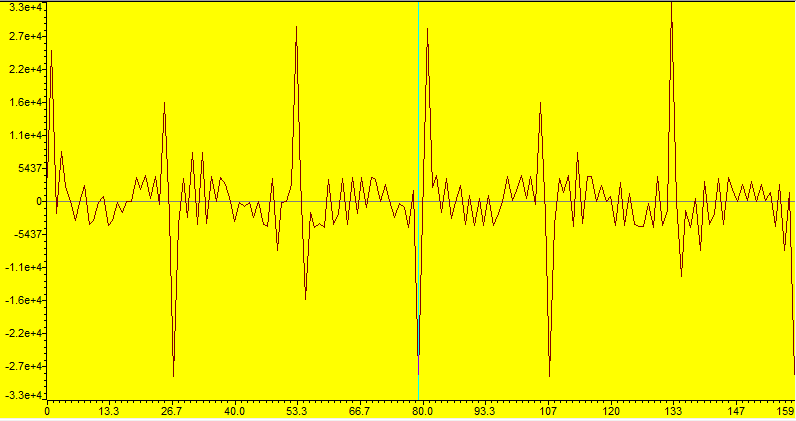
for i=1:160

fprintf(fid,'0x%s%s%s%s\r\n',t(i),t(i+160),t(i+320),t(i+480));

end

fclose(fid)

得到的波形大致如下：

****

## 八．心得体会

FIR滤波器是有限冲击响应滤波器。对于特定参数和特定类型的滤波器，我们只需使用matlab的fdatool工具进行设计并得到其序列即可。

得到序列以后，我们使用代码将输入序列与滤波器的序列进行卷积，即可得到结果。在卷积时，我刚开始理解为了循环卷积，也可以得到结果，但事实证明前面的几个点是错误的。在设计时，我们也可以使用平台自带的函数来实现。

# 实验三 无限冲击响应滤波器（IIR）算法实验

## 一．实验目的

熟悉设计IIR数字滤波器的原理与方法;掌握数字滤波器的计算机仿真方法；通过观察对实际信号的滤波作用，获得对数字滤波的感性认识。

## 二．实验设备

计算机、CSS3.3软件。

## 三．实验原理

1.无限冲击响数字滤波器的基础理论；

2.数字滤波器原理（巴特沃斯滤波器.切比雪夫滤波器、贝塞尔滤波器）；

3.双线性变换的设计原理。

## 四．实验步骤

1.复习有关巴特沃斯滤波器设计和用双线性变换法设计IIR数字滤波器的知识；

2.阅读本实验所提供的样例子程序；

3.运行CCS软件，对样例程序进行跟踪，分析结果；

4.样例程序实验操作说明。

## 五．实验流程图

初始化

滤波器参数nlpass,nlstop

调用低通滤波器设计子程序，完成滤波器a和b参数设计

调用滤波器滤波子程序对输入信号滤波

结束

## 六．主要程序代码

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Filename: test.h

// Version: 0.01

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#define NX 128

#define NBIQ 3

#define FNAME "t3"

#define MAXERROR 10

DATA x[NX];

#pragma DATA\_SECTION (h,".coeffs")

DATA h[5\*NBIQ] ={ /\* C54x: a1/2 a2 b2 b0 b1 ... \*/

-8347.5,

0,

0,

8035,

8035,

-22434,

22693,

2064,

2064,

4128,

17965.5,

11647,

2720,

2720,

5440,

};

DATA rtest[NX] ={

183,

-287,

523,

-55,

35,

164,

224,

-261,

-232,

334,

-1030,

1373,

-1113,

396,

-161,

693,

};

#pragma DATA\_SECTION (dbuffer,".dbuffer")

DATA dbuffer[2\*NBIQ];

DATA \*dp = dbuffer;

DATA r[NX];

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Filename: iir5\_t.c

// Version: 0.01

// Description: test for iircas5 routine

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <math.h>

#include <stdio.h>

#include <tms320.h>

#include <dsplib.h>

#include "test.h"

// generic variables

short i;

short eflag = PASS; // error flag or index into r vector where error

void main(void)

{

while(1){

/\* clear \*/

for (i=0;i<NX;i++) r[i] =0; // clear output buffer (optional)

for (i=0; i<2\*NBIQ; i++) dbuffer[i] = 0; // clear delay buffer (a must)

/\* compute \*/

iircas5(x,h,r,&dp,NBIQ, NX);

/\* test \*/

eflag = test(r, rtest, NX, MAXERROR);

if (eflag != PASS)

{

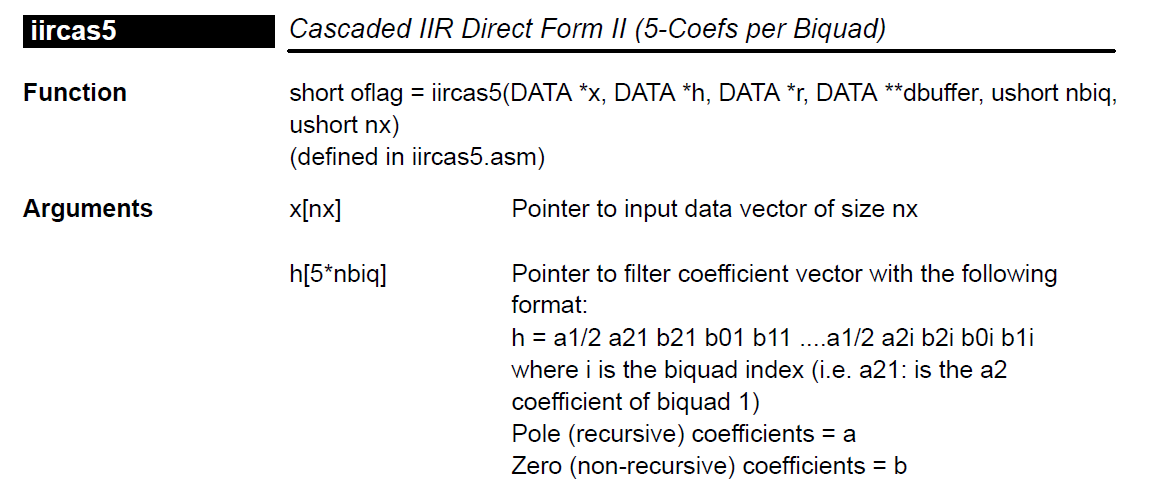
exit(-1);

}

return;

}

平台自带的Iircas5函数语句定义：

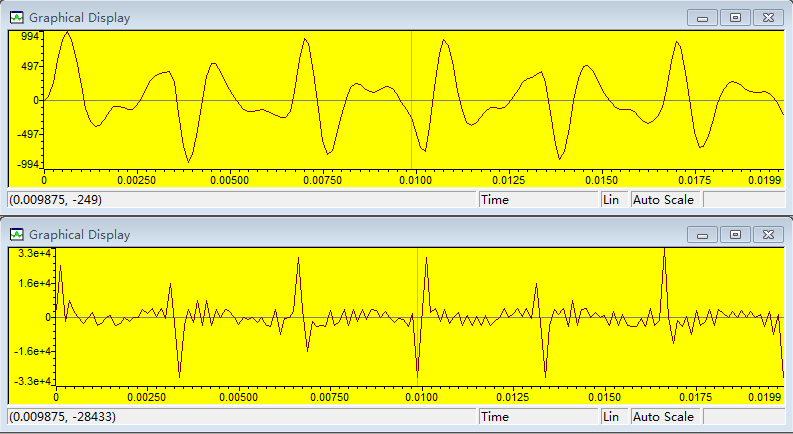


根据定义和生成的正弦函数波形文件，可以使用Matlab计算出巴特沃斯滤波器的所需的二阶结数据。

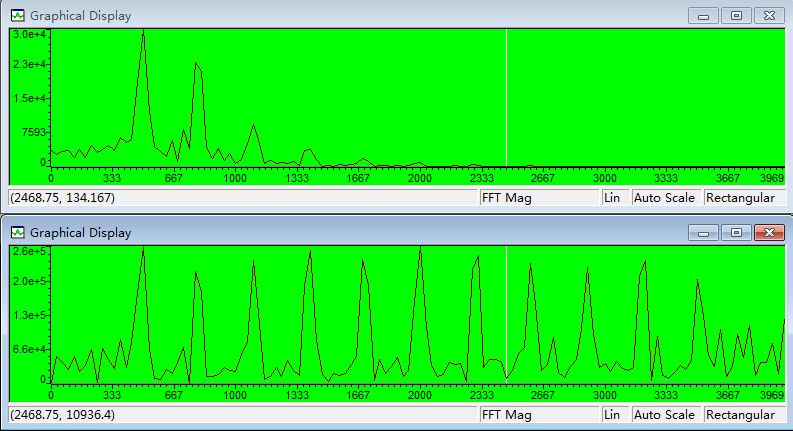
## 七．输入数据和输出数据的对比

我们同样使用实验二生成的11支正弦波的数据文件作为输入，使用fdatool工具生成6阶的截止频率为1000Hz的低通滤波器。

时域（输出/输入）



频域（输出/输入）



从时域波形中可见信号中的高频分量被消除了，体现为信号的毛刺减少了；从频域中可见频率高于1000Hz的信号分量基本被滤除。

## 八．心得体会

在进行IIR滤波时，我们直接使用了平台自带的Iircas5函数。该函数要求提供输入数据，滤波器数据（其中各个参数的排列顺序也有一定的规则，其中每个二阶节提供5个参数），滤波器的阶数，输入数据数组的大小。

特别需要注意的是，我们使用Iircas5函数时，输入的数组都是整形的，所以如果matlab生成的数据是浮点型，我们就需要将其转化为整形。还需要考虑溢出的问题。

# 实验四 双音多频信号的产生与检测

## 一、设计任务书

要求完成的任务：

（1）编写C语言程序，并在CCS集成开发环境下调试通过。

（2）实现设计所要求的各项功能。

（3）按要求撰写设计报告。

要求达到的目的：

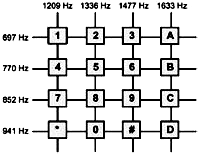
1. (1) 熟悉CCS的编程环境和基本试验调试流程和方法；
2. (2) 熟悉并掌握使用C语言编写较为复杂的程序并用CCS实现；
3. (3) 了解并掌握DTMF信号的产生和检测远离核试验流程；
4. (4) 熟练使用软件CCS5000对程序的完整调试过程。

## 二、设计内容

（1）使用C语言编写DTMF信号的产生程序，要求循环产生0~9、\*、#、A、B、C、D对应的DTMF信号，并且符合CCITT对DTMF信号规定的指标。

（2）使用C语言编写DTMF信号的检测程序，通过Goertzel算法在输入信号中提取频谱信息，通过对检测结果作有效性检查来识别各DTMF信号所对应对应的数字按键信息，并循环显示检测到的数字按键号。

## 三、设计方案、算法原理说明

双音多频DTMF（Dual Tone Multi Frequency）逐渐在全世界范围内使用在按键式电话机上，因其提供更高的拨号速率，迅速取代了传统转盘式电话机使用的拨号脉冲信令。作为实现电话号码快速可靠传输的一种技术，它具有很强的抗干扰能力和较高的传输速度，因此，可广泛用于电话通信系统中。但绝大部分是用作电话的音频拨号。另外，它也可以在数据通信系统中广泛地用来实现各种数据流和语音等信息的远程传输。 近年来DTMF也应用在交互式控制中，诸如语言菜单、语言邮件、电话银行和ATM 终端等。将DTMF信令的产生与检测集成到任一含有数字信号处理器(DSP)的系统中，是一项较有价值的工程应用。一个DTMF 信号由两个频率的音频信号叠加构成。这两个音频信号的频率分别来自两组预定义的频率组：行频组和列频组。每组分别包括4 个频率，分别抽出一个频率进行组合就可以组成16 种DTMF 编码，分别记作0~9、\*、#、A、B、C、D。如下图所示：

要用DSP 产生DTMF 信号，只要通过两个正弦波叠加在一起即可；DTMF 检测时采用改进的Goertzel 算法，从频域搜索两个正弦波的存在。

（一）DTMF信号的产生

DTMF编码器基于两个二阶数字正弦波振荡器，一个用于产生行频，一个用于产生列频。向DSP装入相应的系数和初始条件，就可以只用两个振荡器产生所需的八个音频信号。典型的DTMF信号频率范围是700～1700Hz，选取8000Hz作为采样频率，即可满足Nyquist条件（Nyquist定理：为了正确判定信号频率，信号在一个周期内至少被采样两次。）

正弦波是任何波形构成的基本单元，产生正弦波的方法一般有：采样回放法、查表法、泰勒级数展开法、数字正弦振荡器法。

我们用的是sin函数产生离散的正弦值，因为这种方法我们比较熟悉而且通过查阅资料这种方法的也能达到误差要求，生成DTMF的公式为：buffer[t]=sin(t\*2\*pi\*f1/fs**)+**sin(t\*2\*pi\*f2/fs**)，**其中t为采样序数，由0开始递增；f1，f2为生成DTMF信号的两个正弦波的频率；fs为采样频率；buffer[t]为序数t时的得出的采样值。将这些数据转换为Q15格式然后通过codec发送出去。

　　CCITT对DTMF信号规定的指标是，传送/接收率为每秒10个数字，即每个数字100ms。代表数字的音频信号必须持续至少45ms，但不超过55ms。100ms内其他时间为静音，以便区别连续的两个按键信号。我们使用8000Hz的采样频率（电话信号的典型抽样频率为=8kHZ），即1秒采样8000个点，则100ms采样800个点，我们设置800个点的缓存，其中用400个存产生的DTMF信号值，即音频信号必须持续50ms，另外400个存0值，即静音信号。

（二）DTMF信号的检测

DTMF检测是对进入解码端的信号进行检测，并把双音频信号转换成对应的数字信息。它是一个比DTMF产生更加复杂过程。由于数据流是连续的，为了保证DTMF检测的实时性，因此要求检测过程必须是实时连续的。

在输入信号中检测DTMF 信号，需要在输入的数据信号流中连续地搜索DTMF 信号频谱的存在。整个检测过程分两步：首先采用Goertzel 算法在输入信号中提取频谱信息；接着作检测结果的有效性检查。

DTMF 解码时在输入信号中搜索出有效的行频和列频。计算数字信号的频谱可以采用DFT 及其快速算法FFT，而在实现DTMF 解码时，采用Goertzel 算法要比FFT 更快。通过FFT 可以计算得到信号所有谱线，了解信号整个频域信息，而对于DTMF 信号只需关心其8 个行频/列频及其二次谐波信息即可，二次谐波的信息用于将DTMF 信号与声音信号区别开。此时Goertzel 算法能更加快速的在输入信号中提取频谱信息。Goertzel 算法实质是一个两极点的IIR 滤波器。

Goertzel算法简介

DTMF检测器的核心是Goertzel算法。该算法利用二极点的IIR滤波器计算离散傅立叶变换值，能够快速高效地提取输入信号的频谱信息。由于IIR滤波器是一个递归结构，它利用只有一个实系数的差分方程进行操作，并不像DFT或FFT算法那样需要计算数据块，而是每输入一个样值就执行一次算法。

完成时域到频域的变换可以用离散傅立叶变换(DFT)或快速傅立叶变(FFT)．FFT计算出所有点频率，而DFT可以只计算感兴趣的频率点．如果要计算的频率点数少于log 2N(N为输入信号点数)，采用DFT的计算速度比FFT更快。

直接计算DFT，需要很多复系数，即使只计算一点的DFT也需要N个复系数．采用数字信号处理中的Goertzel算法，则可明显地提高速度。

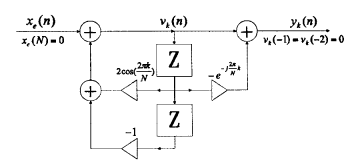


图3.2 Goertzel算法原理框图

利用二阶复共轭极点可以得到只有一个实系数的差分方程：

（1）

在实际的DTMF检测中，只需DFT的幅度（本算法为平方幅度）信息就足够了，因此在Goertzel滤波器中，当N点（相当于DFT数据块的长度）样值输入滤波器后，滤波器输出伪DFT值vk（n），由vk（n）即可确定频谱的平方幅度。

（2）其中k=f\*N/fs，当N取值为125时， k的取值经计算如下表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 信号频率(Hz) | 计算值k | 取整值k | 绝对误差 | 相对误差 |
| 697 | 10.890625 | 11 | 0.109375 | 1.00430% |
| 770 | 12.03125 | 12 | 0.03125 | 0.25974% |
| 852 | 13.3125 | 13 | 0.3125 | 2.34741% |
| 941 | 14.703125 | 15 | 0.296875 | 2.01912% |
| 1209 | 18.890625 | 19 | 0.109375 | 0.57899% |
| 1336 | 20.875 | 21 | 0.125 | 0.59880% |
| 1477 | 23.078125 | 23 | 0.078125 | 0.33852% |
| 1633 | 25.515625 | 25 | 0.515625 | 2.02082% |

## 四、程序主要代码

/\* test.h \*/

#define NX 64

#define NBIQ 1

#define FNAME "t3"

#define MAXERROR 10

DATA x[NX] ={

16384,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

};

#pragma DATA\_SECTION (h,".coeffs")

DATA h1[4][5] ={ /\* C54x: a1 a2 b2 b0 b1 ... \*/

{-27978,

32767,

0,

17052,

0},

{-26956,

32767,

0,

18631,

0},

{-25701,

32767,

0,

20325,

0},

{-24218,

32767,

0,

22072,

0}

};

DATA h2[4][5] ={ /\* C54x: a1 a2 b2 b0 b1 ... \*/

{-19072,

32767,

0,

26643,

0},

{-16325,

32767,

0,

28414,

0},

{-13084,

32767,

0,

30041,

0},

{-9314,

32767,

0,

31627,

0}

};

DATA h3[5] ={ /\* C54x: a1 a2 b2 b0 b1 ... \*/

-19072,

32767,

0,

32767,

0

};

DATA rtest[NX] ={

183,

-287,

523,

-55,

35,

164,

224,

-261,

-232,

334,

-1030,

1373,

-1113,

396,

-161,

693,

};

#pragma DATA\_SECTION (dbuffer,".dbuffer")

DATA dbuffer[2\*NBIQ];

DATA \*dp = dbuffer;

DATA r1[NX];

DATA r2[NX];

DATA r3[NX];

/\* IIR5\_T.c \*/

#include <math.h>

#include <stdio.h>

#include <tms320.h>

#include <dsplib.h>

#include "test.h"

short i,j,k,m,n,h,f,p;

short s[NX];

float c,b,rr[8],l,q;

short eflag = PASS;

char a;

float kn[8]={1.7077,1.6453,1.5687,1.4782,1.1641,0.9964,0.7986,0.5685};

short km[8]={-27979

,-26955

,-25701

,-24218

,-19072

,-16324

,-13084

,-9315};

void main(void)

{

while(1)

{

for(k=0;k<4;k++)

{

for(j=0;j<4;j++)

{

for (m=0;m<NX;m++)

{

r1[m] =0;

r2[m] =0;

}

for (n=0; n<2\*NBIQ; n++) dbuffer[n] = 0;

for (i=0;i<NX;i++) r1[i] =0;

iircas5(x,h1[k],r1,&dp,1, NX);

for (n=0; n<2\*NBIQ; n++) dbuffer[n] = 0;

for (i=0;i<NX;i++) r2[i] =0;

iircas5(x,h2[j],r2,&dp,1, NX);

for(i=0;i<NX;i++)

{

r3[i]=(r1[i]/32+r2[i]/32); }

//for(i=0;i<20000;i++);

/\*-----------\*/

for(p=0;p<8;p++)

{

// s[0]=(float)(r3[0])/32767;

// s[1]=(float)(r3[1])/32767+kn[p]\*s[0];

// for(i=2;i<NX;i++)

// {

// s[i]=kn[p]\*s[i-1]-s[i-2]+(float)(r3[i])/32767;

// }

for (n=0; n<2\*NBIQ; n++) dbuffer[n] = 0;

for (i=0;i<NX;i++) s[i] =0;

h3[0]=km[p];

iircas5(r3,h3,s,&dp,1, NX);

c=(float)(s[NX-2])/32767;

b=(float)(s[NX-1])/32767;

rr[p]=(b\*b)+(c\*c)-(kn[p]\*c\*b);//幅值的平方

}

//赋初值

l=0;//行幅值

q=0;//列幅值

h=0;//行

f=0;//列

for(p=0;p<4;p++)

{

if (rr[p]>l)

{

l=rr[p];

h=p;

}//取最大幅值出频率

if (rr[p+4]>q)

{

q=rr[p+4];

f=p;

}

}

if(h==0 & f==0) a='1';

if(h==0 & f==1) a='2';

if(h==0 & f==2) a='3';

if(h==0 & f==3) a='A';

if(h==1 & f==0) a='4';

if(h==1 & f==1) a='5';

if(h==1 & f==2) a='6';

if(h==1 & f==3) a='B';

if(h==2 & f==0) a='7';

if(h==2 & f==1) a='8';

if(h==2 & f==2) a='9';

if(h==2 & f==3) a='C';

if(h==3 & f==0) a='\*';

if(h==3 & f==1) a='0';

if(h==3 & f==2) a='#';

if(h==3 & f==3) a='D';

}

}

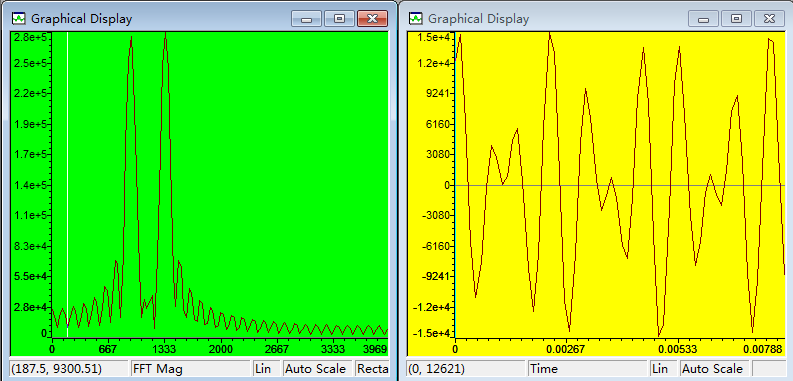
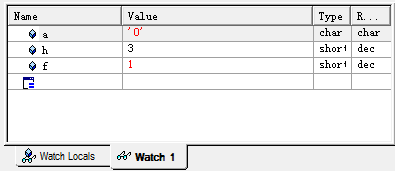
}

}

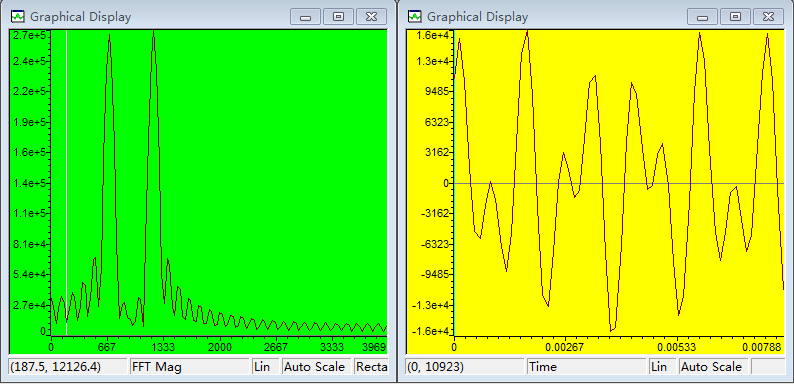
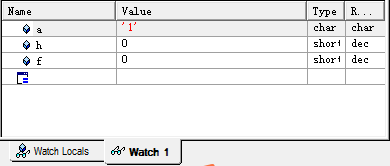
## 五. 结果对比图

对于四行四列的数字键，前四个正弦波表示行数，后四个正弦波表示列数。对于相应的信号，我们只需检测其频谱八个对应点上的值，即可检测出按下的是哪个键。

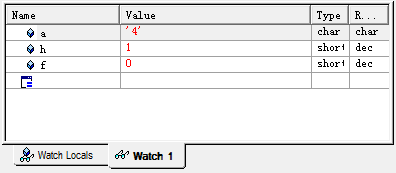
数字键“0”的产生与检测

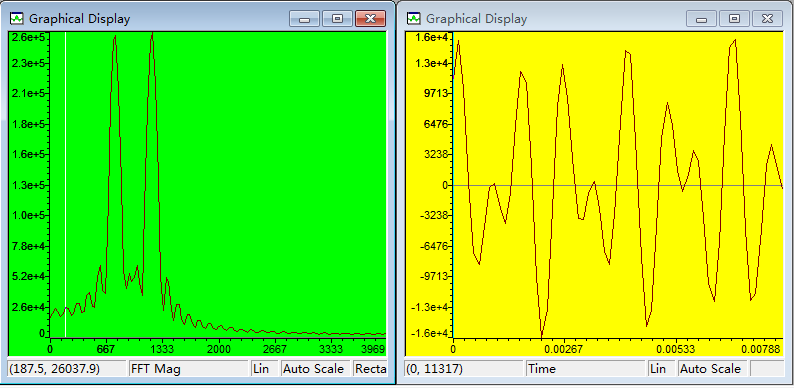


数字键“1”的产生与检测



数字键“4”的产生与检测





## 六．心得体会

双音多频信号由两个正弦波叠加产生，而两个正弦波分别由对应的振荡器产生。

振荡器相当于一个IIR滤波器。我们将一个冲击信号输入振荡器，即可输出对应频率的正弦波。

叠加产生信号后，到达接收端检测信号的方法：先使用Goertzel算法求出频谱上八个点对应的值，其中前四个表示行数，后四个表示列数，然后比较大小，作出判断。Goertzel算法也可以等效为一个IIR滤波器，只需对收到的信号进行八次滤波即可。

可见IIR滤波器的使用贯穿整个数字信号处理的过程，每一个特定的具有相应功能的系统函数，都可以写成一个IIR滤波器的形式。

在整个DSP实验的进行过程中，我们遇到的最大的困难就是c中数制的问题，在编写程序时，我们不止一次地遇到了数制的转换，也经常发生忘记转换或者是溢出而导致的逻辑错误。