

计算机科学学院

课程设计报告

课程名称：单片机综合应用

项目名称：基于51单片机的音乐节拍系统

组成员：王祥通、杨彩霞、杨锦霞

指导老师：郭荣佐

成绩：

时间：

目录

**1.绪论3**

1.1背景3

1.2意义3

1.3 内容安排3

**2.系统总体结构3**

2.1硬件总体结构4

2.2软件总体结构4

**3.硬件设计4**

3.1MCU设计4

3.2电源设计4

3.3通信接口设计4

3.4其他电路设计4

3.5 硬件仿真4

**4.软件设计5**

4.1设备驱动程序5

4.1设备驱动程序5

**5.系统测试5**

5.1硬件焊接和测试5

5.2 软件下载调试6

5.3 系统总体测试6

**6.总结6**

**7.参考文献7**

**8.附录7-23**

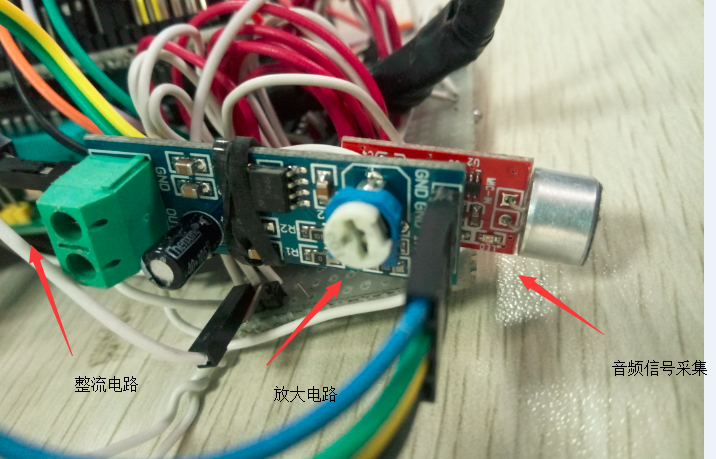
1. **绪论**
   1. **背景** 随着科技的发展，显示部分的点阵LED，用单片机控制的LED显示屏幕将会成为目前高档DISCO酒吧、舞台等夜场最先进的背景灯光产品。由于它应用了单片机技术和数字信号处理技术，所以既不同于普通的灯光装饰，也不同于传统的激光舞台灯。它的出现对夜场传统的灯光装饰提出了极大的挑战，同时也使夜场的灯光装饰步入了一个完全崭新的时代。 通过适当的控制，它能根据音乐的旋律、节奏而产生明暗及色彩的变化并能形成各种动态画面效果，烘托出一种梦幻密离、炫目神秘、华丽斑斓的背景灯光。从而迎合了高档的DISCO酒吧、舞台等夜场对灯光背景的特殊要求，将成为目前夜场装饰的最新亮点。  
       每当看到家里的音箱功放上的几排小灯，随着播放的音乐如波浪般跳跃，或者在电脑上打开千千静听这个音乐播放软件时，看到那动感的频谱跟随音乐节奏优美的舞动着时，不禁思绪万千，要是自己某天能亲手用普通的单片机DIY这么一个东东，那将是多么有成就感的事情，至少对我们电子爱好者来说，这是许多人曾经梦想过的。伴随音箱里传来的美妙音乐，原本只能“听”的音乐，现在却还能“看”，给人带来视觉上的炫酷享受。

* 1. **意义** 在自动化技术控制系统中，单片机的使用实现音乐灯光控制的自动化、智能化，再不需要人为干预的情况下自动用灯光的变换调剂出更好的音乐感觉，实现声光一体化。  
      对于录音棚等专业级别的音乐制作来说，通常都需要对录制的音频信号进行频谱分析来辅助音频的加工制作。当然对于我们普通的爱好者来说，不需要那么精确的频谱分析，我们更多的是用来略显音乐节奏，美化环境，增添气氛。于是，大多数的音乐播放软件，手机，mp3，高档的音箱设备等，都有了音乐频谱显示。
  2. **内容安排** 组内三人学习了频谱图的意义、信号处理相关内容ADC0809的运用等知识，各自分工分别承担了代码编写、proteus仿真、电路焊接、撰写报告与PPT等任务。

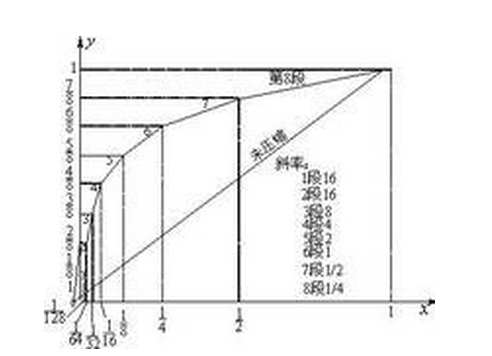
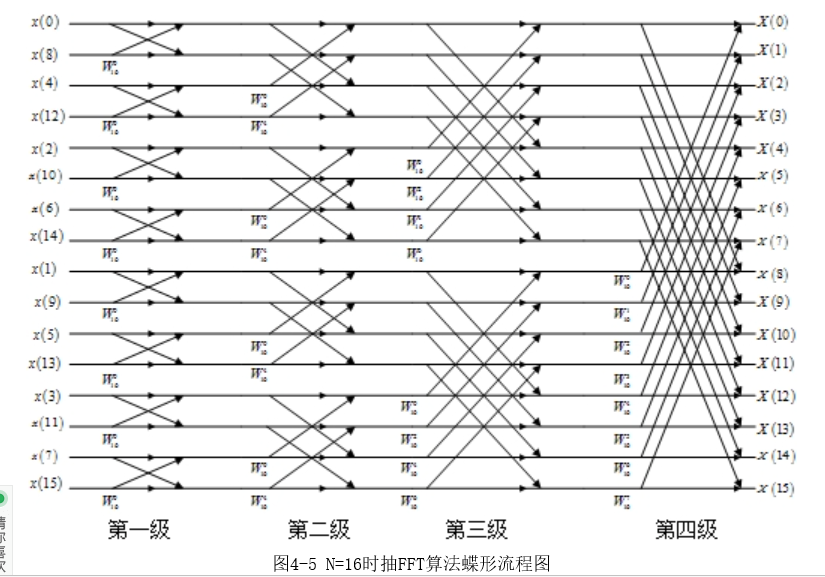
1. **系统总体结构**
   1. **硬件总体设计** 在设计的过程中，由于89C52的数据段只有256个字节，而16点DTFFT计算至少需要存储64个复数变量，每个复变量是实部和虚部为float 的结构体组成，数据段远远不够，所以需要采用6264对承担FFT计算的MCU1进行RAM扩展。  
        
      由于FFT计算需要时间较长，如果采用单独MCU进行计算和显示，会使得屏幕显示断断续续，所以经过讨论系统采用了两块MCU，其中MCU1进行内存扩展后将时域变换后的频域序列，通过8位并行结构传输到MCU2，MCU2主要负责显示，所以不需要内存扩展。  
       在选择布线的过程中,由于系统繁杂不易一次成功连接，整体焊接完成后在进行调试困难很大，找到BUG的难度极高，所以我们组采用了将整体系统拆散成几个独立的子系统，每完成一个部分，单独调试，这样使得调试变得简单。  
        
       关于反相器系统，是用三个74LS04组成的。 译码系统使用两个LS138 组成的4-16译码器， 显示系统是用两个8\*8点阵拼接成的8\*16点阵，用来显示图像。
   2. **软件总体设计** 软件部分主要包括信号采集 、FFT计算、MCU通信、频域谱显示四个部分。
2. **硬件设计**
   1. **MCU设计**

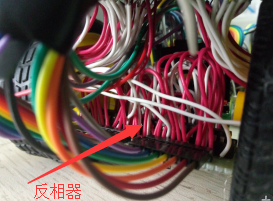
系统采用的是STC89c52，最小系统外围电路见附录

* 1. **电源设计** 电源是通过一个12V的电池，通过转接降压系统（内部集成了稳压环节）得到的5V电源，其中为了使得显示系统的范围可调，在ADC0809的正负参考电压输入阶段，通过电位计又进行了第二次调压，控制显示波形的上下界。
  2. **通信接口设计**采用了八位并行传输，核心代码见附录。
  3. **其他电路设计**
     1. **信号输入电路实物图**



* + 1. **小信号放大电路** 该系统接在时域信号的音频采集系统后端，用以对毫伏量级的小信号进行线性放大，增益可调，1-200倍，使得输出的70-100dB的音频信号电压有0.8V的浮动。
    2. **音频采集电路** 核心部分是极性话筒，采集到的声音信号转化为电压，毫伏级别的交流小信号
    3. **整流电路** 由于采集到的声音信号为交流，而在电源设计中已经定义电压最低的为0V的地线，没有设计负电压引脚，所以ADC0809的负参考电压脚-Vref只能接0V，这就要求输入的模拟信号为直流信号，所以通过整流电路将交流信号转换为直流信号。初始设计时想采用电桥设计整流，但后来考虑到系统本身的低精度特性，干脆直接接了个二极管（IN4007） 将反相部分直接去除。

1. **软件设计**
   1. **设备驱动设计**
      1. **0809驱动** ADC0809的采样子程序返回的是一个0—256的值，要求其显示在高度为8的点阵上，组成柱形图。ADC0809将采集到模拟信号从连续时域转换到离散时域模，内部应该采用了线性均匀量化又把抽样采集到的模拟信号转换到八位的数字信号。  
          但是声音信号的主要集中范围是在比较小的范围，偶尔会有个别采样值远高于平均，若用均匀量化显示在点阵上输出柱形图偏差会比较大，所以根据声音信号普遍的处理方法，这里用了非均匀量化，参考A律13折线压缩特性曲线[文献1]，根据这个系统声音信号被整流成直流，所以实际只用了第一象限的6折线，如图：  
           
           
         该算法核心代码见附录。
      2. **点阵显示驱动** 要使得16条LED灯组成的条按高度表示0-8的值，显示出一个长度为16的数组，这可以通过译码位选、时分复用分别快速显示数组每个元素的值，得到图像，代码见附录。
   2. **16点FFT运算** 在时域到频域的离散傅里叶变换DTFT中，该算法能有效减少算法时间复杂度，由O(n²)到O(nlog(n)),在该设计中，n为采样点数16，蝶形运算算法示意图如下
   3. **主控程序设计** 主程序主要由信号采集、计算、显示三个部分。信号采集和计算为一个线程，在MCU1完成；显示由MCU2完成，代码见附录。
2. **系统测试**
   1. **硬件焊接与测试** 在选择布线的过程中,由于系统繁杂不易一次成功连接，整体焊接完成后在进行调试困难很大，找到BUG的难度极高，所以我们组采用了将整体系统拆散成几个独立的子系统，每完成一个部分，单独调试，这样使得调试变得简单。  
       总系统里焊接脚线最多的几个地方——反相器系统、译码系统、显示系统在测试的时候都出现了和预期结果不一致的情况。  
        
       关于反相器系统，由于总系统需要十六个反相器，一个74LS04有六个集成的反相器，但是脚线没有proteus里面的直观，如图（74LS04反相器），这样在子系统拼接的时候会有线路杂乱，不易理解的情况发生。所以在制作反相系统的时候通过在洞板焊接跳线，调整74LS04的输入输出引脚，使得达到上端输入，下端输出的直观效果，这样也利于子系统板的拼接。



译码系统使用两个LS138 组成的4-16译码器，跳线焊接非常多，在测试的过程中反复出现了译码结果偏差的情况，经过万用表对导线焊接情况依次检查后发现是焊点松动导致短路引起，还有就是线路连接错误引起。

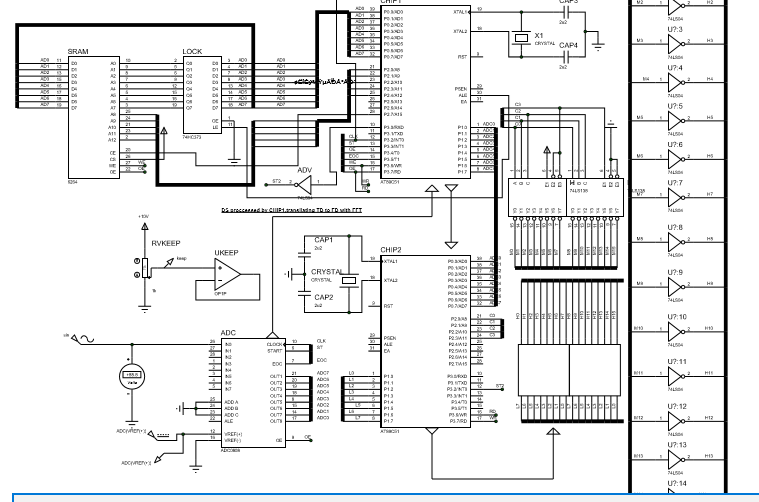
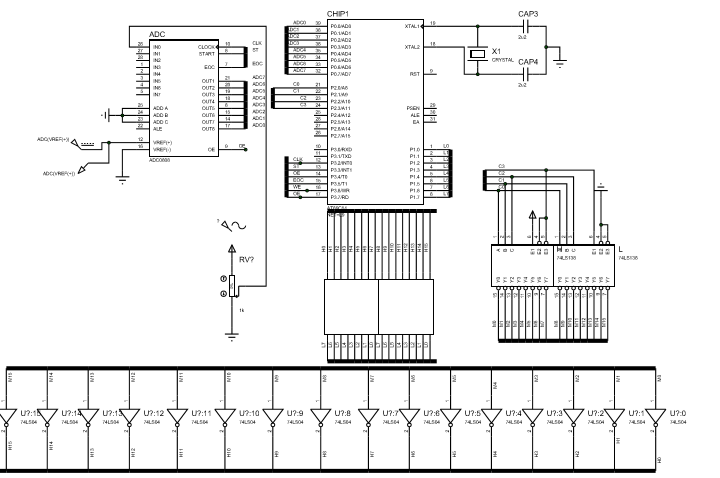
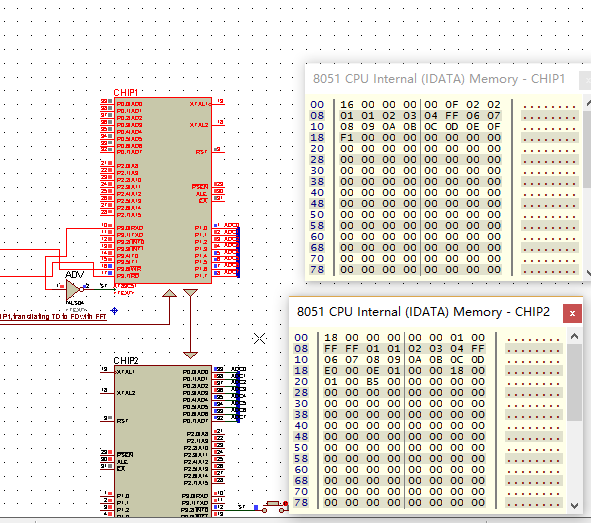
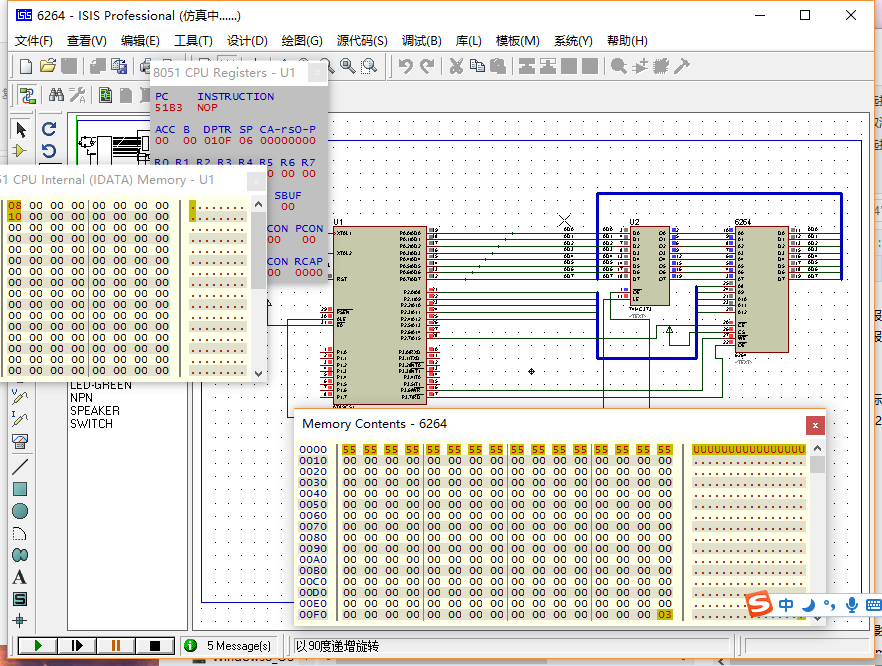
显示系统是用两个8\*8点阵拼接成的8\*16点阵，用来显示时域图像。在焊接过程中，没有搞清楚点阵内部引脚结构，直接焊接上引脚，使得结果大大偏差，后来发现proteus里面的点阵引脚结构和实际中用的完全不同，所以又重新焊了一个，途中又由于点阵引脚断裂没法接跳线，又焊接了第三个显示系统。

* 1. **软件下载调试**无

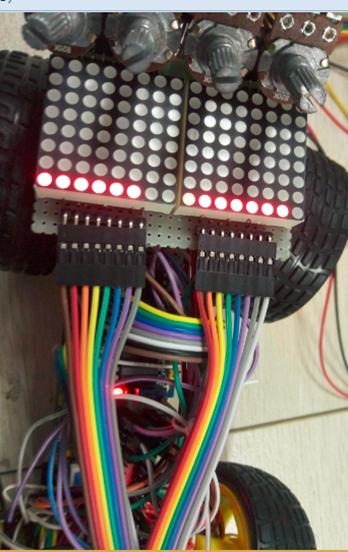
1. **总结** 在这个系统的制作过程中，我们深刻理解到89C52资源匮乏，速度慢效率低的特性，根本无法制作出频谱仪。上述的内存扩展，MCU通信都是单独可以实现，拼接程序后bug连连，更没有自信在我们认为极其复杂实体电路完成后能焊接并且调试成功，所以采用**了精简方案制作音乐节拍显示系统，那就是只显示时域谱，也就是说根本没弄出频域的效果，**这里就是简单的用ADC采样下就完了。
2. **参考文献**

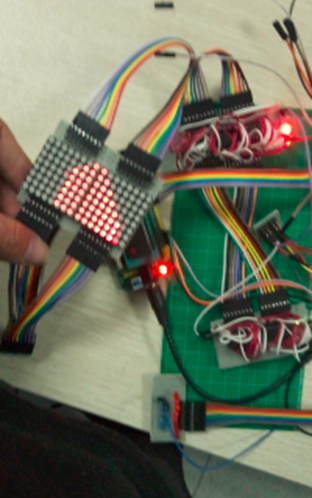
[1]樊昌信,《通信原理》,北京，国防工业出版社,2015年。

[2]文坤 ,《51单片机常用模块设计查询手册》，北京，清华大学出版社 ,2016年。

1. **附录**
   1. **硬件仿真图**
      1. **频域电路仿真**
      2. **时域谱电路**
      3. **MCU通信电路测试结果**从上图可以看出MCU1的数组F传入了MCU2中
      4. **内存扩展电路测试**

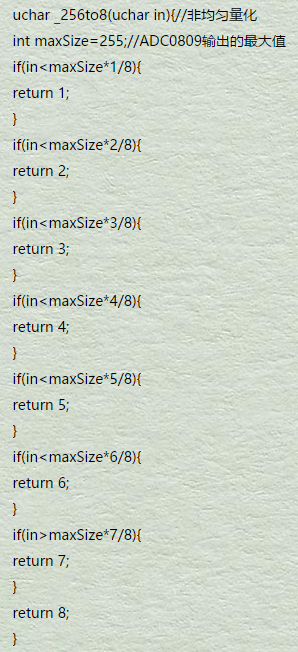
可以看出值存入了外部内存

* 1. **系统实物图  
      **

系统实物图 1 ****

系统实物图 2 ****

系统实物图 3

* 1. **程序源代码**
     1. **非均匀量化核心代码**
     2. **MCU通信代码**

**MCU1代码**

//MCU1

#include <reg51.h>

#define uchar unsigned char

#define OutPut P1

sbit ST= P2^0; //传数组开始的时候 置为高电平

sbit sendEn= P3^6; //传一个元素高电平一下,可以发送==可写

sbit EndRead = P3^7;

data unsigned char F[]={1,1,2,3,4,0xFF,6,7,8,9,10,11,12,13,14,15, }; //可变数组一定要加data关键字，要不然全是零！

void delay( uchar a){

uchar i =0;

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

}

void send() {

uchar i =0;

ST=1;//开始传送数组，此时MCU2进入中断处理函数

// ST=0;

for(i=0;i<16;i++){

while(!EndRead); //d对方收到信息，更新下一帧信息

P1=F[i]; //2us 左右 数据是0-9 四根线

sendEn=1;

delay(2);

sendEn=0;

// delay(2);

}//for

ST=0;//传送结束

}

void main(){

int i=0;

P1=0;P1=0xff;P1=0;P1=0xff;

ST=0;

sendEn=0;

//unsigned char F[]={0xFE,0xFC,0XF8,0XF0,0XE0,0XC0,0X80,0X00,

// 0x00,0x80,0XC0,0XE0,0XF0,0XF8,0XFC,0XFE};

//while(1){

send(); //用send(F)后失灵，而且地址变了.传入的是F

// }

}

**MCU2**

//MCU2

#include <reg51.h>

#define InPut P1;//P1口做八位并行输入

#define N 16

#define uchar unsigned char

#define Input P0

sbit ST = P3^2; //等待开始

sbit InEn =P3^7; //等待输出 sendEn

sbit EndRead=P3^6;//读完确认

unsigned char F[16]={0};

void delay( uchar a){

uchar i =0;

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

}

void initT0(){

TMOD = 0x06; //0000 0010 选择T0定时计数器，选择计数1，选择方式2八位自装填

TH0=254; //0x14 这个是选择给adc的时钟频率

TL0=254;

IE=0X82; //1000 0010 开启T0

TR0=1; //TCON开启T0

}

void initINT0(){

EA=1; // 中断总开关

EX0=1 ; // 外部中断总开关

//用INT0作为中断

IT1=1; //使得!INT0,边沿触发

}

void func\_INT0()interrupt 0{

uchar i =0;

EndRead=1;

for(;i<N;i++){

while(!InEn) ; //读入数据 ,3.7

F[i]=P1; //把P1口的输入存储起来

P2=F[i]; //验证

EndRead=1; //读完确认

delay(2);

EndRead=0;

// delay(2);

}

// P2=0xf0;

}

void main(){

initINT0();

P1=0xFF;

P3=0xFF;

ST=1;//做输入口，这个输入中断信号，来低电平有效，之前用反相器处理过了

InEn=1;//同上 ，做输入口

EndRead=0; //写死了，要不然第一个读不到

while(1);

}

* + 1. **16点FFT代码**

#include <stdio.h>

#include <math.h>

#define typ float

typedef struct {

float real;

float image;

}complex;

//求共轭

complex conjugate(complex a){

complex b;

b.real=a.real;

b.image=-a.image;

return b;

}

//复数加

complex add(complex a,complex b){

complex c;

c.real=a.real+b.real;

c.image=a.image+b.image;

return c;

}

complex minus(complex a,complex b){

complex c;

c.real=a.real-b.real;

c.image=a.image-b.image;

return c;

}

typ amp(complex a){

return sqrt((a.real\*a.real)+(a.image\*a.image));

}

void FFT16(float x[],float F[]){//这里改代码

/\*-------第0级(时域输入)--------------\*/

/\*-----第1级-----------------\*/

complex X7[2];

complex X8[2];

complex X9[2];

complex X10[2];

complex X11[2];

complex X12[2];

complex X13[2];

complex X14[2];

/\*-----第2级-----------------\*/

complex X3[4];

complex X4[4];

complex X5[4];

complex X6[4];

/\*-----第3级-----------------\*/

complex X1[8];

complex X2[8];

/\*------第四级（频域输出）------------\*/

complex X[16];

/\*d蝶形运算\*/

//------------------------

X7[0].real=x[0];

X7[0].image=x[8];

X7[1].real=x[0];

X7[1].image=-x[8];

//-------------------------

X8[0].real=x[4];

X8[0].image=x[12];

X8[1].real=x[4];

X8[1].image=-x[12];

//-----------------------------

X9[0].real=x[2];

X9[0].image=x[10];

X9[1].real=x[2];

X9[1].image=-x[10];

//-------------------------------

X10[0].real=x[6];

X10[0].image=x[14];

X10[1].real=x[6];

X10[1].image=-x[6];

X11[0].real=x[1];

X11[0].image=x[9];

X11[1]=conjugate(X11[0]);

X12[0].real=x[5];

X12[0].image=x[13];

X12[1]=conjugate(X12[0]);

X13[0].real=x[3];

X13[0].image=x[11];

X13[1]=conjugate(X13[0]);

X14[0].real=x[7];

X14[0].image=x[15];

X14[1]=conjugate(X14[0]);

/\*-----------第二级-------------------------\*/

X3[0]=add(X7[0],X8[0]);

X3[1]=add(X7[1],X8[1]);

X3[2]=minus(X7[0],X8[0]);

X3[3]=minus(X7[1],X8[1]);

X4[0]=add(X9[0],X10[0]);

X4[1]=add(X9[1],X10[1]);

X4[2]=minus(X9[0],X10[0]);

X4[3]=minus(X9[1],X10[1]);

X5[0]=add(X11[0],X12[0]);

X5[1]=add(X11[1],X12[1]);

X5[2]=minus(X11[0],X12[0]);

X5[3]=minus(X11[1],X12[1]);

X6[0]=add(X13[0],X14[0]);

X6[1]=add(X13[1],X14[1]);

X6[2]=minus(X13[0],X14[0]);

X6[3]=minus(X13[1],X14[1]);

/\*-----------第3级-------------------------\*/

int j;

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

X1[j]=add(X3[j],X4[j]);

X1[j+4]=minus(X3[0],X4[0]);

}

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

X2[j]=add(X5[j],X6[j]);

X2[j+4]=minus(X5[0],X6[0]);

}

/\*-------------第四级（频域输出）---------------------\*/

for(j=0;j<8;j++){

X[j]=add(X1[j],X2[j]);

X[j+8]=minus(X1[j],X2[j]);

}

int i=0;

for(;i<16;i++){

F[i]=amp(X[i]);

}

}

int main(){

//delta

/\* typ x[16]={0,0,0,0,

1,0,0,0,

0,0,0,0,

0,0,0,0} ;

\*/

//const

typ x[16]={1,1,1,1,

1,1,1,1,

1,1,1,1,

1,1,1,1} ;

/\* typ x[16]={0,0,0,0,

0,0,0,0,

1,1,1,1,

1,1,1,1} ;\*/

float F[16]={0};

FFT16(x,F);

float temp;

int i=0;

for(;i<16;i++){

printf("%f\n",F[i]);

}

return 0;

}

**8.3.4显示驱动代码**

void display(unsigned char F[]){

//就把F打印一次

int i;

for(i =0;i<16;i++){

P1=0xff; //全灭

//位选

P2=i+16;

P1=F[i]; //赋值

delay(7); //这个延时正好 ，时间越长，注入能量越高，越亮

}

P2=0XFF;

P1=0xfe;

delay(1);

}

**8.3.5主程序代码（时域谱）**

void main(){

idata char F[16]={0x18};

int i=0;

unsigned char ADout=0;

//选择7号模拟道

initADC0908();

initT0();

while(1){

for(i=0;i<16;i++){

//0-256之间 变成了0-8 均匀量化

// ADout= ADC0809ConvertC0()/15;

ADout= \_256to8(ADC0809ConvertC0()); //非均匀量化

F[i]= num2Disnum[ADout];

}

display(F);

display(F);

display(F);

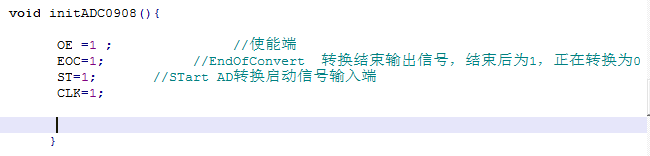
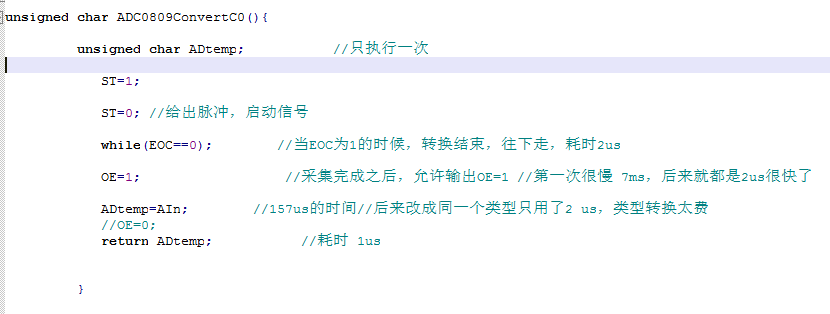
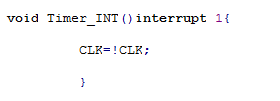
display(F);

display(F);

//不停的传值 采样周期最快现在是12us 大约83KHz

}//while

}

**8.3.6 ADC0809采样代码** 

**8.3.7内存扩展代码**

//6264

#include <reg51.h>

#include <absacc.h>

#define OFFRAMBASE 0x100

unsigned char xdata offRamVal0 \_at\_ 0x300; //地址overFlow ，有效地址直到0x01ff，当超过后，进位，后面位数都为0，故此地址实际值为0

unsigned char xdata offRamVal1 \_at\_ 0x01;

unsigned char xdata offRamVal2 \_at\_ 0xFF;

int main()

{

unsigned char idata index = 1;

//----一般赋值测试

offRamVal0 = 0xAA;

offRamVal1 = 0xCC;

offRamVal2 = 3;

//-----基址变址赋值测试

for(index=0;index<0x10;index++)

{

XBYTE[index+OFFRAMBASE] = 0x55;

}

return 0;

}

// void main(){

// unsigned char temp \_at\_ 0x1020;

// }