

**DSP应用技术实验**

**DSP开发基础实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 作者 | ： | 周鹏 | 学号 | ： | 9181040G0740 |
| 同组人 | ： | 许昕荣 | 学号 | ： | 9181040G1038 |
| 同组人 | ： | 杨霄宇 | 学号 | ： | 9181040G0736 |
| 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 | ： | B6 | | | |
| 题目 | ： | DSP应用技术实验 | | | |
|  |  | DSP开发基础实验报告 | | | |
| 指导者 | ： | 李彧晟 | | | |

2021 年 4 月

目录

[1 实验目的 1](#_Toc69765649)

[2 实验仪器 1](#_Toc69765650)

[2.1 实验仪器清单 1](#_Toc69765651)

[2.2 硬件连接示意图 1](#_Toc69765652)

[3 实验步骤及现象 1](#_Toc69765653)

[4 实验结果汇总及问题回答 4](#_Toc69765656)

[4.1 子程序入口地址与结构体存储地址 4](#_Toc69765657)

[4.2 显示缓冲存储器中的波形 5](#_Toc69765659)

[4.3 查看所有的段在存储空间的地址、长度和含义 5](#_Toc69765661)

[4.4 查看.map文件信息 6](#_Toc69765662)

[4.5 查看.cmd文件信息，比较与.map文件的映射关系 7](#_Toc69765663)

[5 实验总结 9](#_Toc69765664)

[5.1实验中遇到的问题及解决方法 9](#_Toc69765665)

[5.2实验心得体会 10](#_Toc69765666)

# 1 实验目的

1. 了解DSP硬件开发平台基本配置；

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；

# 2 实验仪器

## 2.1 实验仪器清单

|  |  |  |
| --- | --- | --- |
| 1. | TMS320F28335DSP教学实验箱 | 一套 |
| 2. | XDS510 USB仿真器 | 一个 |
| 3. | 电脑 | 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。

|  |
| --- |
| 图2. 1 硬件连接示意图 |

# 3 实验步骤及现象

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“New→Folder”命令，依次新建名为“header”和“source”的文件夹，再点击“Add Files to Project”命令，在弹出的对话框中依次选择“实验范例\LAB9\header”下所有的文件添加，并将添加的文件拖至新建的文件夹内。同样处理 source 文件夹内的相同内容。再次将“实验范例\LAB9”目录下的文件通过该命令添加，包括 main.c、FPU.h、sine.h、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 添加到工程目录中。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”和“source”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。

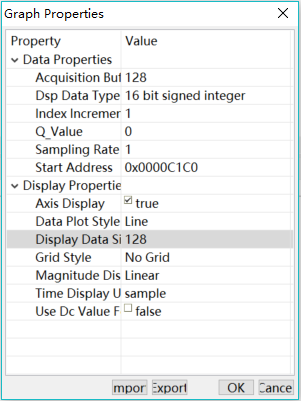
添加结构体变量 currentBuffer 到变量观察窗口（Add Watch Expression），观察currentBuffer.output 和 currentBuffer.input 的地址以及数值。添加 dataIO( )到变量窗口，查看该子程序的入口地址。

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。



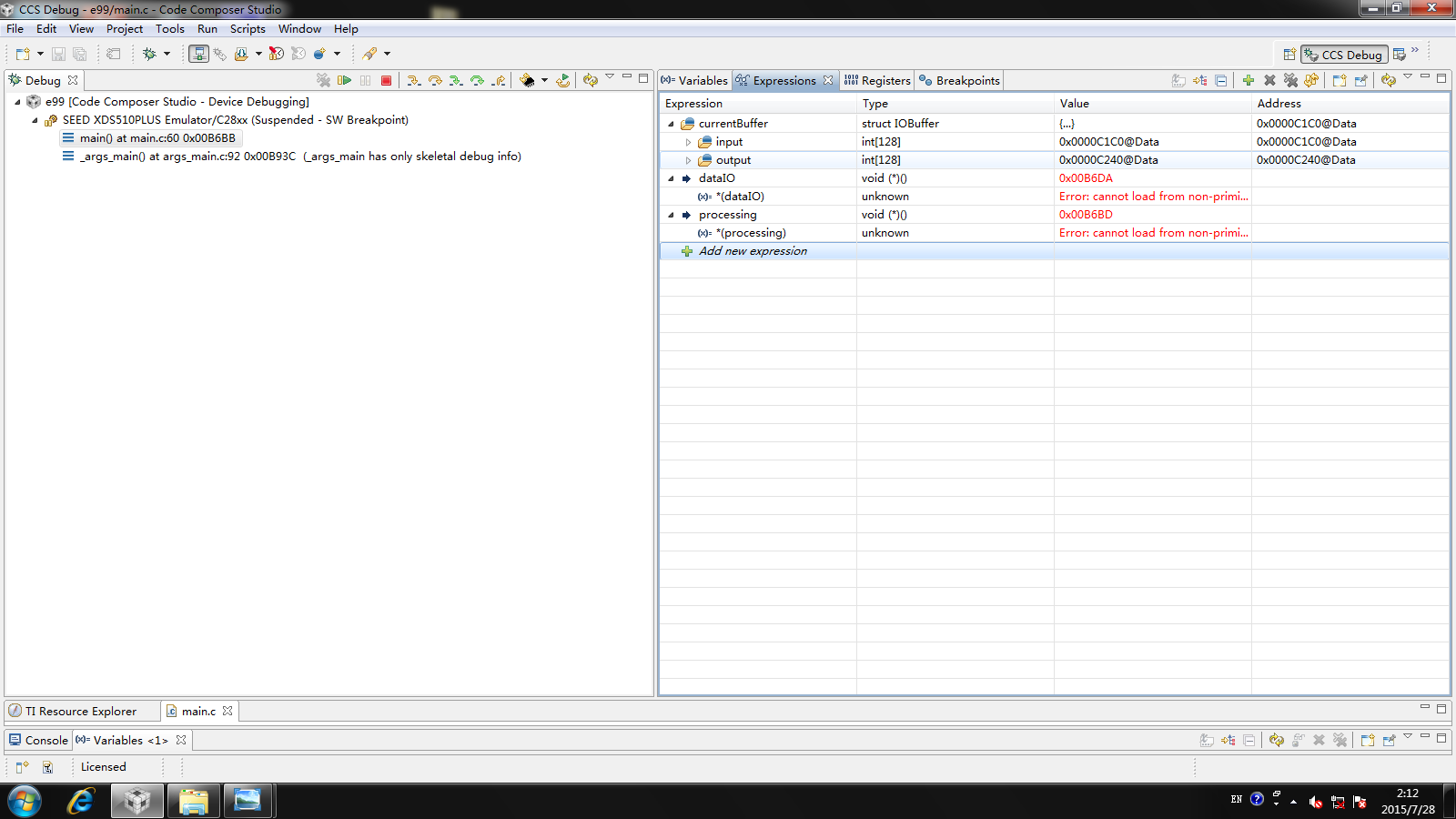
查看存储空间的数值在程序相关语句执行前后的变化。

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。

# 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

|  |  |
| --- | --- |
| 名称 | 地址 |
| dataIO()子程序 | 0x00B6DA |
| Procession()子程序 | 0x00B6BD |
| currentBuffer.input | 0x0000C1C0 |
| currentBuffer.output | 0x0000C240 |



## e99.2

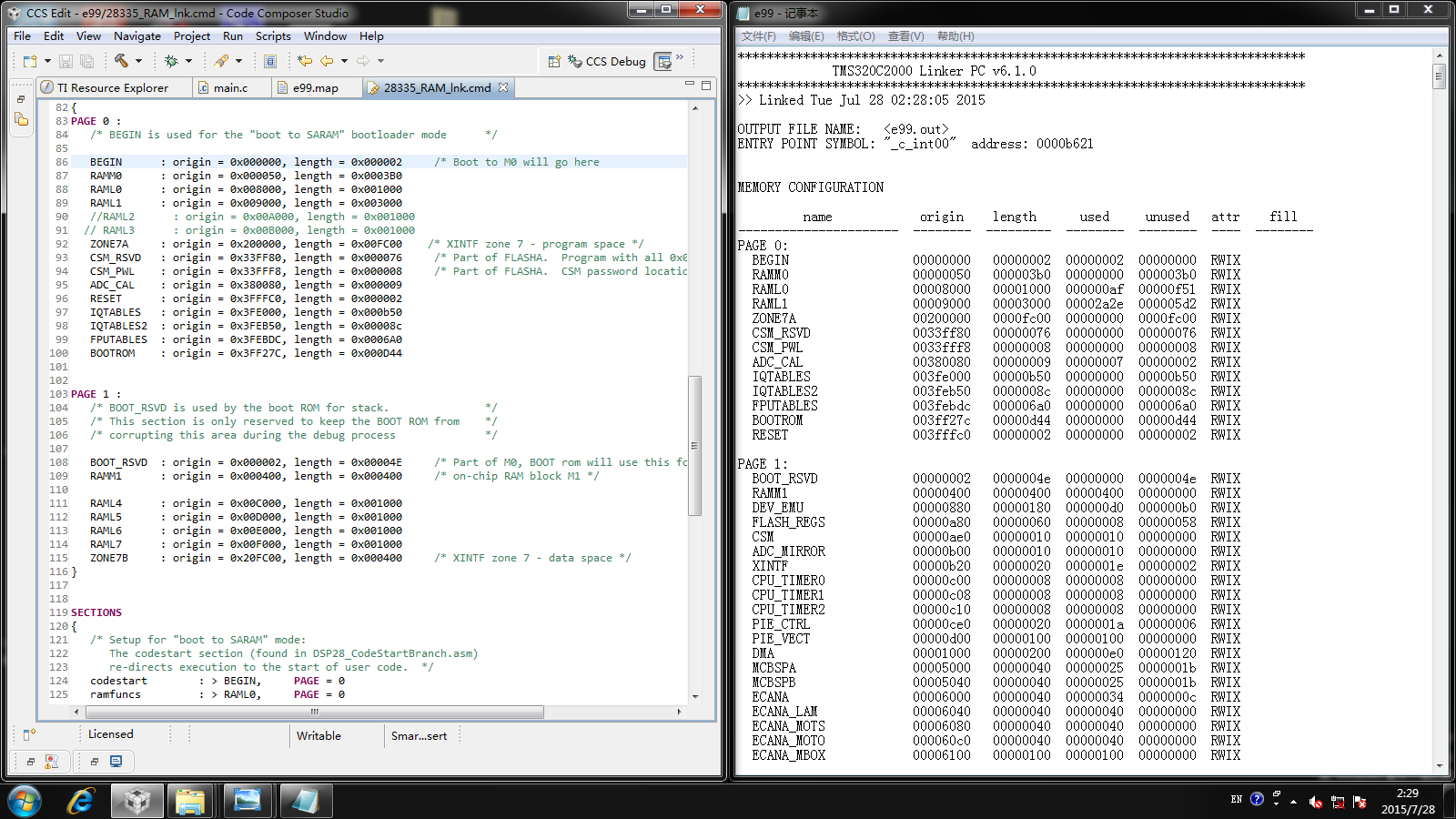
## 4.2 显示缓冲存储器中的波形

## e99.4

## 4.3 查看所有的段在存储空间的地址、长度和含义，指出所处的存储空间及物理存储块名称，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，经过整理，得到下表：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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。

## 4.4 查看.map文件信息

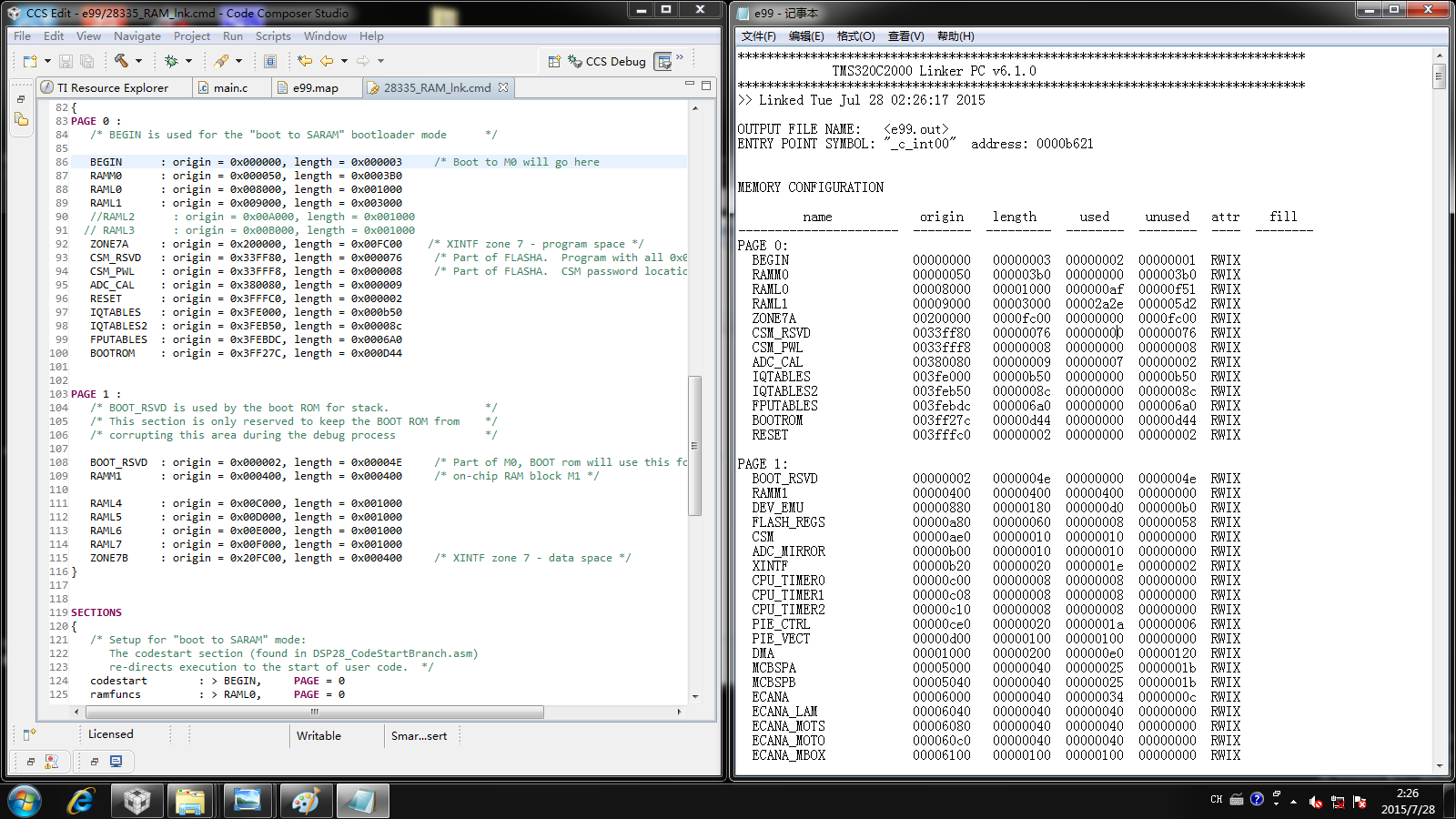
如**错误!未找到引用源。**所示，.map文件的MEMORY CONFIGURATION中，给出了各个存储器空间的首地址、总长度、已用空间和未用空间等信息；在SECTION ALLOCATION MAP中，给出了各段的首地址、长度等信息。查阅资料文件，可得到表4. 1。

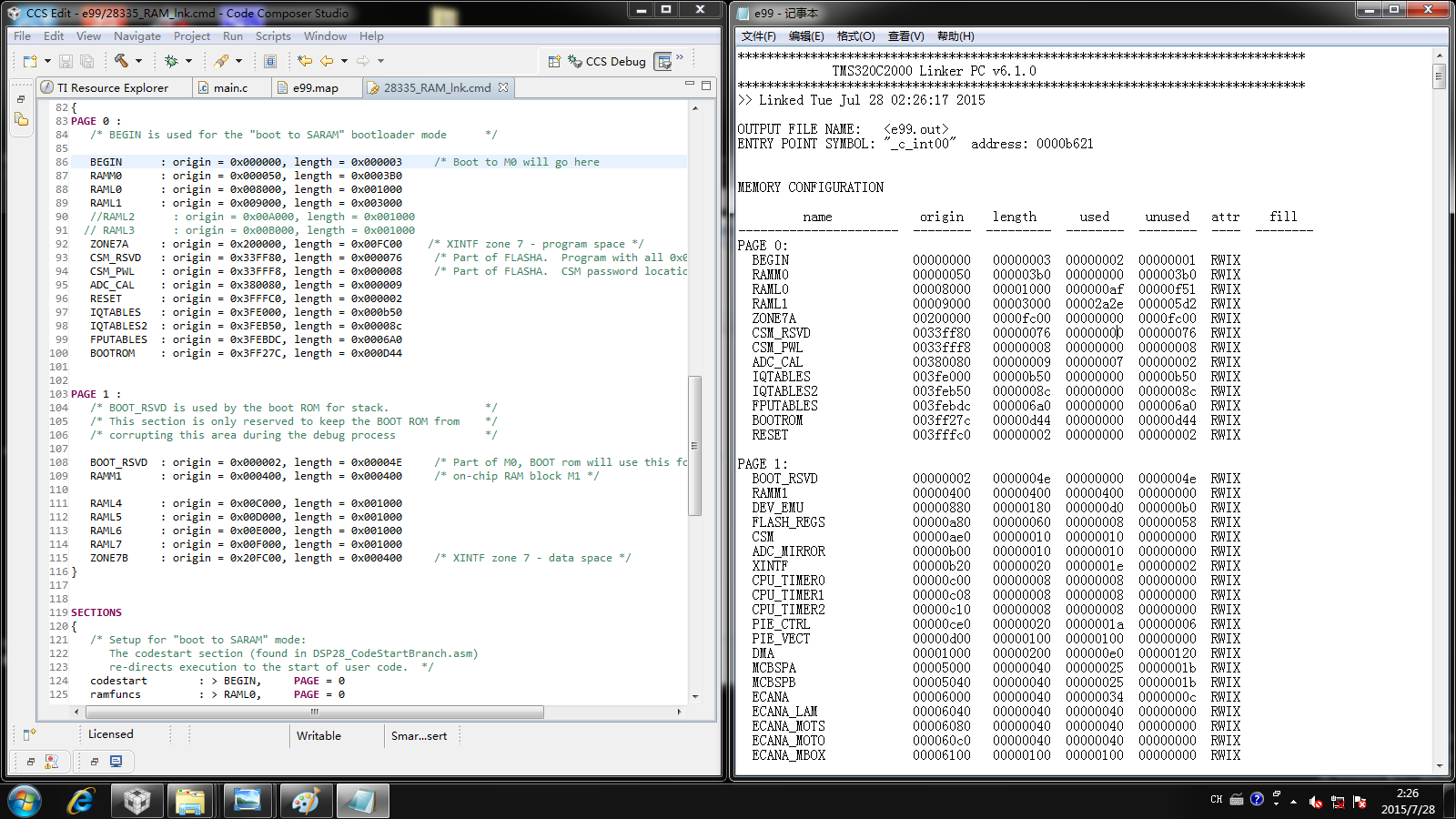
表4. 1 .map文件各段信息

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |

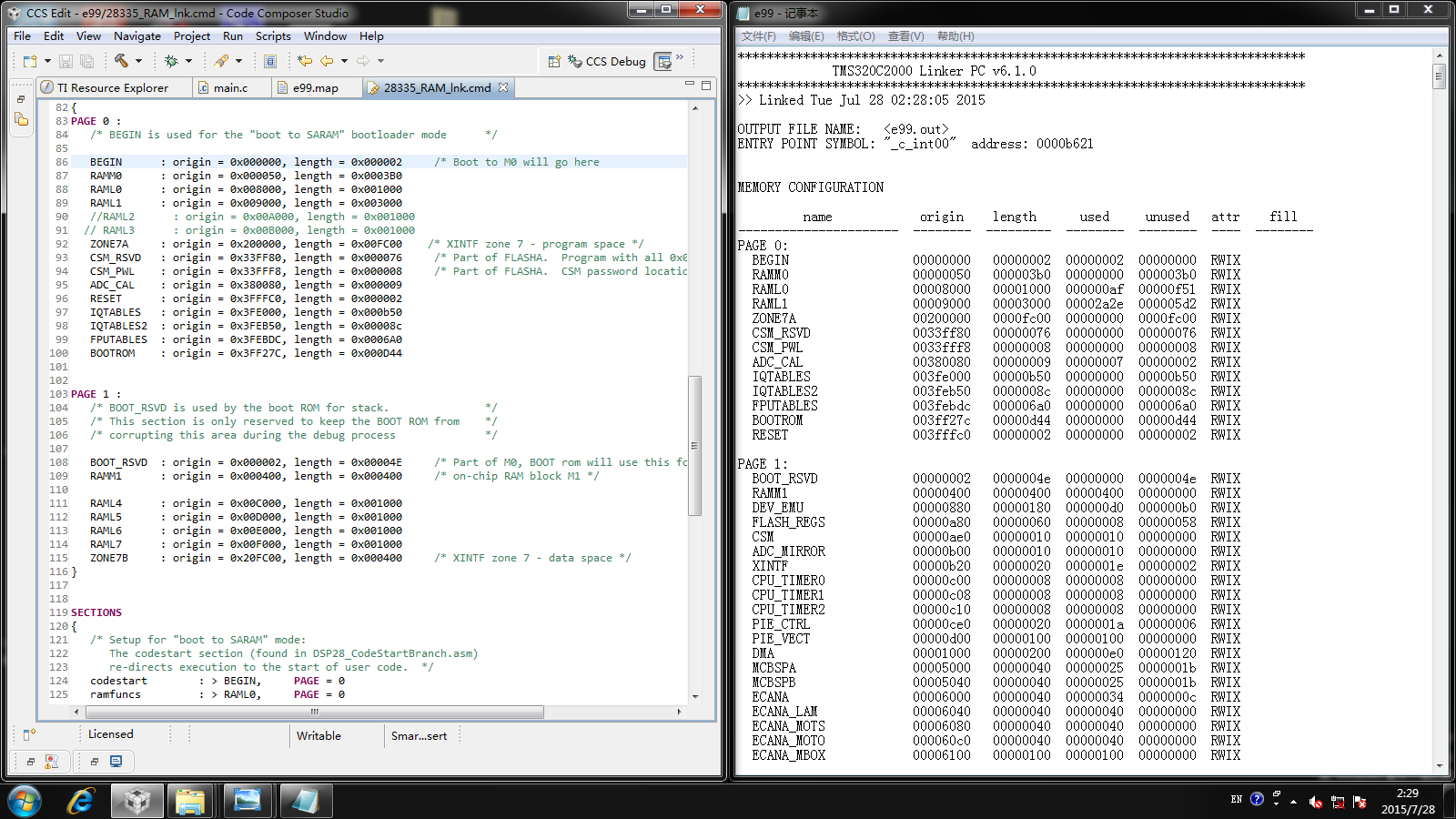
同时，可根据变量的存储地址及程序的入口地址推测它们所在的段，如currentBuffer结构体的input和output在.ebss段；dataIO()子程序和Procession()子程序在.text段。

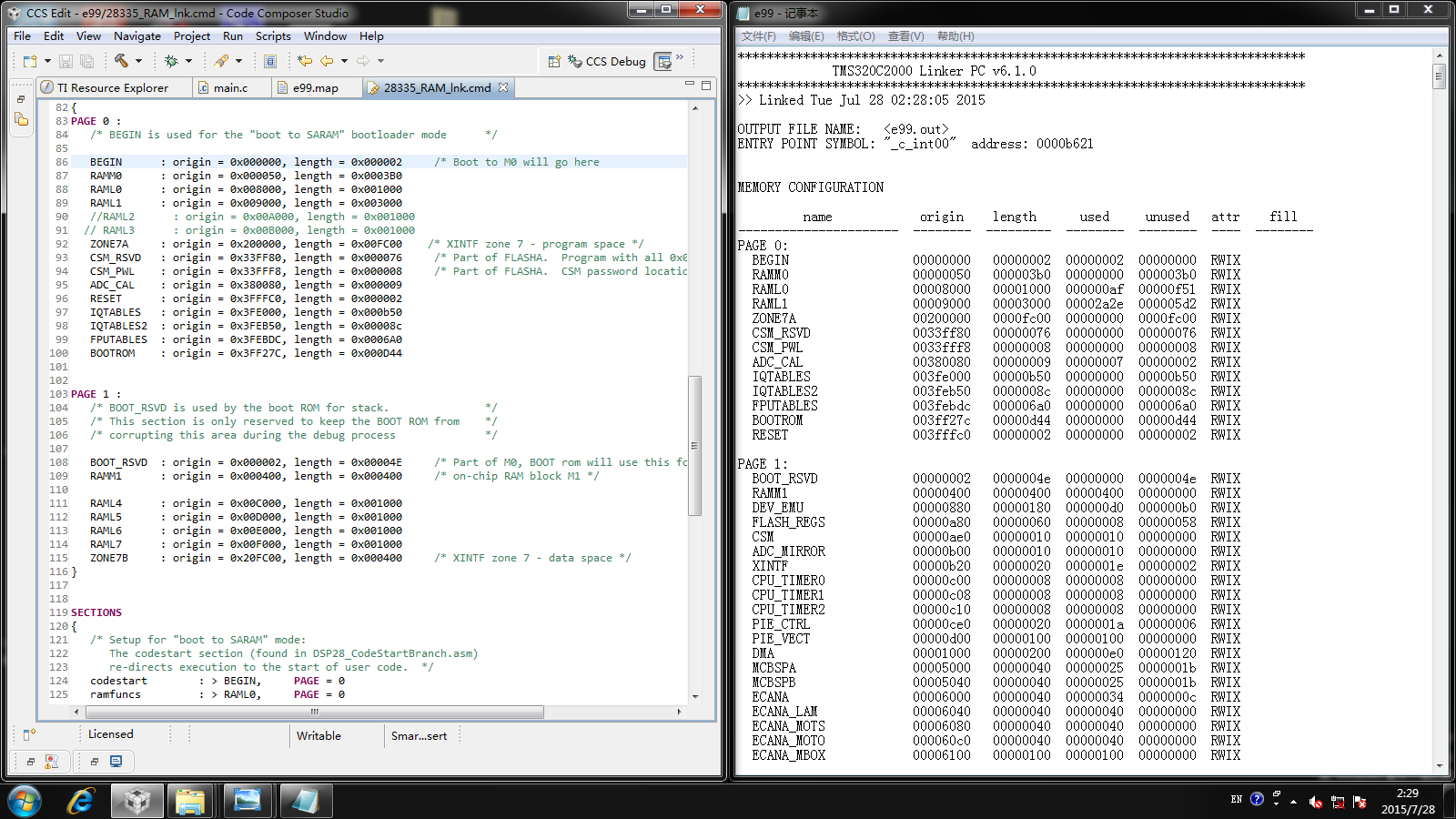
## 4.5 查看.cmd文件信息，比较与.map文件的映射关系

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.PAGE 0中定义BEGIN的首地址为0x000000，长度为0x000003。则在.map文件中，.text段的首地址为0x00000000，且长度0x000003与MEMORY CONFIGURATION中RBEGIN的使用空间一致。



修改.cmd文件中BEGIN的长度为0x000002，重新编译、链接后，.map文件中.text段的相关信息也随之发生变化。





|  |
| --- |
|  |

# 5 实验总结

## 5.1实验中遇到的问题及解决方法

**1. 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 |

1. **build时会出现编译错误，显示未找到studint.h和stdio.h。**

解决方案是我们又重新与实验讲义进行核对，发现了是28335\_RAM\_lnk.cmd文件在添加时没有删除掉原文件自动生成的文件，我们将其删除后，把文件夹中新的.cmd文件重新添加后继续编译，问题得到了解决。

1. **在添加相应的文件后，为了看起来整洁，我们在左侧文件栏中修改了文件的名称，编译时出现了错误。**

解决方案是我们发现，如果改掉了文件栏中的文件名称，会导致ccs中文件的名称与workspace中的文件不对应，出现问题；我们将ccs中拥有新文件名称的文件路径进行重新设置，即右击工程标题，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对应的文件，问题得到解决。

1. **在观察子程序对应的入口地址时，发现dataIO()没有地址只有value，并且颜色为红色。**

在查阅资料后发现，在变量观察表中，观察的对象应为变量，例如currentbuffer这种结构体变量，会显示正确的地址和value，而dataIO()这种函数，在变量观察表中无法显示正确的信息，原因在于其并不是变量，而函数对应的地址就为其value的数值，自然也为红色。

1. **在build成功后的debug中，显示出现-171的错误，内容显示无法发现对应的仿真器。**

我们首先利用CSDN软件进行相应的查找，发现比较好的解决办法时重新建立新的ccs project，之后我们建立了3次，结果依旧出现问题，所以，我们请求了实验室的助教学姐，得到的办法时重新电脑，原因是有病毒的出现，在重启电脑后解决了此问题。

1. **在对所给程序的主函数进行查阅后，发现我们debug后的变量观察表中的数值和程序对应不起来。**

这个问题困扰了大概30多分钟，按照仿真后的结果进行数值的计算，将input中的数值乘以-4给到output，发现怎么都对应不起来，当时还疑惑是不是有数值的溢出，或者补码的换算上出现了问题，在最后结束的时候询问了老师，发现是断点的问题，因为我们程序设置了断点，导致程序没有运行完整，所以存储器中的数据都是随机分配的，导致了数值的不正确，接着，我将程序的断点全部去除重新运行，发现数值对应了起来，bufferinput（127）=70，乘以-4后为-280，正好等于bufferoutput（127）中的-280，问题终于得到了解决。

## 5.2实验心得体会

在之前的电赛和微机实验中，对CCS软件有所接触，因此整体实验上手较快，对软件的适应感觉良好。

但是无论是电赛还是微机实验，都是对单片机的编程，而且大多都是修改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。

整体的实验过程中，出现了许多意想不到的问题，这些问题大多都来源于不规范的操作，例如我们为了看起来整洁，我们在左侧文件栏中修改了文件的名称，编译时出现了错误，这些都是可以通过规范操作避免的。还有一些问题来自于对DSP芯片运作机制了解不充分，例如我们发现我们debug后的变量观察表中的数值和程序对应不起来，这是由于我们的程序中存在断点，导致部分变量存储空间并未赋值，生成的仅仅是随机数。这就需要我们能够对DSP芯片的运作机制有更深的了解，积极思考。

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。