DSP 应用实验报告

实验九：DSP 开发基础

院　　系：电子工程与光电技术学院

专　　业：电子信息工程

姓　　名：赵婧萱

学　　号：920104330118

指导老师：李彧晟

2023 年 5 月 18 日

目录

实验九：DSP 开发基础

9.1 实验目的

1.了解 DSP 开发系统的基础配置

2.熟悉 DSP 集成开发环境（CCS）

3.掌握 C 语言开发的基本流程

4.熟悉代码调试的基本方法

9.2 实验仪器

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

9.3 实验内容

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。

9.4 实验步骤

1.将 TMS320F28335 教学实验箱连接至计算机。

2.点击桌面 CCS5 快捷方式，启动 CCS 集成开发环境。

3.根据实验讲义，熟悉 CCS 集成开发环境各项操作，包括新建工程、添加工程文件、查阅代码、建立工程、调试程序、程序运行、程序调试等步骤。

4.项目编译、链接、调试：

1)将“LAB_9”工程文件添加至目录。导入示范文件后，原工程调试后出现了报错现象。错误信息为：program will not fit into available memory，经查阅相关资料，原因认为是超过了 cmd 中 MEMORY 里定义的地址长度从而导致的内存分配不足，并进行了修改：RAML1 的 length 大小改为 3000。这样修

改后不再报错，但是仍有警告出现。资料上说警告的原因是因为在 SECTIONS 中缺少定义，在其中加上相应定义即可。但由于对本次实验无影响，所以就没有进行此处修改。具体 cmd 文件修改操作如图 1 所示：

```
MEMORY
{
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
   CSM_PWL    : origin = 0x33FFF8, length = 0x000008     /* Part of FLASHA.  CSM password loca
   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

```

图 1 cmd 文件修改

2)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。

3)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。在 CCS Debug 环境界面的主菜单中选择“Run→Resume”运行程序。程序的执行结果依赖外部硬件或查看寄存器存储器的数值加以验证。

5.添加结构体变量至观察窗口：

1)点击 Add new expression，添加变量 currentBuffer 到变量观察窗口

2)选中变量 currentBuffer，点击左侧箭头，即看到 input 与 output 相关内容，可观察变量的类型、内容和地址。同样的操作，将 dataIO( )添加到变量窗口，查看该子程序的入口地址。观察窗口截图如图：

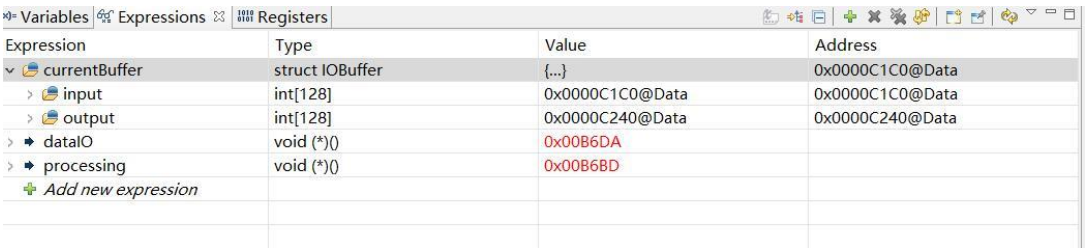

图 2 变量观察窗口

3)通过此操作可以观察到，变量 currentBuffer.input 所在存储器地址为 0x0000C1C0@Data，变量 currentBuffer.output 所在存储器地址为 0x0000C240@Data，子程序 dataIO()的地址为 0x00B6DA。将两个变量和子程序的地址记录下来。

6.在 dataIO()处设置断点，关联输入文件 sine.dat

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(0x0000C1C0@Data)，数据长度为 128，点击“OK”。各项设置如图 3 所示：

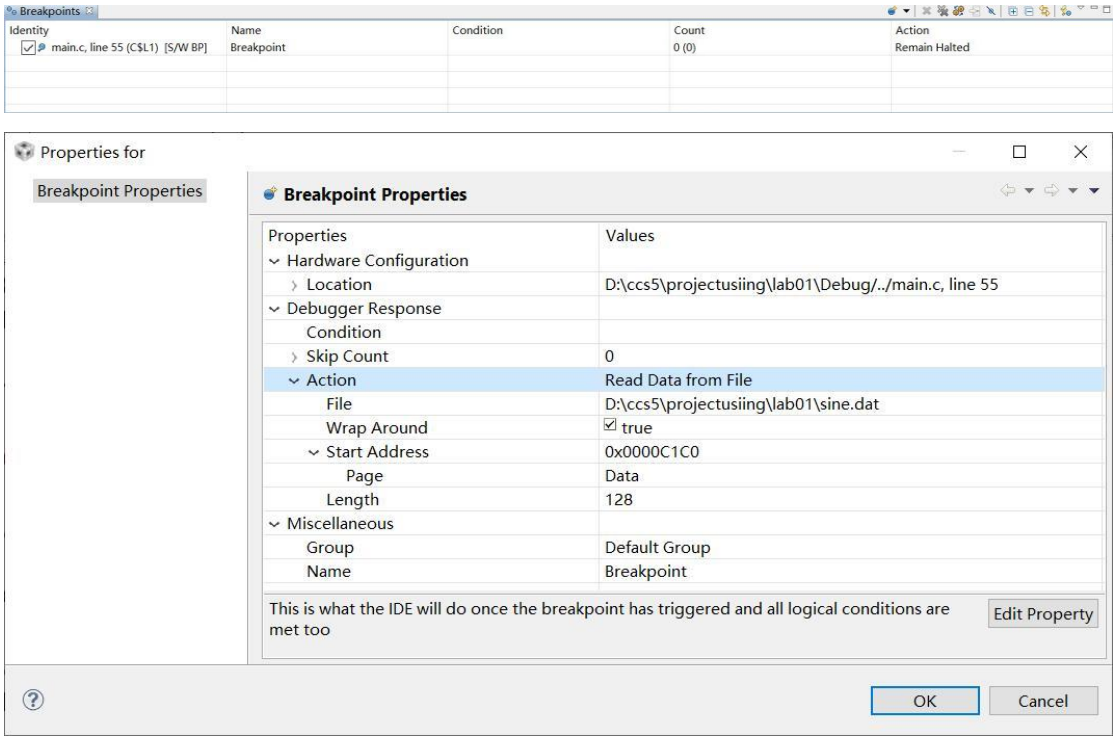

图 3 断点及关联数据设置 DSP

7.图形显示数据空间,打开图形显示功能

在主菜单的点击“Tools→Graph→single time”，按照实验讲义所示设置各项参数。也可直接在观察窗口中选择要绘图的变量，右键选择“Graph”“sign Graph”即可获得想看的图形。设置起始地址为变量 currentBuffer.input 的起始地址 0x0000C1C0@Data，得到存储空间的时域波形如图 4 所示

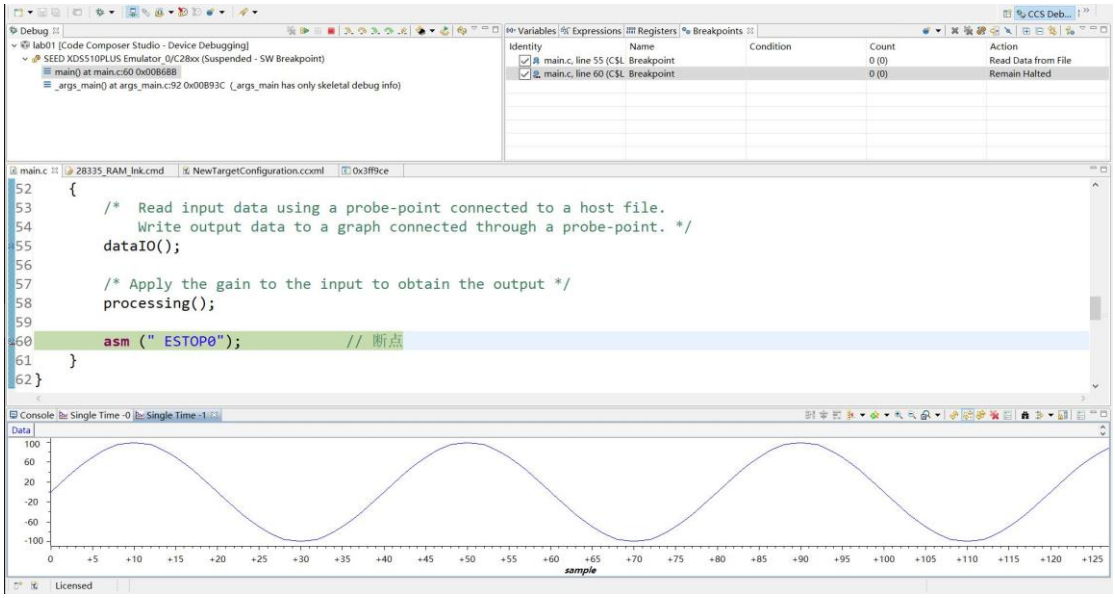

图 4 数据空间 currentBuffer.input 缓冲存储器中的波形

设置起始地址为变量 currentBuffer.output 的起始地址 0x0000C240@Data，得到存储空间的时域波形如图 5 所示

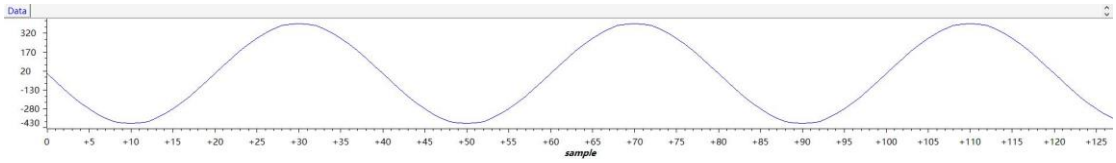

图 5 数据空间 currentBuffer.output 缓冲存储器中的波形

8.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 文件大概分为文件头、内存配置、段映射、全局符号四部分。打开 MAP 文件至 section location map 部分，即可知道该部分程序中所有的段实际映射的起始地址与实际长度。如图 6 所示。

```
SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
codestart
*          0    00000000    00000002
                  00000000    00000002     DSP2833x_CodeStartBranch.obj (codestart)

.pinit     0    00008000    00000000     UNINITIALIZED

.cinit     0    00008000    00000090
                  00008000    0000002d     rts2800_fpu32.lib : lowlev.obj (.cinit)
                  0000802d    0000002a                       : defs.obj (.cinit)
                  00008057    00000017     DSP2833x_Lcd.obj (.cinit)
                  0000806e    0000000a     rts2800_fpu32.lib : _lock.obj (.cinit)
                  00008078    0000000a                       : exit.obj (.cinit)
                  00008082    00000004     main.obj (.cinit)
                  00008086    00000004     rts2800_fpu32.lib : fopen.obj (.cinit)
                  0000808a    00000004                       : memory.obj (.cinit)
                  0000808e    00000002     --HOLE-- [fill = 0]

ramfuncs   0    00008090    0000001f
                  00008090    0000001b     DSP2833x_SysCtrl.obj (ramfuncs)
                  000080ab    00000004     DSP2833x_usDelay.obj (ramfuncs)

.text      0    00009000    00002a2e
                  00009000    00000911     rts2800_fpu32.lib : _printfi.obj (.text)
                  00009911    000003f4     DSP2833x_DMA.obj (.text)
                  00009d05    00000323     DSP2833x_DefaultIsr.obj (.text:retain)
                  0000a028    00000274     DSP2833x_Lcd.obj (.text)
                  0000a29c    0000024b     rts2800_fpu32.lib : lowlev.obj (.text)
                  0000a4e7    00000201                       : trgdrv.obj (.text)
                  0000a6e8    000001d4                       : memory.obj (.text)
                  0000a8bc    00000170     DSP2833x_Mcbsp.obj (.text)
                  0000aa2c    00000130     DSP2833x_ECan.obj (.text)
                  0000ab5c    00000107     rts2800_fpu32.lib : ll_div.obj (.text)
                  0000ac63    00000101                       : fopen.obj (.text)
                  0000ad64    000000fa     DSP2833x_Xintf.obj (.text)
                  0000ae5e    000000f3     DSP2833x_SysCtrl.obj (.text)
                  0000ba2b    00000003     rts2800_fpu32.lib : remove.obj (.text)

csm_rsvd   0    0033ff80    00000076     DSECT
                  0033ff80    00000076     DSP2833x_CSMPasswords.obj (csm_rsvd)

csmpasswds
*          0    0033fff8    00000008     DSECT
                  0033fff8    00000008     DSP2833x_CSMPasswords.obj (csmpasswds)

.adc_cal   0    00380080    00000007     NOLOAD SECTION
                  00380080    00000007     DSP2833x_ADC_cal.obj (.adc_cal)

.reset     0    003fffc0    00000002     DSECT
                  003fffc0    00000002     rts2800_fpu32.lib : boot.obj (.reset)

.sysmem    1    00000400    00000400     UNINITIALIZED
                  00000400    00000001     rts2800_fpu32.lib : memory.obj (.sysmem)
                  00000401    000003ff     --HOLE--

DevEmuRegsFile
*          1    00000880    000000d0     UNINITIALIZED
                  00000880    000000d0     DSP2833x_GlobalVariableDefs.obj (DevEmuRegsFile)
```

```
.ebss      1    0000c000    00000363     UNINITIALIZED
                  0000c000    00000160       rts2800_fpu32.lib : defs.obj (.ebss)
                  0000c160    00000018       DSP2833x_CpuTimers.obj (.ebss)
                  0000c178    00000008       rts2800_fpu32.lib : memory.obj (.ebss)
                  0000c180    00000140       main.obj (.ebss)
                  0000c2c0    00000088       rts2800_fpu32.lib : lowlev.obj (.ebss)
                  0000c348    0000000a       DSP2833x_Lcd.obj (.ebss)
                  0000c352    00000008       rts2800_fpu32.lib : trgdrv.obj (.ebss)
                  0000c35a    00000004                         : _lock.obj (.ebss)
                  0000c35e    00000004                         : exit.obj (.ebss)
                  0000c362    00000001                         : fopen.obj (.ebss)

.cio       1    0000c380    00000120     UNINITIALIZED
                  0000c380    00000120       rts2800_fpu32.lib : ankmsg.obj (.cio)

.econst    1    0000d000    0000025a
                  0000d000    00000101       rts2800_fpu32.lib : ctype.obj (.econst:__ctypes_)
                  0000d101    00000001       --HOLE-- [fill = 0]
                  0000d102    00000100       DSP2833x_PieVect.obj (.econst)
                  0000d202    00000024       rts2800_fpu32.lib : _printfi.obj (.econst:.string)
                  0000d226    0000001a       main.obj (.econst:.string)
                  0000d240    00000018       rts2800_fpu32.lib : _printfi.obj (.econst)
                  0000d258    00000002                         : fputs.obj (.econst:.string)

.stack     1    0000e000    00000300     UNINITIALIZED
                  0000e000    00000300       --HOLE--

CsmPwlFile
*          1    0033fff8    00000008     UNINITIALIZED
                  0033fff8    00000008       DSP2833x_GlobalVariableDefs.obj (CsmPwlFile)


GLOBAL SYMBOLS: SORTED ALPHABETICALLY BY Name
```

图 6 section location map

联系 cmd 文件中 section 指令即可了解该段所在地址。

```
119 SECTIONS
120 {
121    /* Setup for "boot to SARAM" mode:
122       The codestart section (found in DSP28_CodeStartBranch.asm)
123       re-directs execution to the start of user code.  */
124    codestart        : > BEGIN,     PAGE = 0
125    ramfuncs         : > RAML0,     PAGE = 0
126    .text            : > RAML1,     PAGE = 0
127    .cinit           : > RAML0,     PAGE = 0
128    .pinit           : > RAML0,     PAGE = 0
129    .switch          : > RAML0,     PAGE = 0
130
131    //.stack           : > RAMM1,     PAGE = 1
132    .stack           : > RAML6,     PAGE = 1
133    .ebss            : > RAML4,     PAGE = 1
134    .econst          : > RAML5,     PAGE = 1
135    .esysmem         : > RAMM1,     PAGE = 1
136
137    IQmath           : > RAML1,     PAGE = 0
138    IQmathTables     : > IQTABLES,  PAGE = 0, TYPE = NOLOAD
139
```

图 7 cmd 文件 section 指令

下面以 currentBuffer 结构体变量为例，指出该变量所在段和物理存储空间。

法一： 由上文知 currentBuffer 变量地址为 0x0000C1C0,由 MAP 文件的 MEMORY CONFIGURATION 段即可得出 currentBuffer 变量在 RAML4 中。如图 8

所示：

```
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 */
}
```

图 8 变量 currentBuffer 所在段和物理储存空间

法二： 阅读 main 函数，易知 currentBuffer 变量为全局变量，即为.ebss 段，查阅 cmd 文件 section（图 7），即可得出 currentBuffer 变量在 RAML4 中。

主程序中的变量有全局变量 ebss，局部变量.stack，代码.text，初始值.cinint，所在物理存储空间和存储块如上图所示。

9.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。

MAP 文件大概分为文件头、内存配置、段映射、全局符号四部分。内存配置与 cmd 文件中的 MEMORY 指令关联，在 cmd 文件中定义的程序与数据区间定义，在该部分均可以找到对应，与 cmd 文件不同的时，在 MAP 文件中加入了一个实际使用的区间，即在程序中实际用到的空间长度。

```
.text      0    00009000    00002a2e
                  00009000    00000911     rts2800_fpu32.lib : _printfi.obj (.text)
                  00009911    000003f4     DSP2833x_DMA.obj (.text)
                  00009d05    00000323     DSP2833x_DefaultIsr.obj (.text:retain)
                  0000a028    00000274     DSP2833x_Lcd.obj (.text)
                  0000a29c    0000024b     rts2800_fpu32.lib : lowlev.obj (.text)
                  0000a4e7    00000201                       : trgdrv.obj (.text)
                  0000a6e8    000001d4                       : memory.obj (.text)
                  0000a8bc    00000170     DSP2833x_Mcbsp.obj (.text)
                  0000aa2c    00000130     DSP2833x_ECan.obj (.text)
                  0000ab5c    00000107     rts2800_fpu32.lib : ll_div.obj (.text)
                  0000ac63    00000101                       : fopen.obj (.text)
                  0000ad64    000000fa     DSP2833x_Xintf.obj (.text)
                  0000ae5e    000000f3     DSP2833x_SysCtrl.obj (.text)
                  0000af51    000000af     rts2800_fpu32.lib : fputs.obj (.text)
                  0000b000    000000a1     DSP2833x_EPwm.obj (.text)
                  0000b0a1    0000009c     rts2800_fpu32.lib : fd_add.obj (.text)
                  0000b13d    0000008b                       : fd_div.obj (.text)
                  0000b1c8    00000086                       : ankmsg.obj (.text)
                  0000b24e    00000083                       : fd_mpy.obj (.text)
                  0000b2d1    0000007c     DSP2833x_CpuTimers.obj (.text)
                  0000b34d    00000076     rts2800_fpu32.lib : setvbuf.obj (.text)
                  0000b3c3    00000065                       : _io_perm.obj (.text)
                  0000b428    00000060                       : fflush.obj (.text)
                  0000b488    0000005c     DSP2833x_ECap.obj (.text)
                  0000b4e4    00000054     DSP2833x_EQep.obj (.text)
                  0000b538    00000054     rts2800_fpu32.lib : fputc.obj (.text)
```

图 9 map 文件

段映射部分与 cmd 文件中的 SECTION 指令关联，在该部分程序中所有的段实际映射的起始地址与实际长度均有详细说明。可以具体到程序中#pragma 指定的段和各个单独文件产生的 OBJ 文件。修改 cmd 配置文件，在进行断点求地址操作，发现地址产生变化,映射也发生了改变。例如将 RAML1 的起始地址由 0x009000 变化至 0x009002，长度也进行相应改变，查阅 map 文件中.text 起始地址也随之发生改变。改变如下表所示：

	修改前	修改后

CMD	MEMORY
	{
	PAGE 0 :
	/* BEGIN is used for the "boot to SARAM" bootloader
	
	BEGIN : origin = 0x000000, length = 0x000002
	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
	CSM_RSVD : origin = 0x33FF80, length = 0x000076
	CSM_PWL : origin = 0x33FFF8, length = 0x000008
	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
		MEMORY
		{
		PAGE 0 :
		/* BEGIN is used for the "boot to SARAM" bootloader mode */
		
		BEGIN : origin = 0x000000, length = 0x000002 /* Boot to
		RAMM0 : origin = 0x000050, length = 0x0003B0
		RAML0 : origin = 0x008000, length = 0x001000
		RAML1 : origin = 0x009002, length = 0x002ffe
		//RAML2 : origin = 0x00A000, length = 0x001000
		// RAML3 : origin = 0x00B000, length = 0x001000
		ZONE7A : origin = 0x200000, length = 0x00FC00 /* XINTF zon
		CSM_RSVD : origin = 0x33FF80, length = 0x000076 /* Part of
		CSM_PWL : origin = 0x33FFF8, length = 0x000008 /* Part of
		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

MAP	.text 0 00009000 00002a2e 00009000 00000911 rt 00009911 000003f4 DS 00009d05 00000323 DS 0000a028 00000274 DS 0000a29c 0000024b rt 0000a4e7 00000201 0000a6e8 000001d4 0000a8bc 00000170 DS 0000aa2c 00000130 DS 0000ab5c 00000107 rt 0000ac63 00000101 0000ad64 000000fa DS 0000ae5e 000000f3 DS	.text 0 00009002 00002a2e 00009002 00000911 rts 00009913 000003f4 DSP 00009d07 00000323 DSP 0000a02a 00000274 DSP 0000a29e 0000024b rts 0000a4e9 00000201 0000a6ea 000001d4 0000a8be 00000170 DSP 0000aa2e 00000130 DSP 0000ab5e 00000107 rts 0000ac65 00000101

表 1 cmd 与 map 对应关系

9.5 实验总结

在之前的科研竞赛中，曾经接触 DSP 开发板（F2812）和 CCS 开发软件。但在刚本次开始进行调试的时候遇到了许多问题，主要几个问题如下：

1.在实验开始时连接仿真器，出现了设备管理器警告的情况。分析原因，由于 win10 安全状态下无法真正禁用数字签名，故尝试进入 BIOS 之后关闭安全模式，重启进入系统后通过控制台禁用数字签名，重新安装 SEED 驱动，此问题便得到了解决。

2.在完成添加变量至观察窗口、添加断点和数据关联等步骤后，在绘制数据

图形时发现绘制出来的图形是杂乱无序的，确认 sine.data 文件其中数据确是正弦波样点的数据。仔细检查操作步骤后发现，我们的断点位置出现错误，除了 dataIO()，还需要在 asm ("ESTOP0");处打断点。改正之后便得到了正确波形。

```
51    while(TRUE) // loop forever
52    {
53        /*  Read input data using a probe-point connected to a host file.
54            Write output data to a graph connected through a probe-point. */
55        dataIO();
56
57        /* Apply the gain to the input to obtain the output */
58        processing();
59
60        asm (" ESTOP0");            // 断点
61    }
```

4. DSP 文件中源文件的作用

文件名	作用
include 头文件(.h)	定义程序中的函数、参数、变量和宏单元，配合库函数使用；
DSP/BIOS CONFIG FILES	开发基于 DSP/BIOS 的程序保存 BIOS 配置之后自动生成的文件
Source Files	源程序，实现 DSP 系统指定功能的主要代码部分
Program.cmd	链接文件，在源文件通过汇编器（Assembier）输出的.obj files（目标文件）需要通过 Linker（链接器）得到输出文件，在链接阶段 Linker 根据.cmd 内部存储区、section 分配以及 lib 来链接
.cmd	负责定义 memory region，将用户的代码数据 section 映射到区域内
.ccxml 文件	配置仿真器
.map	执行文件的映像信息纪录文件
.pdb（Program Database）	记录程序有关的数据和调试信息

表 2 源文件在 DSP 工程中的应用

通过本次实验，我们在 CCS 的操作环境下跑通了仿真器和板子，初步验证了开发环境无误，为接下来几次实验奠定了基础。本次实验不涉及程序编写，因此较为简单，但这还是我第一次接触在 dsp 中加入查找表功能。希望在接下来几次实验中能够继续深入学习 DSP 芯片编程的其他方面。