北京航空航天大學
BEIHANG UNIVERSITY

数字信号处理硬件部分
实验报告

院(系)名称	高等理工学院
学　　号	16231235
姓　　名	李谨杰
指 导 教 师	崔勇

2019 年 5 月

实验一 指示灯实验

一、实验目的

1．了解DSP 开发系统和计算机与目标系统的连接方法。

2．了解Code Composer Studio 5 软件的操作环境和基本功能，了解TMS320C28xx 软件开发过程。

3．了解ICETEK-F28335-AF 评估板在TMS320F28335DSP外部扩展存储空间上的扩展。

4．了解ICETEK-F28335-AF 评估板上指示灯扩展原理。

二、实验内容

1、通过在CCS 的编程，使DSP 实验箱的指示灯D5-D2 完成二进制的闪烁累加，D5 是最高位，D2 是最低位。

2、自行设计1~2 种指示灯不同的显示方式。

三、实验设备

计算机， ICETEK-F28335-AF 实验箱（ 或ICETEK 仿真器+ICETEK-F28335-AF 系统板+相关连线及电源）。

四、实验原理

1．TMS320F28335DSP 的存储器扩展接口

2．指示灯扩展原理

3．实验程序流程图

详细内容见实验指导书。

五、实验步骤

连接硬件

1. 启动CCS，选择工作空间。

2. 建立目标配置文件：Target Configuration。这一步可以配置DSP芯片和下载器的信息。另外这里要进行Test connection测试，确保仿真器连接顺利。

3. 创建工程：Project。类似单片机，工程下放置各种.c，.h程序。

4. 在main.c 文件中添加程序。

5. 添加头文件路径。这个也和Keil5一样，如果新建的.h文件要添加进工程中，需要添加头文件路径。

6. 添加函数。这里可以拷贝一些已有的函数到文件中。但是由于缺乏对这些函数的描述，不太清楚函数的作用，导致之后的中断设置遇到了困难。

7. 将之前的硬件配置文件链接到工程中，便于生成机器码供芯片运行。

8. 编译。如报错，修改。

9. 下载，Run！

10. 如果现象不对，重复4-10步。

DSP下载程序的流程和单片机还是非常像的。区别在于，我用的Stm32CubeMX会自己配置好硬件与一些初始函数。而在CCS里边需要将Target Configuration单独配置并与工程连接。

六、实验程序流程及说明

1. 例程（关键代码加注释说明）

实验现象：LED灯从0000-1111依次点亮，又从1111-0000依次熄灭。

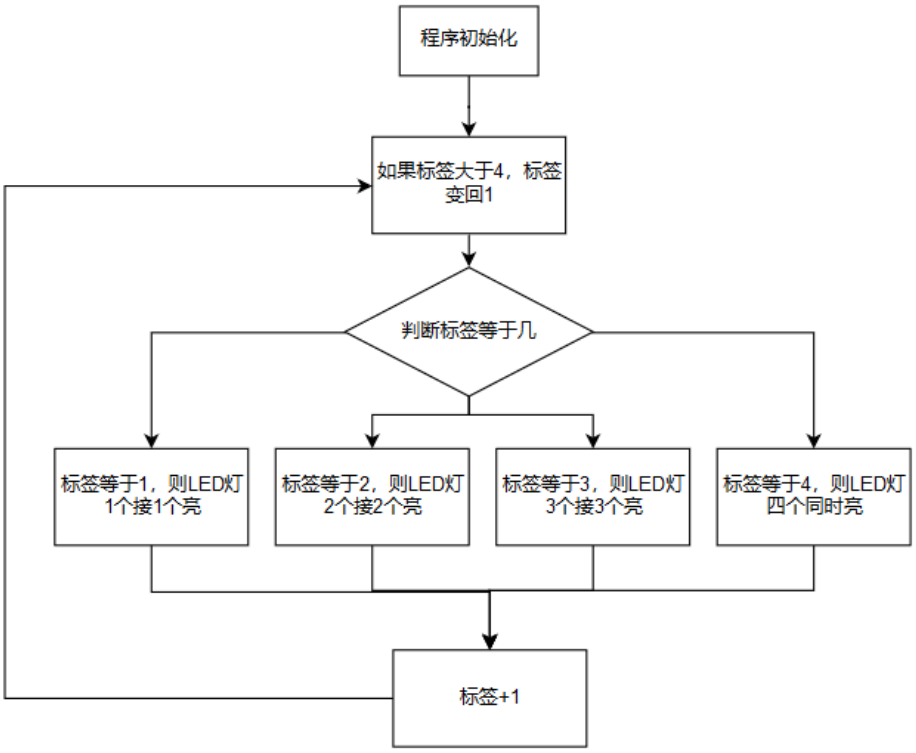
```
#define LED (*(unsigned short int *)0x180000)     // 第一个LED灯对应的GPIO的地址是0x180000

#define SRAM_Base_Adress 0x100000

InitSysCtrl();  // 初始化整个系统的一些配置

InitXintf16Gpio();  // 初始化GPIO

DINT;               // 关中断

InitPieCtrl();     // 初始化Pie寄存器和一些相关的配置   PIE：Peripheral Interrupt Expansion
    IER = 0x0000;   //IER为INT Enable Register，这里关闭全部的中断寄存器
    IFR = 0x0000;   //IFR为INT Flag Register，这里将标志位清零
```

2. 我们自己发挥的程序：

功能描述：设计程序，使得LED灯1个接1个从左至右亮，再从右至左挨个亮；之后变为2个LED灯同时亮，从左往右再从右往左；最后变成3个LED灯同时亮，从左往右再从右往左；最后是四个灯同时亮。

因为程序较简单，这里就不粘贴程序了。

程序流程图如下：



3. 中断程序学习

上课的时候没有调试成功，之后我又上网搜索中断程序进行学习。

程序开始声明函数 interrupt void ISRTimer0(void);

Main函数开头初始化中断：

```
    InitCpuTimers();          //初始化系统时钟，这个TI的DSP2833X_CpuTimes.c文件自带

    #if (CPU_FRQ_150MHZ)     //如果CPU频率是150MHZ，按上述配置
        ConfigCpuTimer(&CpuTimer0, 150, 100000);
    #endif
    #if (CPU_FRQ_100MHZ)     //如果CPU频率是100MHZ，按这个配置
        ConfigCpuTimer(&CpuTimer0, 100, 500000);
    #endif

     CpuTimer0Regs.TCR.bit.TSS=0;      //写0打开定时器；写1停止定时器了
                                       //在设置完时钟之后，一定要记得打开定时器

            EALLOW;
            GpioCtrlRegs.GPAMUX2.bit.GPIO26 = 0;     //IO口的初始化
            GpioCtrlRegs.GPADIR.bit.GPIO26=1;
            EDIS;
            GpioDataRegs.GPADAT.bit.GPIO26=0;
```

```
        EALLOW;  // Edit Allow，表示解除对寄存器的保护，使得可以修改
          GpioCtrlRegs.GPAMUX1.bit.GPIO11= 0;
          GpioCtrlRegs.GPADIR.bit.GPIO11=1;
        EDIS;      // Edit Disable，表示加上对寄存器的保护，使得这些寄存器无法修改。

          EALLOW;
          GpioCtrlRegs.GPAMUX1.bit.GPIO0= 0;
          GpioCtrlRegs.GPADIR.bit.GPIO0=1;
          EDIS;

          IER|=M_INT1;                              //全局使能INT1
          PieCtrlRegs.PIEIER1.bit.INTx7=1;      //使能INT1.7即定时器0中断

          EINT;     //开中断
          ERTM;     //使能调试事件

    for(; ;)
    {};
}

interrupt void ISRTimer0(void)
{
led++;
CpuTimer0.InterruptCount++;

PieCtrlRegs.PIEACK.all=PIEACK_GROUP1;              //使能CPU接受第一组中断

GpioDataRegs.GPATOGGLE.bit.GPIO11=1;    //I/O口翻转，即自定义程序
GpioDataRegs.GPATOGGLE.bit.GPIO0=1;

CpuTimer0Regs.TCR.bit.TIF=1;                       //清除中断标志位
CpuTimer0Regs.TCR.bit.TRB=1;                       //使能重载
}
```

与单片机的区别主要有：

1. 新增函数类型interrupt，表示中断调用函数。
2. 对底层寄存器的操作更细致。
3. 有一些汇编语言与C语言交叉出现的情况。如DINT，EINT，EALLOW，EDIS等等。
4. 当定时器0触发中断后，与8086单片机类似，cpu会去中断向量表中定时器0对应的位置寻找中断程序入口，进而按入口程序地址执行中断。
5. 这个程序也不一定对，希望之后的实验有机会可以跑一下。

七、实验感悟

DSP实验最好提前写好代码，搞明白代码背后的意义，这样才好在两节课的时间内完成尽可能多的实验内容。对于中断配置，需要结合TMS320F28335DSP使用手册才能完全理解对中断向量的配置。在实际程序书写中，main函数开头的初始化内容，中断函数中清除中断标志位和使能重载的代码均相同，需要改变的主要是中断函数中执行操作的内容。因此，在工程实践中要注重积累代码段，这样可以大大缩短编程时间。

另外，基于崔老师的要求，本报告所有指导书上可以找到的内容都进行了一定程度的简化。

实验二 键盘输入液晶屏显示实验

一、 实验目的

1.通过实验学习使用F28335 DSP的扩展I/O 端口控制外围设备的方法，了解液晶显示器的显示控制原理及编程方法。

2. 通过实验学习使用F28335 DSP的扩展端口接收外围设备信息的方法，了解键盘的使用原理及编程方法。

二、 实验内容

1、基本内容：通过在CCS 的编程，在液晶显示屏上显示“ICETEK-F28335-AF液晶显示”和计时时钟，精确到秒，形式为“时时：分分：秒秒”。5分

2、基本内容：获取键值，并显示键值。3分

3、拓展内容：结合键盘和显示，自主设计，完成某一功能。根据难易程度及特殊性。可获得1~2 分

三、 实验设备

计算机， ICETEK-F28335-AF实验箱。

四、 实验原理

1.扩展IO 接口：

ICETEK-F28335-AF 是一块以TMS320F28335DSP 为核心的DSP 扩展评估板，它通过扩展接口与实验箱的显示/控制模块连接，可以控制其各种外围设备。

2. 显示控制方法：

本实验中使用已写好的库函数对液晶屏幕进行操作。需要在工程文件中加入库ICETEK-CTR.lib 以及头文件ICETEK-CTR.h。

下面给出ICETEK-CTR.lib 的控制液晶屏幕的接口函数及其功能描述：

void ICETEKCTR_InitLCD(); //初始化液晶显示屏

void ICETEKCTR_LCDCMD(Byte dbCommand); //向LCD 发送指令

void ICETEKCTR_LCDDAT(Byte cData); //向LCD 发送数据

void ICETEKCTR_LCDCLS(); //LCD 清屏

void ICETEKCTR_LCDPutString(char *sString,int x,int y); //在LCD 屏幕上显示字符串

void ICETEKCTR_LCDDrawPixel(int x,int y,Byte cColor);//写点到屏幕，输入参数坐标值和颜色，颜色0消点，1画点，2异或画点。

3. 键盘连接原理：键盘的扫描码由DSP的扩展地址0x208001给出，当有键盘输入时，读此端口得到扫描码，当无键被按下时读此端口的结果为0。这功能由ICETEKCTR_GetKey()实现，函数返回值就是键盘的扫描码。9个按键分别读回1-9这九个数字。

4．实验程序流程图

在实验报告中给出流程。

五、 实验步骤

1、启动CCS
2、建立目标配置文件（若已建立则可跳过此步骤）
3、创建工程（详见指示灯实验）
4、修改main.c 的内容为所编写代码，点击File->Save 保存
5、添加头文件（添加方法见实验一）
6、添加函数库ICETEK-CTR.lib和头文件ICETEK-CTR.h
7、连接配置文件
8、编译、连接和下载程序；运行程序
9、退出程序

六、 代码流程及说明

1. 液晶屏显示

实验现象：LCD屏上第一行显示“ICETEK-F28335-AF”，第二行显示“液晶显示”，第三行显示时间。

（1）头文件：

```
#include "DSP2833x_Device.h"
#include "DSP2833x_Examples.h"
#include "ICETEK-CTR.h"
```

（2）声明函数：

```
    void InitICETEKF28335Ae(); //初始化：DSP主频、GPIO、中断向量、中断使能
```

进行上一个实验的初始化操作。具体代码如下：

```
void InitICETEKF28335Ae() {
    InitSysCtrl();   //初始化系统
    InitXintf16Gpio(); //初始化扩展空间接口管脚，以便与ICETEK-CTR通讯
    DINT;   //使中断失效
    InitPieCtrl();    //初始化PIE寄存器
    IER = 0x0000;    //失效全部中断，并清除全部中断标志位
    IFR = 0x0000;
    InitPieVectTable();  //初始化PIE向量表
    }
```

（3）主程序，初始化试验箱：

```
    InitICETEKF28335Ae();
    bSuccess=ICETEKCTR_InitCTR(ICETEKCTRModeTeachingResearch); //初始化ICETEK-CTR：教研模式
    while ( bSuccess ); // 如果初始化ICETEK-CTR错误，停止运行，可观察bSuccess取值查找初始化失败原因
```

显示内容分为固定不变的显示和随时间推移改变的显示内容，按字符串为单位显示。应该在初始化的时候显示含有固定内容的字符串，然后实时改变数组中的内容，将含有改变内容的字符串输出到屏幕上。

输出固定内容：

```
    ICETEKCTR_LCDPutString("ICETEK-F28335-AF",0,LCDLINE0); //从第一行第一个字符开始打
    ICETEKCTR_LCDPutString("液晶显示",2,LCDLINE1);
```

主循环中实时更改时间：

```
for(;;){
    s++; if ( s>59 ) { s=0; m++; if ( m>59 ) { m=0; h++; h%=24; } }
    buffer[0]=h/10+'0'; buffer[1]=h%10+'0';
    buffer[3]=m/10+'0'; buffer[4]=m%10+'0';
    buffer[6]=s/10+'0'; buffer[7]=s%10+'0';
    ICETEKCTR_LCDPutString(buffer,2,LCDLINE3); //第三行第二个字符开始
    ICETEKCTR_Delayms(1000);  //延时一秒
        }
```

2. 键盘显示

实验现象：按下按键，液晶屏上会显示按下按键对应的数字。

初始化内容与LCD屏相同，主循环改为如下内容：

```
    uKeyCode=ICETEKCTR_GetKey();
    uDelayCount=20;
    ICETEKCTR_Delayms(uDelayCount);
    if(ICETEKCTR_GetKey()==uKeyCode) //如果依然相等，说明不是误触
    {
    ICETEKCTR_LCDPutString("Key number = ",0,LCDLINE0); //从第一行第一个字符开始打
    ICETEKCTR_LCDPutString('0'+uKeyCode,13,LCDLINE0);
        }
```

在代码中我加入了数字防抖功能，只有持续20ms收到同一个按键信号，才可以判定为按下响应的按键。按下按键后，屏幕上会显示“Key number = a”，a为按下的数字。实际代码稍有改动，这里的代码是我实验前写的。

3. 拓展内容

这里我们想通过屏幕和键盘设计一个简单的游戏，但由于时间限制和预习的不充分，最终没有调试成功。介绍游戏设计思想如下：

游戏开始，LCD屏左上角出现一个数字0，代表玩家所处的位置和得分。在屏幕上随机出现一些数字1，代表食物。其余位置为空格，如下图：

0 1

1

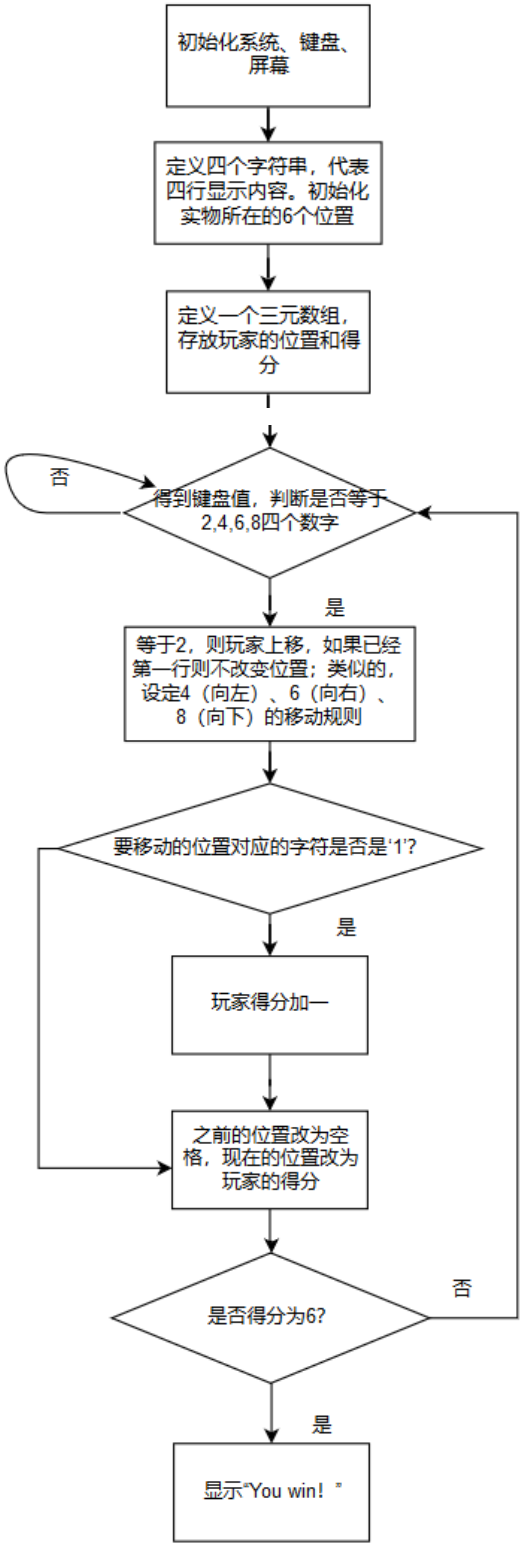
1 1

1 1

1 1

玩家需要通过键盘的上下左右控制坐上角的数字去吃那些数字1，每吃一个数字，玩家身上的数字就会加一。最终全部吃完食物，则玩家胜利。

程序框图如下：



七、 实验感悟

进行 DSP 实验前一定要预习充分。本次实验前两个提前写了代码，第三个是去了才写的，最终就没有调试成功。由于函数库已经封装好 LCD 操作和读取

键值的代码，本次实验并不复杂。实际应用时，要结合 LCD 模块的操作手册，自行编写相应程序。

实验三 键盘输入液晶屏显示实验

一、 实验目的

1. 掌握DSP定时器，采用定时器中断。
2. 掌握DSP片内DA。
3. 掌握FFT应用。

二、 实验内容

1. 基本内容：

采集正弦、方波和三角波信号，并对其进行FFT变换。完成此项基本内容，需要首先分别完成如下内容：（1）定时器调试正常；（2）AD采集正常，数据正确；（3）FFT变换正确。最后将几项内容综合在一起，完成对信号的FFT变换。

2. 拓展内容

拓展内容为研究内容，如：采样周期变化对FFT结果的影响等。

三、 实验设备

计算机， ICETEK-F28335-AF实验箱。

四、 实验原理

1. 片内自带模数转换特性
2. 模数模块介绍
3. 模数转换的程序控制-中断方式
4. FFT的原理和参数生成公式：

$$x(k)=\sum_{r=0}^{\frac{N}{2}-1}x_1(r)W_{\frac{N}{2}}^{rk}+W_N^k\sum_{r=0}^{\frac{N}{2}-1}x_2(r)W_{\frac{N}{2}}^{rk}=X_1(k)+W_N^kX_2(k)$$

FFT即为利用对称特性将DFT连续分成两部分（进包），独立操作，计算完成每一部分后再合成输出“打开”，得到最后的计算结果。

以上内容详见指导书。

5. 根据F28335 DSP datasheet，ADC寄存器组织如下：

Table 4-6. ADC Registers[1]

NAME	ADDRESS[1]	ADDRESS[2]	SIZE (x16)	DESCRIPTION
ADCTRL1	0x7100		1	ADC Control Register 1
ADCTRL2	0x7101		1	ADC Control Register 2
ADCMAXCONV	0x7102		1	ADC Maximum Conversion Channels Register
ADCCHSELSEQ1	0x7103		1	ADC Channel Select Sequencing Control Register 1
ADCCHSELSEQ2	0x7104		1	ADC Channel Select Sequencing Control Register 2
ADCCHSELSEQ3	0x7105		1	ADC Channel Select Sequencing Control Register 3
ADCCHSELSEQ4	0x7106		1	ADC Channel Select Sequencing Control Register 4
ADCASEQSR	0x7107		1	ADC Auto-Sequence Status Register
ADCRESULT0	0x7108	0x0B00	1	ADC Conversion Result Buffer Register 0
ADCRESULT1	0x7109	0x0B01	1	ADC Conversion Result Buffer Register 1
ADCRESULT2	0x710A	0x0B02	1	ADC Conversion Result Buffer Register 2
ADCRESULT3	0x710B	0x0B03	1	ADC Conversion Result Buffer Register 3
ADCRESULT4	0x710C	0x0B04	1	ADC Conversion Result Buffer Register 4
ADCRESULT5	0x710D	0x0B05	1	ADC Conversion Result Buffer Register 5
ADCRESULT6	0x710E	0x0B06	1	ADC Conversion Result Buffer Register 6
ADCRESULT7	0x710F	0x0B07	1	ADC Conversion Result Buffer Register 7
ADCRESULT8	0x7110	0x0B08	1	ADC Conversion Result Buffer Register 8
ADCRESULT9	0x7111	0x0B09	1	ADC Conversion Result Buffer Register 9
ADCRESULT10	0x7112	0x0B0A	1	ADC Conversion Result Buffer Register 10
ADCRESULT11	0x7113	0x0B0B	1	ADC Conversion Result Buffer Register 11
ADCRESULT12	0x7114	0x0B0C	1	ADC Conversion Result Buffer Register 12
ADCRESULT13	0x7115	0x0B0D	1	ADC Conversion Result Buffer Register 13
ADCRESULT14	0x7116	0x0B0E	1	ADC Conversion Result Buffer Register 14
ADCRESULT15	0x7117	0x0B0F	1	ADC Conversion Result Buffer Register 15
ADCTRL3	0x7118		1	ADC Control Register 3
ADCST	0x7119		1	ADC Status Register
Reserved	0x711A – 0x711B		2	
ADCREFSEL	0x711C		1	ADC Reference Select Register
ADCOFFTRIM	0x711D		1	ADC Offset Trim Register
Reserved	0x711E – 0x711F		2	

因此在程序中需要对ADCTRL、ADCMAXCONV、ADCCHSELSEQ控制寄存器进行配置，并从相应的ADCRESULT寄存器中读取数据。另外，由于ADC为12位采样，而寄存器为16位，需要对采集到的数据向右移位四位。

ADC计算公式为：

$$\text{Digital Value} = 0, \quad \text{when input} \le 0\ \text{V}$$

$$\text{Digital Value} = 4096 \times \frac{\text{Input Analog Voltage} - \text{ADCLO}}{3}, \quad \text{when } 0\ \text{V} < \text{input} < 3\ \text{V}$$

$$\text{Digital Value} = 4095, \quad \text{when input} \ge 3\ \text{V}$$

由于ADC的采样值与实际电压值是线性映射关系，直接进行FFT得到的相位谱只会存在比例变换，即幅值放大，整体的形状不受影响。因此可以直接对ADC采样的数据进行FFT变换。

五、 实验步骤

（1） 准备硬件，按实验一硬件连接。
（2） 调试定时器程序。
（3） 熟悉实验箱里信号源的设置。
（4） 将信号源输出连接至AD输入，编写AD程序，调试AD程序。
（5） 编写调试FFT程序。

（6） 检查

六、 代码流程及说明

1. 定时器程序

实验现象：四个LED灯每1秒改变一次显示样式，按照0000, 0001, 0010, 0011……的顺序显示。

定时器程序样例见实验指导书。我将中断子程序更改如下：

```
interrupt void cpu_timer0_isr(void)
{
CpuTimer0.InterruptCount++;     //中断配置
PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
CpuTimer0Regs.TCR.bit.TIF=1;
CpuTimer0Regs.TCR.bit.TRB=1;
 LED=uLBD;
uLBD++;
if(uLBD>15)
    uLBD=0;
       }
```

2. AD程序

实验现象：采集到模拟信号，但每隔固定距离有间断。

AD例程见实验指导书。
例程里有一个延时循环

```
for(j=0;j<100;j++)
    k++;
```

导致采样的信号断断续续的，在最后做综合实验时要重写AD部分。

3. FFT程序

实验现象：采集到模拟信号，将采集信号和FFT变换后的结果存到数组中，用绘图工具显示。

由于时间关系，我没有单独调试FFT程序，而是将FFT程序作为综合程序的一个子程序进行调用。对于实验指导书中的FFT程序理解如下：

```
int INPUT[SAMPLENUMBER],DATA[SAMPLENUMBER];
float fWaveR[SAMPLENUMBER],fWaveI[SAMPLENUMBER],w[SAMPLENUMBER];
float sin_tab[SAMPLENUMBER],cos_tab[SAMPLENUMBER];
```

需要进行FFT数据存放在INPUT中，FFT的结果存放在DATA中。fWaveR存放运算过程中的实部数据，fWaveI存放运算过程中的虚部数据，w存放的是模值，sin_tab, cos_tab按采样数将2pi均分，存储每个采样点对应角度的正弦值与余弦值。

```
void InitForFFT();
```

这个函数给sin_tab, cos_tab幅值。

```
void MakeWave();
```

这个函数将正弦信号存入INPUT中，以便调试。正常使用时无需执行。

```
void FFT(float dataR[SAMPLENUMBER],float dataI[SAMPLENUMBER]);
```

这个函数经过反序与FFT，最终将变换结果存入临时变量dataR（实部）和dataI（虚部）中，并求模值，赋给w数组。

4. 综合程序

（1）头文件引用与函数定义：

```
#include "DSP2833x_Device.h" // DSP2833x Headerfile Include File
#include "DSP2833x_Examples.h" // DSP2833x Examples Include File
#include "math.h"   //包含数学库
#define PI 3.1415926
#define SAMPLENUMBER 128  //DFT点数

#define startCpuTimer0() CpuTimer0Regs.TCR.bit.TSS=0
#define POST_SHIFT 0 // Shift results after the entire sample table
is full
#define INLINE_SHIFT 1 // Shift results as the data is taken from the
results regsiter
#define NO_SHIFT 0 // Do not shift the results

#if (CPU_FRQ_150MHZ) // Default - 150 MHz SYSCLKOUT
#define ADC_MODCLK 0x3 // HSPCLK = SYSCLKOUT/2*ADC_MODCLK2 =
150/(2*3) =25.0 MHz
#endif
#if (CPU_FRQ_100MHZ)
#define ADC_MODCLK 0x2 // HSPCLK = SYSCLKOUT/2*ADC_MODCLK2 =
100/(2*2) =25.0 MHz
#endif

#define ADC_CKPS 0x0 // ADC module clock = HSPCLK/1 = 25.5MHz/(1)
=25.0 MHz
#define ADC_SHCLK 0x1 // S/H width in ADC module periods =2 ADC cycle
#define AVG 1000 // Average sample limit
#define ZOFFSET 0x00 // Average Zero offset
#define BUF_SIZE 1024 // AD采样数组大小
```

这一部分将前三个例程的定义综合起来即可。

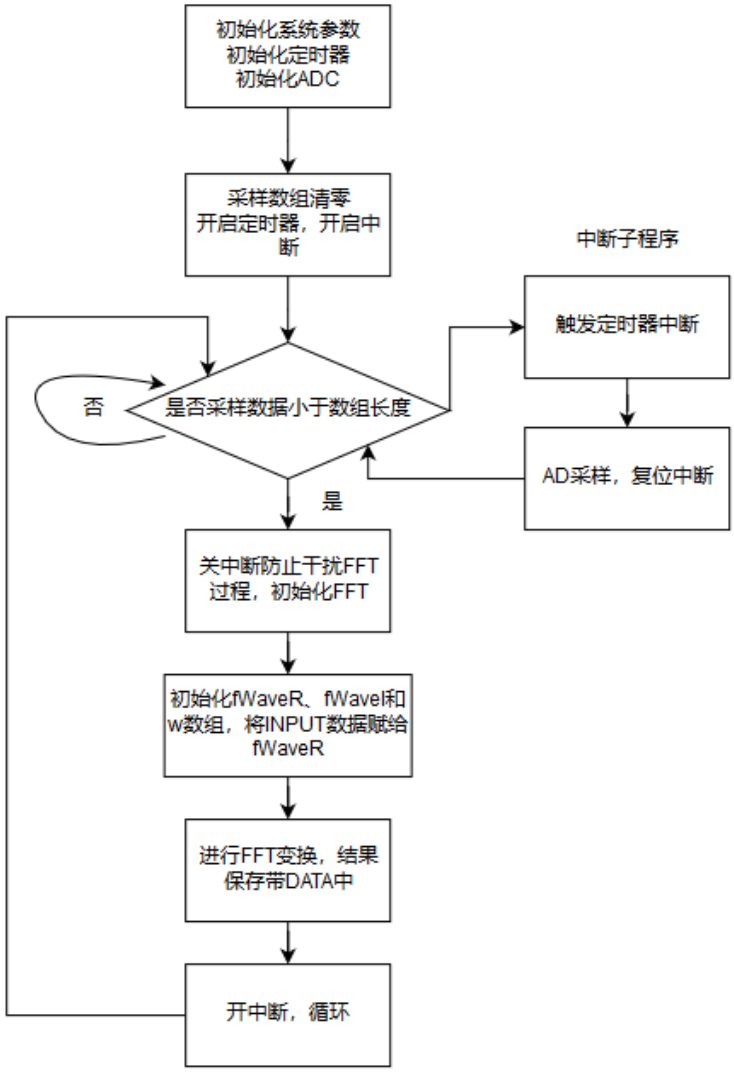
（2）变量定义与函数声明

```
Uint16 SampleTable[BUF_SIZE];  //存放AD采样数据
int array_index = 0;
void InitForFFT();
void FFT(float dataR[SAMPLENUMBER], float dataI[SAMPLENUMBER]);  //来
自FFT例程
int INPUT[SAMPLENUMBER], DATA[SAMPLENUMBER];
float fWaveR[SAMPLENUMBER], fWaveI[SAMPLENUMBER], w[SAMPLENUMBER];
float sin_tab[SAMPLENUMBER], cos_tab[SAMPLENUMBER];
```

```
interrupt void cpu_timer0_isr(void);   //定时器中断处理函数
```

（3）主函数

程序框图：



在实际程序中，忘记在主循环末尾开中断了，导致采样数据无法实时更新。

初始化程序部分即为将前面三个例程的初始化程序合并在一起。值得注意的是，代码：

```
#if (CPU_FRQ_150MHZ)
   ConfigCpuTimer(&CpuTimer0, 150, 1000); //定时器1us
#endif
#if (CPU_FRQ_100MHZ)
   ConfigCpuTimer(&CpuTimer0, 100, 1000);
   #endif
```

配置了采样频率，如图的采样频率为150MHz/（150*1000）=1000Hz，1us采样一次。这里要注意采样频率应满足采样定理，大于信号最高频率的两倍。

主循环开头：

```
   while (array_index< BUF_SIZE)
```

```
        {};  //采够BUF_SIZE的数据才进行下一步，FFT
        GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1;
        array_index = 0;//复位
        DINT;//关外部中断防止干扰FFT过程
        InitForFFT();//初始化FFT相关数组
        MakeWave();//将采集到的数据送到数组INPUT里*/
```

注意第一步程序的意义是，只有经过足够次数的中断程序，将采样数组采满，程序才会继续进行。

-FFT数组初始化：

```
        for (i = 0; i<SAMPLENUMBER; i++)
        {
            fWaveR[i] = INPUT[i];
            fWaveI[i] = 0.0f;
            w[i] = 0.0f;
            }
```

-FFT变换及数据保存：

```
        FFT(fWaveR, fWaveI);//进行FFT变换，fwaveR是实部，fwaveI是虚部
        for (i = 0; i<SAMPLENUMBER; i++)
        {
            DATA[i] = w[i];
            }//把结果放到DATA里保存起来*/
```

-中断子程序：

```
interrupt void cpu_timer0_isr(void)//定时器中断子程序
{
    CpuTimer0.InterruptCount++;
    CpuTimer0Regs.TCR.bit.TIF = 1;
    CpuTimer0Regs.TCR.bit.TRB = 1;

    AdcRegs.ADCTRL2.all = 0x2000;//软件启动AD转换，开启第二个通道
    while (AdcRegs.ADCST.bit.INT_SEQ1 == 0) {}//等待转换完成
    GpioDataRegs.GPBSET.bit.GPIO34 = 1; // Set GPIO34 for monitoring-optional
    AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;//清中断标志位
    SampleTable[array_index++] = ((AdcRegs.ADCRESULT2) >> 4);  //存第二个通道的数据
    // Break point
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
    }
```

由于信号源接在第二个ADC通道上，所以

```
AdcRegs.ADCTRL2.all = 0x2000;
```

对ADC通道开始AD转换之后，通过代码

```
SampleTable[array_index++] = ((AdcRegs.ADCRESULT2) >> 4)
```

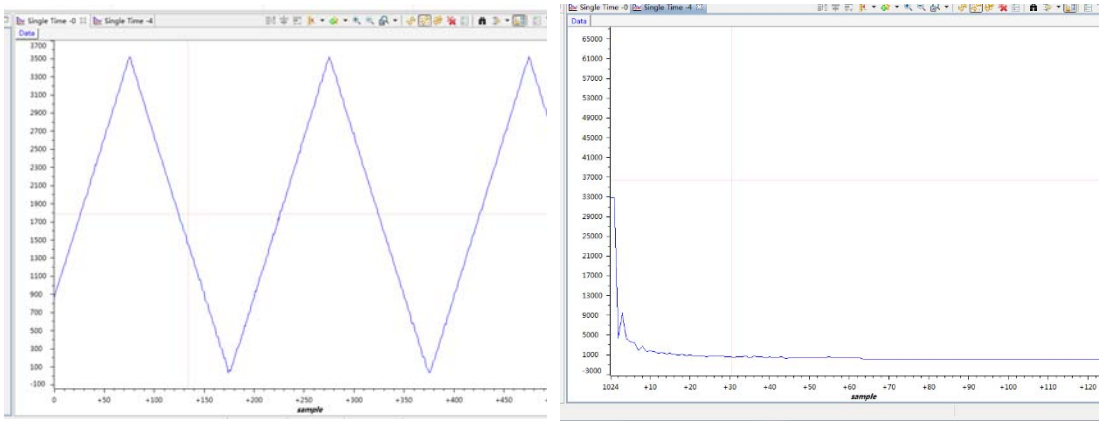
对第二个通道的采样寄存器读取数据。同时因为ADC是12位的，而寄存器是16位的，要将数据右移4位。

-makewave子程序，改为：

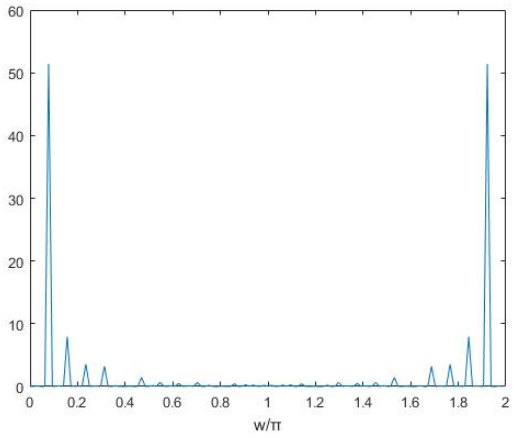
```
void MakeWave()
{ int i;
for ( i=0;i<SAMPLENUMBER;i++ )
{
//INPUT[i]=sin(PI*2*i/SAMPLENUMBER*3)*1024;
    INPUT[i]=SampleTable[i];
}
    }
```

将采样数据保存到INPUT里边。

最终对三角波的信号采样与FFT图像如下图：



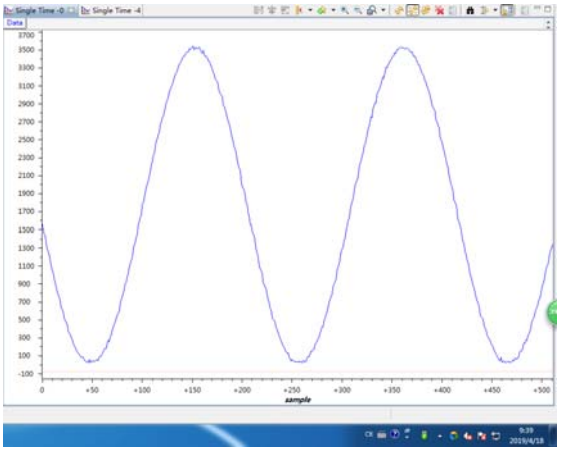
理论仿真如下：

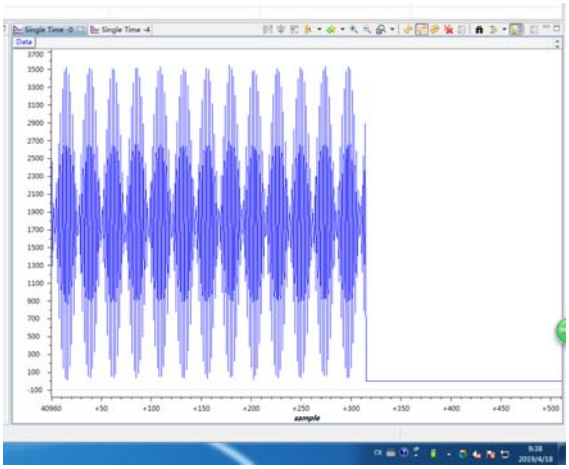


与理论推导相符。

5. 探究采样周期变化对FFT结果的影响

将采样频率有1000Hz改为100Hz，采样图由左图变为右图





可见出现了信号的失真。故采样一定要满足采样定理，否则会造成信号失真，导致严重后果。

七、 实验感悟

本次实验十分复杂，只有实验前做好充分的预习，再加上一定的运气成分，才可以完整做完实验。另外，理解代码一定要结合芯片的数据手册。

实验四 语音信号采集及处理综合实验

一、实验目的

1. 掌握基于DSP的语音信号采集及回放方法。
2. 掌握数字回声产生原理、编程及参数控制方法。
3. 掌握语音信号的频谱分析方法。

二、实验内容

1. 基本内容：完成语音信号的实时采集与回放（8分）

2. 综合拓展内容一：将键盘、液晶与基本实验内容进行综合，完成数字回声实验，要求能够通过按键控制回声的延迟时间及回声叠加幅度。（5分）

3. 综合拓展内容二：选取一段采集到的音频信号，绘制波形并进行播放（1分）；将上述音频信号进行反序，绘制波形并进行播放（1分）；计算并绘制原始的及反序的音频信号的数字幅度谱（1分）；计算并绘制原始的及反序的音频信号的数字相位谱（1分）；比较原始的及反序的音频信号的数字频谱，并对实验现象进行分析（1分）。

4. 自主设计完成一个综合实验（2分）。

三、实验设备

计算机， ICETEK-F28335-AF 实验箱，手机，测试导线

四、实验原理

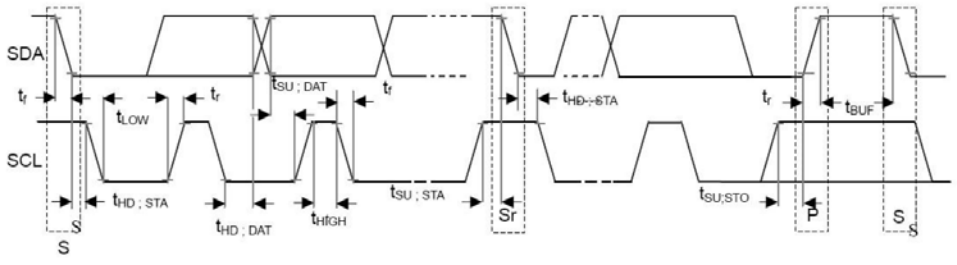
1．TLV320AIC23芯片控制方法

2．数字回声原理

详细内容见实验指导书。

3. DSP芯片与AIC23利用I2C协议进行通讯，对I2C协议进行介绍：

I2C（Inter-Integrated Circuit ,内部集成电路)总线是一种由飞利浦Philip公司开发的串行总线。I2C是两条串行的总线，它由一根数据线（SDA）和一根时钟线（SDL）组成。I2C总线上可以接多个I2C设备，每个器件都有一个唯一的地址识别。同一时间只能有一个主设备，其他为从设备。通常MCU作为主设备控制，外设作为从设备。



I2C时序图

在数据传输过程中，SCL时钟为主设备控制，SCL为高的时候读取SDA的数

据，SCL为低的时候，主设备改变SDA的数据准备传输下一位。数据从高位开始传输，当传输8位后，主设备会释放SDA总线。如果从设备正确接收到数据，则从设备会拉低SDA总线，则产生一个应答信号。如果从设备出错，不拉低SDA总线，由于上拉电阻的作用，SDA的电平会变为高电平，即为非应答信号。数据传输总是以开始信号开始传输，以结束信号终止传输，中间可以传输多个字节的数据。（http://www.waveshare.net/study/article-648-1.html）

五、实验步骤

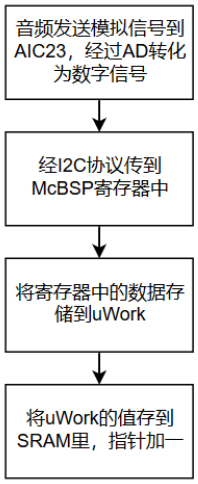
1. 连接硬件及建立工程。
2. 根据实验指导书提供的参考代码完成基本内容。
3. 完成综合实验一。
4. 完成综合实验二。
5. 完成自主设计综合实验。

六、实验程序流程及说明

1. 例程

实验现象：实时播放Beetles的《Let it be》

例程中最有价值的代码是关于声音的记录与播放的，程序框图如下：



存储：

```
while(!McbspaRegs.SPCR1.bit.RRDY);

uWork=McbspaRegs.DRR2.all; // 读取左右声道的数据
uWork=McbspaRegs.DRR1.all; // 因为耳机输入左右声道相同，所以读两次即可。我感觉这里有重复，uWork被重复赋值了

*(int *)(SRAM_Base_Adress+nAudioData)=uWork; //存储到SRAM里
```

播放的流程与存储的流程正好相反，读取SRAM，再发送给AIC23，DA转换后播放。

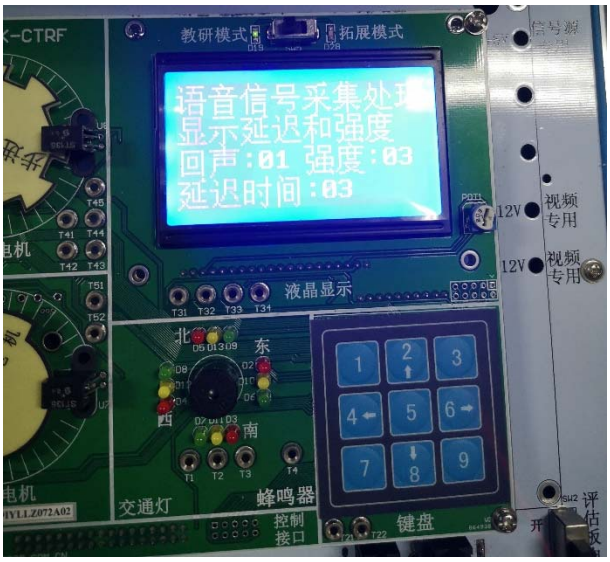
```
while(!McbspaRegs.SPCR2.bit.XRDY);
```

```
McbspaRegs.DXR2.all=uWork;
McbspaRegs.DXR1.all=uWork;
nAudioData++;

if(nAudioData>0x48000)

nAudioData=0;
```

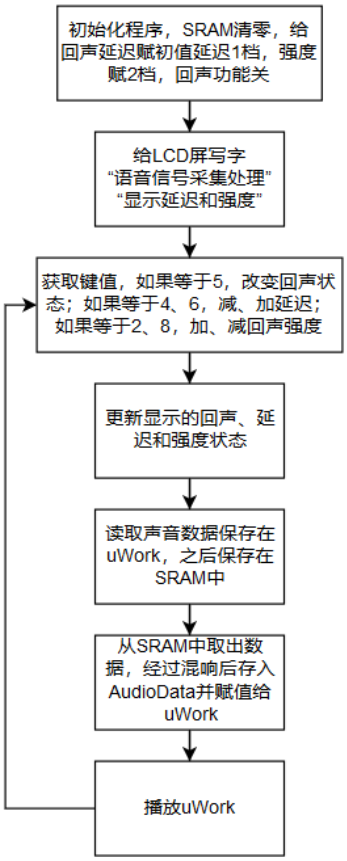
在例程的播放代码中，少了从SRAM中读取的步骤，这是因为uWork在存储的过程中值没有改变，可以直接用来播放。

2. 综合实验一：数字回声

实验现象：播放Beetles的《Let it be》，5键控制回声功能的开启与关闭；4, 6键控制延迟时间，4减少延迟，6增加延迟；2, 8键控制音量大小，2增大音量，8减少音量。音量与延迟均分为8档，1为最低，8为最高，0则消失。界面显示如下：



本实验的程序逻辑并不复杂，核心代码指导书中已经给出，需要注意的是工程的建立，头文件的添加等细节，以及读懂指导书中的程序，框图如下：



附部分实验代码：

初始化LCD屏，清空SRAM：

```
    ICETEKCTR_LCDPutString("语音信号采集处理",0,LCDLINE0); //从第一行第一个字符开始打
    ICETEKCTR_LCDPutString("显示延迟和强度",0,LCDLINE1);

for(i=0;i<=0x48000;i++){
*(int *)(SRAM_Base_Adress+i)=0;
}
uDelay=128; //回声延迟时间
    uEffect=256; //回声强度
```

按键判断与LCD数据更新：

```
uKeyCode=ICETEKCTR_GetKey();
    if (uKeyCode!=0)
    {
    ICETEKCTR_Delayms(20);
    if (uKeyCode==5)
        bEcho=!bEcho;
    else if (uKeyCode==4)
        uDelay -= 128;
    else if (uKeyCode==6)
        uDelay += 128;
```

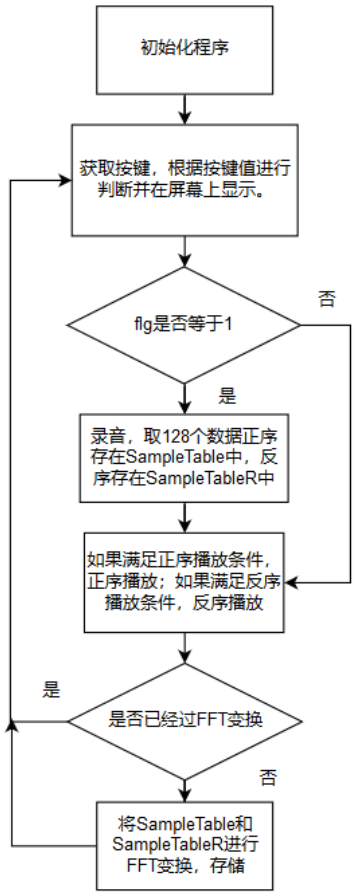
```
else if (uKeyCode==2)
    uEffect += 128;
else if (uKeyCode==8)
    uEffect -= 128;

buffer1[6]=bEcho+'0';
buffer1[14]=uEffect/128+'0';
buffer2[10]=uDelay/128+'0';

ICETEKCTR_LCDPutString(buffer1,0,LCDLINE2); //+ 第三行第1个字符开始
ICETEKCTR_LCDPutString(buffer2,0,LCDLINE3); //+ 第四行第1个字符开始
    }
```

其余代码指导书上均有，在此不再赘述。

3.综合实验二：

实验现象：按下1键，显示“Y”，开始录音；一段时间后，按下2键，显示“N”，停止录音。按下3键，显示“P”，正序播放录音，按下4键停止播放。按下5键，显示“N”，反序播放录音，按下6键停止播放。用编译器工具可以进行作图，幅值图正确，相位图不正确。

程序框图如下：



按键判断相关代码：

```
    uKeyCode=ICETEKCTR_GetKey(); //开始录音
    if(uKeyCode ==1 ){
        flg=1;
        nAudioData=0;
        ICETEKCTR_LCDPutString(buffer1,0,LCDLINE1);
    }
    else if(uKeyCode==2){       //停止录音
        flg=0; tag_fft=0; //重新进行FFT变换
        ICETEKCTR_LCDPutString(buffer2,0,LCDLINE1);
    }
    else if(uKeyCode==3){       //播放正序录音
        flg1=1;
        index=0;
        flg=0;flg2=0;
        ICETEKCTR_LCDPutString(buffer3,0,LCDLINE1);
    }
    else if(uKeyCode==4){       //停止播放正序录音
        flg1=0;flg2=0;
    }
    else if(uKeyCode==5){       //播放反序录音
        flg=0;
        flg2=1;flg1=0;
        index=nAudioData-1;
        ICETEKCTR_LCDPutString(buffer4,0,LCDLINE1);
    }
    else if(uKeyCode==6){       //停止反序录音
        flg2=0;flg1=0;
        }
```

FFT变换相关代码：

```
    if(i==BUF_SIZE&&tag_fft==0) //当sampletable采满了而且没有经过fft
时才会进行fft
    {
        //正序FFT
        InitForFFT();//初始化FFT相关数组
        MakeWave(SampleTable);//将采集到的数据送到数组INPUT里*/
        for (i = 0; i<SAMPLENUMBER; i++)
        {
            fWaveR[i] = INPUT[i];
            fWaveI[i] = 0.0f;
            w[i] = 0.0f;
        }
        FFT(fWaveR, fWaveI);//进行FFT变换，fwaveR是实部，fwaveI是虚部
        for (i = 0; i<SAMPLENUMBER; i++)
```

```
{
    DATA[i] = w[i];
    D_ANGLE[i]= a[i];
    }//把结果保存起来*/
```

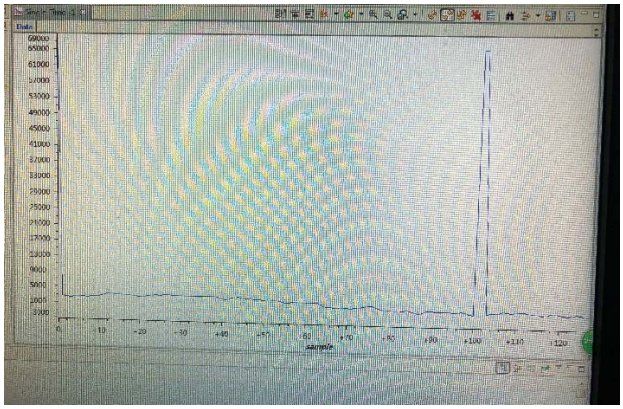
另外，我获取相角的代码如下：

```
a[i]=atan2(dataI[i],dataR[i])*180.0/PI;  //atan2可以求出象限    正180到负180
```

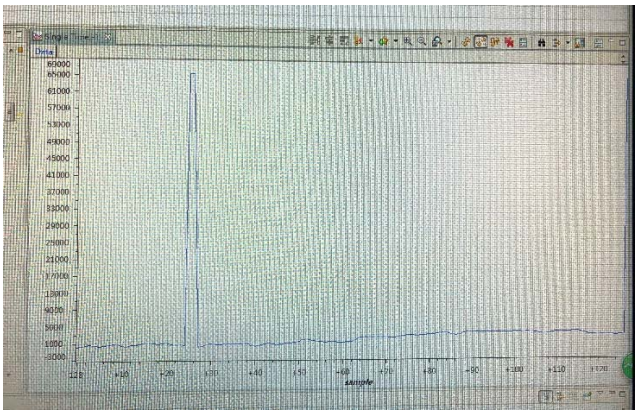
之所以没有选用atan函数，是因为atan无法获取具体的象限。但最后的相位图还是不对。

最终获取的信号图如下：

正序采样信号图

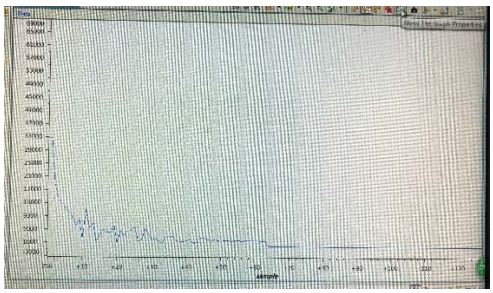


反序采样信号图

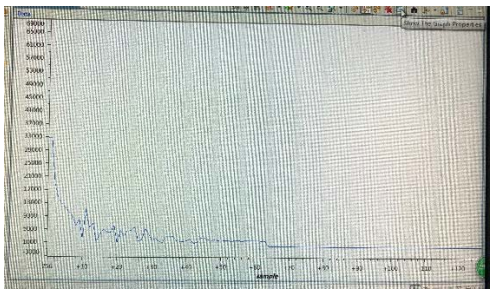


幅值图如下：

正序信号幅值图



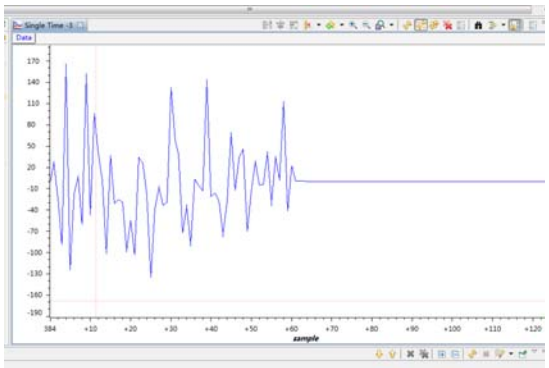
反序信号幅值图



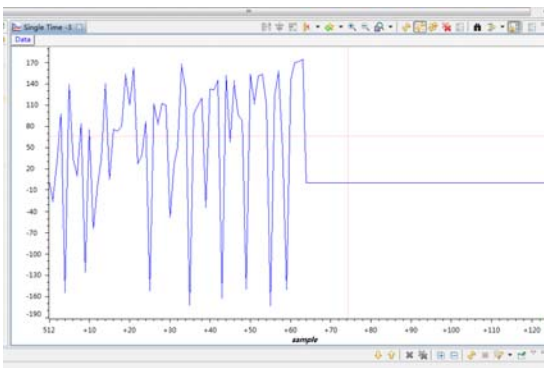
可以看到正序与反序信号的FFT变换幅值结果相同。

相位图如下（第一次录音时两个相位图相同，所以重录另一段发现不同的相位图如下）：

正序信号相位图



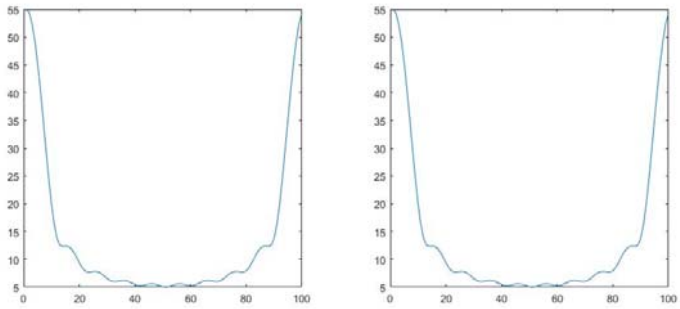
反序信号相位图



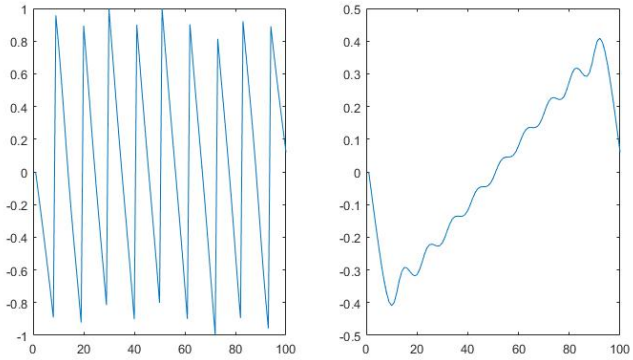
无法看出关系。为探究正序和反序FFT变换的正确关系，我用matlab对

1,2,3…，10信号和它的反信号进行FFT变换。

幅值图：



相位图：



据同学们说相位图应该是相反的，但我做出来是这样的，具体原因还需要进一步思考。

MATLAB验证代码：

```
clc,clear;
for i=1:10
a(i)=i;
end
for i=1:10
ar(i)=11-i;
end
af=fft(a,100);
arf=fft(ar,100);
figure;
subplot(1,2,1);
plot(abs(af));
subplot(1,2,2);
plot(abs(arf));
figure;
subplot(1,2,1);
for i=1:100

k(i)=imag(af(i))/real(af(i)
);
end
plot(angle(af)/pi);
% plot(k);
subplot(1,2,2);
for i=1:100

kr(i)=imag(arf(i))/real(arf
(i));
end
% plot(kr);
plot(angle(arf)/pi);
```

在编译代码的过程中，我们遇到了代码过长，内存不够的问题。我们问了很多同学也问了老师，都说是缩短数组长度，但我们的数组已经非常短了。后来上网搜索了一下，发现可以改变内存的分配来解决问题。

根据网络资源，需要将一个.cmd文件的代码，将PAGE0的指令

RAML2 : origin = 0x00A000, length = 0x001000 去掉，并将RAML2的存储空间与RAML1合并，由0x001000加到0x002000。

同时，要将PAGE1的RAML4和RAML5指令进行修改如下

RAML4 : origin = 0x00C000, length = 0x001800

RAML5 : origin = 0x00D800, length = 0x000800

增加RAML4的内存而减少RAML5的。

我认为在不同PAGE里，RAMLX存储了不同的内容，可能RAML1就是存储代码的。加长这一部分的存储空间，就可以解决程序过长的问题。

七、实验感悟

DSP实验到此结束了，感觉第四个实验特别综合，我们做出来的部分也最少。究其原因，很大程度是因为我们浪费了大量的时间在建工程上。我们做这个实验的时候，刚开始我们建的工程一直无法编译通过，报错也非常莫名其妙，显示找不到一个明明已经添加进去的.h文件。后来试了半个多小时，只好用其他人的工程，把我们的代码拷进去，这样就可以编译通过了。DSP实验非常迷幻，因为出错有可能是代码问题，有可能是编译器问题，也有可能是硬件电路的问题。我觉得这个实验的核心不应该在建工程上，所以最好能在前一二次的指导书上提醒我们拷一下已经建好的工程，让之后的实验在这个工程上跑，这样才能让我们在有限的时间内学到更多的东西，而不是把时间都浪费在建工程上。另外，如果可以更细致地介绍一下DSP芯片的基本原理就更好了。

另，如果报告有问题或者需要源码，请联系我Tel：15652587808或者E-mail：lijinjie@buaa.edu.cn，谢谢老师！