南京理工大学
NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

DSP 应用技术实验

DSP 开发基础实验报告

作　　者：	许晓明	学　　号：	9161040G0734
同 组 人：	李玥	学　　号：	9161040G0703
同 组 人：	陈锦涛	学　　号：	9161040G0614
学　　院：	电子工程与光电技术学院		
专　　业：	电子信息工程		
班　　级：	电信 3 班		
组　　号：	B4		
题　　目：	DSP 应用技术实验		
	DSP 开发基础实验报告		
指 导 者：	李彧晟		

2019 年 11 月

目录

1 实验目的

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。

2 实验仪器

2.1 实验仪器清单

1. DSP 仿真平台（仿真器、DSP 实验箱、计算机） 一套
2. 信号发生器 一台
3. 示波器 一台

2.2 硬件连接示意图

实验硬件连接大致如图 2.1 所示。其中，测试完实验箱后，实际上信号发生器与示波器可以不在与实验箱连接。

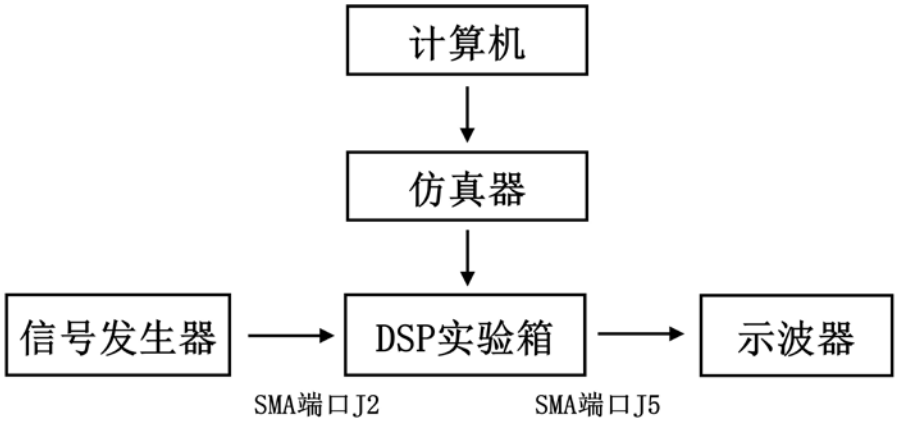


图 2.1 硬件连接示意图

3 实验步骤及现象

3.1 实验箱测试

1.开启示波器、信号发生器，调节信号发生器输出幅度在 0-1V 以内。

2.连接实验设备，确认无误后开启实验箱电源，此时可观察到实验箱电源指示灯亮起。

3.在计算机上通过 CCS 5 创建工程并导入 LAB11 工程。

4. 对项目工程进行编译、链接（Build Project），进入调试（Debug）界面并运行程序（Resume）。

5.可观察到示波器波形与信号发生器一致。改变信号发生器波形及参数，示波器上波形也随之改变，如图 3. 1 所示，说明实验箱正常。

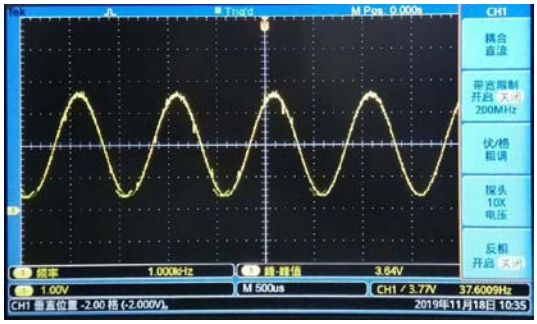


图 3. 1 测试实验箱是否正常

3.2 C 程序基础调试

1.通过 CCS 5 新建工程并设置工程参数信息。

2.添加 LAB9 相关工程文件，为工程添加搜索路径、库文件。

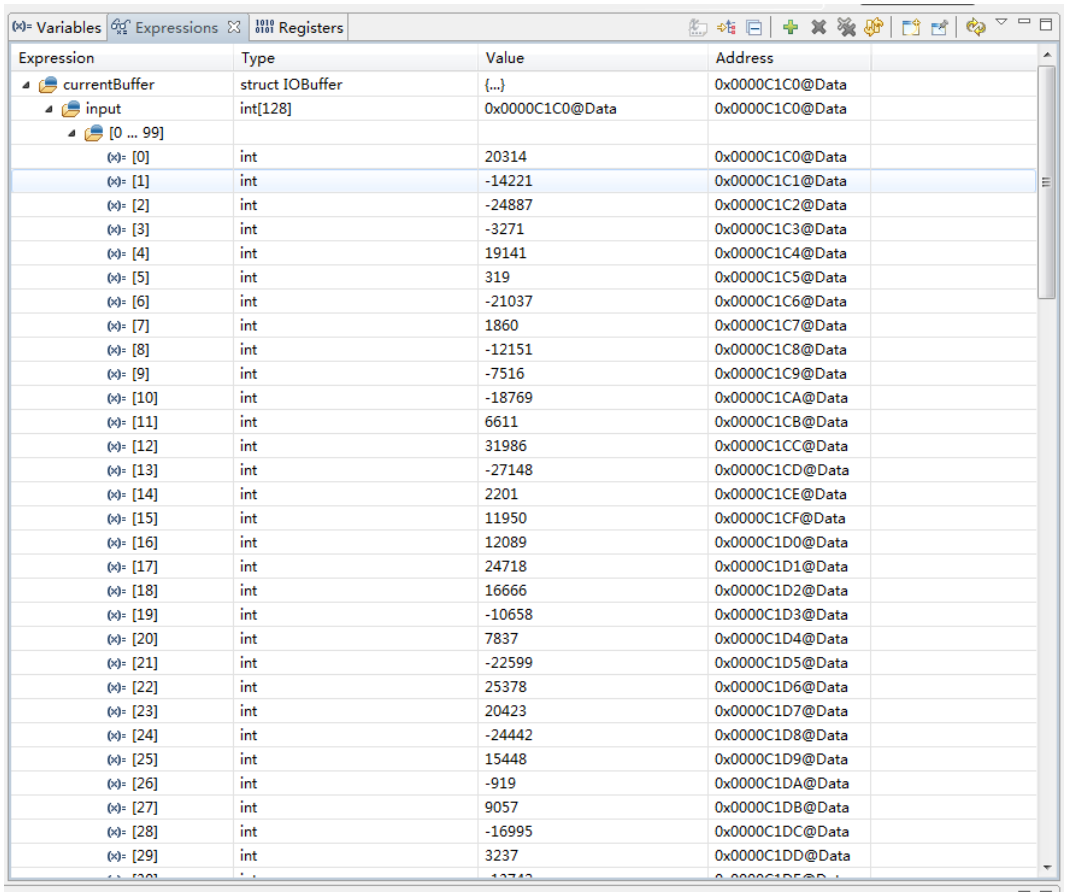
3.对项目工程进行编译、链接（Build Project），进入调试（Debug）界面并运行程序（Resume），此时观察到屏幕上出现“SineWave example started”字样。

4.添加结构体变量 currentBuffer 到变量观察窗口，如图 3. 2 到图 3. 3 所示，可观察到 **currentBuffer.input 和 currentBuffer.output 所在存储器地址分别为 0x0000C1C0 和 0x0000C240**。

5. 添加子程序 dataIO()、子程序 processing()到变量观察窗口，如图 3. 4 所示，可观察到 **dataIO()入口地址为 0x00B6DA；processing()入口地址为 0x00B6BD**。

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

7.重新运行程序，如图 3. 5 所示，可观察到存储空间 currentBuffer.input 和 currentBuffer.output 中的数值发生变化。



Variables | Expressions | Registers

Expression	Type	Value	Address
currentBuffer	struct IOBuffer	{...}	0x0000C1C0@Data
input	int[128]	0x0000C1C0@Data	0x0000C1C0@Data
[0 ... 99]			
[0]	int	20314	0x0000C1C0@Data
[1]	int	-14221	0x0000C1C1@Data
[2]	int	-24887	0x0000C1C2@Data
[3]	int	-3271	0x0000C1C3@Data
[4]	int	19141	0x0000C1C4@Data
[5]	int	319	0x0000C1C5@Data
[6]	int	-21037	0x0000C1C6@Data
[7]	int	1860	0x0000C1C7@Data
[8]	int	-12151	0x0000C1C8@Data
[9]	int	-7516	0x0000C1C9@Data
[10]	int	-18769	0x0000C1CA@Data
[11]	int	6611	0x0000C1CB@Data
[12]	int	31986	0x0000C1CC@Data
[13]	int	-27148	0x0000C1CD@Data
[14]	int	2201	0x0000C1CE@Data
[15]	int	11950	0x0000C1CF@Data
[16]	int	12089	0x0000C1D0@Data
[17]	int	24718	0x0000C1D1@Data
[18]	int	16666	0x0000C1D2@Data
[19]	int	-10658	0x0000C1D3@Data
[20]	int	7837	0x0000C1D4@Data
[21]	int	-22599	0x0000C1D5@Data
[22]	int	25378	0x0000C1D6@Data
[23]	int	20423	0x0000C1D7@Data
[24]	int	-24442	0x0000C1D8@Data
[25]	int	15448	0x0000C1D9@Data
[26]	int	-919	0x0000C1DA@Data
[27]	int	9057	0x0000C1DB@Data
[28]	int	-16995	0x0000C1DC@Data
[29]	int	3237	0x0000C1DD@Data

图 3. 2 currentBuffer.input 地址及部分数值

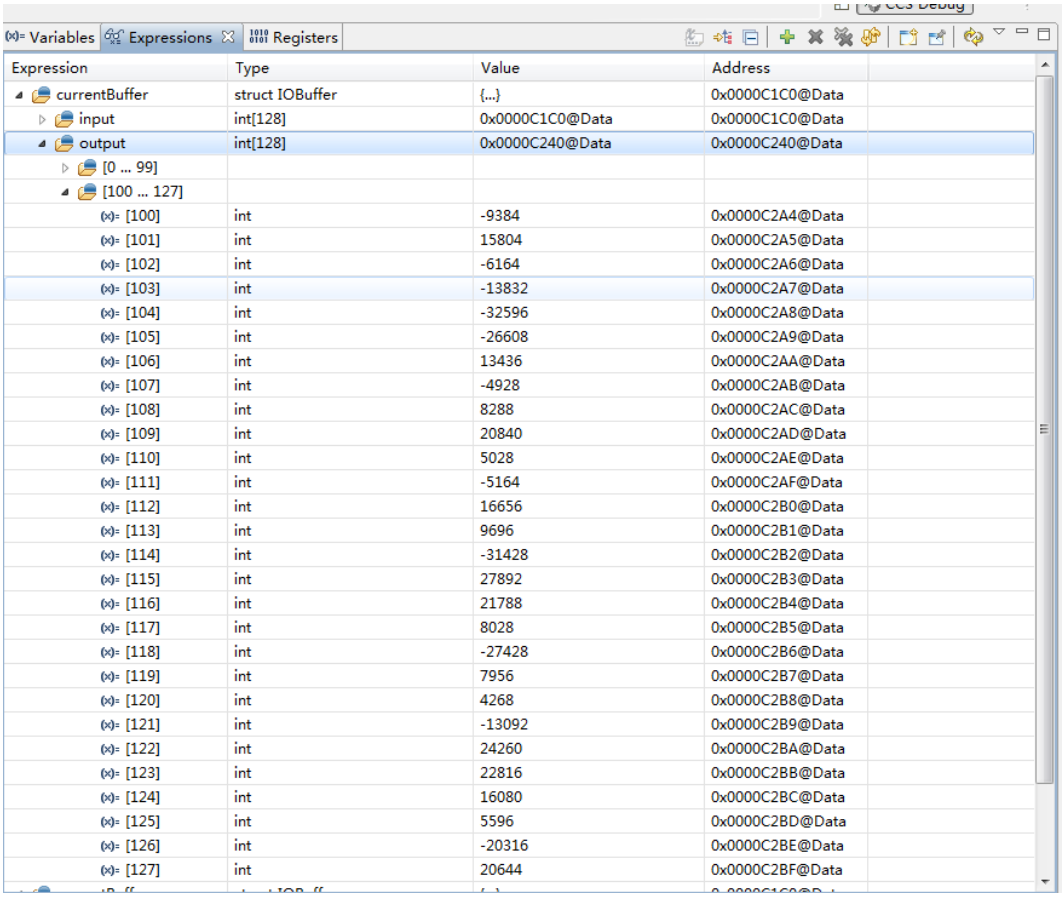


图 3. 3 currentBuffer.output 地址及部分数值

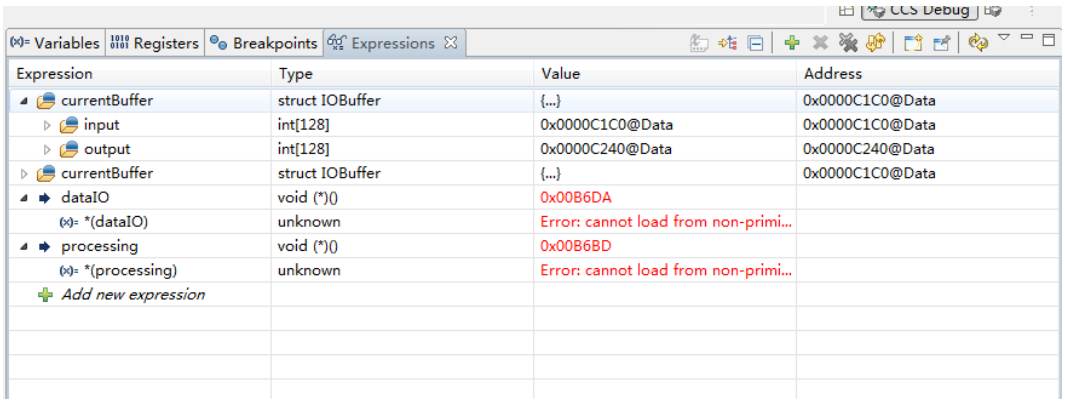
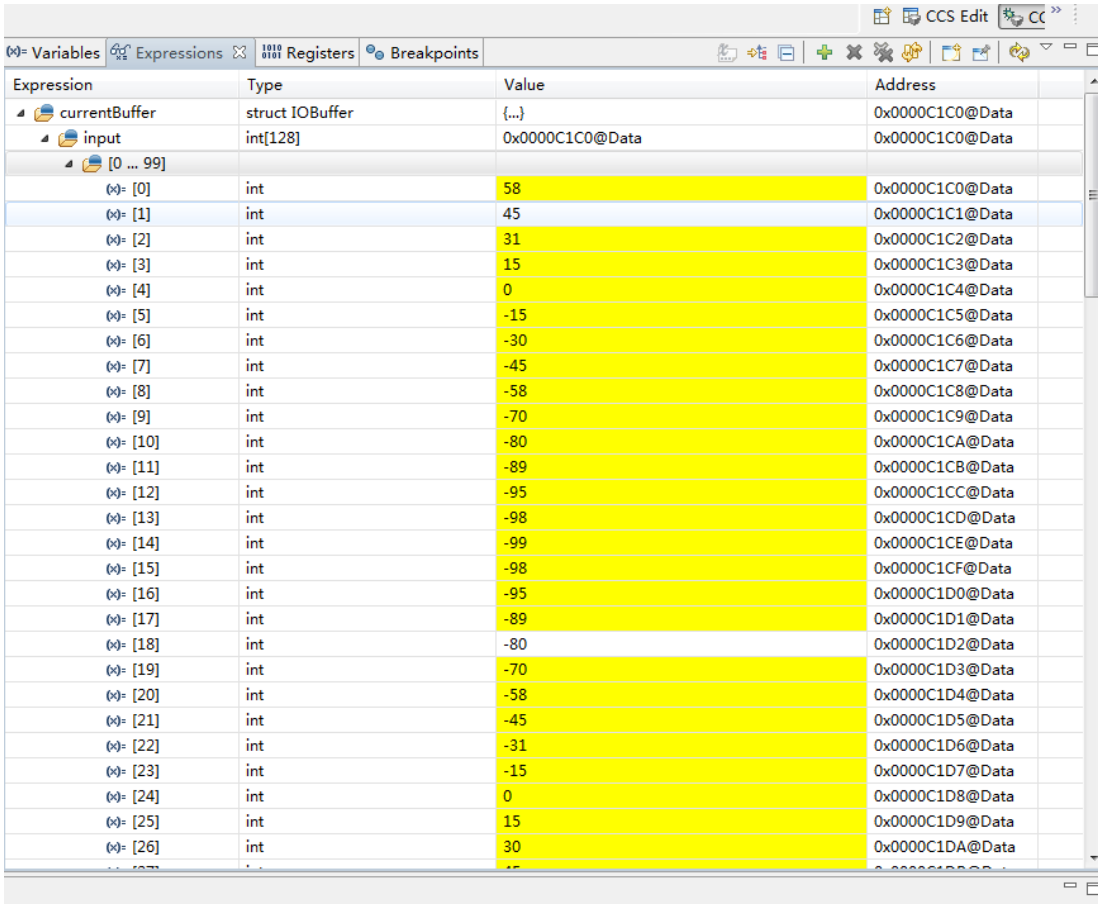


图 3. 4 子程序入口地址



Expression	Type	Value	Address
currentBuffer	struct IOBuffer	{...}	0x0000C1C0@Data
input	int[128]	0x0000C1C0@Data	0x0000C1C0@Data
[0 ... 99]			
[0]	int	58	0x0000C1C0@Data
[1]	int	45	0x0000C1C1@Data
[2]	int	31	0x0000C1C2@Data
[3]	int	15	0x0000C1C3@Data
[4]	int	0	0x0000C1C4@Data
[5]	int	-15	0x0000C1C5@Data
[6]	int	-30	0x0000C1C6@Data
[7]	int	-45	0x0000C1C7@Data
[8]	int	-58	0x0000C1C8@Data
[9]	int	-70	0x0000C1C9@Data
[10]	int	-80	0x0000C1CA@Data
[11]	int	-89	0x0000C1CB@Data
[12]	int	-95	0x0000C1CC@Data
[13]	int	-98	0x0000C1CD@Data
[14]	int	-99	0x0000C1CE@Data
[15]	int	-98	0x0000C1CF@Data
[16]	int	-95	0x0000C1D0@Data
[17]	int	-89	0x0000C1D1@Data
[18]	int	-80	0x0000C1D2@Data
[19]	int	-70	0x0000C1D3@Data
[20]	int	-58	0x0000C1D4@Data
[21]	int	-45	0x0000C1D5@Data
[22]	int	-31	0x0000C1D6@Data
[23]	int	-15	0x0000C1D7@Data
[24]	int	0	0x0000C1D8@Data
[25]	int	15	0x0000C1D9@Data
[26]	int	30	0x0000C1DA@Data

图 3.5 设置断点并关联文件后的程序运行结果

8.通过图形显示功能，查看存储空间 currentBuffer.input 和 currentBuffer.output 的时域波形，如图 3.6 到图 3.7 所示，可知通过增益子程序 processing()后，**output 的波形幅度比 input 增大 4 倍**。同时观察到二者的波形反相，查看头文件 sine.h ，可发现以下代码：

```
19  // buffer constants
20  #define BUFFSIZE 128
21  #define INITIALGAIN -4
```

结合主程序 main.c 中的如下代码：

```
27  // gain control variable
28  int gain = INITIALGAIN;
```

```
72      int size = BUFFSIZE;
73
74      while(size--){
75          currentBuffer.output[size] = currentBuffer.input[size] * gain;
76      }
```

可知：这是**由于 processing()对 input 中的数据乘上了负增益**而产生的结果。

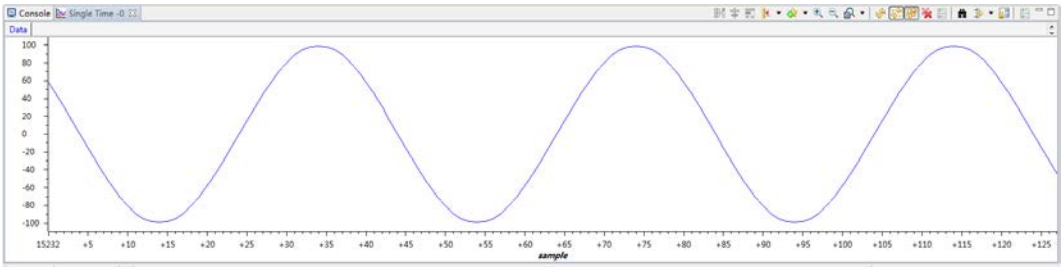


图 3. 6 设置断点并关联文件后 currentBuffer.input 的时域波形

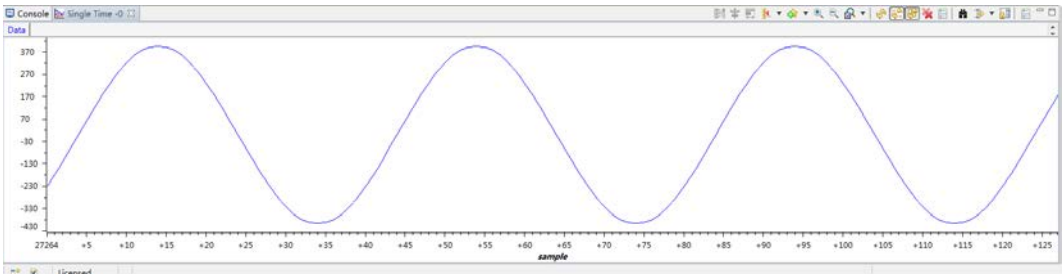


图 3. 7 设置断点并关联文件后 currentBuffer.output 的时域波形

9. 在 processing()子程序中设置断点，分别执行 “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。

10. 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。.map 文件的部分内容如图 3. 8 所示。

11. 查看.cmd 命令文件，部分内容如图 3. 9 所示，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。

```
12          name            origin    length      used     unused   attr    fill
13  ----------------------  --------  ---------  --------  --------  ----  --------
14  PAGE 0:
15    BEGIN                 00000000  00000002  00000002  00000000  RWIX
16    RAMM0                 00000050  000003b0  00000000  000003b0  RWIX
17    RAML0                 00008000  00001000  000000af  00000f51  RWIX
18    RAML1                 00009000  00003000  00002a2e  000005d2  RWIX
19    ZONE7A                00200000  0000fc00  00000000  0000fc00  RWIX
20    CSM_RSVD              0033ff80  00000076  00000000  00000076  RWIX
21    CSM_PWL               0033fff8  00000008  00000000  00000008  RWIX
22    ADC_CAL               00380080  00000009  00000007  00000002  RWIX
23    IQTABLES              003fe000  00000b50  00000000  00000b50  RWIX
24    IQTABLES2             003feb50  0000008c  00000000  0000008c  RWIX
25    FPUTABLES             003febdc  000006a0  00000000  000006a0  RWIX
26    BOOTROM               003ff27c  00000d44  00000000  00000d44  RWIX
27    RESET                 003fffc0  00000002  00000000  00000002  RWIX
88  SECTION ALLOCATION MAP
89
90   output                                  attributes/
91  section   page    origin      length       input sections
92  --------  ----  ----------  ----------   ----------------
93  codestart
94  *           0    00000000    00000002
95                     00000000    00000002     DSP2833x_CodeStartBranch.obj (codestart)
96
97  .pinit      0    00008000    00000000     UNINITIALIZED
98
99  .cinit      0    00008000    00000090
100                    00008000    0000002d     rts2800_fpu32.lib : lowlev.obj (.cinit)
101                    0000802d    0000002a                       : defs.obj (.cinit)
102                    00008057    00000017     DSP2833x_Lcd.obj (.cinit)
103                    0000806e    0000000a     rts2800_fpu32.lib : _lock.obj (.cinit)
104                    00008078    0000000a                       : exit.obj (.cinit)
105                    00008082    00000004     main.obj (.cinit)
106                    00008086    00000004     rts2800_fpu32.lib : fopen.obj (.cinit)
107                    0000808a    00000004                       : memory.obj (.cinit)
108                    0000808e    00000002     --HOLE-- [fill = 0]
109
110 ramfuncs    0    00008090    0000001f
111                    00008090    0000001b     DSP2833x_SysCtrl.obj (ramfuncs)
112                    000080ab    00000004     DSP2833x_usDelay.obj (ramfuncs)
113
114 .text       0    00009000    00002a2e
115                    00009000    00000911     rts2800_fpu32.lib : _printfi.obj (.text)
193 .reset      0    003fffc0    00000002     DSECT
194                    003fffc0    00000002     rts2800_fpu32.lib : boot.obj (.reset)
195
196 .sysmem     1    00000400    00000400     UNINITIALIZED
197                    00000400    00000001     rts2800_fpu32.lib : memory.obj (.sysmem)
198                    00000401    000003ff     --HOLE--
199

392 .ebss       1    0000c000    00000363     UNINITIALIZED
393                    0000c000    00000160     rts2800_fpu32.lib : defs.obj (.ebss)
394                    0000c160    00000018     DSP2833x_CpuTimers.obj (.ebss)
395                    0000c178    00000008     rts2800_fpu32.lib : memory.obj (.ebss)
396                    0000c180    00000140     main.obj (.ebss)
397                    0000c2c0    00000088     rts2800_fpu32.lib : lowlev.obj (.ebss)
398                    0000c348    0000000a     DSP2833x_Lcd.obj (.ebss)
399                    0000c352    00000008     rts2800_fpu32.lib : trgdrv.obj (.ebss)
400                    0000c35a    00000004                       : _lock.obj (.ebss)
401                    0000c35e    00000004                       : exit.obj (.ebss)
402                    0000c362    00000001                       : fopen.obj (.ebss)
403
404 .cio        1    0000c380    00000120     UNINITIALIZED
405                    0000c380    00000120     rts2800_fpu32.lib : ankmsg.obj (.cio)
406
407 .econst     1    0000d000    0000025a
408                    0000d000    00000101     rts2800_fpu32.lib : ctype.obj (.econst:__ctypes_)
409                    0000d101    00000001     --HOLE-- [fill = 0]
410                    0000d102    00000100     DSP2833x_PieVect.obj (.econst)
411                    0000d202    00000024     rts2800_fpu32.lib : _printfi.obj (.econst:.string)
412                    0000d226    0000001a     main.obj (.econst:.string)
413                    0000d240    00000018     rts2800_fpu32.lib : _printfi.obj (.econst)
414                    0000d258    00000002                       : fputs.obj (.econst:.string)
415
416 .stack      1    0000e000    00000300     UNINITIALIZED
417                    0000e000    00000300     --HOLE--
```

图 3. 8 .map 文件部分内容

```
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 a
   CSM_PWL    : origin = 0x33FFF8, length = 0x000008     /* Part of FLASHA.  CSM password l
   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

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
   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
```

```
SECTIONS
{
   /* Setup for "boot to SARAM" mode:
      The codestart section (found in DSP28_CodeStartBranch
      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
```

图 3. 9 .cmd 文件部分内容

4 实验结果汇总及问题回答

4.1 子程序入口地址与结构体存储地址

如图 3. 2 到图 3. 4 所示，可汇总得到表 4. 1。

表 4. 1 子程序入口地址与结构体存储地址

名称	地址
dataIO()子程序	0x00B6DA
Procession()子程序	0x00B6BD
currentBuffer.input	0x0000C1C0

currentBuffer.output	0x0000C240

4.2 显示缓冲存储器中的波形

如图 3. 6 到图 3. 7 所示，由于 processing()对 input 中的数据乘上了负增益，二者的波形反相，output 的波形幅度比 input 增大 4 倍。

4.3 比较不同单步方式的区别

通过设置断点并单步调试可知：Step into 单步执行遇到子程序将进入并且继续单步执行；Step over 单步执行时，遇到子程序时不会进入子程序单步执行，而是将整个子程序执行完毕后再停止，即将子程序整个作为一步。

4.4 查看.map 文件信息

如图 3. 8 所示，.map 文件的 MEMORY CONFIGURATION 中，给出了各个存储器空间的首地址、总长度、已用空间和未用空间等信息；在 SECTION ALLOCATION MAP 中，给出了各段的首地址、长度等信息。查阅资料文件，可得到表 4. 2。

表 4. 2 .map 文件各段信息

段名称	page	首地址	长度	作用	所在位置
.pinit	0	0x00008000	0x00000000		RAML0
.cinit	0	0x00008000	0x00000090	存放程序中的变量初值和常量	RAML0
.text	0	0x00009000	0x00002a2e	存放程序代码	RAML1
.reset	0	0x003fffc0	0x00000002		RESET
.sysmem	1	0x00000400	0x00000400	为动态存储分配保留的空间	RAMM1
.ebss	1	0x0000c000	0x00000363	为程序中的全局和静态变量保留存储空间	RAML4
.econst	1	0x0000d000	0x0000025a	存放常量	RAML5
.stack	1	0x0000e000	0x00000300	为程序系统堆栈保留存储空间	RAML6

同时，可根据变量的存储地址及程序的入口地址推测它们所在的段，如 currentBuffer 结构体的 input 和 output 在.ebss 段；dataIO()子程序和 Procession()子程序在.text 段。

4.5 查看及修改.cmd 文件

如图 3. 9 所示，每个段映射得到的存储器首地址与.map 文件中的地址相同。如.text 段映射在 RAML1，PAGE 0 中定义 RAML1 的首地址为 0x009000，长度为 0x003000。则在.map 文件中，.text 段的首地址为 0x00009000，且长度 0x00002a2e 与 MEMORY CONFIGURATION 中 RAML1 的使用空间一致。

修改.cmd 文件中 RAML1 的首地址为 0x009001，同时将长度修改为 0x002FFFF，重新编译、链接后，.map 文件中.text 段的相关信息如图 4. 1 所示，其中.text 段长度和原来不同，推测是每次编译、链接后本来长度便会不一（由 C 语言转换到汇编语言可能存在多种转换方式）。

```
113 .text      0    00009001    0000214d
114                   00009001    00000bfd     rts2800_fpu32.lib : _printfi.obj (.text)
115                   00009bfe    00000323     DSP2833x_DefaultIsr.obj (.text:retain)
116                   00009f21    00000226     rts2800_fpu32.lib : lowlev.obj (.text)
117                   0000a147    00000206                       : trgdrv.obj (.text)
118                   0000a34d    000001e2                       : memory.obj (.text)
119                   0000a52f    00000107                       : ll_div.obj (.text)
120                   0000a636    00000103                       : fopen.obj (.text)
121                   0000a739    000000f8     DSP2833x_SysCtrl.obj (.text)
122                   0000a831    000000b2     rts2800_fpu32.lib : fputs.obj (.text)
123                   0000a8e3    0000009c                       : fd_add.obj (.text)
124                   0000a97f    00000094                       : ankmsg.obj (.text)
125                   0000aa13    0000008b                       : fd_div.obj (.text)
126                   0000aa9e    0000008a                       : setvbuf.obj (.text)
127                   0000ab28    00000083                       : fd_mpy.obj (.text)
128                   0000abab    00000076                       : fflush.obj (.text)
129                   0000ac21    00000067                       : _io_perm.obj (.text)
130                   0000ac88    0000005f                       : fclose.obj (.text)
```

图 4.1 修改.cmd 后重新编译链接的.map 文件部分内容

5 实验总结

5.1 实验中遇到的问题及解决方法

1.找不到“小锤子”编译链接选项。

在一开始测试实验箱时，找不到“小锤子”编译链接（Build）选项。经过摸索后发现是已经进入了调试（Debug）模式（但至于为什么一开始便进入调试模式，这个问题我还无法回答），回到编辑（Edit）模式后可以进行编译、链接。

2.无法在工程中添加“28335_RAM_lnk.cmd”文件。

添加“28335_RAM_lnk.cmd”文件，报错无法添加，修改添加方式（link 或 copy）都无法解决，后来发现是工程文件中已经存在了 28335_RAM_lnk.cmd，将其删除后可以正常添加。

3.编译时报极多错误。

编译链接时发现错误极多，查看报错信息后发现文件路径乱码，推测是是添加工程文件时因为选择了“link”方式，而该些文件在桌面中文目录下。后来将工程文件创建在英文目录下，同时添加文件方式选择“copy”。再进行编译时，错误变少，但仍然有错，错误提示为找不到 XXX 文件。反复核对后发现是添加工程文件时，由于讲义中的要求是“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 、DSP2833x_Headers_nonBIOS.cmd，添加到当前工程中”，于是我们只添加了这些文件，事实上该目录下还有“header”等文件夹，将一系列文件全部添加后，编译链接可以通过。

4.graph 图形工具绘制波形杂乱

在使用 graph 工具绘图时，得到的波形杂乱无章。摸索相关选项后发现，在使用 graph 工具时输入的属性参数并没有在 graph 界面得到体现，即点击图 5. 1 中标出的选项，弹出的属性对话框中的参数仍然是初始值。在对话框中修改为正确的参数并确认，可以得到正确的波形。（但事实上，第二次实验时，使用 graph 工具时输入的正确属性参数却可以直接产生正确波形，而不需要点击图 5. 1 中的

选项再次修改了，推测是因为第一次实验使用 graph 工具时 CCS 5 运行了较长的时间，CCS 5 中的缓存等相关内容影响了 graph 工具的使用）

图 5. 1 graph 工具部分选项

5.2 实验心得体会

事实上在本学期的《电子信息工程课程设计》实验中，也接触过 DSP 实验箱和 CCS，但那时更多的是修改 C 语言程序并烧录，而不是像此次实验一样深入学习 DSP 调试。这使得我对 DSP 的调试有了更深刻的认识。

CCS 5 软件的界面是全英文，这使得我们在摸索一些功能时出现了一些困难，例如由于不小心关闭了“Project Explorer”栏，在添加工程文件这一步，又需要在“窗口”中找到显示“Project Explorer”的选项；在调试界面下，变量观察窗口没有显示，需要调出变量观察窗口；设置断点动作等等。这些卡壳的地方解决起来并不难、也知道解决的方法，只是在英文界面下查找起来不是十分的方便。这就要求我们对使用的开发环境比较熟悉，才能更快地“对症下药”。

在使用各种工具过程中，如果出现结果和预想不一致的情况，往往查找问题会很花费时间，而问题出现的位置又常常出乎意料。例如在使用 graph 工具时，开始输入的参数对照实验讲义是无误的，但始终出不来正确的波形。经过不断摸索该工具下的各个选项，才发现需要进一步修改属性参数。而同样是使用 graph 工具，第二次实验中，一开始直接输入参数，又能够正常显示波形，不需要再次修改参数了。这种软件中“迷”的地方，常常让人不知所措。

在本次使用中，进行了许多调试步骤，如查看变量地址及内容、查看子程序地址、设置断点动作等等，结合工程.map 文件及.cmd 文件，使得我对 DSP 的程序运行原理有了更深刻的了解，对 DSP 开发有了更进一步的认识，也为接下来 DSP 编程打下了基础，希望自己能够在之后的实验中合理运用本实验中学到的内容。