

摘 要

正弦信号是极具代表性的周期信号,它在通信系统、电力系统、桥梁振动检测、生物医学检测、雷达和声纳技术等众多领域中都有十分重要的应用。在信号处理领域中,对正弦信号参数估计问题具有重要的研究意义。

本文基于 DFT 频谱相位信息的正弦波频率高精度估计方法,该方法通过截取正弦波的两段长度不同的离散序列,利用它们的频谱信息来完成参数的估计,并运用谱峰的相位差来完成参数的估计。此方法运算速度快,频率估计准确,适合于实时信号处理。

关键词: 正弦信号, 幅值估计, 频率估计, 频谱遍历

目 录

第一章 针对复杂工程问题的方案设计与实现	1
1.1 针对复杂工程问题的方案设计	1
1.2 针对复杂工程问题的推理分析	9
1.3 针对复杂工程问题的方案实现	13
第二章 系统测试	19
2.1 系统分段测试	19
2.2 测试案例设计及系统整体测试结果	27
第三章 知识技能学习情况	29
第四章 分工协作与交流情况	31
参考文献	33
致谢	35

第一章 针对复杂工程问题的方案设计与实现

1.1 针对复杂工程问题的方案设计

1.1.1 实验开发环境简述

实验开发环境 CCS5.5:

Code Composer Studio 是一种集成开发环境, 支持 TI 的微控制器和嵌入式处理器产品系列, 包含一整套用于开发和调试嵌入式应用的工具。包含了用于优化的 C/C++ 编译器、源码编辑器、项目构建环境、调试器、描述器以及多种其他功能。

实验开发板 EVMDM6437:

DM6437 是 TI 公司 2006 年推出的、专门为高性能、低成本视频应用开发的、主频 600MHz 的、32 位定点 DSP 达芬奇 DaVinci 技术的处理器系列。采用 TI 第 3 代超长指令集结构(VelociTI.3)的 TMS320C64x+ DSP 内核, 支持 8 个 8 位或 4 个 16 位并行 MAC 运算。

如图 1 所示, DM6437 提供几个外设接口, 运行用户连接外部设备, 实验主要使用到的 AIC33 接口:

DM6437 使用 TI TLV320AIC33 立体声编码器(codec)来做音频信号的输入输出。音频模拟信号通过 microphone 或 line 输入, 经该编码器采样转换成数字信号送 DSP 处理。

DSP 处理完音频数字信号后也是用该编码器再转换为模拟信号输出给用户。

此 codec 使用两个串行通道通信, 一个用于 codec 内部寄存器的配置, 另一个发送和接收数字音频信号。

IIC 总线用于单向控制通道, 该控制通道仅在配置 codec 时使用, 因此在音频数据传输过程中, 该通道处于闲置状态。缺省配置是使用 McBSP 作为双向数据通道, 但也可以选择 McASP 来驱动数据通道。所有音频数据流通过数据通道传输。根据采样数据宽度、时钟信号源和串行数据格式三个变量, 可支持多种数据格式。

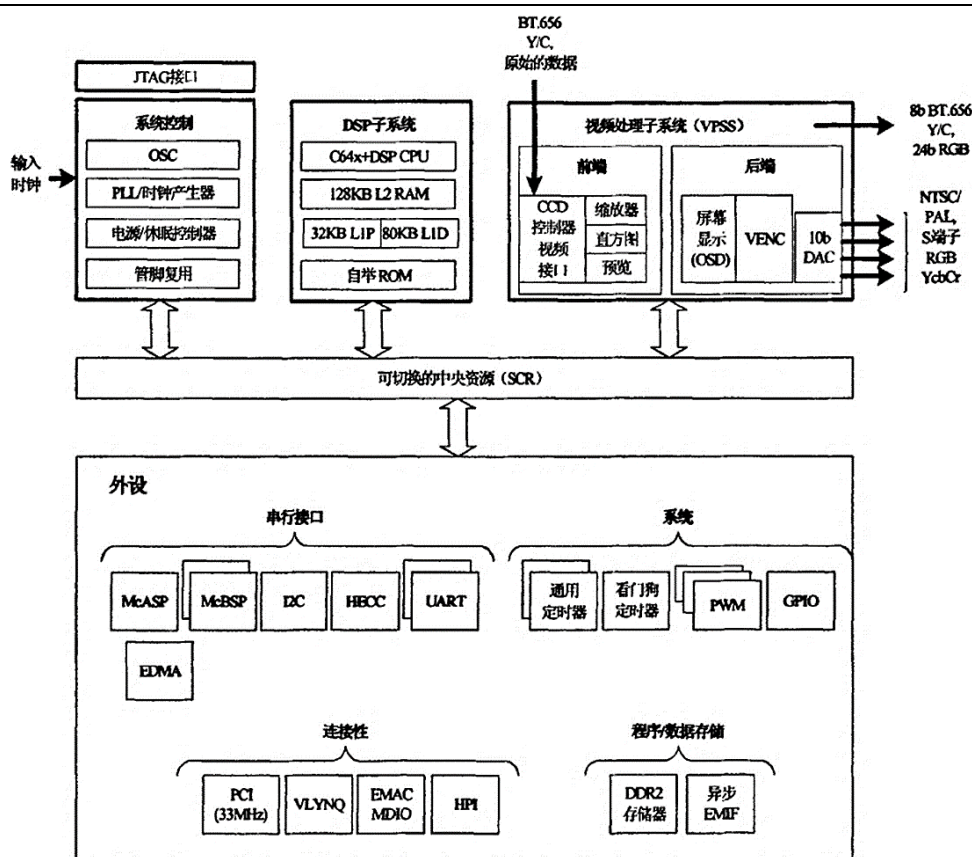


图 1 DM6437 开发板硬件资源框图

在 DM6437EVM 开发板进行对正弦信号进行捕获并进行参数估计，主要分为两个步骤。第一是要将音频输入口的输入信号进行采集，第二是要将采集得到的采样数据进行处理和分析，进一步得到最终的测量数据。

对于 DSP 的音频信号的采集，大多采用 McBSP 多通道同步缓冲串口来实现，音频信号经过分帧输入到波形缓存中，转换为一串数字信号；对于采样信号的检测和估算，可以通过相位差分法和最小二乘拟合法进行计算。

基于 CCS 平台，将可能使用到的硬件资源信息进行分析，总结出了以下可用信息，便于后续环境搭建提供支持；同时，对于相位差分法与最小二乘拟合法进行数学分析，总结出了各自的特点以及适用情况。

1.1.2 McBSP 多通道缓冲串行口：

McBSP 是在标准串行接口的基础之上对功能进行扩展，因此，具有与标准串行接口相同的基本功能。它可以和其他 DSP 器件、编码器等其他串口器件通信。McBSP 的结构图如图 2 所示：

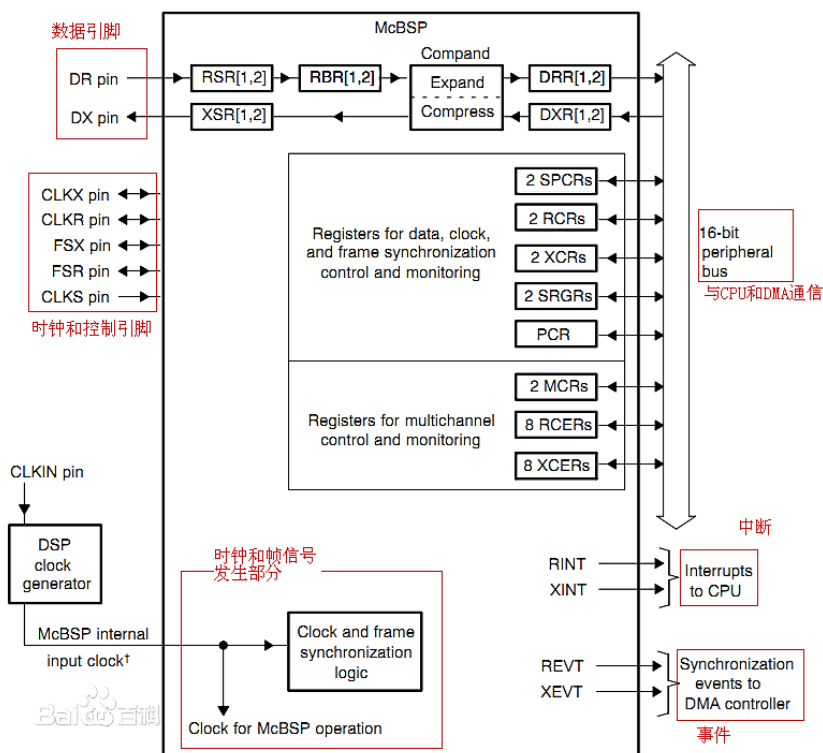


图 2 多通道缓冲串口内部结构图

McBSP 包括一个数据通道和一个控制通道，通过 7 个引脚与外部设备连接。数据发送引脚 DX 负责数据的发送，数据接收引脚 DR 负责数据的接收，发送时钟引脚 CLKX，接收时钟引脚 CLKR，发送帧同步引脚 FSX 和接收帧同步引脚 FSR 提供串行时钟和控制信号。

McBSP 具有以下特点：1. 全双工通信；2. 拥有两级缓冲发送和三级缓冲接收数据寄存器，允许连续数据流传输；3. 串行字长度可选，包括 8、12、16、20、24 和 32 位；

McBSP 可以通过两种方式进行数据传输：

- 1) EDMA 方式：McBSP 发送事件通知 EDMA 通道进行传输，有接收事件 REVT 和输出事件 XEVT，它们都与固定 EDMA 通道进行了绑定；
- 2) CPU 方式：McBSP 通过中断方式通知 CPU 进行数据的传输，有接收中断 RINT 和输出中断 XINT，中断的设定由 SPCR.RINTM 和 SPCR.XINTM 控制，CPU 也可以通过轮询方式来控制数据的传输。

课程设计中需要使用到两个数据引脚，使用的串口资源 DRR 和 DXR 分别完成音频信号的输入和输出，每采集到一次音频信号，就会通过 XINT 完成中断信号的发出，此时数据通过 16bit 总线被 CPU 调用。

1.1.3 DSP/BIOS 实时多任务操作系统内核:

DSP/BIOS 是一个简易的实时嵌入式操作系统，主要面向实时调度与同步、主机/目标系统通信，以及实时监测等应用，具有实时操作系统的诸多功能，如任务的调度管理、任务间的同步和通信、内存管理、实时时钟管理、中断服务管理、外设驱动程序管理等。

项目中 DSP/BIOS 使用的主要模块有以下三部分：**HWI** 硬件中断管理、**LOG** 时间记录显示和 **SWI** 软件中断管理中。处理任务一般要划分为两个部分：一个是控制部分，花时间少，放在 **HWI** 函数中；另一部分是处理部分，放在 **SWI** 函数或任务中处理。

DSP/BIOS 具体模块的使用方式如下：

- 1) **Log 模块**：标准输入输出模块，占用内存少执行速度快。输出到 log 监视窗口，优先级很低。

参数：bufseg：日志缓冲区的存储段名称；buflen：日志缓冲区的大小（以字为单位）；logtype：说明日志类型，循环或者固定；fixed（固定）：只存储其最先接收的信息，当消息缓冲区满时就会拒绝接收新的信息；circular（循环）：当消息缓冲区满时，新日志会自动覆盖原有的日志；datatype：如使用 LOG_printf 函数来打印输出日志信息时，请选择“printf”类型。如使用 LOG_event 函数记录日志信息，请选择“raw data”。

API： LOG_enable(&trace);

LOG_printf(&trace, "Hello DSP/BIOS %d.", 0);

配置细节如图 3 所示：

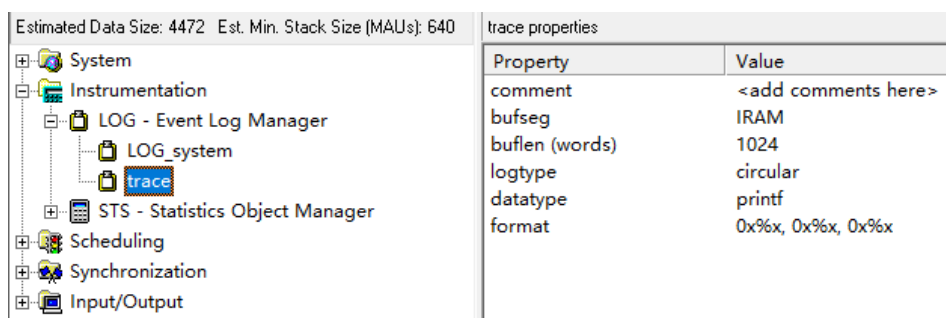


图 3 LOG 模块的配置

- 2) **SWI 模块**：一旦软中断邮箱的值满足一定条件时，触发软件中断，触发后一次执行完成（除非被硬件中断打断），执行完后软中断邮箱恢复到复位值，软中断中绝不会有死循环。

参数：function 相关函数名，priority 优先级，mailbox 软中断邮箱

API : void SWI_post(&estimate_SWI) // 软中段调用

void SWI_andn(swi, mask); // 邮箱值与 mask 进行与非操作, 当值为 0 时触发软中断

void SWI_andn(swi, mask); // 邮箱值与 mask 进行与非操作, 当值为 0 时触发软中断

void SWI_inc(swi); // Increment SWI's mailbox value and post the SWI

void SWI_dec(swi); // Decrement SWI's mailbox value and post if mailbox becomes 0

时限 100us 以上, SWI 允许 HWI 将一些非关键处理在低优先级上延迟执行, 这样可以减少在中断服务程序中的驻留时间每个 SWI 对象有一个邮箱, 可以决定是否触发该中断。

用于触发 SWI 的 API 函数可以对邮箱值作不同的操作, 软中断触发后一次执行完成 (除非被硬件中断打断), 执行完后软中断邮箱恢复到复位值。

配置细节如图 4 所示:

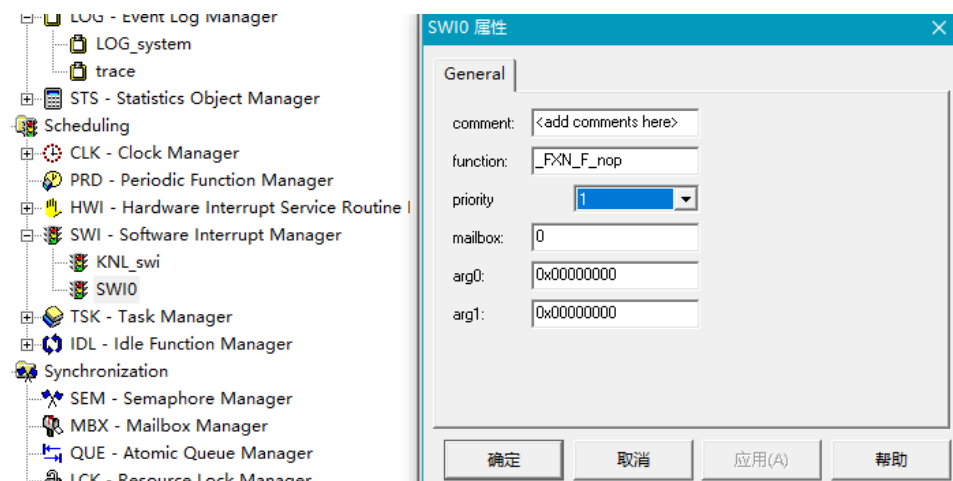


图 4 SWI 线程的配置

3) HWI 模块: 硬中断中包含 INT4 到 INT15 可自定义用户中断, 其中 INT4 优先级最高。

参数 : interrupt selection number 中断事件号, function 中断相应函数名, monitor 监视, Use Dispatcher 调度在调用 HWI 函数的前后自动调用 HWI_enter 和 HWI_exit, Interrupt Mask 中断掩字

频率可达 200KHz (5us), 处理时限在 2us~100us, 包括 CLK 函数。配置细节如图 5 所示:

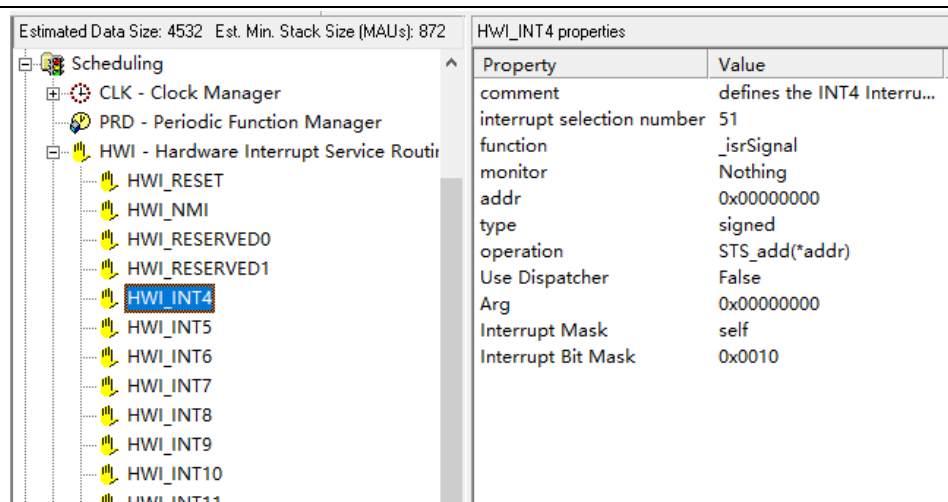


图 5 HWI 线程的配置

1.1.4 使用 DSP/BIOS 实现对音频信号的采样：

首先对立体声编码器 codec 进行初始化，调用 initCode 函数，加载默认配置，采样率设置为 44.1kHz，采样位数 32 位。

在 12 个可屏蔽硬件中断中选择最高级别的 HWI_INT4 设定中断号为 51，使得保证获取采样数据的优先级。设定函数名为 isrSignal，于是在对应的硬件中断函数中可使用 McBSP 的相关函数获取到采集的数据。

每一次音频串口采集到数据就会通过 DSP/BIOS 硬件中断调用 isrSignal 函数，此时通过 MCBSP1_DRR_32BIT 可以接受来自 McBSP 的 DRR 寄存器中两个声道的音频数据，然后通过使用以下语句：

```
buf[0] = (short)dataIn;           // add new L data to delay line
buf[1] = (short)(dataIn >> 16); // add new R data to delay line
```

可以完成对左右两个声道数据的分离，其中一个声道的数值即为正弦信号的采样值。

1.1.5 相位差分法特点：

相位差分法：将一个采样信号序列分为两段，分别做 DFT 变换，然后找出两段 DFT 变换序列的谱线峰值位置的相差，用它来估计信号的频率与相位。其运行流程如下：

- 1) 对 $S1[n]$ 和 $S2[n]$ 等长子序列进行傅立叶变换，频率分辨率为 Δf ;
- 2) 分别检测到 $S1[k]$ 和 $S2[k]$ 的最大谱线，计算得到频率粗测值 $f_k = k_0 * \Delta f$ 和

幅度估计值 \hat{a} ;

- 3) 由检测到的两条最大谱线计算出相位差 $\Delta\varphi=\varphi_2-\varphi_1$ 、频率偏差 $f\delta=\Delta f*\Delta\varphi/2\pi$;
- 4) 计算出频率估计值 $f_0=f_k+f\delta$ 和初相位 $\varphi_0=\varphi_1-(N-2)*\Delta\varphi/2N$;

相位差分法具有以下特点, 以及在进行编程实现时需要注意的一些问题:

- 1) 可以采用 FFT 快速算法, 运算速度较快, 适合实时信号的处理, 但是 DFT 频率分辨率和频率估计精度取决于信号的测量时间长度, 信号测量时间成为了限制因素之一。
- 2) DFT 对正弦信号有显著的信噪比增益, 但是当信号频率不是 DFT 频率分辨率的整数倍时, 由于频谱泄露使得信号幅度的估计值产生较大偏差, 而且该偏差不会随 DFT 长度增加而减小。
- 3) 由于栅栏效应当频率为 DFT 分辨率一半的时候, 频谱内就会有两条幅度相当的谱线, 此时在噪声存在的情况下, 方法就会失效。

总结: 具有一定信噪比增益, 运算量小、实时性好, 幅度估计精准度不足, 对采样有一些要求;

1.1.6 最小二乘法拟合法特点:

最小二乘拟合法: 通过最小化误差的平方和寻找数据的最佳函数匹配。利用最小二乘法可以简便地求得未知的数据, 并使得这些求得的数据与实际数据之间误差的平方和为最小。

其运行流程如下:

四参数正弦曲线拟合 `curvefit_4para()`:

- 1) 设定停止条件方差范围 he , 用圆周计点法获取每周期点数 m 和周期数 p ;
- 2) 获得频率估计值 $f_0=fs/m$ 极其收敛区间界 $\Delta f=fs/n$, 确定迭代边界 fl 、 fr , 中值频率 fm 、 ft ;
- 3) 在 fl 、 fr 、 fm 和 ft 上执行三参数正弦曲线拟合 `curvefit()`, 获得四组拟合参数;
- 4) 进行残差的比较, 确定下次迭代参数, 判定拟合残差大小;
- 5) 当残差平方和满足停止条件即满足测量要求。

三参数正弦曲线拟合 curvefit():

理想正弦信号: $y(t) = E1 * \cos(2\pi ft) + E2 * \sin(2\pi ft) + Q$

有效方差: $\varepsilon = \sum_{i=1}^n [y_i - A_1 \cos(2\pi f t_i) - B_1 \sin(2\pi f t_i) - C]^2$

- 1) 根据序列时刻 t_i , 计算 $\alpha_i = \cos(2\pi f t_i)$ 和 $\beta_i = \sin(2\pi f t_i)$, 和其平均值 α , β , y ;
- 2) 代入方程计算 AN、AD、BN 和 BD 四个参数;
- 3) 参数 $A1 = AN/AD$ 、 $B1 = BN/BD$ 、 $C = y - A1 * \alpha - B1 * \beta$ 则为理想正弦信号的拟合参数, 使得有效方差最小。

最小二乘拟合法具有以下特点, 以及在进行编程实现时需要注意的一些问题:

- 1) 计算量较大, 幅度值和频率值估计较相位值精确;
- 2) 可手动设置迭代条件调节精度, 但会带来更多的计算耗时;
- 3) 三参数拟合法在频率值给定的条件下, 计算的幅度值几乎没有误差, 误差来源于拟合算式;
- 4) 四参数拟合法的误差来源于迭代中值频率与实际频率的差值, 依赖于迭代条件;
- 5) 算法计算时间与迭代次数有关, 更优的 首次迭代中值频率的计算方法 可使迭代次数有效减少;

1.2 针对复杂工程问题的推理分析

1.2.1 两种估算方法的选择：

基于最小二乘法拟合与相位差两种方法的分析，由于在 DSP 上采用相位差算法能够减少估算的时间，且根据相位差得出的频率估计值更为精确，计算过程短且思路清晰，便于工程实现等特点，小组选用相位差方法完成采样后续的估算过程。

进行工程中的相位差分法估算，首先需要将时域正弦信号中频率、幅值和相位在频谱中对应的表现形式找到，根据频谱特点来推断正弦参数的具体数值。相位差分法的思路主要是首先直接通过 FFT 参数测值，然后使用相位差分对粗测估值进行矫正。

1.2.2 傅立叶变换物理意义分析

傅里叶变换是数字信号处理中常用的重要数字变换。对于有限长序列的离散傅里叶变换 DFT，其实质是有限长序列傅里叶变换的有限点离散采样，从而实现了频域离散化，使数字信号处理可以在频域采用数值运算的方法进行，增加了数字信号处理的灵活性。

任何连续测量的时序或信号，都可以表示为不同频率的正弦波信号的无限叠加。而根据该原理创立的傅立叶变换算法利用直接测量到的原始信号，以累加方式来计算该信号中不同正弦波信号的频率、振幅和相位。

假设采样频率为 F_s ，信号频率 F ，采样点数为 N 。那么 FFT 之后结果就是一个为 N 点的复数。每一个点就对应着一个频率点，每个点的模就是该频率值下的幅值。

假设原始信号的峰值为 A ，那么 FFT 的结果的每个点的模值就是 A 的 $N/2$ 倍。而第一个点就是直流分量，它的模值就是直流分量的 N 倍。而每个点的相位呢，就是在该频率下的信号的相位。

1.2.3 直接 FFT 算法分析

假设采样频率为 F_s ，信号频率 F ，采样点数为 N 。

原始信号做 FFT 之后，某一点 n 表示的频率为：
$$F_n = (n - 1) * \frac{F_s}{N};$$

该点的模值的表达形式为 $A_k = \frac{2}{N} \sqrt{a_n^2 + b_n^2}$, a 和 b 分别为该点傅立叶变换后的实部和虚部部分数值, 模值除以 $N/2$ 就是对应该频率下的信号的幅度 (对于直流信号是除以 N): $A = A_k * \frac{2}{N}$ 。

由于 FFT 有明显的栅栏效应, 只有很少的频率点不存在频谱泄露, 于是对那些存在频谱泄露的点来说, 用上式来估计信号幅度会有很大的误差, 最大可达 36.3%;

该点的相位即是对应该频率下的信号的相位; 相位的计算可用反正切函数 atan2 计算。atan2(b,a)是求坐标为(a,b)点的角度值, 范围从 -pi 到 pi;

$$\hat{\theta} = \tan^{-1} \left[-\frac{\sum_{n=0}^{N-1} s(n) \cos(2\pi \hat{f}_0 n \Delta t)}{\sum_{n=0}^{N-1} s(n) \sin(2\pi \hat{f}_0 n \Delta t)} \right]$$

如果要精确到 $x\text{Hz}$, 则需要采样长度为 $1/x$ 秒的信号, 并做 FFT。

如果要提高频率分辨率, 就需要增加采样点数, 这在一些实际的应用中是不现实的, 需要在较短的时间内完成分析。

解决这个问题的方法有频率细分法, 比较简单的方法是采样比较短时间的信号, 然后在后面补充一定数量的 0, 使其长度达到需要的点数, 再做 FFT, 这在一定程度上能够提高频率分辨力。

1.2.4 相位差分法分析

相位差分法是在 FFT 粗测结果上的进一步校正, 这种算法无需在频谱的最大和次大谱线间进行频率的搜索, 只需对采样点分组后进行两次 FFT 就可以在不高的信噪比下获得精度相当高的频率和初相估值, 而且初相和频率的估计精度是彼此独立的, 十分有利于工程的实现。

相位差分法的基本原理是将频率为 f_0 , 初相 θ 的正弦信号

$$s(t) = \exp[j(2\pi f_0 t + \theta)]$$

以采样率 f_s 进行采样, 设信号的记录时间为 T , 采样点数为 N , 将得到的序列分为前后两个等长的子序列 $s_1(n)$ 和 $s_2(n)$, 分别求出它们的 FFT, 得:

$$S_1(k) = A_k \exp(j\phi_k)$$

$$S_2(k) = S_1(k) \exp(j\pi f_0 T)$$

其中 A_k 和 ϕ_k 分别为幅度项和相位项, 有如下形式:

$$A_k = \frac{\sin[\pi(k - f_0 T / 2)]}{\sin[2\pi(k - f_0 T / 2) / N]}$$

$$\varphi_k = \theta + (1 - 2/N)(f_0 T / 2 - k)\pi$$

可知幅度最大值处对应的离散频率为 $k_0 = [f_0 T / 2]$ ，用 φ_1 和 φ_2 表示 $S_1(k)$ 和 $S_2(k)$ 在最大谱线处的相位，由文献^[2]可知，利用 $\Delta\varphi = \varphi_2 - \varphi_1$ 可以对粗测频率

$$\hat{f}_k = k_0 \frac{2f_s}{N} = k_0 \Delta f$$

进行校准，偏差的估计为

$$\hat{f}_\delta = \frac{\Delta\varphi}{2\pi} \Delta f$$

从而信号频率 f_0 的估值

$$\begin{aligned} \hat{f}_0 &= \hat{f}_k + \hat{f}_\delta \\ &= \left(k_0 + \frac{\Delta\varphi}{2\pi} \right) \cdot \Delta f \end{aligned}$$

同时得到初相 θ 的估值

$$\hat{\theta} = \frac{3N-2}{2N} \varphi_1 - \frac{N-2}{2N} \varphi_2$$

1.3 针对复杂工程问题的方案实现

1.3.1 使用 DSP/BIOS 实现对音频信号的采样

在 main 函数中首先完成对 EVMDM6437 和立体声编码器 codec 的初始化，配置采样频率、滤波器系数和串口寄存器等，首先对进行数据存放的数组进行初始化，包括直接缓存、单声道数组缓存、定点转浮点和频谱模值缓存。

然后配置控制状态寄存器 CSR、中断清零寄存器 ICR 和中断使能寄存器 IER 的值来启动串口，允许硬件中断执行。此时主函数运行完毕，程序运行将始终等待 McBSP 产生的中断请求返回数据，直到采样点数达到运算需求。

其中每一次采集到的 32 位双声道数据 dataIn 中的右声道数值将被依次存放于 buffer 数组中，与此同时两个声道通过矩形窗滤波后再次合成赋值 dataOut 传出至音频输出口。

代码 1 采样功能的实现

```
// ===== MAIN - init and return to begin BIOS Scheduler environment...=====
void main(void){                                     // Main: setup prior to BIOS
    EVMDM6437_init();                                // Init EVM6437 HW
    initCodec();                                     // Init McBSP1; s/u AIC via I2C
    initDipSwitches();

    for (i = 0; i < 2*Npoint; i++)                  // For size of stereo buffer
        buf[i] = 0; buffer[i] = 0; buffer2[i] = 0; A[i] = 0;
    ICR = 0x10;                                       // Clear INT4 (precaution)
    IER |= 0x10;                                     // Enable INT4 as CPU interrupt
    MCBSP1_SPCR = 0x00010001;                         // Start McBSP
}

// ===== isrAudio() - Serial Port Interrupt Service Routine: BIOS HWI =====
void isrAudio(void){                                 // Get, process, send data: IPO
    dataIn = MCBSP1_DRR_32BIT;                       // get one stereo sample
    buf[0] = (short)dataIn;                          // add new L data to delay line
    buf[1] = (short)(dataIn >> 16);                 // add new R data to delay line
    buffer[counter++] = (short)(dataIn >> 16);      // add new audio data to buffer

    for (i = FIRSZ-2; i >= 0; i--)                  // for 2*(#coeffs-1)
        buf[i+2] = buf[i];                          // move all data down 1 pair
```

```

if( sw0 == 1 ){                                     // if SW0 is down...
    fir(&buf[0], &coeffs[sw1][0], &dataOutL, FIRSZ, 1); // FIR: 1 sample L
    fir(&buf[1], &coeffs[sw1][0], &dataOutR, FIRSZ, 1); // FIR: 1 sample R
    dataOut = 0x0000FFFF & dataOutL;                // left result is 16 LSBs
    dataOut |= 0xFFFF0000 & (dataOutR<<16); // right result is 16 MSBs
}
else dataOut = dataIn;                               // new input copied to output
MCBSP1_DXR_32BIT = dataOut;                          // send 1 stereo value to codec
if(counter==Npoint) SWI_post(&estimate_SWI); // Switch to software interrupt
}}

```

1.3.2 直接 FFT 进行估计

直接 FFT 可以放置于软中断函数 estimate 中，函数通过硬件中断采样完毕后调用 SWI_post 函数触发执行，估算之前首先做数据结构的转换，将原有的 16 位整形数依次转换赋值到复数结构体中的 float 浮点实数中。然后将右声道 buffer 数组清空准备下一次的数据采集。

复数结构准备好后进行 FFT 变换，然后计算频谱上各点对应的模的值，寻找频谱中的最大的模值 max，将其做频域到时域的变换之后得到相应的最大谱线处幅度值，此幅度值可能是信号的幅度估计值。

由于加矩形窗直接做 FFT 变换后的幅度估计值误差较大，在大信噪比下采集到的正弦信号其时域的最大值也接近于实际信号的幅度值，遍历采样值后也可以寻找到幅度的估计值。

查找到频谱最大值点位 k 后可以依据算式 $F_n = (k - 1) * \frac{F_s}{N}$ 计算出频率的大致估计值，但此方法计算出的频率偏差可以达到 10% 左右，导致依赖于频率估计值的相位估计算式误差进一步扩大。

因此采取将 o 替代 1 进行频率计算的方法进行误差矫正，由 k 点对应复数结构的虚实两部计算得到，当 k 被定位到 N 点右侧频谱处时， o 为负数使计算结果偏离原式，需要将 k 调整至频谱前半部分对称位， o 值恢复正数计算，此时计算的频率估计值接近于实际的频率值。

频率估计值确定之后，使用算式 $\theta = \arg(s[f_0])$ 通过时域数值计算出较为准确的初相估计值。

代码 2 直接估算方法的实现

```

// ===== estimate() - Parameter estimation and preparation work: BIOS SWI =====
void estimate(void){
    for(j = 0; j < Npoint; j++){
        buffer2[j]=short2float(buffer[j]);
        s[j].real=buffer2[j]; s[j].imag=0;
    }
    // Convert to frequency domain analysis
    fft(s, Npoint, -1); // FFT calculation
    for (q = 0; q < Npoint; q++){ // Gets the spectral value
        A[q] = sqrt(s[q].real*s[q].real + s[q].imag*s[q].imag);
        if (A[q] > max) {
            max = A[q]; k = q; // Find the corresponding point
        }
    }
    // Direct search for the maximum time domain
    for(q = 0; q < Npoint; q++){
        if(fabs(buffer2[q])>maxx) { maxx=fabs(buffer2[q]); }
    }
    max = max * 2 / Npoint /cor; // Convert to maximum range
    float range=(maxx+max)/2;
    // Frequency calculation using correction parameters
    if (s[k].real > s[k].imag)
        o = (s[k + 1].real - s[k - 1].real) / (2 * s[k].real - s[k - 1].real - s[k + 1].real);
    else
        o = (s[k + 1].imag - s[k - 1].imag) / (2 * s[k].imag - s[k - 1].imag - s[k + 1].imag);
    if(2*k>Npoint){k = Npoint - k; o = fabs(o);} // Relative orthodontic correct
    freq = (k - o)*Samplerate / Npoint; // Sinusoidal frequency
    // Using accurate frequency value to calculate phase
    for (q = 0; q < Npoint; q++) {
        a = a + buffer2[q] * cos(2 * PI*freq*q / Samplerate);
        b = b + buffer2[q] * sin(2 * PI*freq*q / Samplerate);
    }
    phi = atan(a / b); // Calculat initial phase
    counter=0; // Restart the sampling process
}

```

1.3.3 相位差分法改进

相位差分法主要是通过两次频谱的谱峰做相位差值得到 $\Delta\varphi = \varphi_2 - \varphi_1$ ，来达到对频率估计值和相位估计值的进一步矫正，项目中引入相位差方法适合于对初相估计值和频率估计值进行一次修正。其具体步骤如下：

分两段各 512 点进行 FFT 计算，两端频谱的谱峰求均值即求出对应的幅度，幅度较直接法测量值误差更小，求取频率的方法与直接方法保持一致，求相位时需要求出第一段信号的初相、第一段信号的谱峰点相位，第二段信号的谱峰点相位，两端谱峰信号的差值即为相位差。按照式

$$f_0 = \left(k_0 + \frac{\Delta\varphi}{2\pi}\right)\Delta f \text{ 和 } \varphi_0 = \varphi_1 - \frac{N-2}{2N}\Delta\varphi \text{ 即可完成偏差纠正。}$$

代码 3 相位差算法的实现

```
fft(s, N, -1);fft(s2, N, -1); //分段 FFT
A[i] = sqrt(s[i].real*s[i].real + s[i].imag*s[i].imag); //频谱模值计算
A2[i] = sqrt(s2[i].real*s2[i].real + s2[i].imag*s2[i].imag);
max = A[k]; max2 = A2[k2]; //查找谱峰和点位
A0 = (max + max2) / N / cor; //计算幅度
//频率计算
if (2 * k > N) { k = N - k; o = fabs(o); } if (2 * k2 > N) k2 = N - k2;
f00 = (k - o)*fs / N;
//两段 k 点相位计算
fi1 = atan(a / b); fk1 = atan(a / b) + k * f00 * 2*PI / fs;
fk2 = atan(a2 / b2) + k2 * f00 * 2*PI / fs;
//相位差和频率偏差值
deltaphi = fk2 - fk1; deltaf = fs * deltaphi / N / PI;
//相位和频率纠正
fi0 = fi1 - (N - 2)* deltaphi / 2 / N;
freq = f00+deltaf0;
```

1.3.4 复数结构体的 FFT 实现

N 点 FFT 运算可以分成 $\log_2 N$ 级，每一级都有 $N/2$ 个蝶形，FFT 的基本思想是用 3 层循环完成全部运算。第一层循环对运算的级数进行控制，一共需要 m 级计算；第二层循环根据乘数进行控制，保证对于每一个蝶形因子第三层循环要执行一次；第三层循环要将本级中每个群中具有这一乘数的蝶形计算一次。

傅立叶变换计算完成后的实部和虚部会依次存放于对应的复数结构中。

代码 4 傅立叶变换的实现

```
void fft(complex x[]) {
    while (1) {                                     /*一级蝶形运算 stage */
        if (1 >= n) {                               /*计算完成判断条件 */
            if (isign == -1) return;
            for (q = 0; q < n; q++) {
                x[q].real = x[q].real / n; x[q].imag = x[q].imag / n;
            } return;
        }

        for (m = 0; m < l; m++) {                   /*一组蝶形运算 group*/
            for (h = m; h < n; h = h + 2 * l) {      /*一次蝶形运算 */
                z.imag = m * pisin / l; ce = cexp(z);
                t.real = x[h + l].real * ce.real - x[h + l].imag * ce.imag;
                t.imag = x[h + l].real * ce.imag + x[h + l].imag * ce.real;
                x[h + l].real = x[h].real - t.real;
                x[h + l].imag = x[h].imag - t.imag;
                x[h].real = x[h].real + t.real;
                x[h].imag = x[h].imag + t.imag;
            } l = 2 * l;
        }
    }
}
```

1.3.5 使用 DFT 库函数估计

上一小节中的傅立叶变换函数是针对复数结构体进行计算变换的，以便于估算算法的直接调用实现。

由于 C64x+库中有 FFT 相关库函数可以调用，可以使用到 32 位和 16 位的傅立叶变换及其反变换等十种变换函数，且调用这些优化的函数，可以获得比标准 C 语言程序快得多的执行速度。因此也可以使用库中的相关函数进行计算。

首先需要添加 DSP_fft16x16.h, DSP_q15tofl.h 和 gen_twiddle_fft16x16.h 三个头文件及其.c 源文件以进行调用，使用 DSP_fft16x16 函数可以完成 16bit 数据的复输入输出计算，gen_twiddle_fft16x16 函数用于生成 N 点旋转因子用于 FFT 变换，DSP_q15tofl 函数则是对完成了变换后的整形数据完成浮点转换运算。

估算函数中首先对 fftindata 数据进行压缩以防变换时溢出，然后生成旋转因子，进行傅立叶变换后转为浮点交织赋值给复数结构体中的实部和虚部，完成数据结

构的转换，之后进行频谱计算、谱峰查找、幅度频率和初相的估计计算。

此问题可以有效解决 2.1.2 节中硬件中断抢占所带来的浮点转换问题。

代码 5 使用库函数方式实现

```
void curvefitpara(void){
    for( p = 0; p < Npoint; p++ )
        fftindata[2*p] = fftindata[2*p]>>7; //压缩输入数据 7bit, 防止 FFT 运算后溢出
    gen_twiddle_fft16x16(w_16x16, Npoint); //旋转因子的产生
    DSP_fft16x16(w_16x16, Npoint,fftindata, fftoutdata);
    //FFT 变换(旋转因子,点数,输入数据,输出数据)
    DSP_q15tofl(fftoutdata, fftdata, Npoint); //结果转浮点
    //数据结构变换
    for (q = 0, r = 0; r < Npoint; q+=2, r++)
        s[r].real = fftdata[q]; s[r].image = fftdata[q + 1];
    //频谱计算
    for(findmax=0;findmax<Npoint;findmax++)
        AA[findmax]=sqrt(s[findmax].real*s[findmax].real+s[findmax].image*s[findmax].image);
    //参数估计
    max=max*2/Npoint; //幅度估计值
    freq=(k-op)*Samplerate/Npoint; //频率估计值
    phi = atan(a / b); //相位估计值
}
```

第二章 系统测试

2.1 系统分段测试

系统分为两段实现，前一段为正弦信号的波形采样，后一段为采样数据的 FFT 变换和参数估算，中间需要通过整型数值到浮点数值的转换完成复数结构的建立进行连接。

关于系统测量的范围，对于初相的测量，测量的结果应当位于区间 $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ ，且不存在测量范围。对于频率的测量，系统采用 CD 音频采样率 44100Hz，比特 16Bits，根据奈奎斯特采样定律，正弦信号的频率不应高于 22kHz；对于幅度的测量，由于采样使用到的整型数值 buffer 为 16Bits 的 short 型数据，其单边幅度的最大值为 32767，对应转化为信号的伏特约为 3.81V，为可能采集到的最大信号幅值，实际使用中建议不要超过 3.8V，可能会造成数据的溢出造成顶端波形截断。

2.1.1 正弦信号采样部分的测试

正弦信号由 AIC33 经编码器采样转换成数字信号，采样频率恒定，经硬件中断发生后采集得到。所以采样达到点数要求后进行右声道数值对应 buffer 数组的数据查验，数据需要满足以下特点：

1. 数值需要满足正负两极循环交替的分布特点；
2. 对于低频正弦波的数值需要呈现稳定的规律的波形走势变动；
3. 数值呈现周期分布特点，且周期之间数值变动幅度相近，频率恒定。

设计三组采样波形，对百千赫兹和万赫兹量级的波形采样分别进行验证：

1. freq=300Hz, phase=30°, range=2V;
2. freq=1.2kHz, phase=30°, range=3V;
3. freq=12kHz, phase=30°, range=1V;

以下是数据采集的结果：

对于 300Hz 的低频信号，采集到的数据能够看出明显的缓慢的波形变化趋势，

其幅度值大多在正负 17200 之间；对于 1200Hz 的信号，采集到的数据呈现的变化趋势更陡，幅度值位于正负 25800 之间；对于频率更高的 12kHz，也呈现了相应的波形变化趋势。

sion	Type	Value	sion	Type	Value
{0} [0]	short	-13247	{0} [0]	short	-10944
{0} [1]	short	-13636	{0} [1]	short	-11447
{0} [2]	short	-14000	{0} [2]	short	12447
{0} [3]	short	-14339	{0} [3]	short	8224
{0} [4]	short	-14652	{0} [4]	short	10605
{0} [5]	short	-14934	{0} [5]	short	12688
{0} [6]	short	-15192	{0} [6]	short	14409
{0} [7]	short	-15419	{0} [7]	short	15715
{0} [8]	short	-15618	{0} [8]	short	16572
{0} [9]	short	-15789	{0} [9]	short	16951
{0} [10]	short	-15929	{0} [10]	short	16845
{0} [11]	short	-16042	{0} [11]	short	16257
{0} [12]	short	-16123	{0} [12]	short	15201
{0} [13]	short	-16177	{0} [13]	short	13707
{0} [14]	short	-16198	{0} [14]	short	11822
{0} [15]	short	-16190	{0} [15]	short	9601
{0} [16]	short	-16152	{0} [16]	short	7106
{0} [17]	short	-16083	{0} [17]	short	4411
{0} [18]	short	-15985	{0} [18]	short	1597
{0} [19]	short	-15857	{0} [19]	short	-1254
{0} [20]	short	-15700	{0} [20]	short	-4063
{0} [21]	short	-15514	{0} [21]	short	-6747
{0} [22]	short	-15299	{0} [22]	short	-9229
{0} [23]	short	-15055	{0} [23]	short	-11434
{0} [24]	short	-14785	{0} [24]	short	-13293
{0} [25]	short	-14486	{0} [25]	short	-14761
{0} [26]	short	-14160	{0} [26]	short	-15791
{0} [27]	short	-13810	{0} [27]	short	-16350
{0} [28]	short	-13430	{0} [28]	short	-16426
{0} [29]	short	-13027	{0} [29]	short	-16016
{0} [30]	short	-12601	{0} [30]	short	-15134
buffer			{0} [31]	short	-13798
			buffer		

图 6 百千赫兹级数据的采集结果

图 6 中，左图为 300Hz 时的采集数值，呈现下降后反弹的趋势，经历了正弦信号的负半轴的波峰，右图为 1200Hz 时的采集数值，32 个点几乎呈现了一个周期内的变化趋势。

图 7 为 12kHz 时的采集数值，数据在一个周期内采到的点数较为有限，仍能看出一定的变化趋势，从右侧的参数窗口中也能看见参数的估计值。。

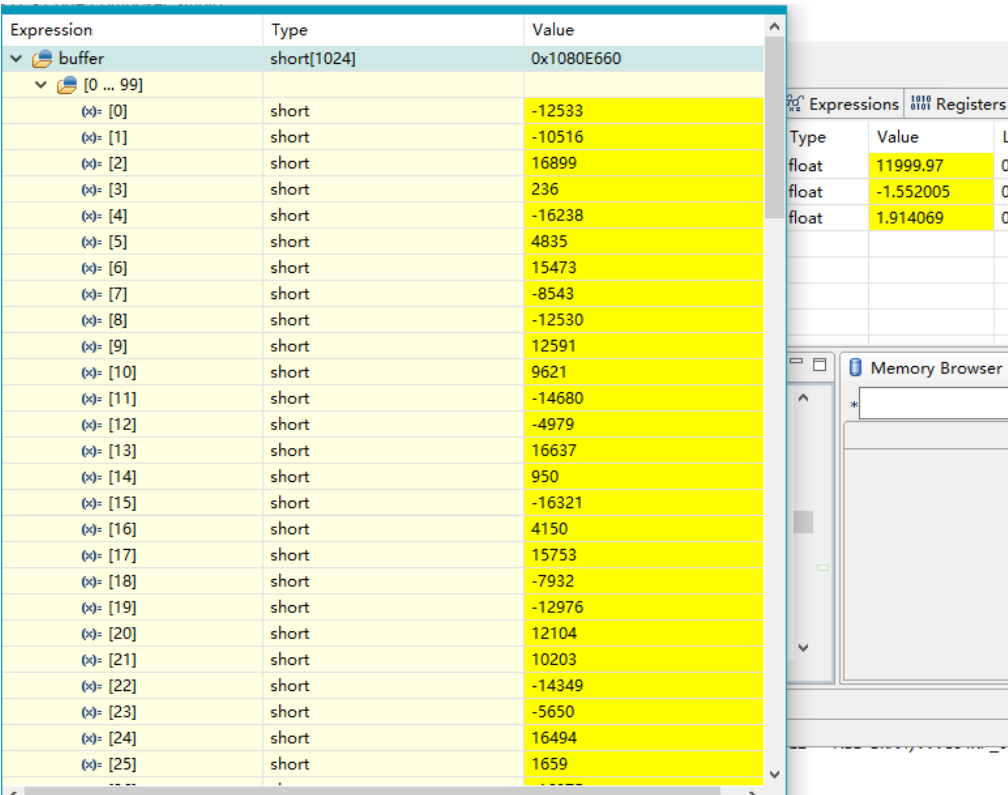


图 7 12kHz 时的采集数值

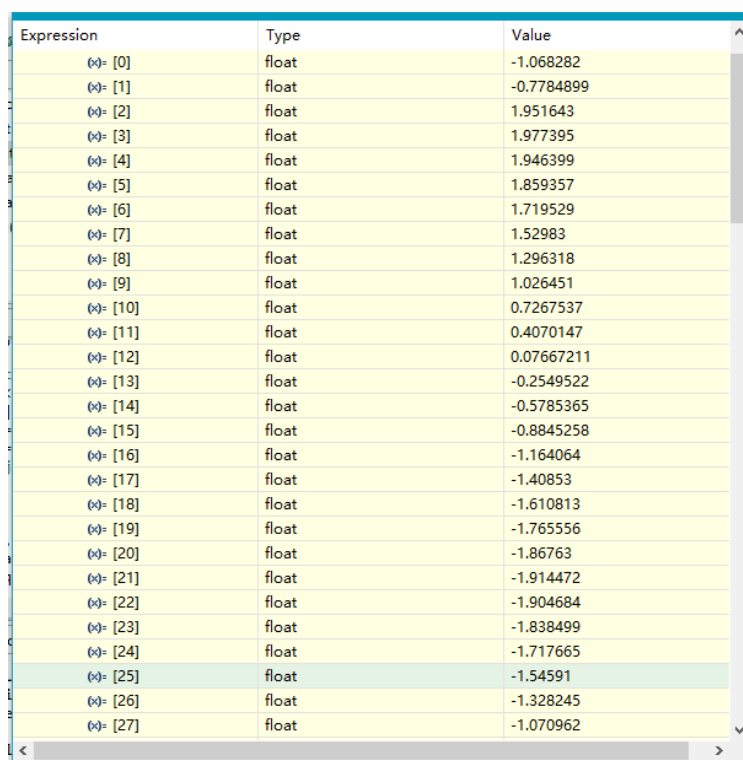
2.1.2 正弦信号采样值浮点转换验证

正弦信号经编码器采样转换成的数字信号只能表示数据的相对值，并不能直接表示实际的信号源产生的以伏特为单位的信号幅度。所以需要对于编码的数字信号到伏特的信号发生单位进行数据的映射，同时需要完成 short 型到复数结构中 float 数据类型的转换。

在工程中设计 short2float 函数，函数中第一部首先完成整型数对浮点数的强制转换，然后根据频谱映射的规律完成对伏特单位的转换。tran 为单位的转换系数，需要大量的实验得出经验值。

```
float short2float(short input){return ((float)input)/trans;}
```

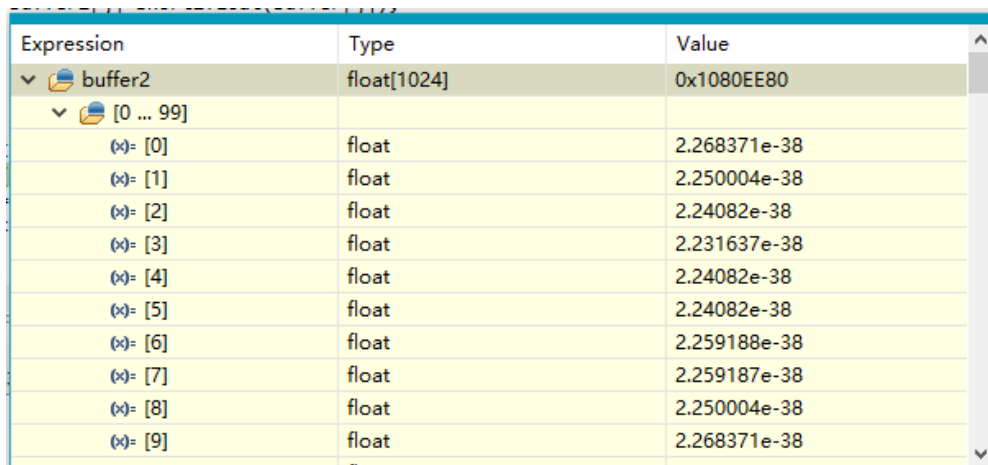
同样对于上一小节进行采样的第二组信号，在存储浮点信号 buffer2 的数组中进行数据的查验，以确定是否完成对应的工作。以下是数据采集和转换的结果：



Expression	Type	Value
(x)= [0]	float	-1.068282
(x)= [1]	float	-0.7784899
(x)= [2]	float	1.951643
(x)= [3]	float	1.977395
(x)= [4]	float	1.946399
(x)= [5]	float	1.859357
(x)= [6]	float	1.719529
(x)= [7]	float	1.52983
(x)= [8]	float	1.296318
(x)= [9]	float	1.026451
(x)= [10]	float	0.7267537
(x)= [11]	float	0.4070147
(x)= [12]	float	0.07667211
(x)= [13]	float	-0.2549522
(x)= [14]	float	-0.5785365
(x)= [15]	float	-0.8845258
(x)= [16]	float	-1.164064
(x)= [17]	float	-1.40853
(x)= [18]	float	-1.610813
(x)= [19]	float	-1.765556
(x)= [20]	float	-1.86763
(x)= [21]	float	-1.914472
(x)= [22]	float	-1.904684
(x)= [23]	float	-1.838499
(x)= [24]	float	-1.717665
(x)= [25]	float	-1.54591
(x)= [26]	float	-1.328245
(x)= [27]	float	-1.070962

图 8 1200Hz 时浮点转换结果

转换实现时遇到的问题：如图 9 所示，转换功能在开发板上运行时，会遇到数值转换错误的情况，出现大量的不可预料的数据，而右声道数组 `buffer` 中数值一切正常。



Expression	Type	Value
buffer2	float[1024]	0x1080EE80
(x)= [0]	float	2.268371e-38
(x)= [1]	float	2.250004e-38
(x)= [2]	float	2.24082e-38
(x)= [3]	float	2.231637e-38
(x)= [4]	float	2.24082e-38
(x)= [5]	float	2.24082e-38
(x)= [6]	float	2.259188e-38
(x)= [7]	float	2.259187e-38
(x)= [8]	float	2.250004e-38
(x)= [9]	float	2.268371e-38

图 9 转浮点数异常结果

此问题是由于位于软件中断的转换函数在转换过程中被硬件中断抢断，强制转换 float 在中断切换的过程中可能未保护被中断进程现场相应数据，导致现场恢复时计算结果错误。

问题解决方案：由于强制转换会由于硬件中断的打断而发生错误，则避免错误的方法有两个：一是使用 1.3.5 节中代码 5 库函数方法实现，采用 DSP_q15tofl 完成转换；二是在转换过程中或者估算过程中暂时屏蔽掉硬件中断，因为此时不需要再采集任何波形数据，在不需要打断处使用 HWI_disable 和 HWI_enable 即可。

2.1.3 估算算法的准确度验证

估算算法的 VS 平台 C 语言测试代码实现：需要通过正弦函数生成器模拟采样过程，得到采样的数据赋值给复数结构体进行计算。

代码 6 VS 平台估算法测量代码

```
complex cexp(complex a) ;
void fft(complex x[], int n, int isign);

int main(void) {
    float fi = 0.25, f0 = 800, AA = 2.4, fs = 44100;
    //初相位, 频率, 幅度, 采样频率
    for (i = 0; i < N; i++) {
        x[i] = AA * sin(2 * PI*f0*i / fs + fi);
        s[i].real = x[i]; s[i].imag = 0;
    }
    fft(s, N, -1);
    A[i] = sqrt(s[i].real*s[i].real + s[i].imag*s[i].imag);
    max = max * 2 / N /correct
    f00 = (k - o)*fs / N;
    fi1 = atan(a / b);
}
```

测试结果数据如表 1 所示，频率何初相的估计值都与实际值相近，幅度的估计值偶尔会出现计算结果的偏离：

表 1 估算法测试结果

	频率 Hz		幅度		初相 rad	
	原始值	估计值	原始值	估计值	原始值	估计值
1	300	299.992	2	2.026	0.2	0.199
2	850	849.993	2	1.788	0.2	0.208
3	1200	1200.001	2	1.951	0.2	0.202
4	1200	1200.001	3	2.926	0.2	0.202
5	1200	1200.001	3	2.926	0.4	0.4008

幅度估计值偏离原因分析：正弦波采样值直接作 FFT 之后，在频率 f 处会出现一个谱峰，幅度用 M 表示。在没有频谱泄露的情况下，有如下关系：

$$M_{peak} = \frac{An}{2}$$

由于 FFT 有明显的栅栏效应，只有很少的频率点不存在频谱泄露，如图 10(a) 所示。实际上谱峰所在的数值区间应该位于：

$$\frac{2}{\pi} \cdot \frac{An}{2} \leq M_{peak} \leq \frac{An}{2}$$

于是，对那些存在频谱泄露的点来说，用上式来估计信号幅度会有很大的误差，最大可达 36.3%，如图 10(b) 所示。

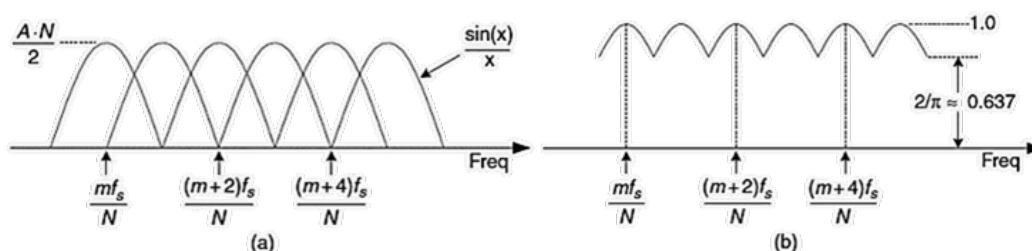


图 10 FFT 频谱泄露误差

这个误差是由于采用矩形窗导致的，如果要减小误差则有两种方式，一是使用汉宁或汉明窗，两者可以将频谱泄露误差减少约一半。二是在选取谱峰之后，对其幅度值进行一定程度的校正，得到幅度的更好的估计。校正的方法是对谱峰及其附近的时域信号值进行加权，来达到加窗的目的：

$$X_{ft}(m) = \frac{h_4}{2}X(m-4) - \frac{h_3}{2}X(m-3) + \frac{h_2}{2}X(m-2) - \frac{h_1}{2}X(m-1) + h_0X(m) \\ - \frac{h_1}{2}X(m+1) + \frac{h_2}{2}X(m+2) - \frac{h_3}{2}X(m+3) + \frac{h_4}{2}X(m+4)$$

$h_0 = 0.2156, h_1 = 0.4160, h_0 = 0.2781, h_0 = 0.0836, h_0 = 0.0069;$

加权之后最大估计误差约为 0.0166dB，更多关于误差矫正的信息参考^[5]。

2.2 测试案例设计及系统整体测试结果

由于系统测试的初相数据由采集信号的第一点确定，每一次测量条件下都可能会导致不相同，存在很大的偶然性，因此在估算的过程中很难直接验证初相的计算值是否正确。但估算算法的准确度验证小节中，在精确的频率估计条件下，总是可以得出准确的初相估计值，此处可以根据频率的估计值的精准度在一定程度上表现了初相估计值精准与否。

设计的测试用例如下：

1. freq=500Hz, phase=30°, range=1V;

Name	Type	Value	Location
(x)= freq	float	499.9746	0x1081C2A8
(x)= phi	float	-1.169648	0x1081C2A4
(x)= range	float	0.9514431	0x1081C2AC

图 11 实验测试结果 1

2. freq=1000Hz, phase=30°, range=1V;

Name	Type	Value	Location
(x)= freq	float	999.9946	0x1081C2A8
(x)= phi	float	0.4755884	0x1081C2A4
(x)= range	float	1.041042	0x1081C2AC

图 12 实验测试结果 2

3. freq=1000Hz, phase=30°, range=2V;

Name	Type	Value	Location
(x)= freq	float	1000.001	0x1081C2A8
(x)= phi	float	0.1698443	0x1081C2A4
(x)= range	float	2.058992	0x1081C2AC

图 13 实验测试结果 3

4. freq=12kHz, phase=30°, range=3V;

Name	Type	Value	Location
(x)= freq	float	11999.97	0x1081C2A8
(x)= phi	float	0.1325572	0x1081C2A4
(x)= range	float	2.960573	0x1081C2AC

图 14 实验测试结果 4

测试实拍图:

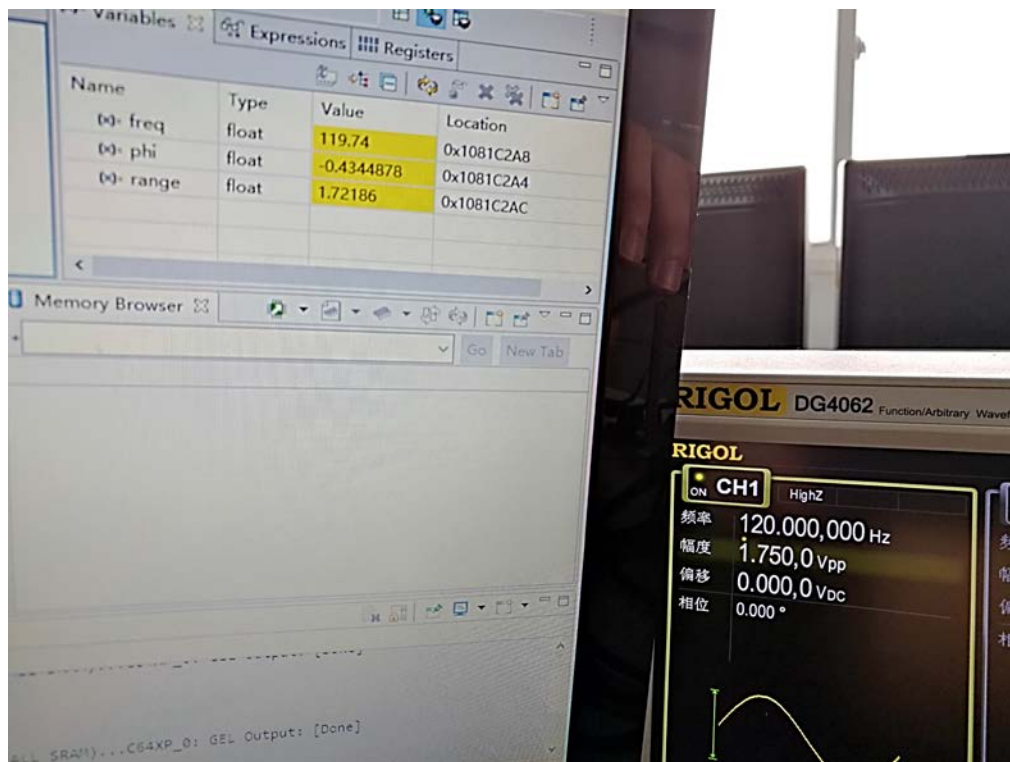


图 15 实验测试结果 5

第三章 知识技能学习情况

在本次实验中首先掌握了 DSP/BIOS 的配置问题,即 HWI, SWI 的主要作用以及参数配置,对于 DSP/BIOS 中的线程管理,各线程类型的优先级及其特点做了深入的了解,选择了硬件中断与软件中断配合的方式。

学会了如何从一个正弦波信号中采样得到多个数据,并且将数据进行筛选处理,进行数据的移位完成双声道分离和双声道的合并输出等操作。在数据估算的算法方面,首先对多个算法的理论知识进行了深入学习,比较了各个算法的性能优劣以及试用的场合,最后我们选择了相位差分法。

在具体实现时,我们并没有一开始就在 CCS 上进行代码的编写以及调试,而是先在 VS 上进行仿真操作,从而验证算法的正确性。相对于 CCS 而言 VS 是我们熟悉的开发工具,而且整个工程模块相对简单,方便我们进行代码的编译以及调试。

代码的实现是建立在 C 语言的基础之上,在整个编写过程中,要体现完整的工程思想,即将功能实现模块化。在整个验证过程中,在 VS 平台通过正弦函数生成器模拟采样,将生成的数据保存下来,再进行运算,这里保存数据的数据结构应该是一个复数结构。将根据采样算出数据算出的参数与初始设定的参数进行比较,从而验证算法的正确性。

将代码从 VS 移植到 CCS 上后,又会出现新的问题。首先采样所得的数据是 16 位整形数,估算之前首先做数据结构的转换,需要转换赋值到复数结构体中的 float 浮点实数中,然后才能进行相应的傅里叶变换。

项目后期,我们发现由于 C64x+库中有 FFT 相关库函数可以调用,可以使用到 32 位和 16 位的傅立叶变换及其反变换等十种变换函数,且调用这些优化的函数,可以获得比标准 C 语言程序快得多的执行速度。因此也可以使用库中的相关函数进行计算。

第四章 分工协作与交流情况

本次实验中部分成员第一次接触 CCS，对于 DSP/BIOS 的线程配置不太了解，相关材料的搜集和具体的配置实现主要由组长负责，平台搭建于 CCS5.5 上，能够通过计算效率不太高的软仿真实现算法在开发板上的计算过程，组员开发测试能够不受开发板数量限制。

项目的准备阶段以确定思路与学习为主，首先确定了实验的运行流程和每个模块的具体功能，在收集资料的同时需要组员快速学习 CCS 的基础知识，对于库函数的查找和使用有了清晰的认识，最后设计出了使用 DSP 库函数和自写 FFT 两种不同的开发思路。

部分组员在 MATLAB 上进行仿真，可以通过少量的语句测试估算算法的可靠性，完成对两种方法的比较，对于最终确立实现方式有较大帮助。

在整个工程项目的过程中组内的分工具体如表 2 所示：

表 2 项目小组分工安排

刘恒利	完成整个项目主要流程模块的设计 DSP/BIOS 的配置实现 相位差算法的实现
何劲帆	前期收集相应的学习资料 两种算法的 C 语言仿真和正确性验证 参与 CCS 上算法的实现工作
胡震	相位差分法前期资料收集 相位差方法数据测试和方法比较
唐浩	CCS 平台和 MATLAB 平台的实验的调试与测试 DSP 库函数的查找和整理
滑文博	傅立叶变换实现方法的查找和确定 代码调试以及测试工作
楚蕃源	MATLAB 平台的实验的调试与测试 最小二乘拟合数据测试和方法比较

分工合作中也遇到了部分问题，即前期的分工并没有得到很好的实现，部分成员前中期积极性不好，能力不足等不能完成组内安排的工作导致延期，相位差分法部分实现过程中困难较大，出现项目进度较慢的情况。

参考文献

- [1] 张涛, 杨炳恒, 樊向党. 基于 FFT 的正弦信号幅值估计研究[J]. 兵器装备工程学报, 2016, 37(7):90-93.
- [2] 张晓威, 孟凡明. 正弦信号幅值和初相位估计的问题研究[J]. 计算机工程与应用, 2013, 49(5):216-219.
- [3] 李春宇, 张晓林, 张展,等. 基于 DFT 的正弦波初相估计算法及误差分析[J]. 北京航空航天大学学报, 2007, 33(5):580-584.
- [4] 黄琳琳, 刘仲. 基于定点处理器的浮点 FFT 算法设计与实现[C]// 微处理器技术论坛. 2014.
- [5] F. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform", Proc. IEEE, vol. 66, no. 1, pp. 51-83, 1978-Jan.

致谢

本报告的工作是在我的指导教师邓建华老师的悉心指导下完成的，邓建华老师严谨的学术态度、独到的研究视角、高标准和严要求使我在此次综合课程的完成有了清晰的认识。本论文完成每一步都是在邓老师的指导下完成的，倾注了邓老师大量的心血。在此，谨向导师表示崇高的敬意和衷心的感谢！