北京交通大学

DSP课程设计

实 验 报 告

利用 DSP 实现实时滤波

学　院：电子信息工程学院

老　师：高海林老师

班　级：通信 0705

设计者：陈川峰　07221125

　　刁苏蒙　07211066

联系方式：51689590

北京交通大学

目　录

一、实验任务

1、实验背景

在信号与信息处理中，提取有用信息就要对信号进行滤波。利用DSP可以实时地对信号进行数字滤波。本设计要求利用DSP的DMA方式进行信号采集和信号输出，同时对外部输入的信号进行数字滤波。

自适应滤波不仅能够选择信号，而且能够控制信号的特性。自适应滤波器具有跟踪信号和噪声变化的能力，它的系数能够被一种自适应算法所修改。利用 DSP 可以实时地对信号进行自适应滤波。DSP 利用直接存储器访问方式 DMA 采集数据时不打扰 CPU，因此 CPU 可以对信号进行实时地滤波。本设计要求利用 DSP 的 DMA 方式进行信号采集和信号输出，同时对外部输入的信号进行数字滤波。

2、实验要求

1.建立信号处理系统的概念，学会使用DSP处理器；
2.了解DSP处理系统的关键器件的使用方法；
3.掌握DSP课程设计的基本方法，巩固信号处理的基本理论;
4.掌握查阅有关资料和使用器件手册的基本方法，学会阅读原版英文资料；
5.掌握DSP集成开发环境的使用和调试方法；
6.掌握DSP片外资源和片上资源访问的基本方法，如存储器、McBSP、DMA、A/D和 D/A转换器等。

二、设计内容

1、基本部分：

（1）对DMA进行初始化；
（2）对A/D、D/A进行初始化；
（3）编写DMA中断服务程序，实现信号的实时滤波；
（4）利用CCS信号分析工具分析信号的频谱成分，确定滤波器的参数，利用MATLAB设计数字滤波器，提取滤波器参数；
（5）设计数字滤波算法，或调用DSPLIB中的滤波函数，实现对信号的滤波。
（6）比较加不同窗和阶数时滤波器的滤波效果；
（7）测试所设计滤波器的幅频特性和相频特性，并与MATLIB下的设计结果进行比较。

2、拓展部分：

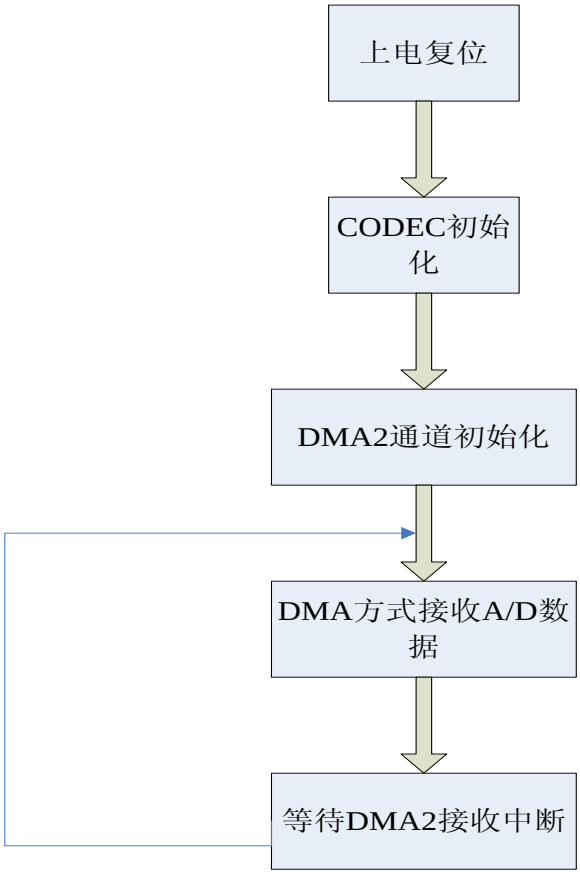
（1）滤波后信号实时输出的同时，将数据存放在数据文件中；
（2）利用自适应滤波实现语音信号回波对消。

三、设计方案、算法原理说明

1、设计方案

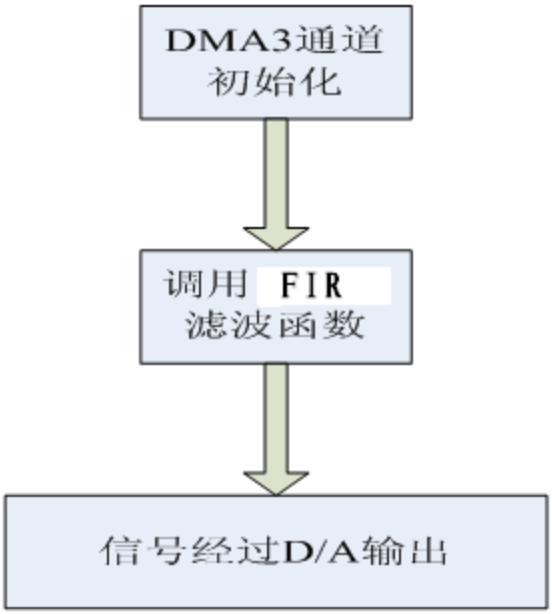
流图如下：

【1】 主程序流图



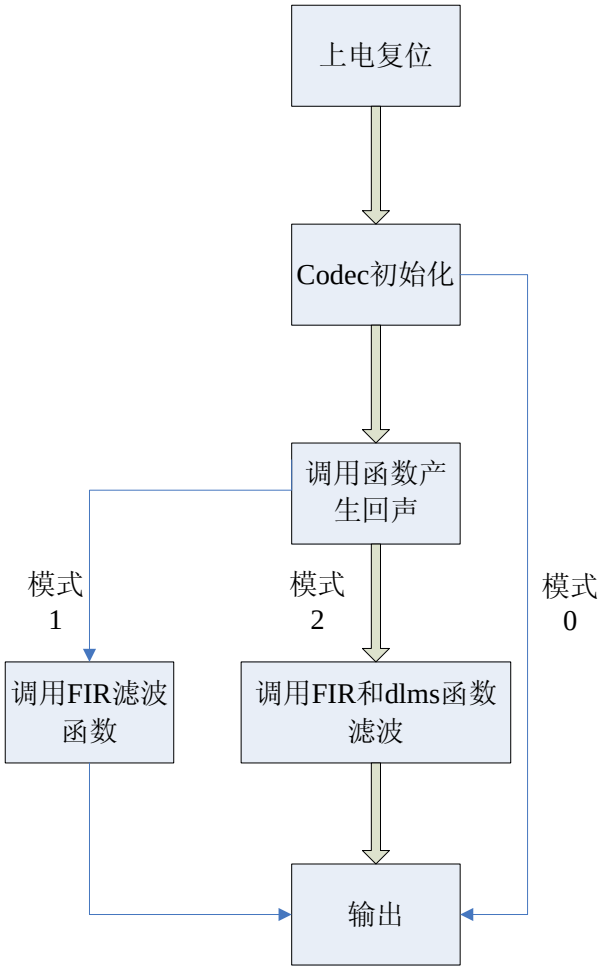
简要说明：模拟音频进过 codec 电路（codec 已设置好初值），转化为数据流，存放于缓冲区中，用于对数据处理。DMA 初始化，设置初值，用于控制 DSK 板相应模块。利用 DMA 实现对信号的实时处理，减少占用 CPU 的资源。等待 DMA2 中断，无限循环控制对信号实时处理。

【2】 中断服务程序流图



简要说明：初始化 DMA 通道 3,然后启动 FIR 滤波函数后数据的发送，经过处理后的数据经过 codec 电路中的 D/A 模块输出。中断服务程序用于主函数的入口处，供实时调用。调用的 FIR 滤波函数。

【3】 回波产生于消除流图

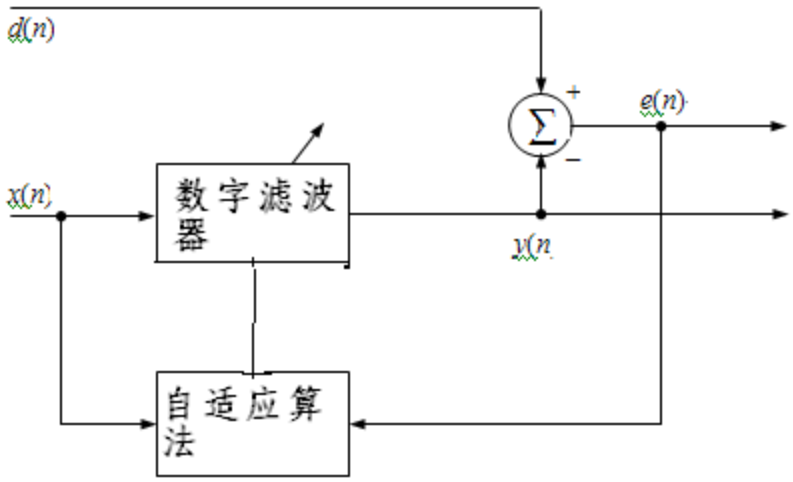


简要说明：模拟的语音信号先经过初始化过的 codec 电路，然后转为数据流，存放于指定的位置，用于处理。数据流经过处理变为带有回声的数据信号，然后通过调用相应的 FIR 滤波函数和自适应滤波函数 dlms 对数据进行处理，即又转化为与输入基本相同的语音信号，达到了实现产生回波，然后再消除的结果。中间设置不同的模式，课根据不同模式观察有回波，没加 dlms 函数处理后和有回波，加 dlms 函数处理后相互比较，以便比较程序的正确性和可实践性。

2、算法原理：

【1】自适应滤波器的结构

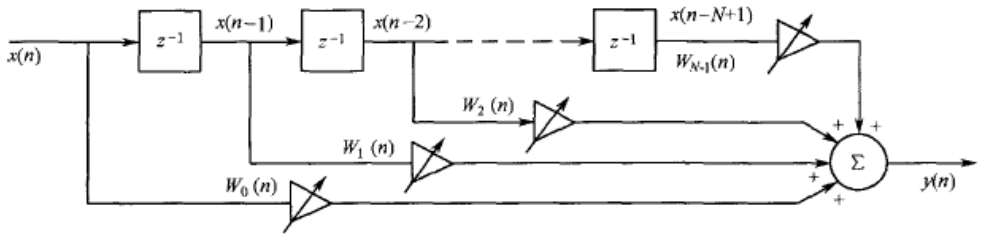
自适应滤波器的特性变化是由自适应算法通过调整滤波器的系数来实现的. 所以，自适应滤波器一般都由两部分组成：一是滤波器结构，它为完成期望的处理功能而设计；二是自适应算法，它调节滤波器系数以改进性能. 自适应滤波器的一般形式如下图，图中输入信号 (n) 加权到数字滤波器产生输出信号 y(n)，自适应算法调节滤波器权系数使输出 y(n)和滤波器期望的响应 x(n)之间的误差信号 e(n)为最小. 自适应滤波器的系数受误差信号的控制，根据 e(n)的值和自适应算法自动调整. 一旦输入信号的统计规律发生了变化，滤波器能够自动跟踪输入信号的变化，自动调整滤波器的权系数，实现自适应过程，最终达到滤波效果。



自适应滤波的一般形式

x(n)———— 自适应滤波器的输入；
y(n)———— 自适应滤波器的输出；
d(n) ———— 期望响应；
e(n) ————估计误差；
e(n)=d(n)－y(n)；

自适应滤波器的结构可以采用 FIR 或 IIR 结构，但实际应用中一般采用 FIR 型结构，主要原因是 FIR 结构的滤波器只包含正向通路，是固有稳定的，实现更容易，其权系数的修正就调节了滤波器的性能，计算量小且很稳定；而 IIR 滤波器同时兼有正向通路和反馈通路，内部反馈可能导致滤波器震荡，存在稳定性问题。所以，在自适应滤波器的应用中，一般采用 FIR 滤波器．故本次实验选择 FIR 型滤波器。而自适应 FIR 滤波器结构又可分为三种结构类型：横向型结构（Transversal Structure）、对称横向型结构（Symmetric Transversal Structure）、格型结构（Lattice Structure）。本实验所采用的是自适应滤波器设计中最常用的 FIR 横向型结构。结构图如下：



横向型滤波器结构

横向型滤波器的结构示意图

其中 x(n)——自适应滤波器的输入
w(n)——自适应滤波器的冲激响应: w(n)={w(0),w(1),…,w(N-1)}
y(n)——自适应滤波器的输出: y(n)=x(n)*w(n)

【2】LMS 算法简介

自适应算法的种类较多，最常用的自适应算法是最小均方误差算法，即 LMS 算法（Least Mean Square），LMS 算法是一种易于实现、性能稳健、应用广泛的算法。所有的滤波器系数调整算法都是设法使 y(n) 接近 d(n)，所不同的只是对于这种接近的评价标准不同。LMS 算法的目标是通过调整系数，使输出误差序列 e(n)=d(n)−y(n) 的均方值最小化，并且根据这个判据来修改权系数，该算法因此而得名。误差序列的均方值又叫“均方误差”MSE（Mean Square Error）

最常用的判据是最小均方误差，即理想信号 d(n)与滤波器输出 y(n)之差 e(n)的期望值最小，并且根据这个判据来修改权系数 wi(n)。由此产生的算法称为 LMS。均方误差表示为：

$$\varepsilon = \mathrm{MSE} = \mathrm{E}[e^{w}(n)] = \mathrm{E}[(d(n) - y(n))^2]$$

对于横向结构的滤波器，代入的表达式有：

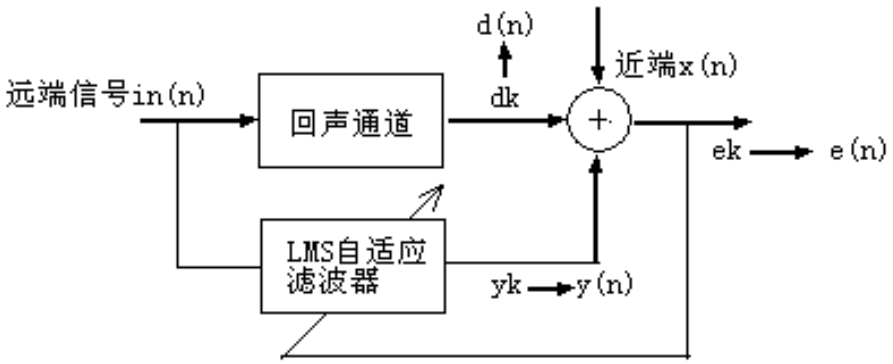
$$\varepsilon = \mathrm{MSE} = \mathrm{E}[d^2(n)] + \mathbf{W}^T(n)\mathbf{R}\mathbf{W}(n) - 2\mathbf{W}^T(n)\mathbf{P}$$

其中，$\mathbf{R} = [\mathbf{X}(n)\ \mathbf{X}^T(n)]$为 N×N 自相关矩阵，是输入信号采样值间的相关性矩阵。P＝E[d(n)X(n)]为 N×1 互相关矢量，代表理想信号 d(n)与输入矢量的相关性。在均方误差达到最小时，得到最佳权系数 $\mathbf{W}^* = [w_0^*, w_1^*, \lambda\ w_{N-1}^*]^T$ 。

说明：本次实验通过调用 displib.h 中的函数：short dlms（）来实现自适应滤波。具体见程序。

【3】回波原理与设计

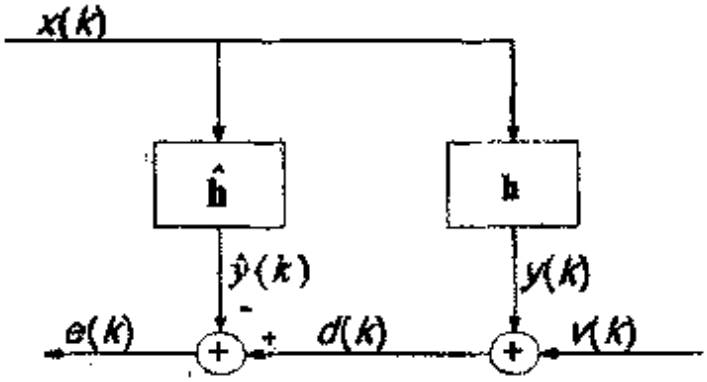
a、声学回声消除的功能原理框图如下图所示：



b、回声消除的基本原理可以概括如下:

自适应地合成回声，并从有回声干扰地信号中减去该合成回声。

回声消除器的基本原理图如下：



一般回声消除算法通过自适应滤波来完成，其基本原理如上图所示。其中，远端信号城 x（k)通过回声信道 h 产生回声 y(k)，近端信号 d(k)是由回声 y(k)
馄和近端声音代句(可包含噪声信号)得到。通过使用 M 抽头的 FIR 自适应滤波器来模拟回声信道 h，可以使所得 y(k)通近回声信号，进而达到回声消除的目的由此可见，回声消除的关键是自适应地调整$\tilde{h}$使其通近 h，可通过现有的各种自适应滤波算法实现。本实验自适应滤波调用的是系统的库函数 dlms 函数来实现的

四、程序设计、调试与结果分析

1、程序设计

基本要求部分【1】和【2】

【1】滤波主函数源程序

```
/*****************************************************************************/
/* FIRlab.C    AIC,McBSP,DMA initialization for FIR lab Filter                */
/*****************************************************************************/

/*****************************************************************************/
/*包含文件                                                                   */
/*****************************************************************************/

#include <type.h>
#include <board.h>
#include <codec.h>
#include <dma54xx.h>
#include <timer.h>
#include <intr.h>
#include <string.h>
void delay(s16 period);
extern void DMAC2ISR();

/*****************************************************************************/
/*  全局变量                                                                 */
```

```
/*****************************************************************************/

HANDLE hHandset;
unsigned int dmsefc, dmmcr, dmctr, src_addr, dst_addr;
unsigned int dmpre, dmsrcp, dmdstp, dmidx0, dmidx1, dmfri0, dmfri1, dmgsa, dmgda, dmgcr,
dmgfr;

/*************************************************************/

/*  建立输入数据段存放输入数据(inp_buffer)*/
#pragma DATA_SECTION(inp_buffer,"audio_buffer");
int inp_buffer[0x200];

/*建立输入数据段存放输入数据(out_buffer)*/
#pragma DATA_SECTION(out_buffer,"outt_buffer");
int out_buffer[0x200];

/*建立脉冲相应数据区*/
#pragma DATA_SECTION(coeffs,"coefficients");
/* Low    Pass Filter fs=16000 fc=4000Hz */
  int coeffs[21]={38,        78,        0,    -303,    -723,    -817,        0,    2006,    4775,
    7226,    8208,    7226,    4775,    2006,        0,    -817,    -723,    -303,
            0,        78,        38};                                        //kasier 窗设计，低通
/* High Pass Filter */
//int coeffs[21]={ -27,      102,        0,    -396,      510,      441,   -1799,     1083,     3369,
  //    -9420,   12284,   -9420,     3369,     1083,   -1799,      441,      510,     -396,
  //          0,      102,      -27};                                         //kasier 窗设计，高通

/*  建立延迟缓冲数据段，存放延迟数据(delaybuff)*/
#pragma DATA_SECTION(delaybuff,"delayb");
int delaybuff[21]={0};

/*为 DMAC ISR 申明和初始化 2 个变量*/
int frame=0;
int flag=0;
int temp;
int currbuff = 0;

/* delayptr 保持延迟缓冲区地址*/

int *delayptr1 = &(delaybuff[0]);

interrupt void DMAC2ISR();
```

```
/*****************************************************************************/
/*                                主程序                                     */
/*****************************************************************************/

void main()
{
    s16 cnt=1;
    BSCR = 0x8806;
    XPC = 0;
    PMST = 0xA0;//设置 IPTR(中断指针)
    brd_set_cpu_freq(100);
    TIMER_HALT(0);
    TIMER_RESET(0);
    IMR=0;              //禁止所有中断


/* ------------------------------------------- */

    if(brd_init_bios())
        return;
while(cnt--)
    {
        brd_led_toggle(BRD_LED0);     //切换 LED 指示灯 0 的显示状态
        delay(1000);
        brd_led_toggle(BRD_LED1);     //切换 LED 指示灯 1 的显示状态
        delay(1000);
        brd_led_toggle(BRD_LED2);     //切换 LED 指示灯 2 的显示状态
        delay(1000);
    }

/*********************************************************
*                                                        *
*                    CODEC 初始化                        *
*                                                        *
*********************************************************/

/* Codec 初始化 */

    hHandset = codec_open(HANDSET_CODEC);              /* 创建 codec 句柄 */

/* 设置 codec 参数 */
```

```
    codec_dac_mode(hHandset, CODEC_DAC_15BIT);              /* DAC 转化器以 15-bit 模式*/
    codec_adc_mode(hHandset, CODEC_ADC_15BIT);              /* ADC 转换器以 15-bit 模式*/
    codec_ain_gain(hHandset, CODEC_AIN_6dB);                /* 模拟输入 DAC 到 6dB 增益*/
    codec_aout_gain(hHandset, CODEC_AOUT_MINUS_6dB);        /* 模拟输入 DAC 到-6dB 增益*/
    codec_sample_rate(hHandset,SR_16000);                   /* 16KHz 采样率*/

/*********************************************************
*                                                        *
*                    DMA 通道 2  初始化                  *
*                                                        *
*********************************************************/

/*清除 IFR 中断标志寄存器寄存器 */

   INTR_CLR_FLAG(DMAC2);

/*  复位所有通道（DMA)*/

    dma_reset_all();

/***************************************/
/*        初始化 DMA 通道 2 的通道寄存器        */
/***************************************/

    dmsefc = ((DSYNC_REVT1 <<12));//与 MCBSP1 接受事件同步
    dmmcr = ((AUTOINIT_ENABLE << 15) | (DINM_ENABLE << 14) | (IMOD_HALFBLOCK <<13) | (CTMOD_DEC <<12) | (INDEXMODE_NOMOD << 8) | (SPACE_DATA << 6) | (INDEXMODE_INC << 2) | (SPACE_DATA));
    //设置传输控制模式寄存器 DMMCR
    dmctr = 0xFF;//单元计数，256 单元/frame
    src_addr = DRR1_ADDR(HANDSET_CODEC);//设置源地址，DMA 源地址为 mcbsp1 接受寄存器
    dst_addr = (unsigned int) &buffer;//设置目的地址，DMA 目的地址为 buffer

    dma_init(DMA_CH2, dmsefc, dmmcr, dmctr, SPACE_DATA, src_addr, SPACE_DATA, dst_addr);//调用 DMA 初始化函数

/*  设置通道 2 的 frame*/
```

```
    DMA_FRAMECOUNT(DMA_CH2, 2);

/*****************************************/
/*          为 DMA 通道 2 初始化全局寄存器          */
/*****************************************/

/* 为 DMA CH2 输入设置全局自动初始化寄存器*/

    dmgsa = src_addr;
    dmgda = dst_addr;
    dmgcr = 0xFF;//设置每 frame 大小为 256
    dmgfr = 2;

/* 为 CH2 设置全局优先级和使能控制寄存器*/

    dmpre = ((HIGH_PRIORITY << 10) | (INTSEL_01 << 6));
    dmsrcp = SPACE_DATA;
    dmdstp = SPACE_DATA;
    dmidx0 = 0;
    dmidx1 = 0;
    dmfri0 = 0;
    dmfri1 = 0;

    dma_global_init(dmpre, dmsrcp, dmdstp, dmidx0, dmidx1, dmfri0, dmfri1, dmgsa, dmgda,
dmgcr, dmgfr);

/*********************************/
/*          使能 DMA 通道 2              */
/*********************************/

/* 使能通道 2 */

    DMA_ENABLE(DMA_CH2);

/* 开始输入缓冲流的主串行口，读 MCBSP 的数据接收寄存器*/
    temp = *(volatile u16*)DRR1_ADDR(HANDSET_CODEC);

/*********************************/
/*          DMA 通道 2 中断使能        */
/*********************************/

/* 使能 DMAC2 中断 */
```

```
    INTR_ENABLE(DMAC2);

/*  使能全局中断  */

    INTR_GLOBAL_ENABLE;

/*********************************/
/*          等待 DMA 通道 2 中断       */
/*********************************/

/*  等待 DMAC2 中断，无限循环*/

    for(;;);
}


/*****************************************************************************/
/*                                    延迟子程序                                 */
/*****************************************************************************/

void delay(s16 period)
{
    int i, j;

    for(i=0; i<period; i++)
    {
        for(j=0; j<period>>1; j++);
    }
}
```

【2】FIR 滤波的中断服务程序

```
#include <codec.h>
#include <dma54xx.h>

extern    unsigned int channel;          //DMA 通道
extern    unsigned int dmsefc;           //设置同步事件和 frame 计数寄存器的初始值
extern    unsigned int dmmcr;            //设置模式寄存器值
extern    unsigned int dmctr;            //设置单元计数寄存器值
extern    unsigned int src_page;         //设置源页寄存器值
extern    unsigned int src_addr;         //设置源地址寄存器值
extern    unsigned int dst_page;         //设置目的页寄存器值
extern    unsigned int dst_addr;         //设置目的地址寄存器值
```

```
extern int buffer[0x200];
extern int coeffs[21];
extern int delaybuff[21];
extern int frame;
extern int flag;
extern int currbuff;
extern int *delayptr1;
extern int step = 1;

/*********************************************************
*                                                        *
*          初始化 DMA 通道 3,启动 FIR 滤波函数后数据的发送   *
*                                                        *
*********************************************************/

void init_dma3(void)
{
        while(DMPREC&0x0008) {};     //DMA3 的传送是否结束

        /*  初始化 DMA 通道 3 */

        dmsefc = ((DSYNC_REVT1 <<12));//与 MCBSP1 接收事件同步
        dmmcr = 0x4141;
    //AUTOINIT=0,DINM=1,IMOD=0,CTMOD=0,SIND=0x001;DMS=0x01,DIND=0x000,D
MD=0x01（传输模式控制寄存器）
        dmctr = 0xFF;            //256 单元/frame（单元计数器）
        src_addr = (unsigned int) &buffer+0x300+((unsigned int)currbuff*0x100);//设置
DMA3 通道的源地址，即读取此地址上的数据
        dst_addr = DXR1_ADDR(HANDSET_CODEC);//将数据放到 MCBSP1 的发送
寄存器发送出去

        dma_init(DMA_CH3, dmsefc, dmmcr, dmctr, SPACE_DATA, src_addr,
SPACE_DATA, dst_addr);//调用初始化函数

         /*设置通道 3 的 frame*/

        DMA_FRAMECOUNT(DMA_CH3, 1);          //2 frame/block

        /*使能通道 3*/

        DMA_ENABLE(DMA_CH3);
}
```

```
/*********************************************************
*                                                        *
*DMA 通道 2 的中断服务程序,对接收到的 DMA 数据进行滤波   *
*                                                        *
*********************************************************/
 interrupt void DMAC2ISR(void)
{
      int *p_inp,*p_out;
      p_inp=inp_buffer+frame*0x100;
      p_out=out_buffer+frame*0x100;
      fir(p_inp,coeffs,p_out,&delayptr1,21,256);
      init_dma3();
      frame^=1;

}
```

拓展部分

程序说明：

a、本程序设置了三种模式，模式 0（mode=0）的输出结果是语音经过一个全通滤波器后直接输出，模式 1（mode=1）输出结果是语音信号经过变换产生回波，然后经过 FIR 滤波后输出，模式 2（mode=2）输出结果是语音信号经过变换产生回波，然后经过 FIR 滤波和 dlms 自适应滤波后输出。三种模式相互对比，效果显著，符合实验要求。

b、步骤：先初始化 FIR 延迟输出数组 dbuffer_h[]、FIR 输出数组、自适应滤波器系数矢量 coff_w[]和自适应滤波器的输出数组 out_w[]，再进行 DSK 板的初始化。然后采集输入的语音信号，根据模式的不同进行不同的处理以输出不同的信号来进行效果的比较。

【3】回波产生与消除的主函数（源程序）

```
#include <type.h>
#include <board.h>
#include <codec.h>
#include <mcbsp54.h>
#include <tms320.h>
#include <dsplib.h>
#include <setm.h>

void delay(void);
void update(DATA x[],DATA dk);
void initarray(DATA x[]);
/*************************************************************************/
/*  定义全局变量                                                         */
/*************************************************************************/
```

```
HANDLE hHandset;        //CODEC 句柄
DATA bf1[5000];         //缓冲区
/*****************************************************************************/
/*                                  主函数                                   */
/*****************************************************************************/
DATA *dp_w = &dbuffer_w[0];
DATA *dp_h = &dbuffer_h[0];
void main()
{
    s16 j;
    s16 m;
    s16 cnt=2;                          //灯循环闪次数,初始化等待时钟周期数
    DATA dk,out_delay,yk;//ek
    s16 mode=2;                         //决定是否产生回波是否使用回波抵消
    s16 i;
    if (brd_init(100))                  //板的初始化
        return;
    for (i=0; i<LENGTH_IN; i++) dbuffer_h[i] = 0;
    for (i=0;i<LENGTH_IN;i++) out_h[i] =0;    //清除 output buffer (optional)
    for (i=0;i<LENGTH_W;i++) coff_w[i] =0;    //清除 coeff buffer (optional)
    for (i=0;i<LENGTH_IN;i++) out_w[i] =0;    //清除 output buffer (optional)

    while ( cnt-- )                     /*板子初始化，用灯闪来表示*/
    {
        brd_led_toggle(BRD_LED0);
        delay();
        brd_led_toggle(BRD_LED1);
        delay();
        brd_led_toggle(BRD_LED2);
        delay();
    }

    /* 打开 Handset Codec */
    hHandset = codec_open(HANDSET_CODEC);           /*codec 需要一个句柄 */
    /*设置 codec*/
    codec_dac_mode(hHandset, CODEC_DAC_15BIT);      /* DAC 转化器以 15-bit 模
式*/
    codec_adc_mode(hHandset, CODEC_ADC_15BIT);      /* ADC 转换器以 15-bit 模
式*/
    codec_ain_gain(hHandset, CODEC_AIN_6dB);        /* 模拟输入 DAC 到 6dB 增
益*/
    codec_aout_gain(hHandset, CODEC_AOUT_MINUS_6dB);    /* 模拟输出 DAC 到-6dB
增益*/
```

```
codec_sample_rate(hHandset,SR_16000);                    /* 16KHz 采样率*/
brd_led_toggle(BRD_LED0);

initarray(x);                                            //初始化所有数组
  i=0,j=3000,m=1500;

    while(1)                                             //循环
  {
    while (!MCBSP_RRDY(HANDSET_CODEC)) {};
    dk = *(volatile u16*)DRR1_ADDR(HANDSET_CODEC); //读 DXR1 寄存器
    update(x,dk);

    if(mode==0)          //模式 0
    {
    *(volatile u16*)DXR1_ADDR(HANDSET_CODEC)=dk;   //写 DXR1 寄存器
    }
    else if(mode==1) //模式 1
    {
    fir(x,coff_h,out_h,&dp_h,LENGTH_H,LENGTH_IN);
    if(i==5000) i=0;
    if(j==5000) j=0;
    if(m==5000) m=0;
    bf1[i]=out_h[0];i++;                                 //放入缓冲区

    out_delay=0.5*bf1[i+1]+1.5*bf1[j++]+bf1[m++];
    yk=dk+out_delay;
    *(volatile u16*)DXR1_ADDR(HANDSET_CODEC)=yk;

    }
    else if(mode==2)//模式 2
    {

    fir(x,coff_h,out_h,&dp_h,LENGTH_H,LENGTH_IN);        //调用 FIR 滤波函
数

    if(i==5000) i=0;
    if(j==5000) j=0;
    if(m==5000) m=0;

    bf1[i]=out_h[0];i++;

    out_delay=0.5*bf1[i+1]+1.5*bf1[j++]+bf1[m++];
    yk=dk+out_delay;
```

```
            update(y,yk);

            dlms(y,coff_w,out_w,&dp_w,out_h,STEP,LENGTH_W,LENGTH_IN);// 调用自适应滤
波函数
            *(volatile u16*)DXR1_ADDR(HANDSET_CODEC)=out_w[i];
            }

          }

}




void delay(void)                    //延迟
{
   long int j;
   for(j=0; j<100000; j++)
   asm("_nop");
}

void update(DATA x[],DATA dk)//改变数组
{
      s16 j,k;
      for(j=1;j<LENGTH_IN;++j)
      {
            k=LENGTH_IN-j;
            x[k]=x[k-1];
      }
      x[0]=dk;
}

void initarray(DATA x[])      //初始化数组
{
       s16 i;
       for(i=0;i<LENGTH_IN;++i)
       {
           x[i]=0;
       }

}
```

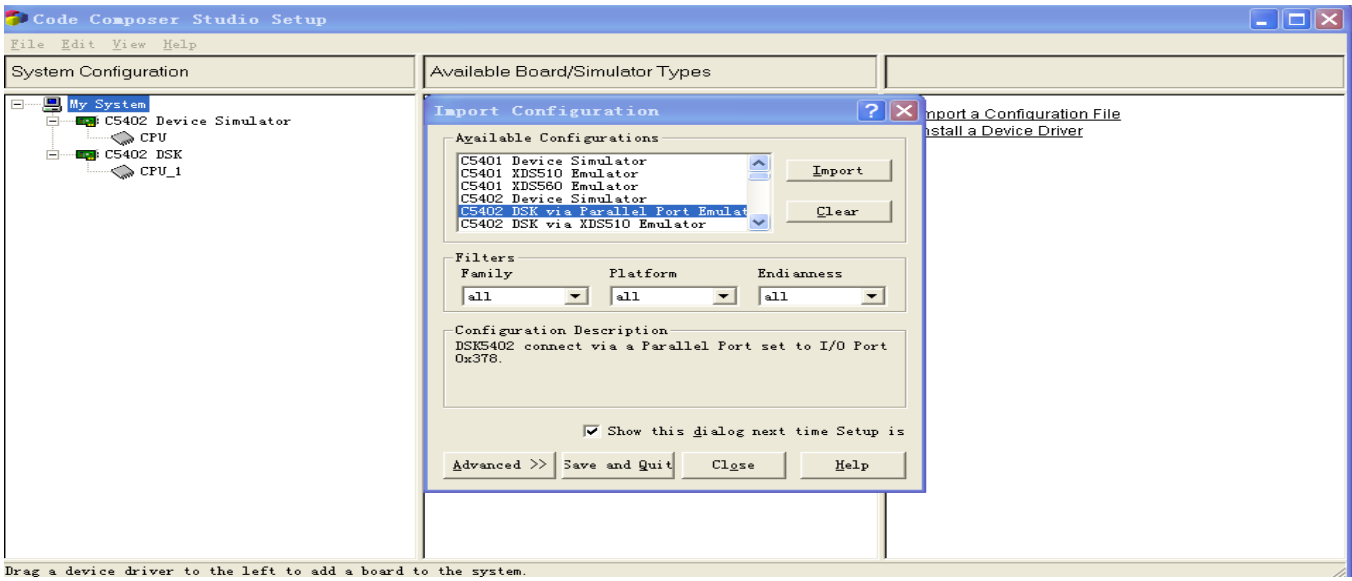
【4】定义的 stem.h 库函数源程序如下：

```
#define LENGTH_W    10                          //LMS 滤波器长度,
```

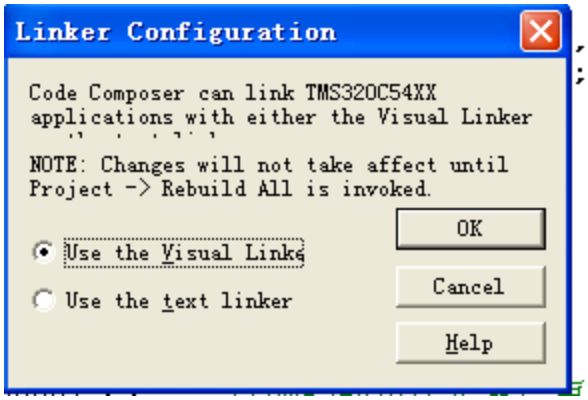
```
#define LENGTH_H    10                          //自适应滤波器长度
#define STEP 100                                //自适应步长
#define LENGTH_IN 5                             //输入缓冲数组 x[]长度
DATA x[LENGTH_IN];                              //声明输入缓冲数组
DATA y[LENGTH_IN];                              //声明有回波时输入数组
DATA out_h[LENGTH_IN];                          //FIR 滤波器即回声消除通道输出
DATA out_w[LENGTH_IN];                          //自适应滤波器输出
DATA e[LENGTH_IN];
#pragma DATA_SECTION(coff_w,".coffw")           //将数组 coff_w 指定到内存段.coffw
DATA coff_w[LENGTH_W];                          //声明自适应滤波器系数矢量
#pragma DATA_SECTION(coff_h,".coffh")           //将数组 coff_h 指定到内存段.coffh
DATA coff_h[LENGTH_H]={790,2661,4629,6919,8210,8210,6919,4629,2661,790};
                                                //定义 FIR 滤波器的系数即回声通道的权
系数
#pragma DATA_SECTION(dbuffer_h,".dbufferh")     //将数组 dbuffer_h 指定到内存段.dbufferh
DATA dbuffer_h[LENGTH_IN];    //该数组存放 FIR 上一时刻的输出
#pragma DATA_SECTION(dbuffer_w,".dbufferw")     //将数组指定到内存段.dbufferw
DATA dbuffer_w[LENGTH_IN];                      //该数组存放自适应滤波器上一时刻输出
```

2、调试与观察

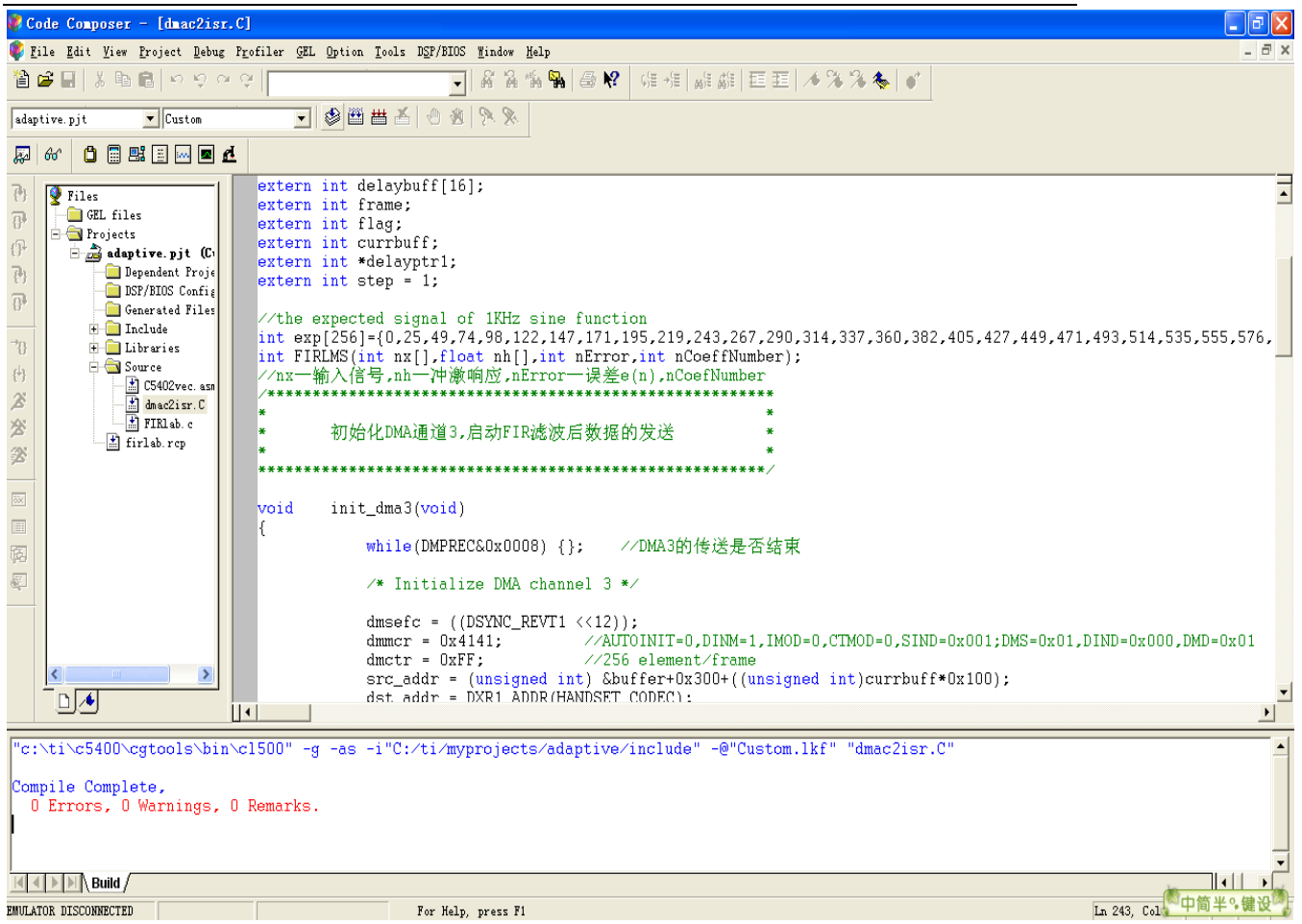
【1】 CCS2.0 的设置

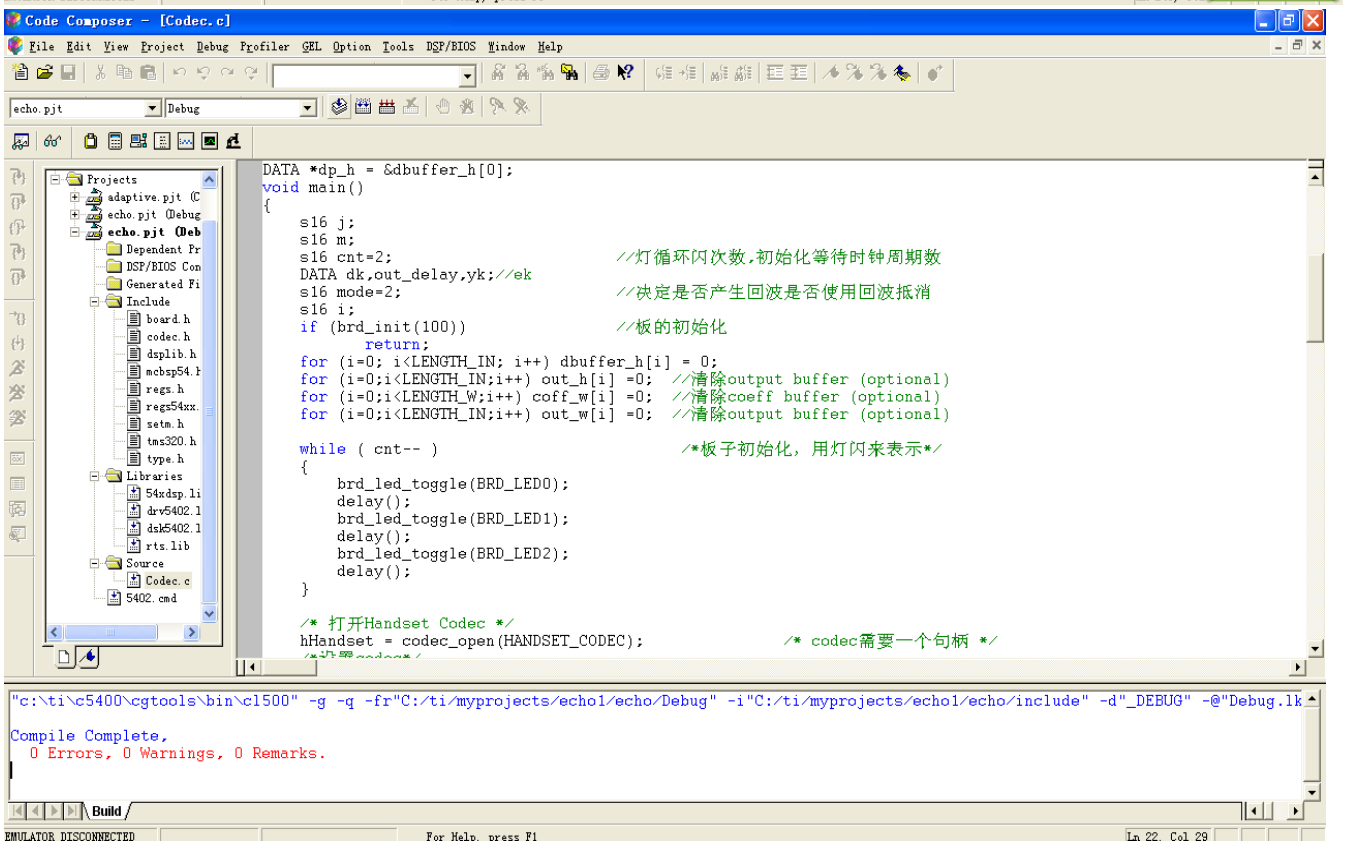


【2】linker configuration 设置：



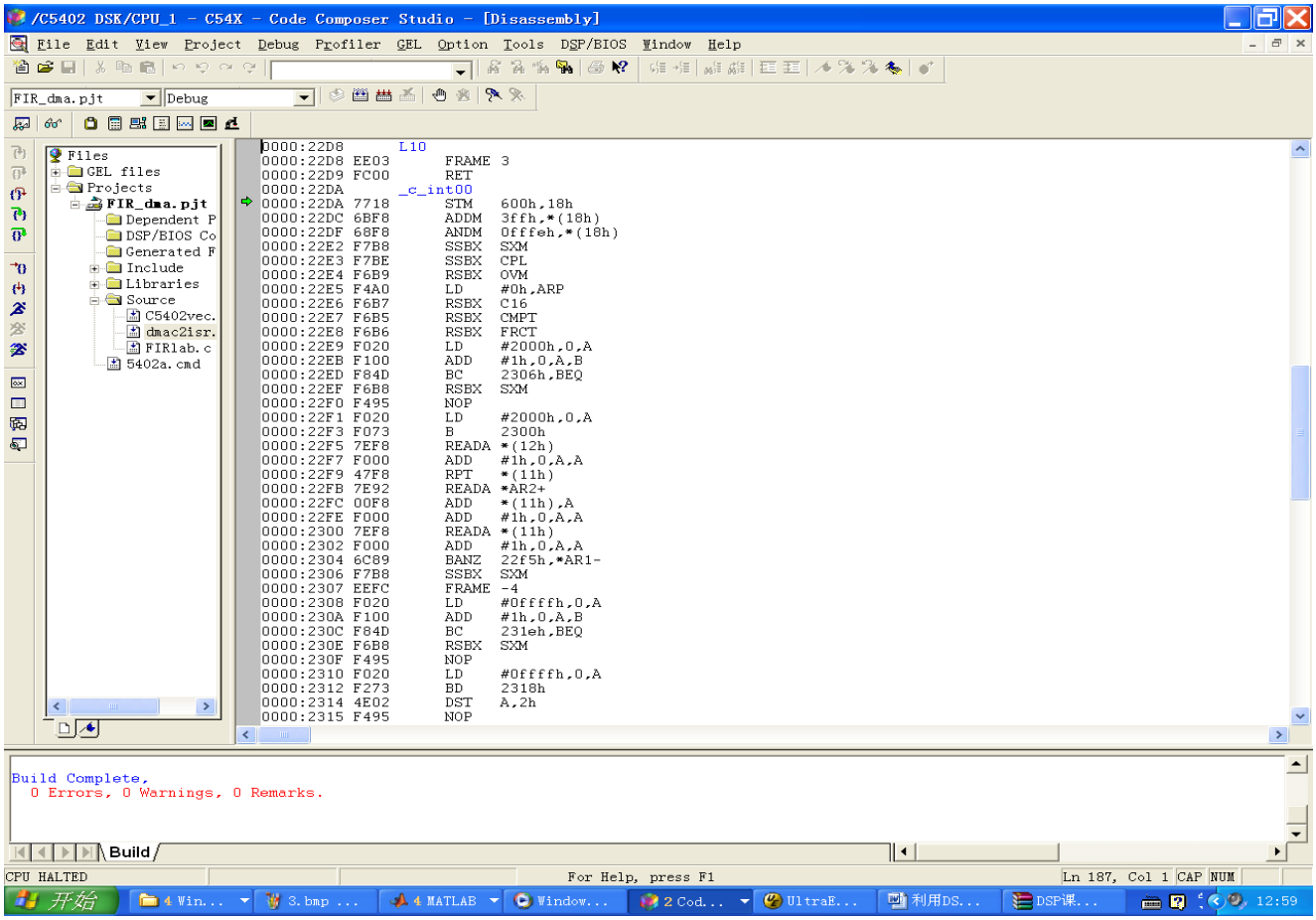
【3】程序的编译，连接（无误）：


Code Composer - [dmac2isr.C]
File Edit View Project Debug Profiler GEL Option Tools DSP/BIOS Window Help
adaptive.pjt
Custom
Files
GEL files
Projects
adaptive.pjt
Dependent Proje
DSP/BIOS Config
Generated Files
Include
Libraries
Source
C5402vec.asm
dmac2isr.C
FIRlab.c
firlab.rcp
extern int delaybuff[16];
extern int frame;
extern int flag;
extern int currbuff;
extern int *delayptr1;
extern int step = 1;
//the expected signal of 1KHz sine function
int exp[256]={0,25,49,74,98,122,147,171,195,219,243,267,290,314,337,360,382,405,427,449,471,493,514,535,555,576,
int FIRLMS(int nx[],float nh[],int nError,int nCoeffNumber);
//nx—输入信号,nh—冲激响应,nError—误差e(n),nCoefNumber
/***
*
*
初始化DMA通道3,启动FIR滤波后数据的发送
*
*
***/
void
init_dma3(void)
{
while(DMPREC&0x0008) {};
//DMA3的传送是否结束
/* Initialize DMA channel 3 */
dmsefc = ((DSYNC_REVT1 <<12));
dmmcr = 0x4141;
//AUTOINIT=0,DINM=1,IMOD=0,CTMOD=0,SIND=0x001;DMS=0x01,DIND=0x000,DMD=0x01
dmctr = 0xFF;
//256 element/frame
src_addr = (unsigned int) &buffer+0x300+((unsigned int)currbuff*0x100);
"c:\ti\c5400\cgtools\bin\cl500" -g -as -i"C:/ti/myprojects/adaptive/include" -@"Custom.lkf" "dmac2isr.C"
Compile Complete,
0 Errors, 0 Warnings, 0 Remarks.
Build
EMULATOR DISCONNECTED
For Help, press F1
Ln 243, Col


Code Composer - [Codec.c]
File Edit View Project Debug Profiler GEL Option Tools DSP/BIOS Window Help
echo.pjt
Debug
Projects
adaptive.pjt
echo.pjt (Debug
echo.pjt (Deb
Dependent Pr
DSP/BIOS Con
Generated Fi
Include
board.h
codec.h
dsplib.h
mcbsp54.h
regs.h
regs54xx.
setm.h
tms320.h
type.h
Libraries
54xdsp.li
drv5402.l
dsk5402.l
rts.lib
Source
Codec.c
5402.cmd
DATA *dp_h = &dbuffer_h[0];
void main()
{
s16 j;
s16 m;
s16 cnt=2;
//灯循环闪次数,初始化等待时钟周期数
DATA dk,out_delay,yk;//ek
s16 mode=2;
//决定是否产生回波是否使用回波抵消
s16 i;
if (brd_init(100))
//板的初始化
return;
for (i=0; i<LENGTH_IN; i++) dbuffer_h[i] = 0;
for (i=0;i<LENGTH_IN;i++) out_h[i] =0;
//清除output buffer (optional)
for (i=0;i<LENGTH_W;i++) coff_w[i] =0;
//清除coeff buffer (optional)
for (i=0;i<LENGTH_IN;i++) out_w[i] =0;
//清除output buffer (optional)
while (cnt--)
/*板子初始化，用灯闪来表示*/
{
brd_led_toggle(BRD_LED0);
delay();
brd_led_toggle(BRD_LED1);
delay();
brd_led_toggle(BRD_LED2);
delay();
}
/* 打开Handset Codec */
hHandset = codec_open(HANDSET_CODEC);
/* codec需要一个句柄 */
"c:\ti\c5400\cgtools\bin\cl500" -g -q -fr"C:/ti/myprojects/echo1/echo/Debug" -i"C:/ti/myprojects/echo1/echo/include" -d"_DEBUG" -@"Debug.lk
Compile Complete,
0 Errors, 0 Warnings, 0 Remarks.
Build
EMULATOR DISCONNECTED
For Help, press F1
Ln 22, Col 29

【4】程序载入

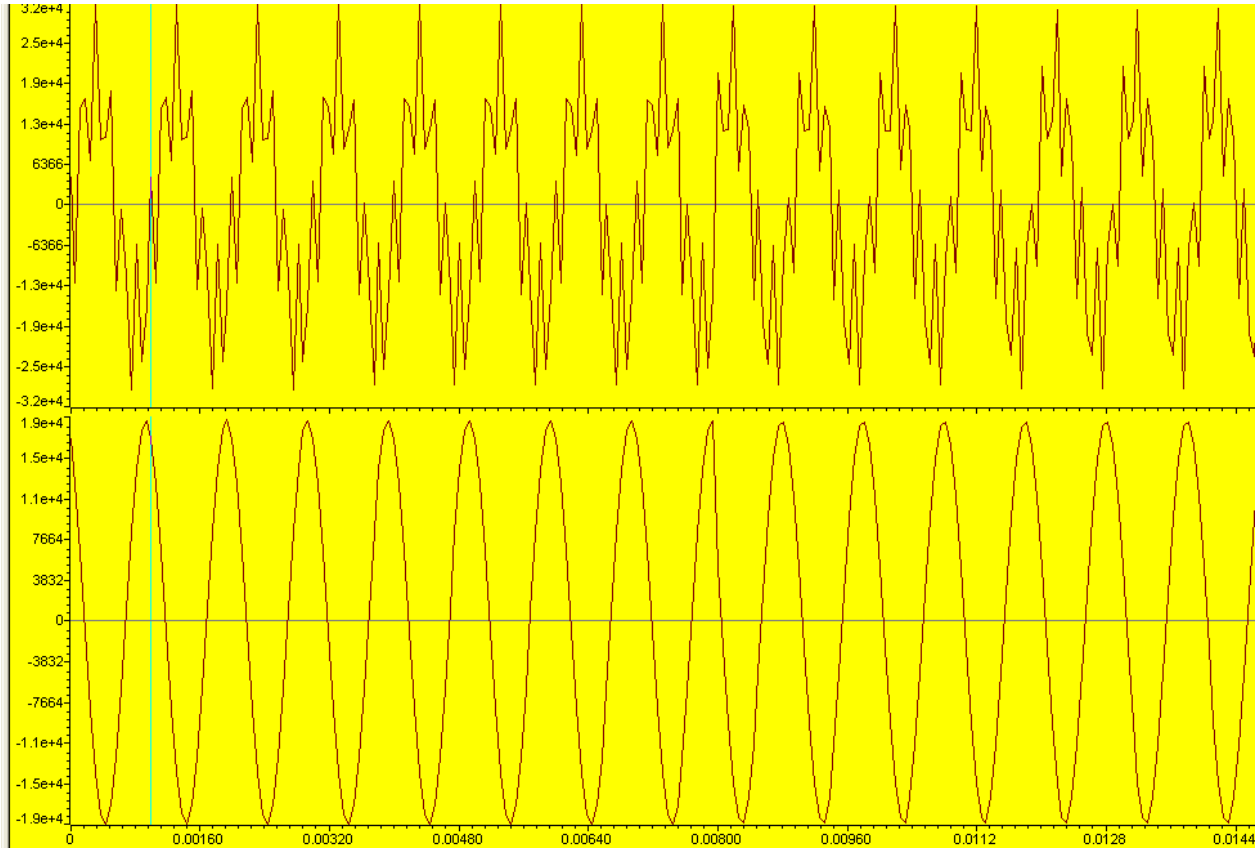
编译，连接无误后，即可产生.out 的文件，然后在菜单栏中 File 选项选择 Load Program 即可把.out 文件载入到 DSK 板子上，如下图：



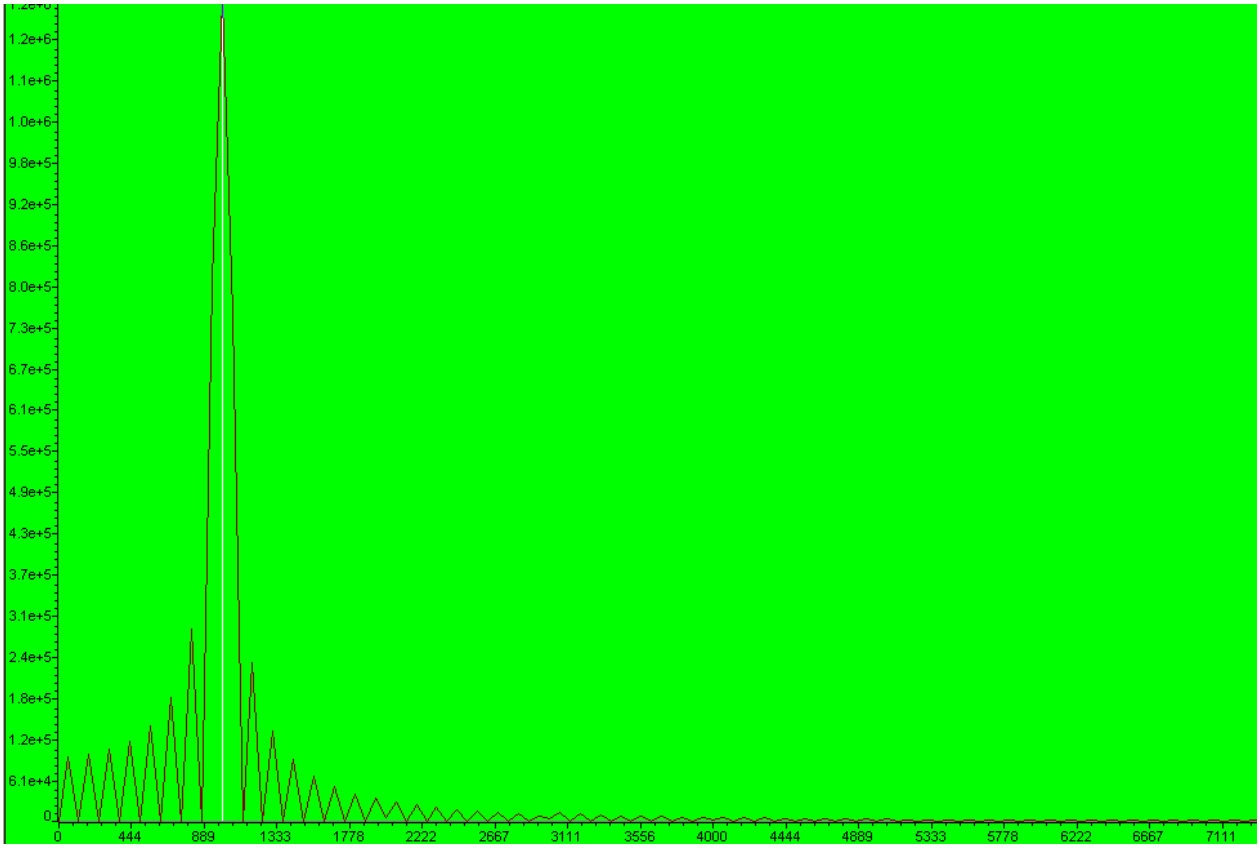
【5】利用 Graph 观察输入输出波形：

20 阶 kasier 窗低通滤波器（滤 6khz）

a/双通道时域图：

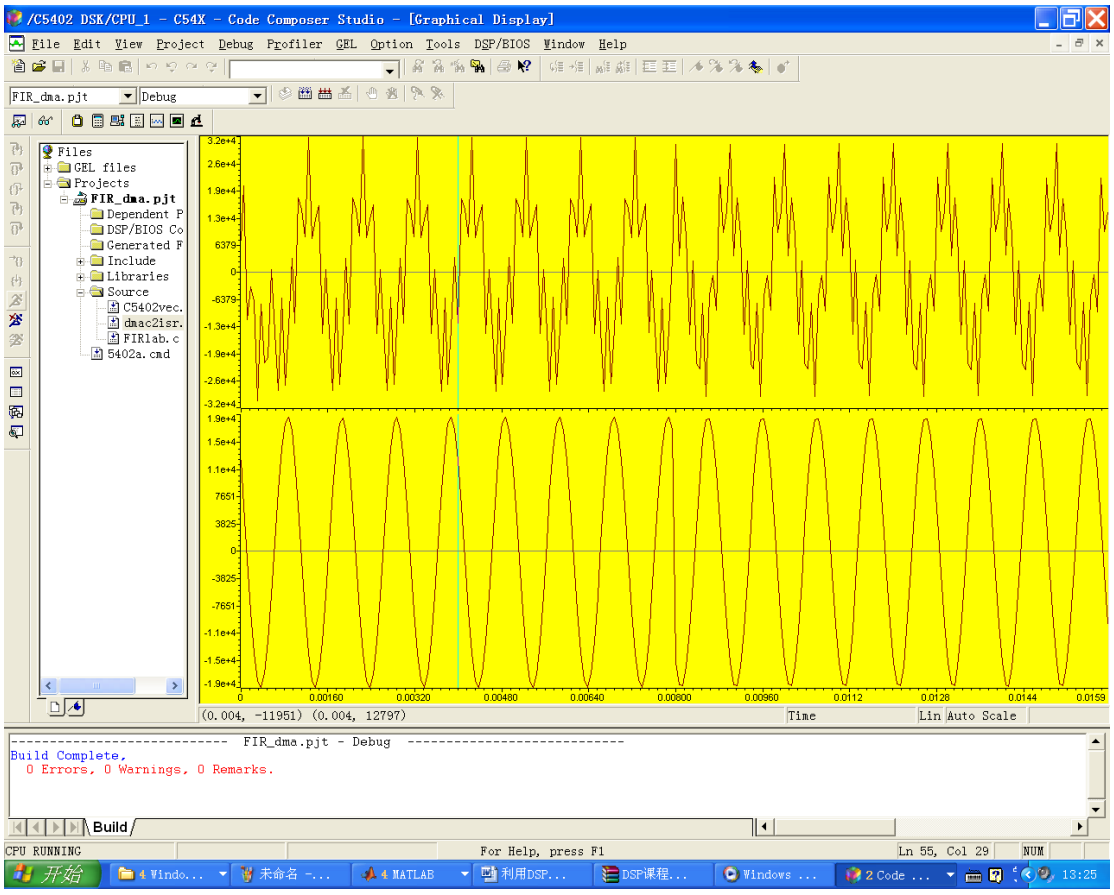

3.2e+4
2.5e+4
1.9e+4
1.3e+4
6366
0
-6366
-1.3e+4
-1.9e+4
-2.5e+4
-3.2e+4
1.9e+4
1.5e+4
1.1e+4
7664
3832
0
-3832
-7664
-1.1e+4
-1.5e+4
-1.9e+4
0
0.00160
0.00320
0.00480
0.00640
0.00800
0.00960
0.0112
0.0128
0.0144

b/单通道频域图（输出）：

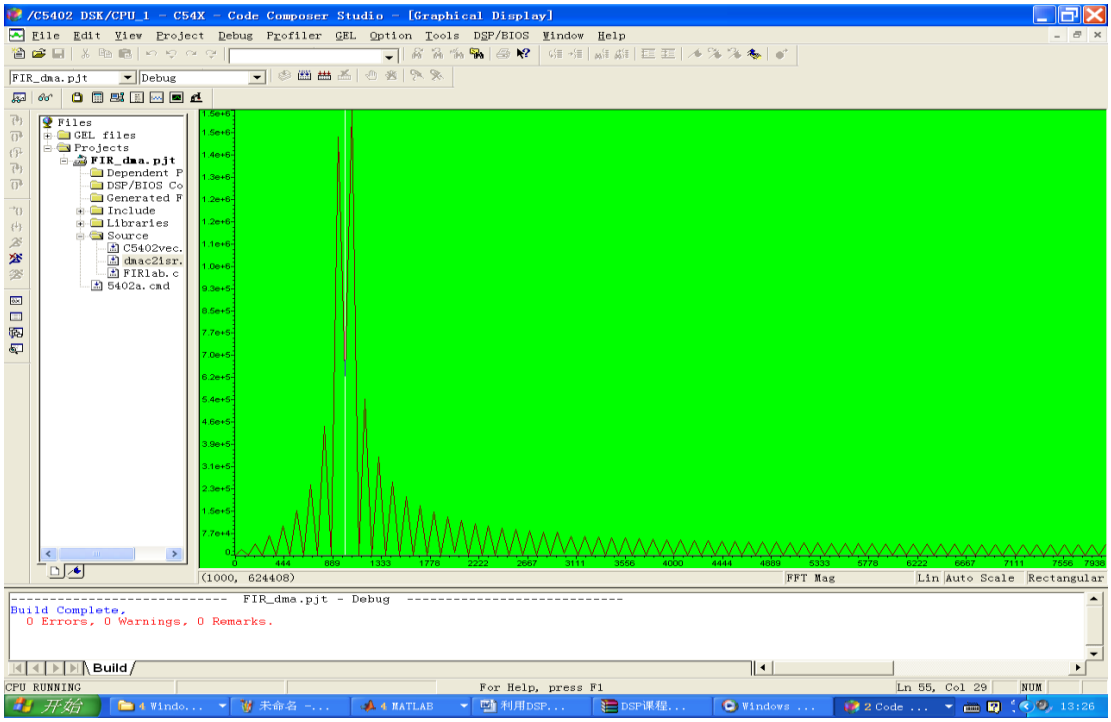

1.2e+6
1.1e+6
1.0e+6
9.8e+5
9.2e+5
8.6e+5
8.0e+5
7.3e+5
6.7e+5
6.1e+5
5.5e+5
4.9e+5
4.3e+5
3.7e+5
3.1e+5
2.4e+5
1.8e+5
1.2e+5
6.1e+4
0
0
444
889
1333
1778
2222
2667
3111
3556
4000
4444
4889
5333
5778
6222
6667
7111

20 阶 hamming 窗低通滤波器（滤 6khz）

a/双通道时域图：

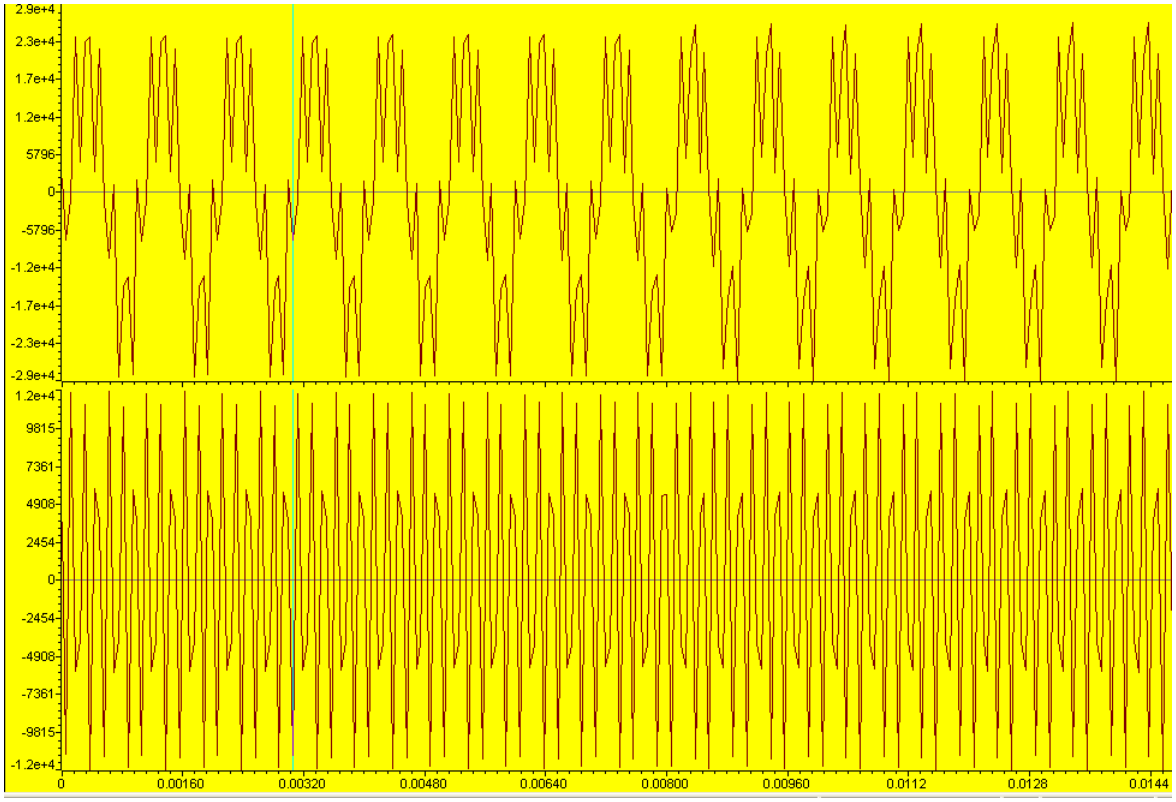


b/单通道频域图（输出）：

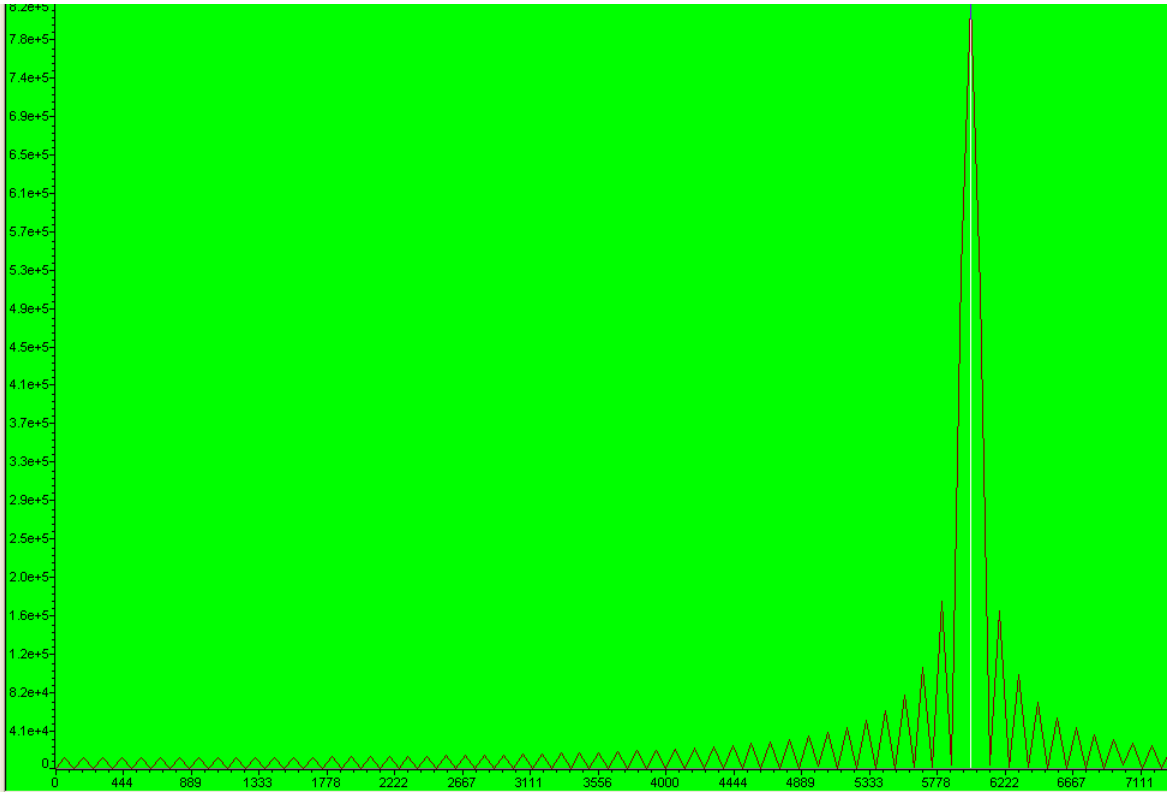


20 阶 kasier 窗高通滤波器（滤 1khz）

a/双通道时域图：



b/单通道频域输出图：



3、结果分析

a、基本实现了实验的基本要求，能实现实时滤波。

同阶比较 kasier 窗与 hamming 窗：

由图可知 kasier 窗的效果比 hamming 窗好，波形失真小

不同阶数比较：

阶数越高，效果也明显。

b、拓展部分效果显著，各个模式均能实现。

五、设计（安装）与调试的体会

陈川峰总结与感想:

DSP 课程设计是一门综合的实验课程。理论的学习和实践均是本课程的关键。几个节课的理论学习，我对 DSP 课程设计有了初步的了解。在此期间我也到过实验室调过几次板子，不过掌握的不是很深刻，只知道如何利用老师的程序调试。课程设计需要自己完成设计，编写与调试。因此我把小学期规划为三个部分。

小学期前段期间，我主要把精力用在学习 DSP 的理论知识上，学习的关键是理论知识的掌握与理解。主要涉及到硬件部分的编程，包括：硬件部分 codec 的应用与设置，多通道串行口 McBSP 的作用与设置，DMA 的应用与初始化等，软件部分包括 C 语言的在 C54XX 中的编写，涉及到 I/O 空间访问，中断服务程序的编写，cmd 文件的应用与编写等。前期的主要是一些基础知识，为后期的工作做准备。

小学期中段期间。主要是程序的设计与编写。由于初次接触 DSP，对它的了解不是很深，自己的想法比较少，有些想法还不知道是否能实现，是否具有可行性。因此，我和我的搭档查找了很多关于 DSP 课程设计的资料，同时还不定期的和老师进性沟通和交流。在此期间，我学到了很多的东西，对 DSP 课程也有了深入的理解。结合我们现有的水平，我们小组选着了利用 DSP 实现自适应滤波这个题目，难度适中。题目是选好了，可是我们还不知道如何进行下一步的工作。如何设计整个框图，进而编写每个模块的程序更是关键。我们结合书本上的知识，加上自己的见解和认识，一步步的剖析题目，最后完成题目框图的设计。有了框图，我们对比老师上课给我们演示程序和我们自己设计的程序，慢慢的编写每个部分的程序，期间我们遇到了诸多困难，不过经过我们的不懈努力，最后都一一解决了。我们再此基础上还增加了拓展的功能，就是回波的产生于消除。最后经过好几天的编写，我们基本上完成了各个模块的编程。

小学期后期期间。主要是程序的调试与修改。经过前期的学习，对 CCS2.0 的应用与操作我们是比较熟练的，应用起来也是得心应手。在中期期间，我们小组也经常来到实验室调程序，过程不像我们想象中那么顺利，经常出现各种错误。有时即使没出现错误可是总是输不出要的结果。我们询问了老师，也自己在课下不断的寻找问题的所在，经过好几天的调试，我们有了实质性的进展，基本要求基本得到实现。我们不仅局限于基本功能，在剩下的时间里我们还尽量寻找进一步的拓展，希望能进一步拓展程序。最后也取得了一定的效果。

本次课程，我学到了很多的东西。也更加深刻的明白自主学习，自己动手等的重要性。实验过程中虽然遇到了许许多多的问题，不过经过我们小组成员的共同努力，困难也被我们一一克服了。课程的要求就是要求学生有自己的想法，自己动脑设计，自己动手操作，我感觉我们小组在这方面已经做的很不错了。不过，需要学习的知识还有很多，现在我们学到的还只是课程的一部分，需要我们自己在接下去的时间里自己深入地专研下去。

刁苏蒙总结与感想

这次小学期 DSP 应用课程设计，对我们来说是一个非常大的挑战，这次实验课程比较紧张，尤其是讲课阶段，由于 DSP 设计需要对 DSK 板子的了解，对编程软件 CCS 的掌握，尤其是 C 语言的灵活运用，这些方面都是必不可少的，但是只从三次授课中全部都能灵活掌握运用几乎是不可能的,所需要的是更多的查阅资料,同时要不断的进行软件与硬件结合的操作、实践，对各个部分的调试、改动，想通过资料直接找到适合的程序几乎是不可能的，查阅的资料中很大的一部分只是提供了大体的思路，需要我们不断的进行添枝加叶，同时也有一部分资料中可以找到可取的那一部分，将其整合、修改，并将自适应算法以及所需要的部分改动与程序串联起来，不论是老师提供的滤波框架，还是查阅的自适应滤波器资料，都要加以整合修改。

实验中每一步的进展都使我们付出心血努力的结果，实验过程中碰到了种种问题，实验 DSK 板的问题，程序中的问题，比如.out 文件有时不能正常生成，编译连接不能通过，也有程序结构上了问题，自适应算法的原理，如何应用到程序当中，又如何正确的调用库函数中的 DLMS 函数，以达到要求的结果，面对种种问题，必须要能够静下心来，一个一个的加以调试。通过 CCS 软件上提供的帮助信息，查阅帮助文件，可能是变量定义或者调用时出现矛盾，可能是不能找到相应的文件，有时需要加设断点，一步一步运行，寻找程序中出现差错的步骤，不厌其烦的一次次的调试，有时实验板也会出现这样那样的异常情况，但都被我们一一攻破。通过这种不断的查错、纠错的过程，虽然有些均是细小的错误，但是却很好的锻炼了我们的能力，对复杂嵌套程序的全面的把握、了解，对实验过程、步骤的详细认识。通过这次 DSP 课程设计，不仅使我对数字信号处理有了更深的了解，掌握了更多的动手实验的知识，同时又教会了我对于设计性实验应该如何着手，如何准备，注意事项，怎样来做能避免绕更多的圈子，少走弯路，出现问题，如何进行分析，再如何来解决，最后也要由衷的感谢高老师对我们耐心的讲解，遇到问题是提供帮助，解决问题的思路等等，实验的成功离不开您的指导。在今后的学习道路上我们会再接再厉，以优异的成绩作为回报。

六、参考文献

参考书目一览

【1】高海林，钱满义编：DSP技术及其应用，北京：北京交通大学电工电子教学基地，2009

【2】黄汉卿，宁永海著：基于 DSP 的自适应滤波 LMS 算法的研究和实现，河南，河南科技大学出版社，2007

【3】亓淑敏，梁文家，张美娟等编：基于DSP的自适应噪声消除系统，西安：长安大学信息工程学院 2008.6

【4】冬雷编：DSP原理及开放技术，北京：清华大学出版社 北京交通大学出版社，2007.7

【5】谭浩强编：C 程序设计（第二版，北京，清华大学出版社，2003

【6】Paulo S.R.Diniz 著，刘郁林等译：自适应滤波算法与实现（第二版），成都，电子工业出版社，2004

【7】TMS320C54x DSP CPU and Peripherals. Texas Instrument Inc，2001

【8】王树恩，宋彦，汪萌等：一种快速的回声消除算法及DSP实现，合肥：中国科学技术大学电子工程与信息科学系，2007.6