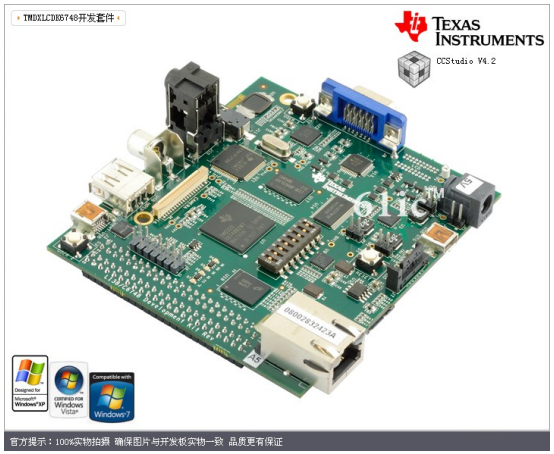
LCDK6748 DSP LAB
Part II
指导材料



杜伟韬 duweitao@cuc.edu.cn
广播电视数字化工程中心 ECDAV

杨刚 gangy@cuc.edu.cn
信息工程学院 电子工程系

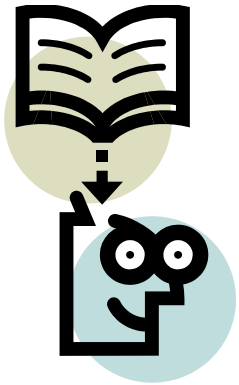
中国传媒大学 Communication University of China

本课程实验内容

- Hello World on DSP
- LED 系列实验
  - 点灯实验-直接配置寄存器
  - 点灯实验-使用GEL函数
  - 点灯实验-使用StarterWare库函数
  - 通过按键中断控制走马灯实验-使用StarterWare库函数
- 音频实时采集回放实验：使用中断、I2C控制器、McASP控制器、AIC31 Codec、DMA控制器和StarterWare库函数
- 使用FIR滤波器的实时音频均衡器实验，本实验基于音频采集回放实验
- 存储器布局和访存性能测试实验：使用片内RAM，片外DDR、malloc函数、定时器，观察MAP文件，本实验基于音频采集回放实验
- DSP/BIOS: Hello World 和系统时间打印实验
- DSP/BIOS: 线程任务（TSK）和信号量（SEM）实验
- DSP/BIOS: 按键中断和点灯实验：使用BIOS中断调度器
- DSP/BIOS: 音频实时采集、回放实验：使用BIOS中断调度器

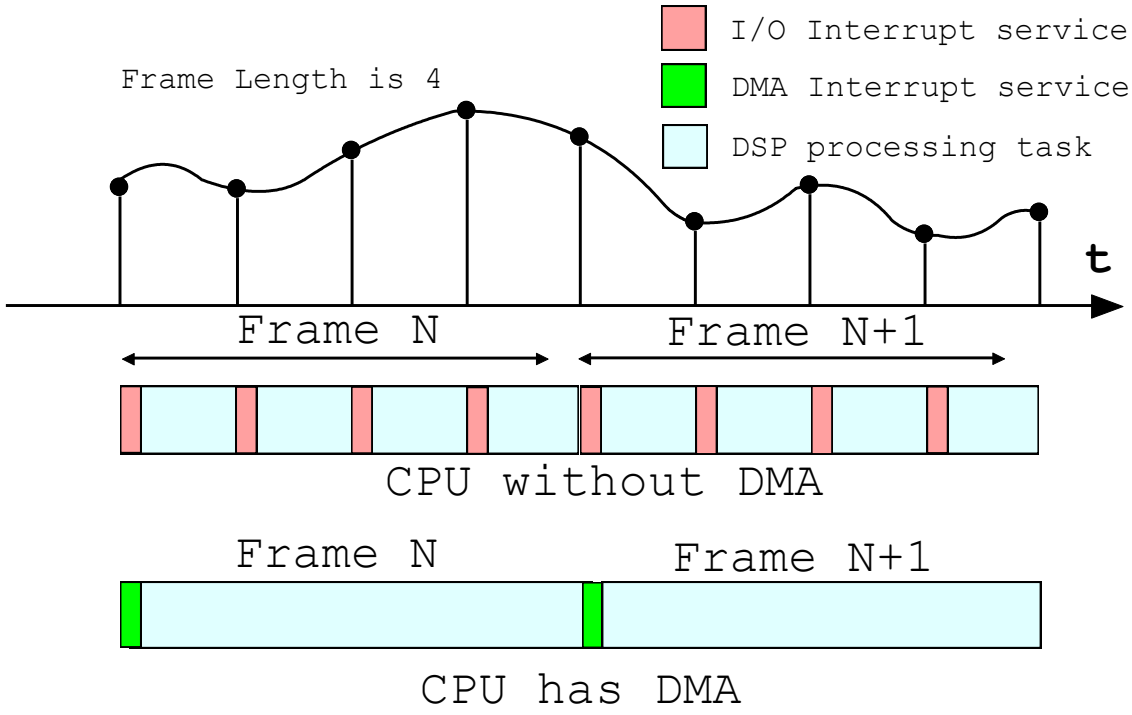
LAB1

Audio DMA playback



背景知识

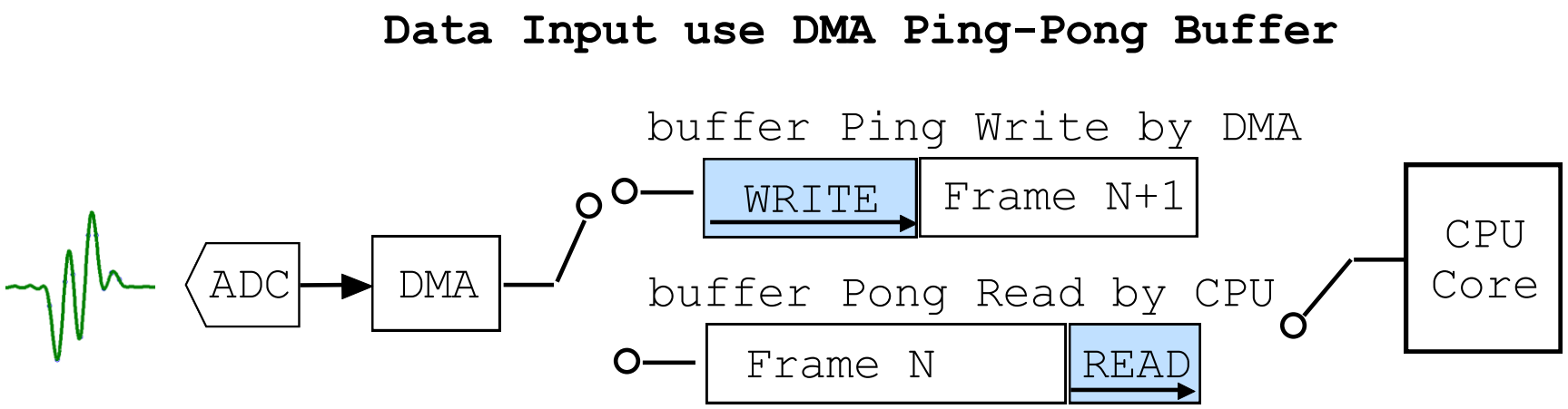
什么是DMA？



- DMA
  - Direct-Memory-Access
- 是一种用来搬数据的硬件模块
- 典型情况，外设数据接收
  - DMA响应外设（例如一个ADC）的中断信号，将外设数据搬移到某RAM区段。
  - 收集好一帧数据之后，DMA发出中断请求。
  - CPU响应DMA中断请求，处理当前的数据帧。

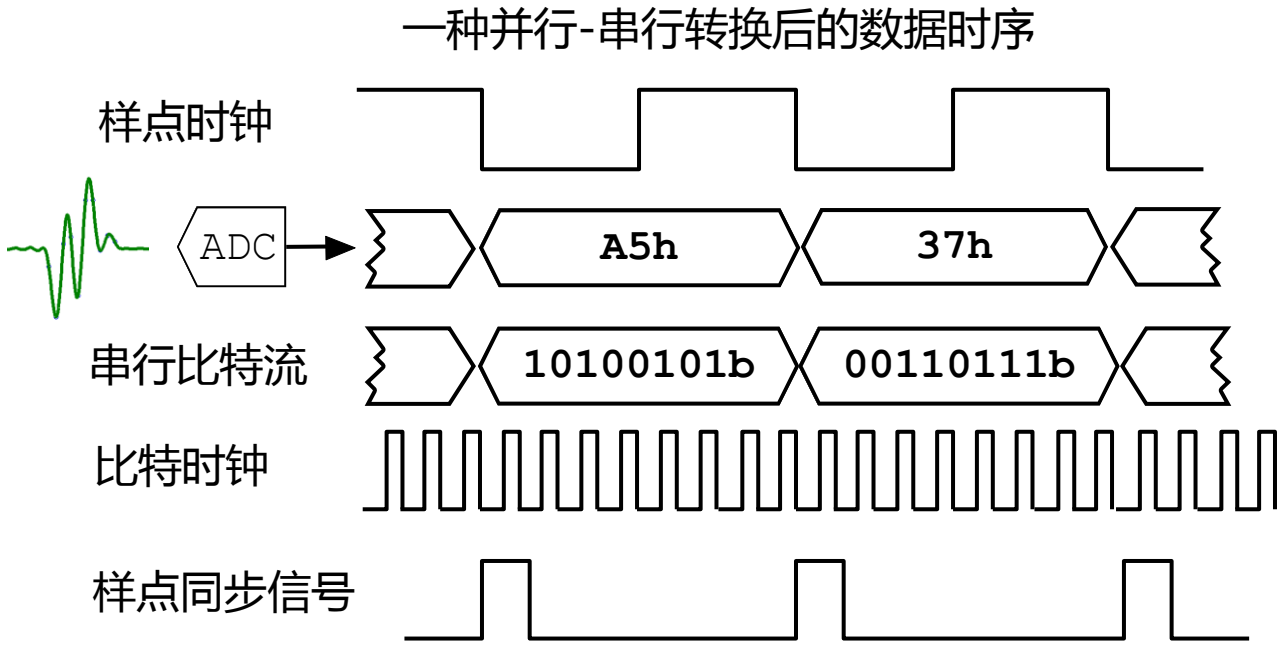
- 为什么要有DMA？
- 因为频繁的打断处理器响应I/O事件，会极大的降低处理效率
- 处理器跳转到中断服务程序的时候，需要压堆栈，返回时需要弹出堆栈
- 处理器执行跳转指令的时候，需要清空流水线

什么是乒乓（Ping-Pong)缓冲？



- 以数据的输入为例，2个缓冲区
  - 其中一个被DMA写入
  - 另外一个被CPU读出
  - DMA写满之后则切换到另外一个缓冲继续写入
  - 此时CPU应当已经把其中的数据都已经读出到处理缓冲区
  - 以上假设需要CPU的速度足够快作为前提
  - 否则就会出现未处理完的数据被新数据覆盖

什么是串并-并串转换？



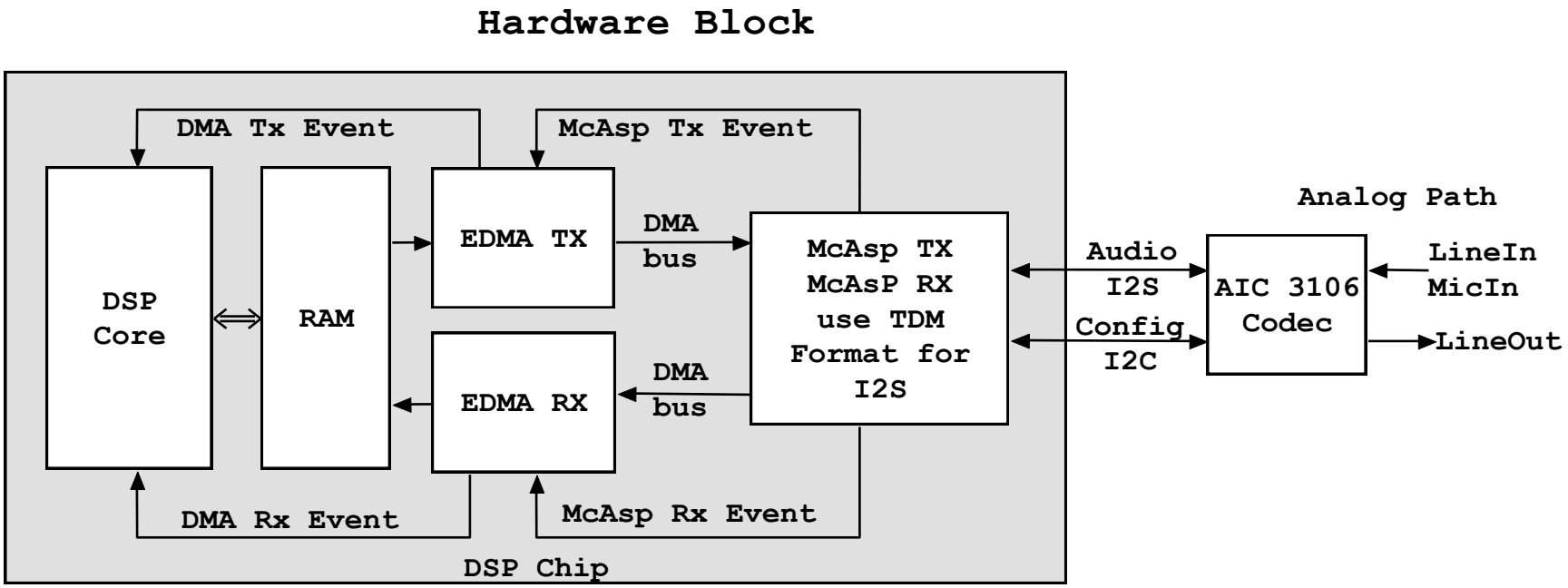
- 串行信号格式的好处
  - 节省管脚
  - 容易控制电路走线时的信号传输质量

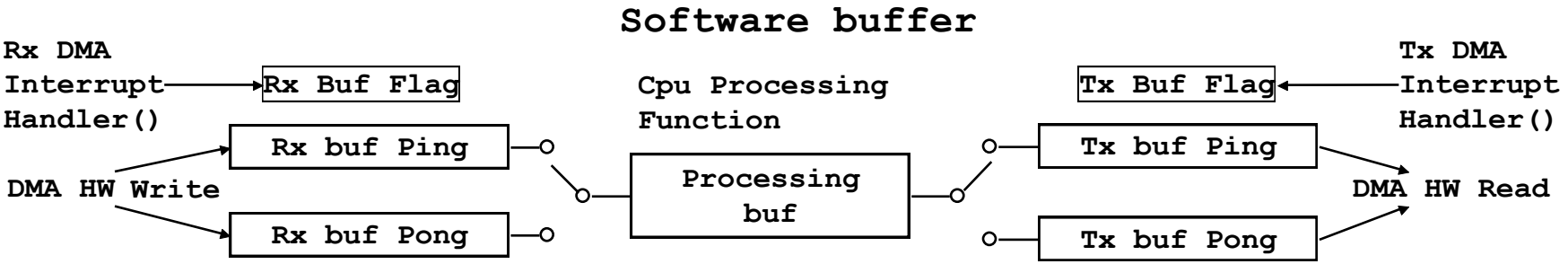
- 上图展示了一个 1通道的ADC输出串行比特流可能的信号格式。
- 对于多通道的ADC，其输出串行比特流可能需要其他格式或者更多的辅助信号
- 最常见的信号格式为：串行数据、串行时钟、帧同步

DSP算法的软件实现

- 通常以帧为单位
- 需要多个缓冲区以及管理策略
  - 输入输出的数据缓冲区
  - 算法的数据缓冲区
  - 通常使用乒乓（ping-pong）缓冲
- 需要接口硬件的DMA配合
  - 有操作系统的情况，使用多线程和信号量
  - 无操作系统的情况，使用处理器配置寄存器+中断函数

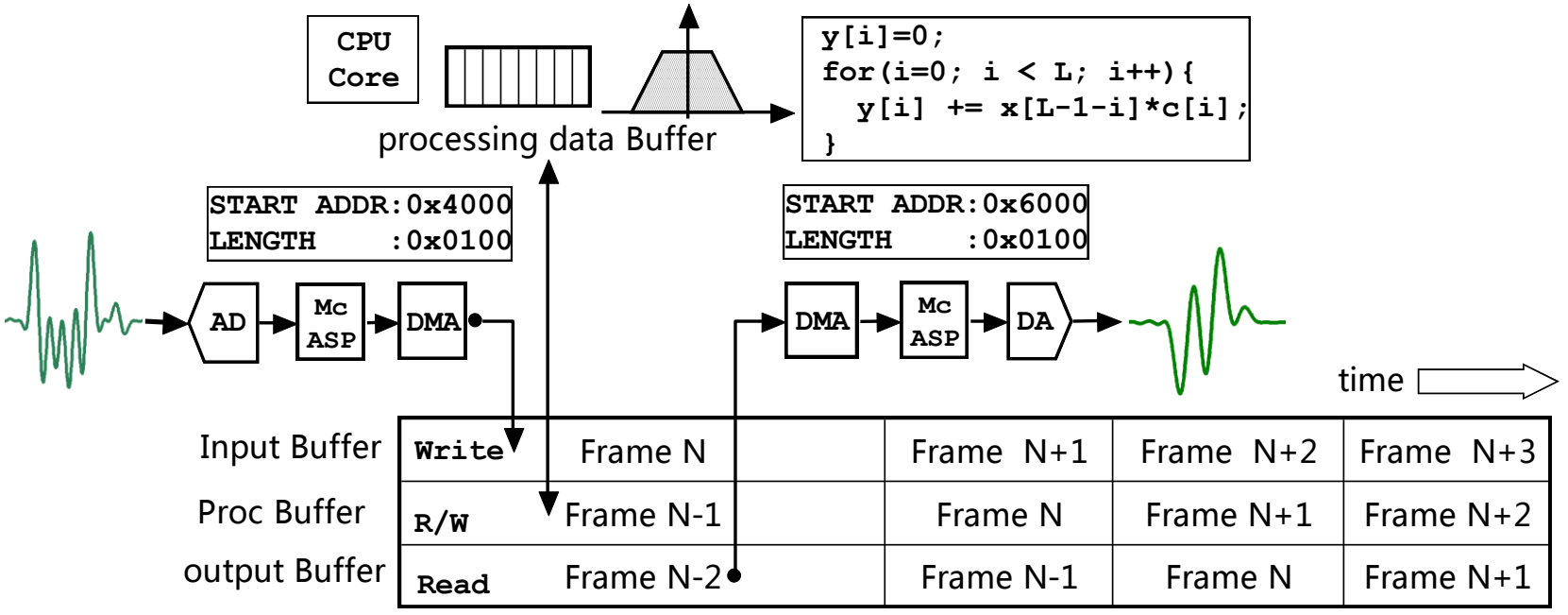
本音频处理实验的软硬件结构





DSP算法软件工作过程

- 第N帧正在被采集：
  - 由输入DMA控制器来响应ADC的输入中断，
  - DMA将输入缓冲区填满后发送BUFFER满中断
- 第N-1帧被处理
  - 处理器读取输入缓冲区（可能有多个）样点至计算缓冲区，进行滤波计算
  - 滤波结果被送至输出缓冲区（可能有多个）
- 第N-2帧被输出
  - 输出DMA响应DAC的输出中断
  - 处理器填充输出缓冲区

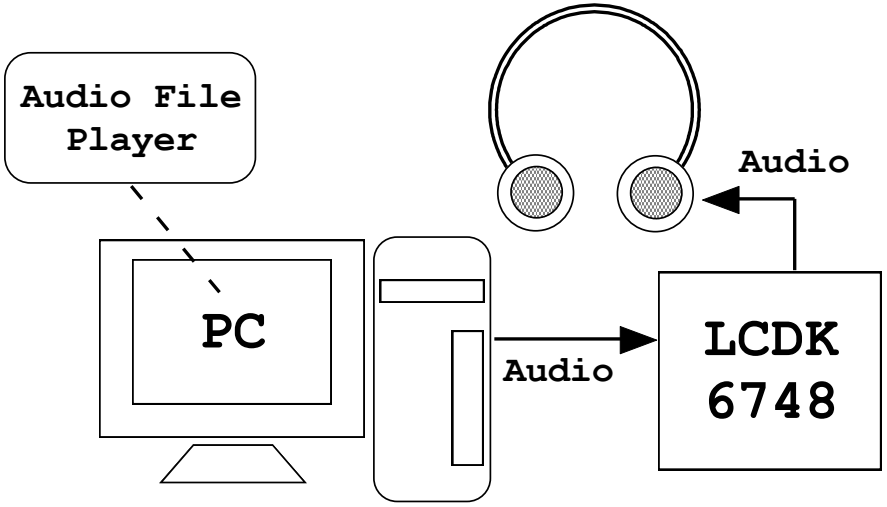


上机操作

初级测试方法

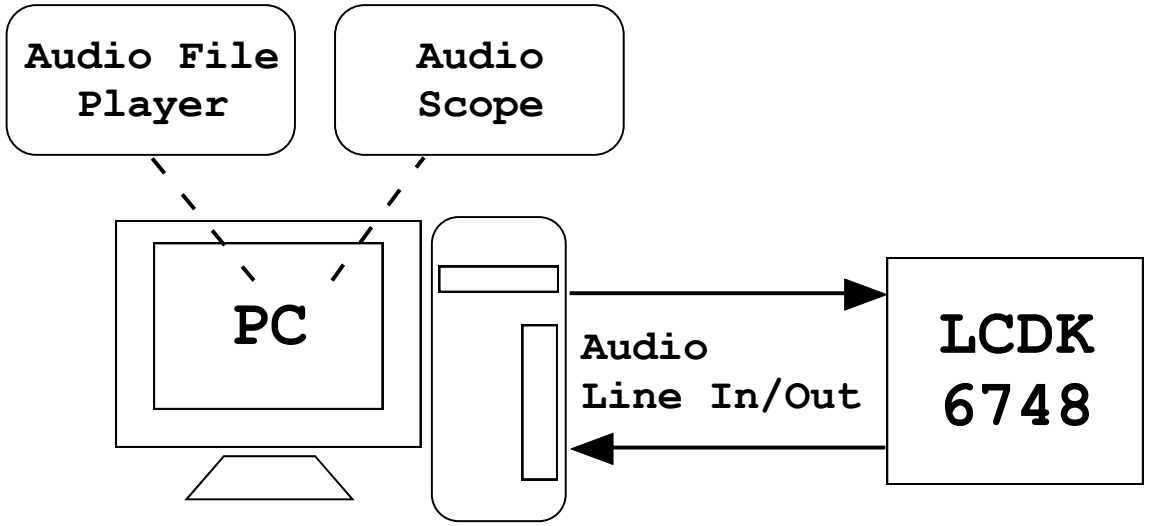
- 需要1根双公头音频线，一个耳机
- 连接电脑音频 送至 DSP板 再送至 耳机
- 注意调节电脑的输出音量，防止DSP板ADC饱和溢出
- 在电脑上播放音频文件（比如1KHz的立体声）作为激励
- **注意保护耳朵，当信号音量较大时，不要长时间佩戴耳机，以免造成听力损伤，大音量单音信号尤需注意。**

- 注：部分LCDK6748电路板的信号干扰抑制不好，在链接USB仿真器时会有干扰信号泄露到模拟音频线路中，用耳机可以听到明显噪声



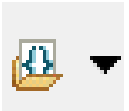
高级测试方法

- 需要2根双公头音频线
- 连接电脑和DSP板的Line In / Line Out
- 注意：LCDK6748的Line In在TOP，Line Out在BOT。
- 在电脑上播放音频文件（比如1KHz的立体声）作为激励
- 使用声卡示波器监测DSP的输出信号



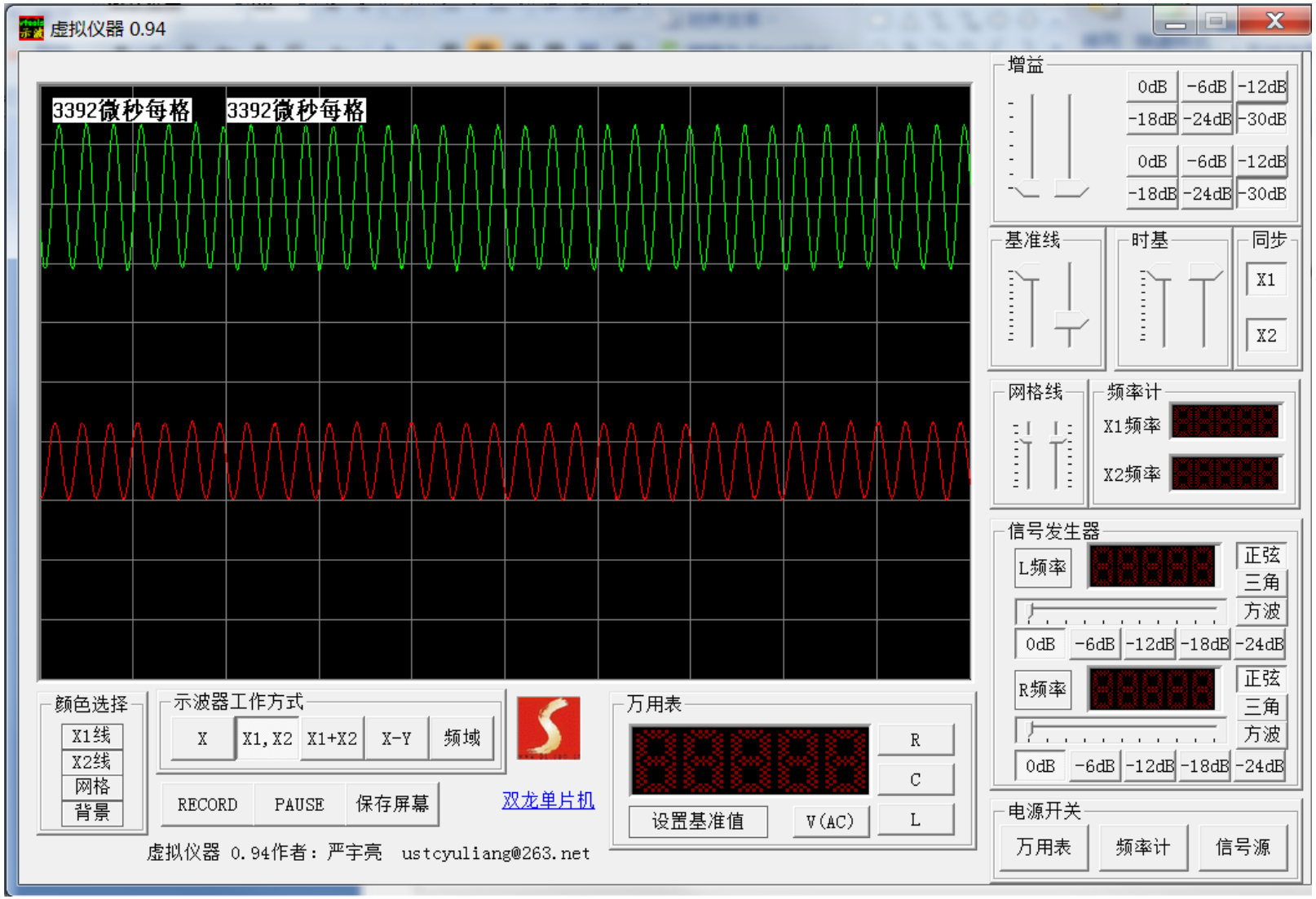
切换至LAB1，编译下载

LAB1_prj_auido_inout_StarterWare_HWI

- 如果下载代码后，运行DSP无反应。
- 可以尝试，system reset  ，以及reload 
- 用耳机听一下DSP的输出音频
- 再使用虚拟示波器观察DSP输出信号的时域和频域特征。

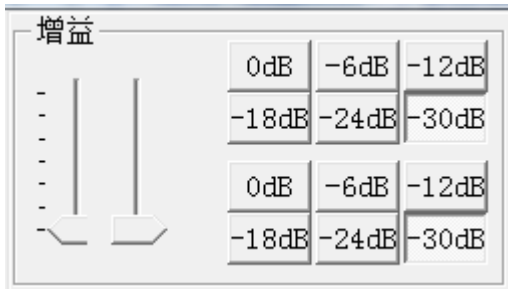
- **注意保护耳朵，当信号音量较大时，不要长时间佩戴耳机，以免造成听力损伤，大音量单音信号尤需注意。**

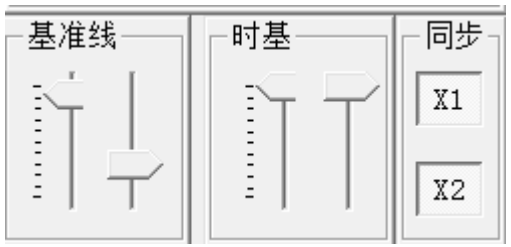
关于虚拟示波器使用（特别感谢免费软件作者）

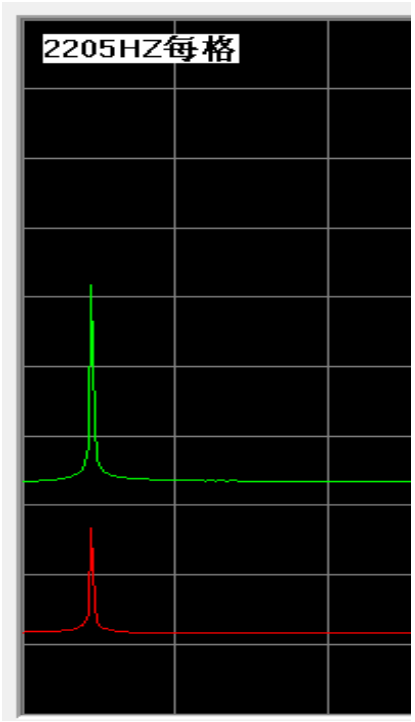


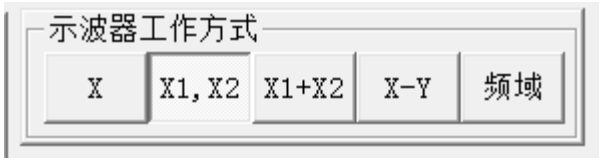
虚拟示波器的使用方法

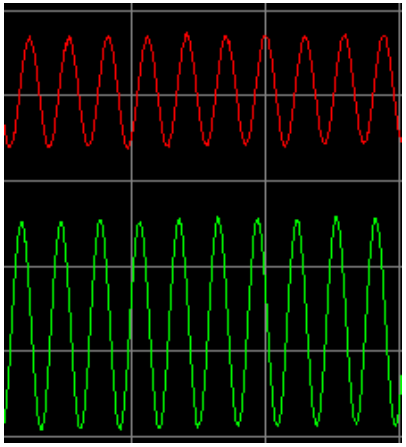
- 调整增益，改变输入衰减量以防溢出
- 基准线用来上下移动信号，时基用来改变time/div 同步用来设定触发通道
- 使用示波器工作方式选项，可以选择时域、频域、X-Y李萨如图形模式

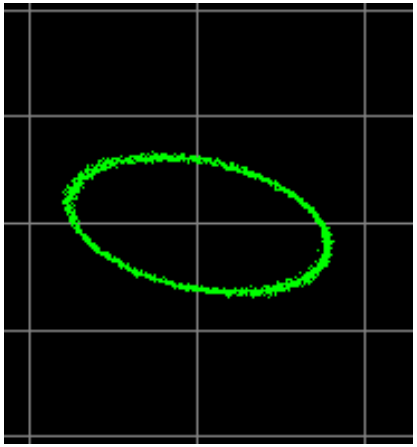










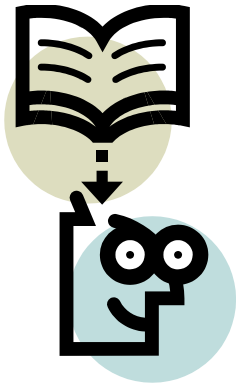


LAB1 作业

- 观察一下C代码结构，猜测一下各部分代码的用途。
- 学习虚拟仪器的用法，可以在电脑声卡上做一个音频自环（声卡的Line Out – Line In）
- 使用不同的激励信号文件，用虚拟示波器观察输出信号。
- 修改main函数中以下部分的代码，使得左右声道的输出音量不同。

```
for(i = 0 ; i < NUM_SAMPLES_PER_AUDIO_CH; i ++){
  // copy left and right channel input data to proc buf
  float temp_L, temp_R;
  proc_buf_L[i] = AuIn2Float(src_ptr[i*2]);
  proc_buf_R[i] = AuIn2Float(src_ptr[i*2+1]);
  // proc data: scale the sample
  temp_L = proc_buf_L[i] * 0.3;
  temp_R = proc_buf_R[i] * 0.6;
  // copy the left and right channel proc data to output buffer
  dst_ptr[i*2]   = Float2AuOut(temp_L);
  dst_ptr[i*2+1] = Float2AuOut(temp_R);
}
```

LAB2
Audio FIR



背景知识

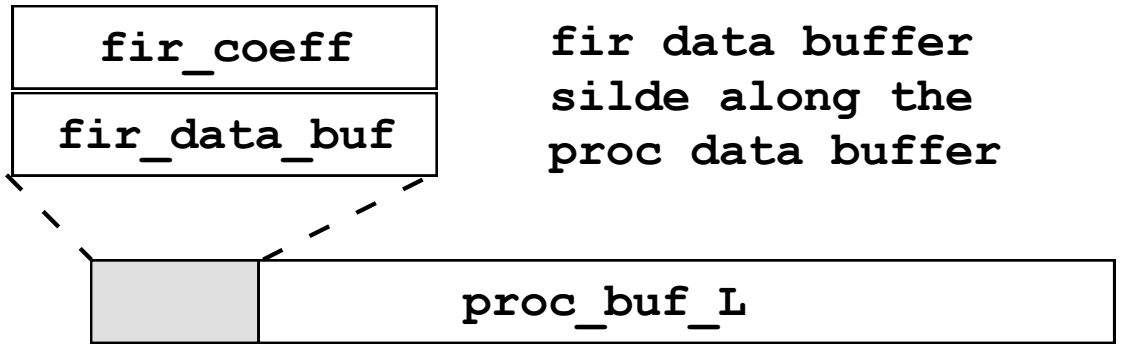
本实验和音频回放实验的区别

- 在DSP的INPUT DMA与OUTPUT DMA之间
- 添加一个了对左声道进行FIR滤波的算法程序
- 主函数的处理算法调用如下图

```
// copy input data to proc buf, perform proc, then output
for(i = 0 ; i < NUM_SAMPLES_PER_AUDIO_CH; i ++){
  // copy left and right channel input data to proc buf
  proc_buf_L[i] = AuIn2Float(src_ptr[i*2]);
  proc_buf_R[i] = AuIn2Float(src_ptr[i*2+1]);
}
fir_proc(proc_buf_L, proc_buf_L, NUM_SAMPLES_PER_AUDIO_CH);
for(i = 0 ; i < NUM_SAMPLES_PER_AUDIO_CH; i ++){
  float temp_L, temp_R;
  // copy the left and right channel proc data to output buffer
  temp_L = proc_buf_L[i] * 0.9;
  temp_R = proc_buf_R[i] * 0.9;
  dst_ptr[i*2]   = Float2AuOut(temp_L);
  dst_ptr[i*2+1] = Float2AuOut(temp_R);
}
```

滤波代码缓冲区结构

- FIR输入数据缓冲区
- FIR移位寄存缓冲区，FIR系数缓冲区
  - FIR的循序移位模型通过移动头数据和尾数据的指针实现循环移位的效果
  - 即：移动的是指针而不是真实数据
  - 滤波器的数据缓冲在输入数据缓冲上逐点向后滑动，直到所有输入数据均经过滤波。
  - 滤波程序是一个2层循环结构，外层循环为滤波样点，内层为滤波乘加
- FIR输出数据缓冲区
- 本实验的FIR滤波代码支持同址的输入与输出

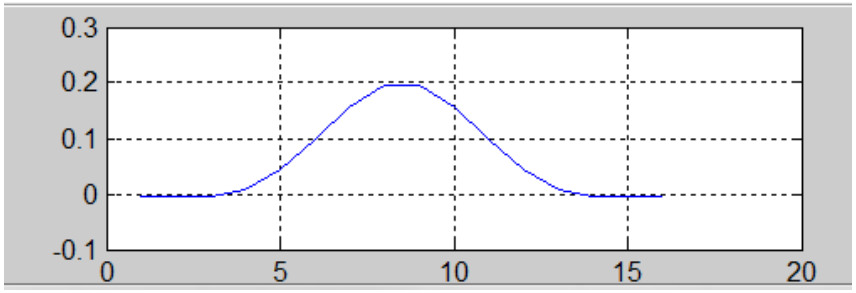


如何使用matlab生成FIR滤波系数

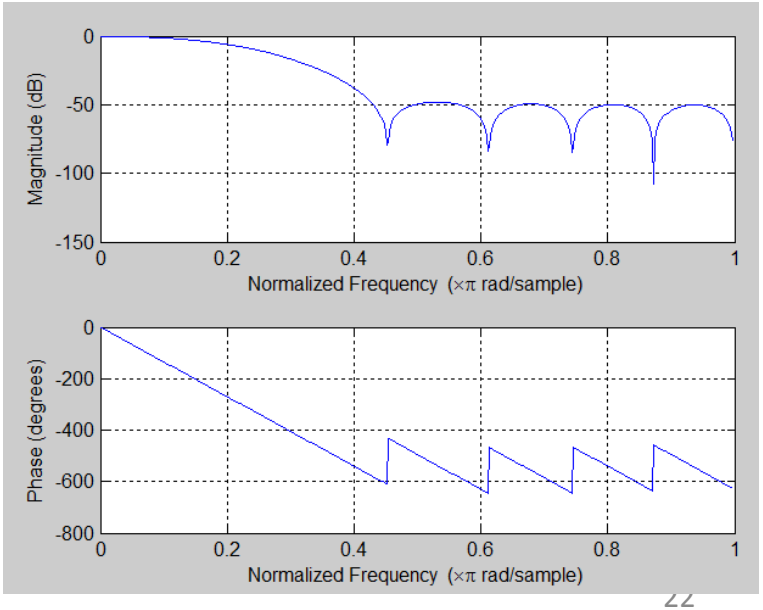
```
clear
clc
f = [0 0.2 0.2 1]; m = [1 1 0 0];
b = fir2(15,f,m);
freqz(b);
b.'
```

清内存
清屏
设定频响（低通）
得到15阶滤波器抽头系数b
绘制滤波器频响
得到列向量，便于拷贝

右图为一个16抽头的低通滤波器的幅频和相频响应。下图为该滤波器系数的时域波形



	1
1	-0.0033
2	-0.0046
3	-0.0039
4	0.0089
5	0.0432
6	0.0977
7	0.1560
8	0.1940
9	0.1940
10	0.1560
11	0.0977
12	0.0432
13	0.0089
14	-0.0039
15	-0.0046
16	-0.0033



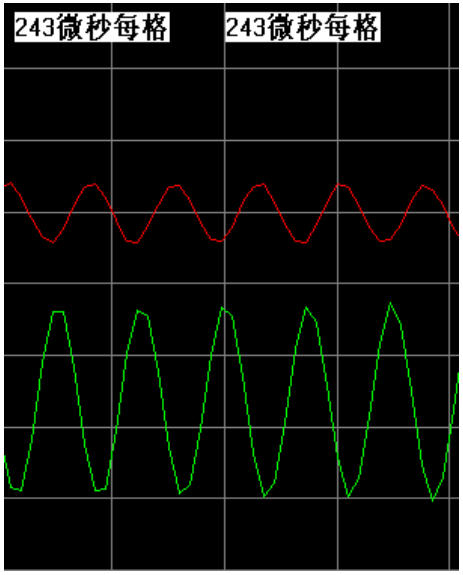
上机操作

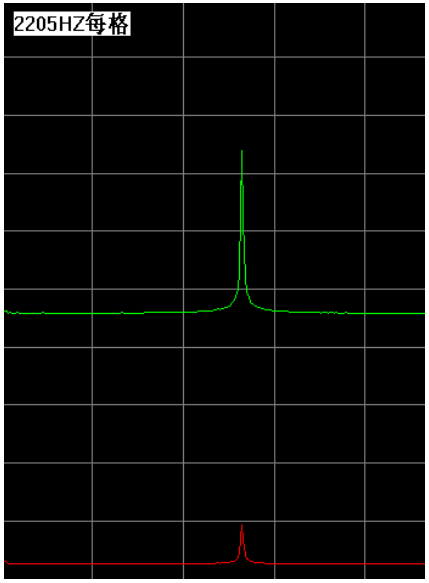
切换当前项目到LAB2

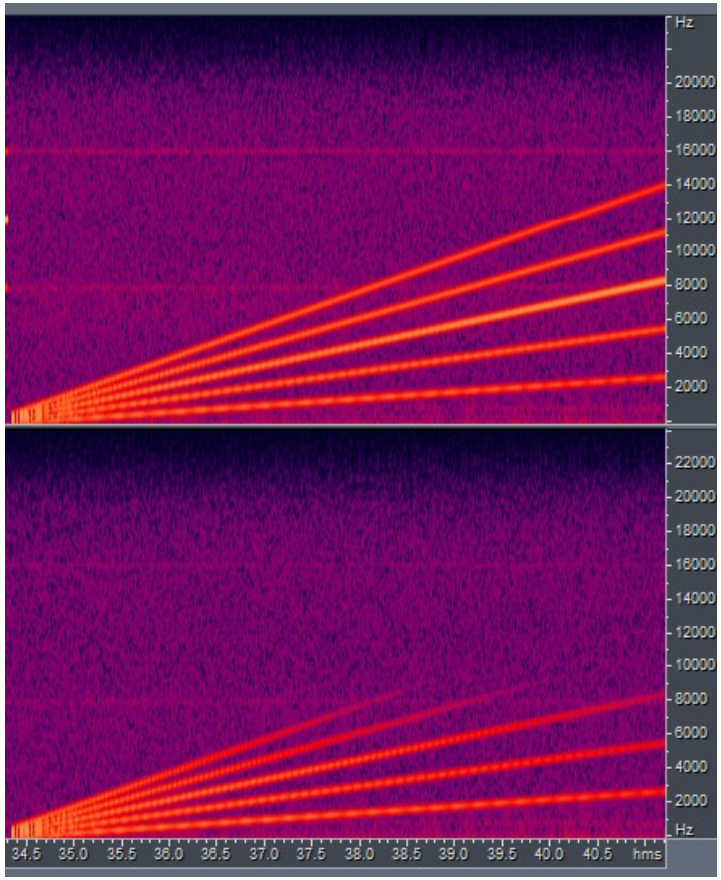
- 编译项目代码
- System reset
- Reload Program
- 使用提供的音频文件作为激励信号送给DSP开发板
- 使用虚拟仪器观察滤波后输出的音频信号
- 对于音乐文件可以听一下获取主观感受（注意音量，保护听力）

滤波器执行的观察方法

- 使用虚拟仪器观察时域或者频域
- 或者使用CoolEdit录音软件观察输入信号瀑布图







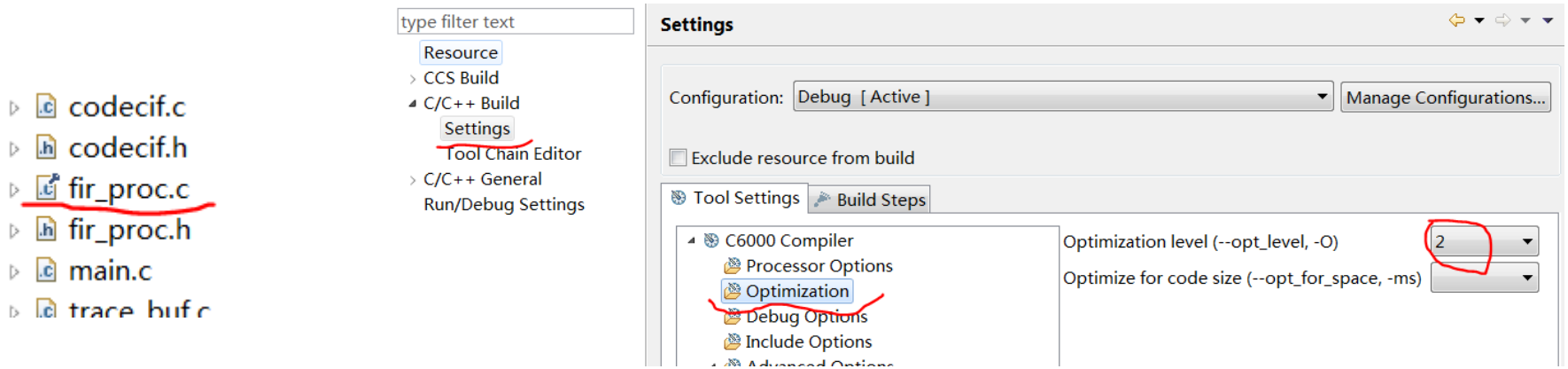
LAB2 作业 1

- 观察一下C代码结构，猜测一下各部分代码的用途。
- 编译下载代码，使用虚拟仪器，分析滤波后的效果
- 观察滤波器部分的代码fir_proc.c ，更换滤波器系数，如下图所示，把截止频率为0.1的滤波器系数更换为截止频率为0.2的系数。
- 使用虚拟仪器，分析滤波器更换之后的滤波效果。

```
46 void fir_proc(float *pIn, float *pOut, int dataLen){
47   float *pCoeff = 0 ;
48   int nCoeff = 0;
49   nCoeff = N_COEFF;
50   // pCoeff = fir_coeff_LP_0_2_N16;
51   pCoeff = fir_coeff_LP_0_1_N16;
52
```

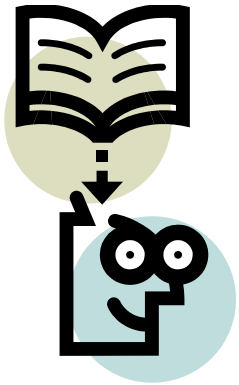
LAB2 作业 2

- 配置编译器优化选项
- 观察项目左边的文件列表，你会发现fir_proc.c文件的图标上多了一个小扳手的图样。这说明该文件有独立配置的编译选项。
- 在该文件上点击右键 选择 Properties，打开对话框。
- 找到编译优化选项，参考代码的优化级别是Level2
- 请把该优化级别设定为 off
- 重新编译、下载代码、观察滤波实验现象
- 再把该优化级别设为0、1、3，编译、下载



LAB3

存储器定位与访存测速



背景知识

存储的分段

- 从处理器（processor）的角度只有2种信息
  - 代码信息，即要执行的指令（操作码）的集合
  - 数据信息，即要被处理的数据（操作数）的集合
- 从编译器（compiler）的角度，则有多种信息
  - 这些信息以段（section）为单位存放，详情需查阅编译器手册），例如
  - .cinit　存放程序中的变量初值和常量
  - .const　存放程序中的字符常量、浮点常量和用const声明的常量
  - .switch　存放程序中switch语句的跳转地址表
  - .text　存放程序代码
  - .bss　为程序中的全局和静态变量保留存储空间
  - .far　为程序中用far声明的全局和静态变量保留空间
  - .stack　为程序系统堆栈保留存储空间，用于保存返回地址、函数间的参数传递、存储局部变量和保存中间结果
  - .sysmem 用于程序中的malloc 、calloc 、和realoc 函数动态分配存储空间
- 从链接器（linker）则以物理内存为单位，例如
  - DDR2，片内RAM，等等
  - 每个区段需要定义好物理地址的范围
  - 需要和处理器的结构，以及电路板上的存储器区保持一致

CMD文件

- TI DSP软件开发过程中，存储器使用的描述文件
- MEMORY部分描述了存储区的物理区段
- SECTION部分描述了编译器生成的区段放在哪个物理区段中

注：此处为方便排版把MEMORY和SECTION两部分双栏排列

```
MEMORY
{
    DSPL2ROM      o = 0x00700000  l = 0x00100000
    DSPL2RAM      o = 0x00800000  l = 0x00040000
    DSPL1PRAM     o = 0x00E00000  l = 0x00008000
    DSPL1DRAM     o = 0x00F00000  l = 0x00008000
    SHDSPL2ROM    o = 0x11700000  l = 0x00100000
    SHDSPL2RAM    o = 0x11800000  l = 0x00040000
    SHDSPL1PRAM   o = 0x11E00000  l = 0x00008000
    SHDSPL1DRAM   o = 0x11F00000  l = 0x00008000
    EMIFACS0      o = 0x40000000  l = 0x20000000
    EMIFACS2      o = 0x60000000  l = 0x02000000
    EMIFACS3      o = 0x62000000  l = 0x02000000
    EMIFACS4      o = 0x64000000  l = 0x02000000
    EMIFACS5      o = 0x66000000  l = 0x02000000
    SHRAM         o = 0x80000000  l = 0x00020000
    DDR2          o = 0xC0000000  l = 0x20000000
}
```

```
SECTIONS
{
    PROC_BUF        >  SHDSPL2RAM
    .text           >  SHDSPL2RAM
    .stack          >  SHDSPL2RAM
    .bss            >  SHDSPL2RAM
    .cio            >  SHDSPL2RAM
    .const          >  SHDSPL2RAM
    .data           >  SHDSPL2RAM
    .switch         >  SHDSPL2RAM
    .sysmem         >  DDR2
    .far            >  SHDSPL2RAM
    .args           >  SHDSPL2RAM
    .ppinfo         >  SHDSPL2RAM
    .ppdata         >  SHDSPL2RAM

    /* COFF sections */
    .pinit          >  SHDSPL2RAM
    .cinit          >  SHDSPL2RAM

    /* EABI sections */
    .binit          >  SHDSPL2RAM
    .init_array     >  SHDSPL2RAM
    .neardata       >  SHDSPL2RAM
    .fardata        >  SHDSPL2RAM
    .rodata         >  SHDSPL2RAM
    .c6xabi.exidx   >  SHDSPL2RAM
    .c6xabi.extab   >  SHDSPL2RAM
}
```

MAP 文件

- CCS编译之后的输出文件
- 包含以下几方面内容
  - 每个物理内存的使用情况
  - 每个section的定位情况
    - 起始地址和长度
    - 里面的各个目标模块的占用情况
  - 全局符号的物理地址
    - 所谓全局符号是指
    - 函数
    - 全局变量
  - 注意，有2个全局符号的地址表
    - 一个是按照符号名称排序
    - 一个是按照地址顺序排序

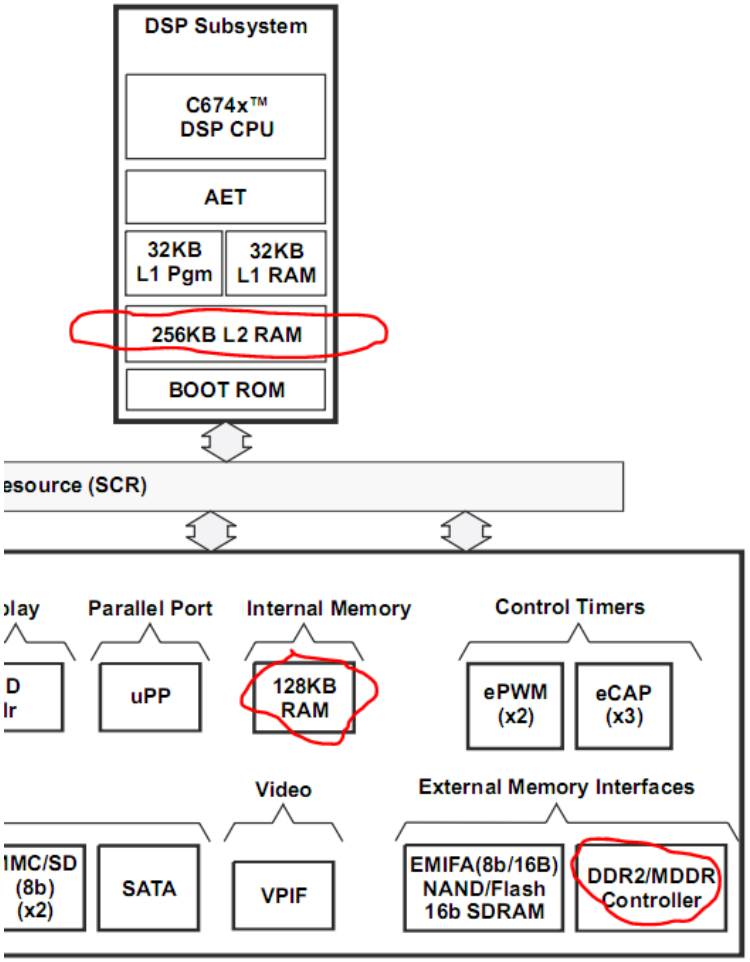
```
MEMORY CONFIGURATION

        name            origin    length      used     unused   attr    fill
----------------------  --------  ---------  --------  --------  ----  --------
  DSPL2ROM              00700000  00100000  00000000  00100000  RWIX
  DSPL2RAM              00800000  00040000  00000000  00040000  RWIX
  DSPL1PRAM             00e00000  00008000  00000000  00008000  RWIX
  DSPL1DRAM             00f00000  00008000  00000000  00008000  RWIX
  SHDSPL2ROM            11700000  00100000  00000000  00100000  RWIX
  SHDSPL2RAM            11800000  00040000  0001cc9c  00023364  RWIX
```

```
SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
.pinit     0    11800000    00000000     UNINITIALIZED

.data      0    11800000    00000000     UNINITIALIZED

.far       0    11800000    00011a28     UNINITIALIZED
                  11800000    00009500       main.obj (.far)
                  11809500    00008280       board.obj (.far)
                  11811780    00000240       interrupt.obj (.far)
                  118119c0    00000040       rts6740.lib : sin.obj (.far)
                  11811a00    0000000c       codecif.obj (.far)
                  11811a0c    0000000c       rts6740.lib : exit.obj (.far)
                  11811a18    00000008                   : _lock.obj (.far)
                  11811a20    00000008                   : memory.obj (.far)
```

```
GLOBAL SYMBOLS: SORTED ALPHABETICALLY BY Name

address    name
--------   ----
11811a78   $bss
11811a78   .bss
11800000   .data
11811c00   .text
1181b3e0   C$$EXIT
11819bc4   _AIC31ADCInit
11819c20   _AIC31DACInit
```

存储器的性能差异

- 同样的代码、数据放在不同的物理存储器中，性能差异很大。
  - 原因1，离处理器内核越近的存储器访问速度越快。
  - 原因2，不同的存储器的访存性能差异很大。例如DDR SDRAM，SDR SDRAM，SRAM等
- 如何测试访存性能
  - 使用定时器
  - 在访存代码之前记录定时器计数值
  - 在访存代码之后记录定时器计数值
  - 前后计数值相减，得到访存的周期数



C代码运行环境的初始化

- main函数是用户的C代码的入口
- 在main函数之前，编译器会插入一部分代码，用来初始化C代码的运行环境
- 初始化工作包括
  - 初始化各种全局变量的初始值
  - 初始化函数调用的栈指针初始值
  - 初始化malloc堆指针的初始值
- 使用外部存储器的注意事项
  - 如果使用了外部存储器，则可能该存储器中会包含部分的C环境的初始化数据。
  - 则需要保证在进入main函数之前，该存储器的控制器被正确的初始化
  - 以上工作通常由bootloader完成（在系统从外部的固化芯片如flash启动的情况下），或是由GEL脚本完成（在调试模式的情况下）

上机操作

切换当前项目至LAB3

- 打开该项目的*.cmd文件
  - 找到 PROC_BUF和 .sysmem这两个SECTION
  - 观察并指出它们分别属于哪个物理存储器
- 打开项目的*.map文件
  - 找到PROC_BUF和.sysmem这两个区段
  - 观察并指出它们的具体起始物理地址和长度
- 打开项目的main.c 文件
  - 找到用来定义proc_buf_L区段的伪指令

```
#pragma DATA_SECTION(proc_buf_L, "PROC_BUF");
```

  - 然后在map文件中找到proc_buf_L说出它的物理地址
  - 在main.c中找到给proc_buf_R申请内存的代码

```
proc_buf_R = (float *) malloc(NUM_SAMPLES_PER_AUDIO_CH * sizeof(float));
```

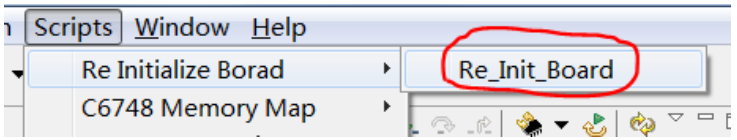
  - 其中proc_buf_L 和proc_buf_R分别是左声道、右声道的处理缓冲区

首先请思考

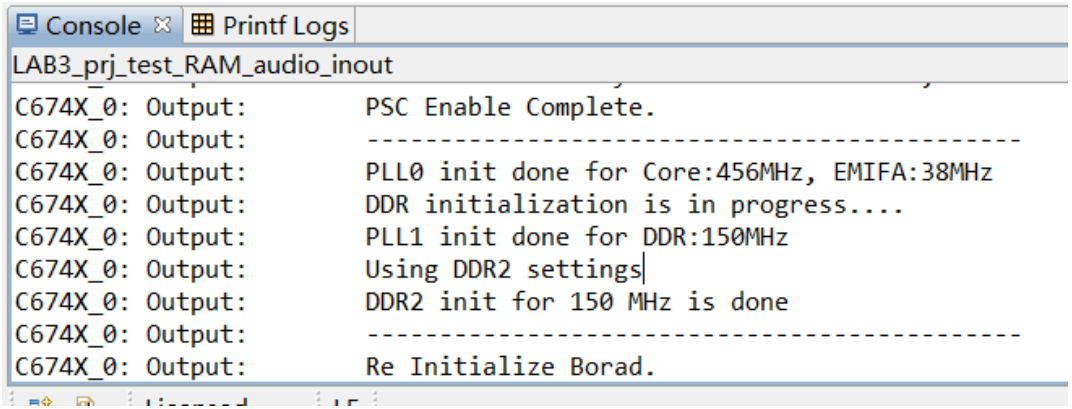
- proc_buf_L 和proc_buf_R是C语言里面的2个数组
  - 请问，哪一个的地址是在编译之后就确定了的？
  - 请问，哪一个的地址是要在运行之后才确定的，但是这个地址的范围你能确定么？
- 以上两个数组，分别位于哪个物理存储器中。
- 如果你想改变这两个数组的位置，使其位于其他的物理存储器中。
- 你应该怎么做？
- 注意：此处的代码已经展示了足够多的设定方法，以及检查方法。

切换当前项目进入debug模式

- 首先，system reset系统  ， 然后Load代码 
- 运行，然后暂停  ，观察程序是否跑飞



- 其次， system reset系统， 然后用GEL脚本重新初始化电路。
- 然后重新Load程序，运行。观察程序是否跑飞
- 请观察CCS底部的Console窗口，思考系统重新初始化过程中GEL做了什么。为什么在运行程序之前需要重新初始化电路。

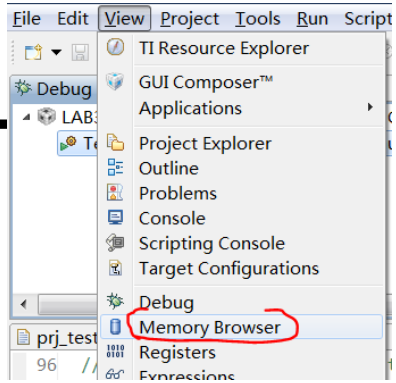


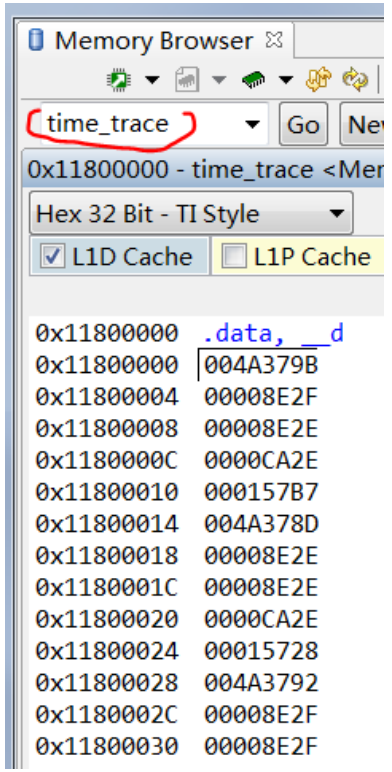
LAB3 作业1

- 修改cmd文件
- 把.sysmem段定位到和.text段同样的物理存储器
- 重复上面的实验操作，回答以下问题。
- 把sysmem放到片内或片外 和 初始化外存储器有什么关系呢？

继续进行实验

- 首先修改cmd文件
- 把.sysmem段定位到DDR2
- 编译程序，system reset，Re Init board，然后下载程序运行
- 运行程序，开启内存观察窗口
- 在地址栏中输入 time_trace
- 这是一个数组，保存了不同监测点的定时器计数值信息。
- 音频处理任务是一个无限循环，每次循环会记录5个监测点的定时器计数值。
- 请阅读main函数中while(1)循环中的代码，了解5个监测点的数据的意义。





注意：右边2个图标用来刷新显示内容

LAB3 作业2

- 首先检查工程代码，确保数组`proc_buf_L[ ]`位于片内RAM，`proc_buf_R[ ]`位于DDR RAM 中。
- 修改main.c
- 找到以下2行代码，取消其注释

```
// proc_buf_L[i] = proc_buf_L[i] * 0.95;
// proc_buf_R[i] = proc_buf_R[i] * 0.95;
```

- 编译代码，重新运行代码
- 观察memory browser中的每帧5个时间监测节点，和之前的结果进行对比
- 你有什么结论？
- 根据实验现象，请回答

1、在访问外部存储器的过程中，如果在访存的同时还伴有数据的运算操作，会对访存的速度造成怎样的影响？

2、为了提高访存的速度，应该如何使用外部存储器

That's ALL

Happy DSP Life