南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

DSP 应用技术实验报告

——实验 9、11

课程名称：DSP应用技术

实验名称：DSP开发基础实验

班级：9161042101

姓名：李镇洋

学号：9161010E0121

指导老师：李彧晟

2019 年 11 月 21 日

目　录

1 实验目的

1.1 实验 9 ： DSP 开发基础

（1）了解 DSP 硬件开发平台基本配置
（2）熟悉 TI DSP 软件集成开发环境
（3）学习 DSP 软件开发流程
（4）掌握工程代码产生方法
（5）学习 DSP 软件调试方法

1.2 实验 11：DSP 数据采集

（1）熟悉 DSP 的软硬件开发平台
（2）掌握 TMS320F28335 的 ePWM 中时间基准子模块和事件触发子模块的基本使用方法
（3）熟悉 TMS320F28335 的中断的设置
（4）掌握 TMS320F28335 的 ADC 模块的基本使用方法
（5）掌握代码调试的基本方法

2 实验仪器

计算机、TMS320F28335 DSP 教学实验箱、XDS510 USB 仿真器

3 实验内容

3.1 实验 9 ： DSP 开发基础

建立工程，对工程进行编译、 链接，载入可执行程序，在 DSP 硬件平台上进行实时调试，利用代码调试工具，查看程序运行结果。

1. 独立完成项目编译、链接、调试的全过程。

2. 记录 dataIO()、processing()子程序的入口地址，记录 currentBuffer.input 和 currentBuffer.output 所在存储器地址。

3. 记录增益控制处理后，以图形方式显示数据空间 currentBuffer.input

和 currentBuffer.output 缓冲存储器中的波形。

4. 打开工程的.map 文件，查看所有的段在存储空间的地址、长度和含义，指出分别位于 TMS320F28335 的什么存储空间以及物理存储块名称，主程序中所用的变量分别属于什么段？

5. 查看.cmd 命令文件，比较其与上述.map 中的映射关系。试图修改.cmd 文件，再次编译链接，查看配置命令与各段的映射关系。

3.2 实验 11：DSP 数据采集

建立工程，编写 DSP 的主程序，对工程进行编译、链接，利用现有 DSP 平台实现数据的采集、存储以及模拟还原，并采取多种方法予以验证。

1. 独立完成项目编译、链接、调试的全过程。

2. 根据范例程序，给出 ADC 的采样频率计算公式，修改 ADC 的采样频率，并验证。

3. 指出波形数据保存的空间地址，并以图形方式显示采集的信号波形，并保存，附在实验报告中。

4. 利用上述图形，给出采样频率的验证方法，以此检验数据采集程序的正确性。

4 实验设计

（1）开发环境搭建以及程序调试

TI 的 CCS 5 集成开发环境，不仅支持汇编的编译、链接，还支持对 C/C++汇编、编译、链接以及优化。同时强大的 IDE 开发环境也为代码的调试提供了强大的功能支持，已经成为 TI 各 DSP 系列的程序设计、制作、调试、优化的主流工具。TMS320F283x 软件开发流程如下图所示。

下面简单介绍各主要模块功能：

• C/C++ Compiler C/C++编译器

C/C++编译器把 C/C++程序 代码编译为基于 DSP 汇编指令集的汇编代码。这种转换并非一一对应，甚至会产生冗余的汇编代码，在某些场合需要使用优化器（Optimizer）来提高转换的效率，使得汇编代码长度尽可能的短小，程序所使用的资源尽可能的少。优化器是编译器的一部分。 编程效率与 编译器直接相关。

• Assembler 汇编器

汇编器负责将汇编语言代码 转换为符合公共目标格式（COFF）的机器语言，这种转换是一一对应的，每一条汇编指令都对应了唯一的机器代码。源文件中还包括汇编指令、伪指令和宏指令。 这里的汇编代码包括了 由 C/C++编

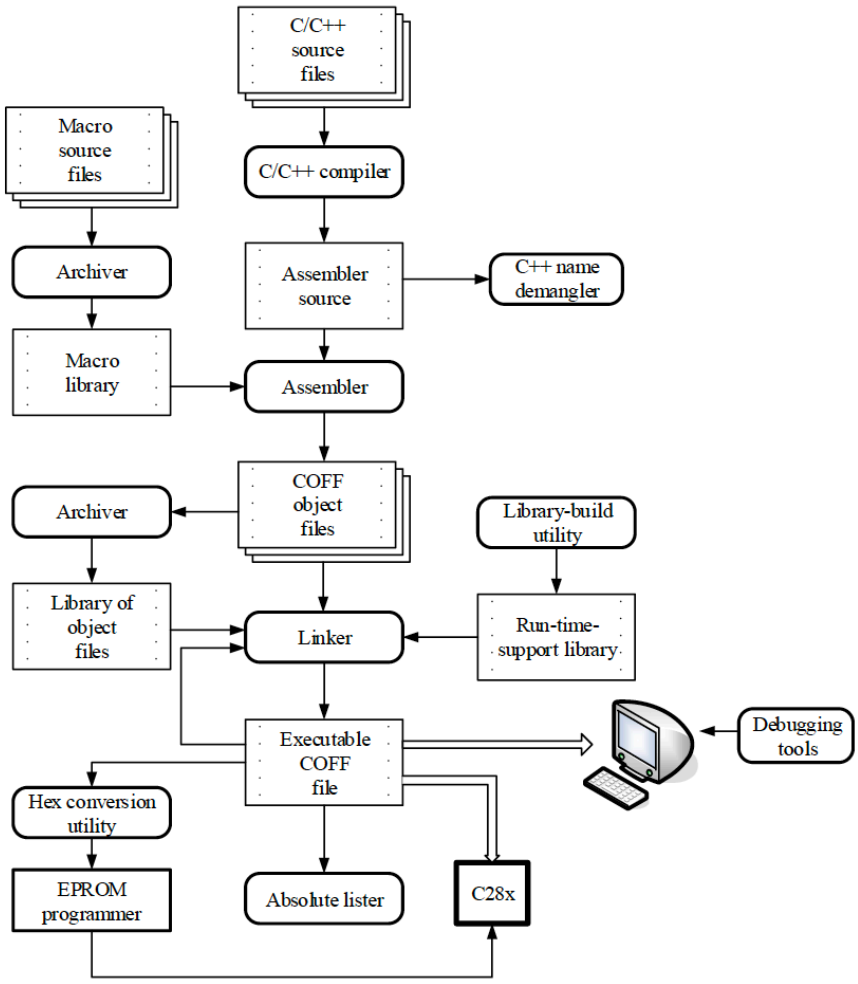
译器生 成的汇编代码和直接编写的汇编代码。

• Linker 链接器

链接器负责把可重定位的多个目标文件和目标库文件转换为一个 DSP 可执行程序，其中包含程序的机器代码、数据以及其他用来链接和加载程序所需要的信息。链接器必须依赖配置命令文件（CMD）的指令，实现对目标文件中各段的定位。

• Run-time-support library 运行支持库

对于用 C/C++语言中编写 DSP 程序中的某些功能（例如存储器的寻址定位、字符串转换等）并不属于 C/C++语言所能描述对象，包含在 C/C++编译器中的运行支持库却可以很好的支持这些算法的标准 ANSI/ISO C 函数描述。函数运行支持库包含有 ANSI/ISO C 的标准运行支持库函数、编译器功能函数、浮点算术函数和系统初始化子程序（这些函数都集成在汇编源文件 rts.src 中）。当对 C/C++编写的 DSP 程序进行链接时，必须根据不同型号的 DSP 芯片添加相应的运行支持库到工程中。除此之外，在使用运行支持库中的函数时，必须在程序起始处用 include 语句包含相应的头文件（如使用数学运算 sin、cos 时，必须包含 math.h）。而采用汇编语言编写程序时，却不需要这个运行支持库。因此 C 语言编写的 DSP 程序链接后，会产生大量的“冗余”汇编程序。



由此可见，用C/C++语言来开发DSP 程序，一般在工程中必须包含以下文件：

• .c 或者.cpp：C 或C++程序，是主程序或函数，用于描述用户特定的算法功能；

• .cmd：配置命令文件，用于对编译生成的COFF 格式目标文件（.obj）定位，安排各段的物理存储空间；

• .lib：运行支持库文件，不同芯片有不同的运行支持库，必须根据具体芯片加以选择，例如TMS320F283x 的运行支持库文件命为rts2800_fpu32.lib。（后缀fpu32 含义是支持32 位浮点运算）。

至于头文件（.h），只有当使用了运行支持库中相应的函数时，才需要在C文件的主程序中用include 语句指定相应的头文件（math.h、stdlib.h、float.h 等）。具体内容参见TI 公司的TMS320C28x Optimizing C/C++ Compiler User's Guide。

其次用户自定义函数、寄存器地址、常量定义等信息也可以编制到头文件中，使用时也同样需要在C 主程序中指定。

例如本实验中，需要的文件：

• main.c：C 语言主程序。

• 28335_RAM_lnk.cmd，DSP2833x_Headers_nonBIOS.cmd：配置命令文件。

• rts2800_fpu32.lib：运行支持库。

• Sine.h：常量定义头文件。

• FPU.h：浮点运算库头文件。

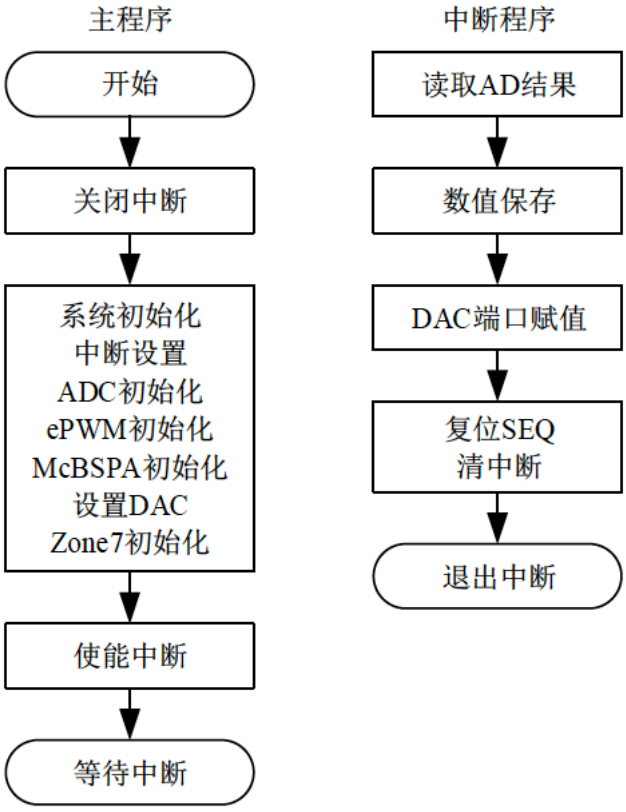
• sine.dat：实验中需要的数据文件。

对于使用CCS以工程为单位进行DSP程序的项目开发时，一般为每个工程建立一个独立的目录，将项目中所需要的文件都存放在该工程目录下，便于程序的管理。rts2800_fpu32.lib在TI的安装目录…\TI\c2000\cgtools\lib中可以找到。

（2）DSP 程序设计

为实现DSP的数据采集存储以及模拟的还原，必须依赖于ADC、DSP以及DAC三大基本部件，而TMS320F28335芯片上集成了ADC模块，因此实现该功能较为简单，数据采集的工作可以由DSP单独完成，只需要对相关外设模块进行合理配置。模拟还原由实验箱中DAC（AD9747）来完成。TMS320F28335中的ADC模块与DSP内核之间的通信可以通过查询方式或中断方式，在此，我们采用ADC的中断功能实现数据的交换。

TMS320F28335中ADC的转换频率和采样频率可以独立设置，分别位于ADC模块和ePWM的时间基准子模块中，因此要使ADC工作，必须掌握ADC模块和ePWM模块中的相关设置。程序流程图如下所示：



【DSP初始化】

一般而言，DSP 要正常工作，必须首先设置时钟，时钟确定了DSP 工作主频。TMS320F28335 中时钟设置大致分为三个主要寄存器，它们分别是锁相环控制寄存器（PLLCR）、外设时钟使能控制寄存器（PCLKCR0，PCLKCR1，PCLKCR2）和外设时钟预定标设置寄存器（HISPCP、LOSPCP）。

【配置数模转换模块ADC】

TMS320F28335 内部有一个16 通道、采样精度为12bit 的ADC 模块。这16通道可配置两个独立的8 通道模块，具有同步采样和顺序采样模式，模拟输入范围0～3V，最快转换时间为80ns，具有多个触发源用于启动AD 的转换，采用灵活的中断控制。

【配置ePWM模块】

TMS320F28335 中ePWM 模块的事件可产生ADC 转换启动脉冲信号SOC，本次实验采用时间基准子模块的产生周期事件，通过事件触发子模块的设置来产生ADC 转换启动脉冲信号SOC。

【配置TMS320F28335中断】

TMS320F283x 的外设中断扩展（PIE）单元通过少量中断输入信号的复用来扩展大量的中断源，PIE 单元支持多达96 个独立的中断，这些中断以8 个为一组进行分类，每组中的所有中断共用一个CPU 级中断（INT1～INT12）。96 个中断对应的中断向量表存储在专用RAM 区域中。PIE 向量表用来存储系统中每个中断服务程序（ISR）的入口地址。一般来说，在设备初始化时就要设置PIE 向

量表，并可在程序执行期间根据需要对其进行更新。

在实验中，当我们设置VMAP＝1（ST1寄存器的bit3），ENPIE＝1（PIECTRL寄存器的bit0）后，TMS320F28335的中断向量表地址范围0x000D00～0x000DFF。例如ADC外设模块SEQ1INT中断向量地址是0x000D40，SEQ2INT中断向量地址是0x000D42，ADCINT中断向量地址为0x000D4A（ADCINT是SEQ1INT和SEQ2INT的逻辑或）。

要想正确使用中断，首先应该合理设置中断向量表，在对应地址填入中断服务子程序的入口地址。其次，必须对上述三个级别的中断作出正确的设置。比如实验中，要想实现CPU利用中断方式读取ADC的采样数据，可以使能ADC模块的中断SEQ1INT，其次使能外设使能寄存器PIEIER1.1，保证中断发生时PIEACK1.1位清零，最后使能CPU中断使能寄存器IER中的INT1，以及全局中断使能位INTM。这些工作必须在系统初始化时完成。退出中断服务程序前，清除ADCST中的INT SEQ1以及相应的PIEACKx。

5 实验步骤

5.1 实验 9 ： DSP 开发基础

1. 设备检查

检查仿真器、F28335 DSP教学实验箱、计算机之间的连接是否正确，打开计算机和实验箱电源。

2. 启动集成开发环境

点击桌面CCS5快捷方式，启动CCS集成开发环境。

3. 新建工程

新建一个工程“Project →New CCS Project”命令，弹出“CCS Project”对话框。在第一项Project Name中输入新建的工程名称，第二项Project Type中选择输出文件格式“Executable（.out）”，在第三项Location中选择工程所在目录，在第四项Device中选择与当前DSP芯片吻合的“2833x Delfino →TMS320F28335”，在Connection中选择仿真器型号“SEED XDS510PLUS Emulator”。在“Project templates and examples”中选择“Empty Project”，单击“完成”按钮确定。则在工程指定的目录中，建立了一个以工程命名的工程文件，它会存储有关该工程的所有设置。

4. 添加工程文件 _

选中工程文件后右键选择“Add Files to Project”命令，在弹出的对话框中依次选择当前工程目录下main.c、source目录夹下所有的文件、以及28335_RAM_lnk.cmd（原工程产生的cmd文件内存分配不够会报错，需要修改，

将它替换成改动过的cmd文件)、DSP2833x_Headers_nonBIOS.cmd，添加到当前工程中。在工程浏览窗口中，展开工程文件列表，可看到刚刚所添加的文件。如果错误的添加了文件，可以在工程浏览窗口中的文件名中单击鼠标右键，在弹出的菜单中选择“Delete”。

当然，CCS也支持文件编辑功能，可以在主菜单选择“File → New”新建一个文件，编辑完成保存为所需要相应格式的C语言程序、汇编程序、cmd配置命令或者头文件，然后添加到工程中。

在添加完文件后，需要为工程添加搜索路径。右击工程标题，在弹出的对话框选“Properties”，进入工程配置对话框，选中左侧的“include Options选项卡”，在右侧的“Add dir to #include path”中点击该框右上侧的“+”，选择“Workspace”，在新建的工程的目录下选择“header”文件夹，点击“Ok”，完成搜索路径的添加。

为工程添加库文件，在工程浏览窗口中的文件名中单击鼠标右键，在弹出的菜单中选择“Properties”，进入工程配置对话框，选中左侧的“General”选项卡”，在“Runtime support library”选项中通过下拉框选择“rts2800_fpu32.lib”后点击“OK”完成库文件的添加。

5. 查阅代码

在 build 工程之前，先阅读一下源代码，明白各文件的内容：在“Project Explorer”栏展开“LAB_9”工程，双击“main.c”文件，即可在 CCS 的编辑窗口看到 c 程序的源代码，代码中有以下四个部分：

- 系统初始化函数 InitSysCtrl();
- 在主函数输出消息“SineWave example started”之后，进入一个无限循环，在循环体内调用了两个函数 dataIO( )和 processing( )。
- 函数 dataIO( )在本实验中，DSP 不作任何实际操作而直接返回。
- 函数 processing( )对输入缓冲区的每个数据进行增益控制，并将结果存入输出缓冲区中。

6. 建立工程（Build 工程）

建立工程（build）是指对 asm、c 源程序文件进行编译（Compile)、汇编（Assemble)，并结合配置命令文件对工程进行链接（Link)，输出可执行程序（.out)。在主菜单选择“Project → Build Project”命令进行编译链接，生成的可执行.out 程序位于工程目录的 debug 子目录下。

对工程文件中的语法或是链接错误，CCS 会终止当前的 build，在底部消息窗口指示出程序包含的编译链接错误，或是警告信息。根据错误提示修改源程序文件或者配置命令文件，直至编译链接正确。

以上的工作称为目标代码生成。

7. 调试程序

当工程被正确建立以后，只有将程序通过仿真器下载到 DSP 芯片上，才能够进行实时的代码调试。

在“LAB_9”工程中双击“TMS320F28335.ccxml”，在弹出的“Basic”界面中“connection”选项中选择“SEED XDS510PLUS Emulator”，在“Board or Device”选项选择“TMS320F28335”后，点击右侧“Save Configuration”下的“Save”保存设置。

打开实验箱电源，在主菜单下选择“Run → Debug”，若仿真器正确连接后，进入“CCS Debug”界面。

8. 程序的运行

在CCS Debug环境界面的主菜单中选择“Run → Resume”可以让DSP从main函数的第一条语句开始执行程序。由于DSP程序输出并不具备GUI界面，由此执行结果只有依赖外部硬件或者查看寄存器、存储器的数值加以验证。在主菜单中选择“Run → Suspend”,，可以暂停程序的执行。DSP指令的执行严格按照指令流的顺序。

当想再次运行程序，可以执行菜单命令“Run → Restart”，使程序指针PC重新指向_c_int00，也可以重新加载程序（“Run → Load → Reload Program”）。当执行菜单命令“Run → Reset”时，DSP复位，内部寄存器恢复默认值，程序指针PC指向中断矢量表的复位向量处。

9. 程序的调试

在程序的开发与测试过程中，常常需要检查某个变量、或者是存储器的数值在程序运行过程中变化情况，这就需要暂停程序执行，用断点与观察窗口等方式来验证数值的正确性。这就是DSP目标代码的调试。

添加结构体变量currentBuffer到变量观察窗口（Add Watch Expression)，观察currentBuffer.output和currentBuffer.input的地址以及数值。添加dataIO( )到变量窗口，查看该子程序的入口地址。

在dataIO( )处设立断点，在断点属性中关联输入文件sine.dat，设置数据加载的起始地址为currentBuffer.input，长度为128。

鼠标移动到断点所在行，右键选择“Breakpoint Properties”，在“Action”选项中选择“Read Data from file”,在“File”选项中选择工程文件夹中的“sine.dat”文件，勾选“Wrap Around”选项为“true”，起始地址“Start Address”为currentBuffer.input的起始地址，数据长度为128，点击“OK”。

打开图形显示功能，在主菜单的“Tools → Graph → single time”查看存储空间currentBuffer.input和currentBuffer.output的时域波形。

查看存储空间的数值在程序相关语句执行前后的变化。

在processing( )子程序中设置断点，分别执行主菜单命令“Run → Step into”和“Run → Step over”单步执行程序，查看并比较这些单步执行方式的区别。

5.2 实验 11：DSP 数据采集

1. 设备检查

检查仿真器、F28335 DSP 教学实验箱、计算机之间的连接是否正确，打开计算机和实验箱电源。

2. 启动集成开发环境

点击桌面 CCS 5 快捷方式，进入集成开发环境 CCS。

3. 新建工程

新建一个 DSP 工程，编辑源程序、配置命令等相关文件，并在工程中添加这些程序文件。

在源程序中，通过对中断、ADC 外设以及事件管理通用时钟的设置，利用中断方式读取 ADC 的采样结果，并用 DAC 实现模拟信号的还原。在程序中，开辟一段数据空间，用于保存 ADC 的采样结果，要求保存 1024 点数据，且该空间的数据不断刷新。

源程序的编写可参照工程 LAB11 中的相关内容。

4. 建立工程（Build）

建立工程（build），若出错，则根据错误提示，修改源程序文件或者配置命令文件，直至编译链接正确，生成可执行的.out 文件。

5. 连接外部电路

打开信号源，产生一个合适的频率（ADC 的采样频率必须满足奈奎斯特采样定律），信号幅度控制在 0-3V 以内，验证后将信号通过接口输入到 DSP 中。

打开示波器，将实验箱中的 SMA 接口 J5 输出到示波器上，并正确设置。

6. 调试程序

在工程中合理配置 ccxml 文件，打开实验箱电源，在主菜单下选择“Run → Debug”，若仿真器正确连接后，进入“CCS Debug”调试界面。

首先验证中断设置是否正确。可以在 ADC 中断服务程序的入口地址处添加断点，全速或者动画运行程序，检查程序计数器 PC 能否间隔性的停留在中断服务入口地址处。若能，说明中断设置基本正确。

若以上步骤正确，其次，验证数据采集的正确性。程序连续运行一段时间后，暂停程序执行，打开图形显示功能，查看存储空间中保存的时域波形，是否为信号源输出的信号波形。

若上述步骤正确，则调节示波器，观察信号波形，是否为信号源的输入波形。若是，则实验调试结束。

以上步骤如果出错，则可以利用各种调试手段，比如打开寄存器窗口、变量窗口等辅助手段，根据数值以及实验原理，查找错误原因，重新修改程序，直至正确为止。

7. 运行程序

若第 6 步正确，可去掉断点，重新全速运行程序。

连接实验箱 SMA 输出口 J5 至示波器，调节示波器，观察信号的输出。可以实时的改变信号源的输入信号（注意信号幅度不要随意修改，超出输入范围易烧毁实验电路），示波器上显示的波形亦会随之变化。

数据直通通道就是最简单的实时信号处理电路。

6 实验结果

1.将信号源输出端接至实验箱 SMA 端口 J2，将实验箱 SMA 端口 J5 连接至示波器。

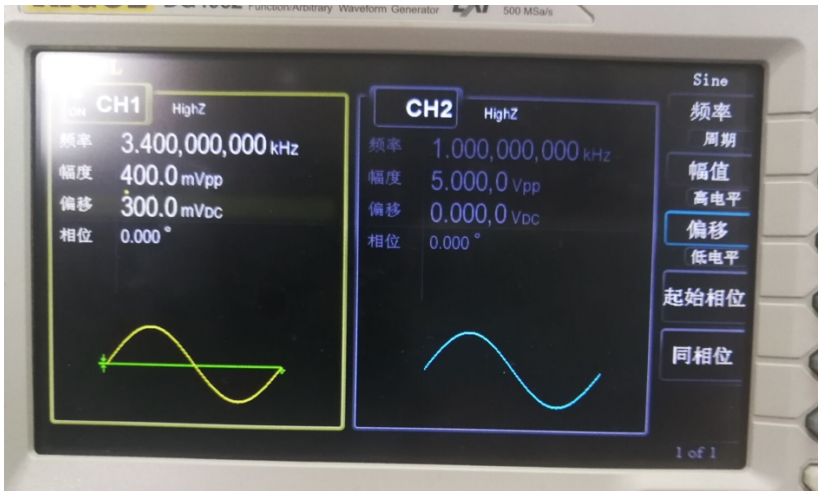
2.打开示波器和信号发生器，调节信号发生器的输出，控制幅度峰峰值在 1V 左右；

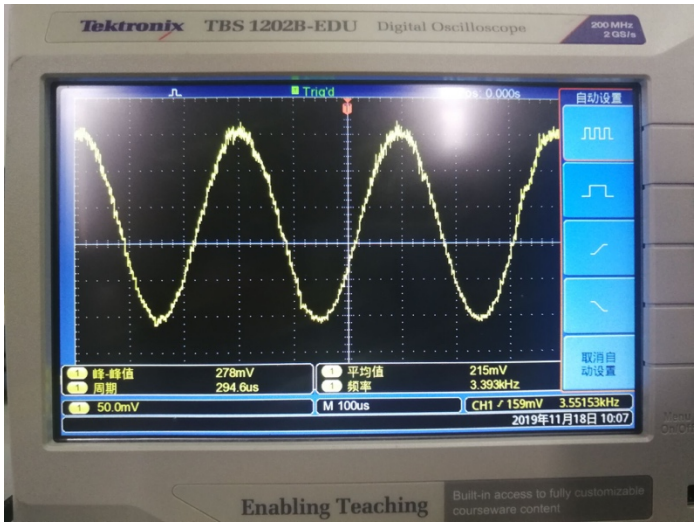
3.打开实验箱电源，检查实验箱电源指示灯是否正常指示；

4.通过仿真器将实验箱与 PC 机连在一起，点击 PC 机上的 CCS5 配置程序，配置完成后成功打开 F28335 集成开发环境；

5.创建工程，导入测试文件后重新编译生成.out 文件，加载到 DSP 中并全速运行，检查实验箱上示波器波形等；

6.最终观察到示波器上的波形和信号发生器产生的波形一致，由此判断实验箱正常工作，可以进行接下来的实验。





6.1 实验 9 ： DSP 开发基础（C 程序基础调试）

（1）dataIO()、processing()子程序的入口地址

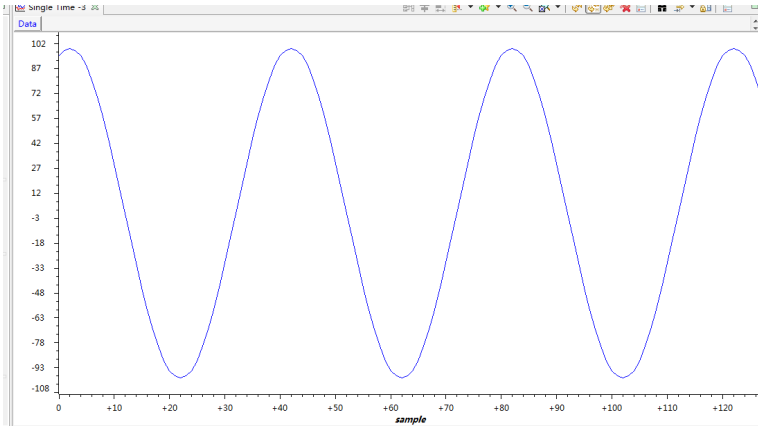
dataIO	void (*)()	0x00AF87	
processing	void (*)()	0x00AF6A	

（2）currentBuffer.input和currentBuffer.output所在存储器地址

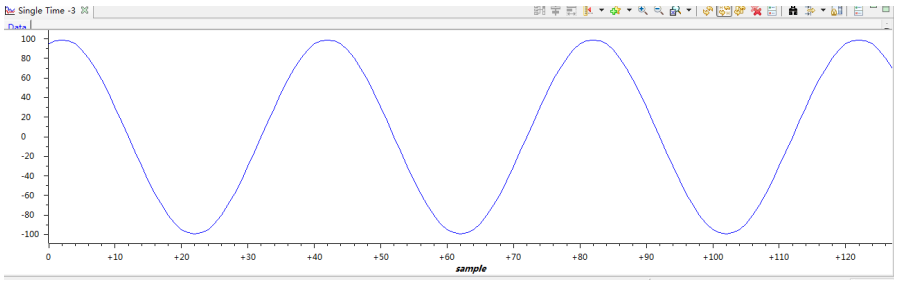
Expression	Type	Value	Address
currentBuffer.input	int[128]	0x0000C1C0@Data	0x0000C1C0@Data
currentBuffer.output	int[128]	0x0000C240@Data	0x0000C240@Data

（3）显示数据空间currentBuffer.input和currentBuffer.output缓冲存储器中的波形

currentBuffer.input:



currentBuffer.output:



（4）.map文件中查看所有的段在存储空间的地址、长度和含义，指出分别位于TMS320F28335的什么存储空间以及物理存储块名称，主程序中所用的变量分别属于什么段

```
113 .text      0    00009000    000022c0
114                   00009000    00000bd9     rts2800_fpu32.lib : _printfi.obj (.text)
115                   00009bd9    00000323     DSP2833x_DefaultIsr.obj (.text:retain)
116                   00009efc    0000027c     rts2800_fpu32.lib : trgdrv.obj (.text)
117                   0000a178    0000021b                       : lowlev.obj (.text)
118                   0000a393    000001d2                       : memory.obj (.text)
119                   0000a565    00000107                       : ll_div.obj (.text)
120                   0000a66c    000000f9                       : fopen.obj (.text)
121                   0000a765    000000f8     DSP2833x_SysCtrl.obj (.text)
122                   0000a85d    000000bc     rts2800_fpu32.lib : s_scalbn.obj (.text)
123                   0000a919    000000b0                       : fputs.obj (.text)
124                   0000a9c9    0000009c                       : fd_add.obj (.text)
125                   0000aa65    00000001                       : startup.obj (.text)
126                   0000aa66    00000094                       : trgmsg.obj (.text)
127                   0000aafa    0000008b                       : fd_div.obj (.text)
128                   0000ab85    00000089                       : setvbuf.obj (.text)
129                   0000ac0e    00000083                       : fd_mpy.obj (.text)
130                   0000ac91    00000073                       : fflush.obj (.text)
131                   0000ad04    00000067                       : _io_perm.obj (.text)
132                   0000ad6b    0000005f                       : fclose.obj (.text)
133                   0000adca    0000005b                       : s_frexp.obj (.text)
134                   0000ae25    00000056                       : boot.obj (.text)
135                   0000ae7b    00000053                       : fputc.obj (.text)
136                   0000aece    00000046                       : cpy_tbl.obj (.text)
137                   0000af14    0000003d                       : fseek.obj (.text)
138                   0000af51    00000037     main.obj (.text)
139                   0000af88    00000034     rts2800_fpu32.lib : printf.obj (.text)
```

段名称	Page	段首地址	段长度	含义	物理存储块
.text	0	00009000	000022c0	代码，片外存储空间的数据存储空间	Flash
.ebss	1	0000c000	00000366	内部存储空间数据存储空间	RAM
.stack	1	0000e000	00000300	堆栈空间	低 64K 字的 RAM
.cinit	0	00008000	0000009c	全局与静态变量的初始值	Flash
.econst	1	0000d000	00000284	常数	Flash
.pinit	0	00008000	00000000	全局结构函数表	Flash
.esysmem	1	00000400	00000400	Farmalloc 函数的存储空间	RAM

（5）.cmd命令文件，比较其与上述.map中的映射关系。修改.cmd文件后，再次编译链接，查看配置命令与各段的映射关系

Page0中对寄存器变量RAMLX的首地址和长度得定义列表：

```
PAGE 0 :
   /* BEGIN is used for the "boot to SARAM" bootloader mode      */

   BEGIN      : origin = 0x000000, length = 0x000002     /* Boot to M0 will go here                    */
   RAMM0      : origin = 0x000050, length = 0x0003B0
   RAML0      : origin = 0x008000, length = 0x001000
   RAML1      : origin = 0x009000, length = 0x003000
   //RAML2      : origin = 0x00A000, length = 0x001000
  // RAML3      : origin = 0x00B000, length = 0x001000
   ZONE7A     : origin = 0x200000, length = 0x00FC00    /* XINTF zone 7 - program space */
   CSM_RSVD   : origin = 0x33FF80, length = 0x000076     /* Part of FLASHA.  Program with all 0x0000 when
   CSM_PWL    : origin = 0x33FFF8, length = 0x000008     /* Part of FLASHA.  CSM password locations in FLA
   ADC_CAL    : origin = 0x380080, length = 0x000009
   RESET      : origin = 0x3FFFC0, length = 0x000002
   IQTABLES   : origin = 0x3FE000, length = 0x000b50
   IQTABLES2  : origin = 0x3FEB50, length = 0x00008c
   FPUTABLES  : origin = 0x3FEBDC, length = 0x0006A0
   BOOTROM    : origin = 0x3FF27C, length = 0x000D44
```

Page1中对寄存器变量RAMLX的首地址和长度得定义列表：

```
PAGE 1 :
   /* BOOT_RSVD is used by the boot ROM for stack.               */
   /* This section is only reserved to keep the BOOT ROM from    */
   /* corrupting this area during the debug process              */

   BOOT_RSVD  : origin = 0x000002, length = 0x00004E     /* Part of M0, BOOT rom will use this for stack
   RAMM1      : origin = 0x000400, length = 0x000400     /* on-chip RAM block M1 */

   RAML4      : origin = 0x00C000, length = 0x001000
   RAML5      : origin = 0x00D000, length = 0x001000
   RAML6      : origin = 0x00E000, length = 0x001000
   RAML7      : origin = 0x00F000, length = 0x001000
   ZONE7B     : origin = 0x20FC00, length = 0x000400     /* XINTF zone 7 - data space */
}
```

每个段得地址映射：

```
SECTIONS
{
   /* Setup for "boot to SARAM" mode:
      The codestart section (found in DSP28_CodeStartBranch.asm)
      re-directs execution to the start of user code.  */
   codestart        : > BEGIN,     PAGE = 0
   ramfuncs         : > RAML0,     PAGE = 0
   .text            : > RAML1,     PAGE = 0
   .cinit           : > RAML0,     PAGE = 0
   .pinit           : > RAML0,     PAGE = 0
   .switch          : > RAML0,     PAGE = 0

   //.stack           : > RAMM1,     PAGE = 1
   .stack           : > RAML6,     PAGE = 1
   .ebss            : > RAML4,     PAGE = 1
   .econst          : > RAML5,     PAGE = 1
   .esysmem         : > RAMM1,     PAGE = 1

   IQmath           : > RAML1,     PAGE = 0
   IQmathTables     : > IQTABLES,  PAGE = 0, TYPE = NOLOAD
```

每个段得映射寄存器地址与.map文件中得寄存器相同。

在.cmd文文件中修改相应段的起始地址等信息，重新编译、链接，可以在.map文件中发现相应的变化：

对BEGIN映射的寄存器得首地址进行更改：

更改前：

.cmd文件

```
   /* BEGIN is used for the "boot to SARAM" bootloader mode      */

   BEGIN      : origin = 0x000000, length = 0x000002     /* Boot to M0 will go here
   RAMM0      : origin = 0x000050, length = 0x0003B0
   RAML0      : origin = 0x008000, length = 0x001000
```

.map文件

```
1
2         name             origin    length      used     unused   attr    fill
3----------------------   --------  ---------  --------  --------  ----  --------
4PAGE 0:
5   BEGIN                 00000000   00000002  00000002  00000000  RWIX
```

更改后：

.cmd文件

```
PAGE 0 :
   /* BEGIN is used for the "boot to SARAM" bootloader mode      */

   BEGIN      : origin = 0x000001, length = 0x000002     /* Boot to M0 will go her
   RAMM0      : origin = 0x000050, length = 0x0003B0
   RAML0      : origin = 0x008000, length = 0x001000
   //RAML1      : origin = 0x009000, length = 0x003000
```

.map文件

```
--
12         name             origin    length      used     unused   attr    fill
13----------------------   --------  ---------  --------  --------  ----  --------
14PAGE 0:
15   BEGIN                 00000001   00000002  00000002  00000000  RWIX
16   RAMM0                 00000050   000003b0  00000000  000003b0  RWIX
17   RAML0                 00008000   00001000  000000a0  00000f60  RWIX
```

6.2 实验 11：DSP 数据采集（实验箱测试）

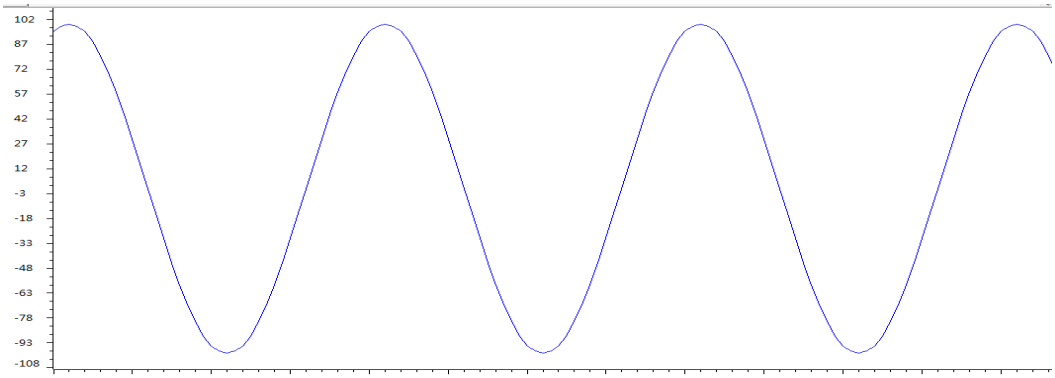
（1）根据范例程序，给出ADC的采样频率计算公式，修改ADC的采样频率，并验证

采样频率：T(PWM1)=TBCLK/(TBPRD*2*3)=25/(208*3*2)=0.02MHz=20KHz

（2）指出波形数据保存的空间地址，并显示采集的信号波形

SampleTable1	unsigned int[1024]	0x0000C040@Data	0x0000C040@Data
Da_out	unsigned int *	0x00200400	0x0000C004@Data
xn	int	-29328	0x0000C003@Data

SampleTable1为储存采样样本点得储存空间
Da_out是输出至示波器得临时储存空间
xn为接收样本点数据得临时储存空间



（3）利用上述图形，给出采样频率的验证方法，以此检验数据采集程序的正确性。

将输出端连接到示波器上，调节好波形，观察波形每个周期内的取点个数，一般来说，取多少个点就意味着多少K的采样频率。

7 实验总结

7.1 实验思考

1. 观察输入信号与示波器显示信号、存储器中存储波形信号幅度的差异，解释差异产生的原因。

（1）在运行程序中，当通过graph窗口观察程序时，断点会使得AD采样传送到存储空间的信号暂时不显示到模拟图形中，而此时外界输入信号依旧在变化，使得输入信号与存储器中存储波形信号产生一定的差异。
（2）*DA_out= (unsigned int)((*(RamAddr+1*x))<<4) + 0x8000该输出程序中，若是对RamAddr的算法处理不是最优化的也会使得数据有所失真。

3. 除了中断方式，DSP内核还可以采用查询方式获取ADC外设的采样数据。如果采样查询方式，则需要查询哪些标志位，给出程序流程并编程实现。

```
#include "DSP2833x_Device.h"     // DSP2833x Headerfile Include File
#include "DSP2833x_Examples.h"   // DSP2833x Examples Include File

Uint16 ConversionCount;
Uint16 Voltage1[10];
Uint16 Voltage2[10];

main()
{
```

```
    InitSysCtrl();

    DINT;
    InitPieCtrl();
    IER = 0x0000;
    IFR = 0x0000;
    InitPieVectTable();

    InitAdc();

    ConversionCount = 0;

    EALLOW;
    AdcRegs.ADCCTL1.bit.INTPULSEPOS = 1;    //转换完成前一个ADC时钟周期产生EOC
    AdcRegs.INTSEL1N2.bit.INT1E     = 1;    //使能ADCINT1
    AdcRegs.INTSEL1N2.bit.INT1CONT  = 0;    //关闭连续模式
    AdcRegs.INTSEL1N2.bit.INT1SEL   = 1;    //将ADCINT1映射到EOC1
    AdcRegs.ADCSOC0CTL.bit.CHSEL    = 0;    //将ADCINA0映射到通道0
    AdcRegs.ADCSOC1CTL.bit.CHSEL    = 1;    //将ADCINA1映射到通道1
    AdcRegs.ADCSOC0CTL.bit.TRIGSEL  = 0;    //软件触发SOC0
    AdcRegs.ADCSOC1CTL.bit.TRIGSEL  = 0;    //软件触发SOC1
    AdcRegs.ADCSOC0CTL.bit.ACQPS    = 6;    //设置窗口采样次数
    AdcRegs.ADCSOC1CTL.bit.ACQPS    = 6;    //设置窗口采样次数
    EDIS;

    AdcRegs.ADCSOCFRC1.all = 0x0003;//强制给通道0和1产生SOC信号

   for(;;)
   {
      while(AdcRegs.ADCINTFLG.bit.ADCINT1 == 0){}     //等待EOC1信号（ADCINT1）
      AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;          //清除EOC1信号（ADCINT1）

      AdcRegs.ADCSOCFRC1.all = 0x0003;//强制给通道0和1产生SOC信号

      if(ConversionCount == 9)
      {
          ConversionCount = 0;
      }
      else ConversionCount++;

      Voltage1[ConversionCount] = AdcResult.ADCRESULT0;
      Voltage2[ConversionCount] = AdcResult.ADCRESULT1;
   }
}
```

4. 如何将存储的采样数据保存到数据文件中，并利用动态有效位 ENOB 测试方法分析实验平台数据采集的性能。

保存数据的思路：

（1）运行软件 cybulk.exe

（2）选择 DSP 板与 PC 连接得 USB 端口

（3）DSP 发送数据、软件接收数据并转化成数据包.dat 文件

（4）编写 MATLAB 程序验证数据的正确性

附 DSP 板发送 USB 串口 main.c 程序：

```
#include "DSP2833x_Device.h"     // DSP2833x Headerfile Include File
#include "DSP2833x_Examples.h"   // DSP2833x Examples Include File

#include "leds.h"
#include "time.h"
#include "uart.h"
#include "rs485.h"
/*********************************************************************************
*
* 函 数 名         : main
* 函数功能         : 主函数
* 输    入         : 无
* 输    出         : 无
*********************************************************************************
/
void main()
{
    Uint16 ReceivedChar;
    char *msg;

    InitSysCtrl();
    InitPieCtrl();
    IER = 0x0000;
    IFR = 0x0000;
    InitPieVectTable();

    LED_Init();
    TIM0_Init(150,200000);//200ms
    RS485_Init(4800);
    RS485_DIR_SETH;
    DELAY_US(5);
    msg =  "\r\n*******welcome to prechin**********\0";
    RS485_SendString(msg);
```

```
    while(1)
    {
        msg = "\r\nEnter a character: \0";
        RS485_SendString(msg);
        DELAY_US(2);
        RS485_DIR_SETL;
        ScibRegs.SCICTL1.bit.SWRESET=0;
        DELAY_US(2);
        ScibRegs.SCICTL1.bit.SWRESET=1;
        // Wait for inc character
        while(ScibRegs.SCIRXST.bit.RXRDY !=1); // wait for XRDY =1 for empty
state
        // Get character
        ReceivedChar = ScibRegs.SCIRXBUF.all;
        RS485_DIR_SETH;
        DELAY_US(5);
        // Echo character back
        msg = "you enter is:\0";
        RS485_SendString(msg);
        RS485_SendByte(ReceivedChar);
    }
}
```

$$SINAD = 10log_{10}[\frac{\text{基频信号能量}}{\text{噪声能量}}]$$

$$ENOB = \frac{SINAD - 1.76}{6.02}$$

7.2 实验中遇到的问题与解决方案

（1）图形工具画出的波形错误

使用 CCS 中的图形工具，绘制出的图像波形前没有数据，波形后有杂乱的波形，经过研究发现是在绘制图象时将 16 位的数据误认为 32 位的数据，从而导致了图像错误，最终将图像数据选择为 16 位符号数，即绘制出了正确的图像。

（2）不同版本的CCS对工程编译不兼容

编译过的低版本的CCS工程在高版本的编译器中打开会无法进行Debug，于是将Debug文件夹和.project文件删除后重新启动CCS编译器就能对工程进行编译。

（3）无法从在工程中添加文件

为了测试实验箱完整性，需要添加外部文件，但是在添加时缺提示报错，后猜测由于路径中有中文名字导致无法添加，修改了文件路径后可以加入文件。

7.3 实验总结

通过这次DSP实验我熟悉DSP的软硬件开发平台，掌握TMS320F28335的ADC外设的使用，熟悉TMS320F28335的中断的设置，掌握代码调试的基本方法。通过数码管显示实验，我学会了建立、编译程序，并生成.out文件，把程序加载到DSP芯片上。在信号采集实验中，我学会了通过调节信号源的频率，来实时观察示波器上的输出信号。

后来在软件验证ADC的过程中，是通过改变EvaRegs.T1CON.bit.TPS的值，来改变采样频率的，并且通过CCS的图形显示功能显示其中存储的波形。而后的硬件验证ADC采样频率实验中，我们采取的方法是在中断服务程序开始时，输出高电平；在中断服务程序结束时，输出低电平，这样可以通过观察两次高电平的时间间隔，便可得到采样频率。

通过这次的DSP实验，让我对DSP开发中的软件和硬件有了大概的了解，任何事物的学习都是由浅入深，相信通过后期的学习和实验，自己在编程、程序调试和硬件测试方面的能力会进一步提升，并且能够独立地完成工程的建立、程序的建立、编译和调试。同时由于本次的实验是三人合作完成，因此通过这次实验，也加强了三人之间的默契程度和三人之间的合作能力。

这次的DSP实验我对DSP开发有了一定的认识，希望通过以后的DSP的实验，自己的能力也能够得到进一步提升，希望能在这条道路上越走越远。