



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 : 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
2 实验仪器	1
2.1 实验仪器清单	1
2.2 硬件连接示意图	1
3 实验步骤及现象	1
4 实验结果汇总及问题回答	4
4.1 子程序入口地址与结构体存储地址	4
4.2 显示缓冲存储器中的波形	5
4.3 查看所有的段在存储空间的地址、长度和含义	5
4.4 查看.map 文件信息	6
4.5 查看.cmd 文件信息，比较与.map 文件的映射关系 ...	7
5 实验总结	9
5.1 实验中遇到的问题及解决方法	9
5.2 实验心得体会	10

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。

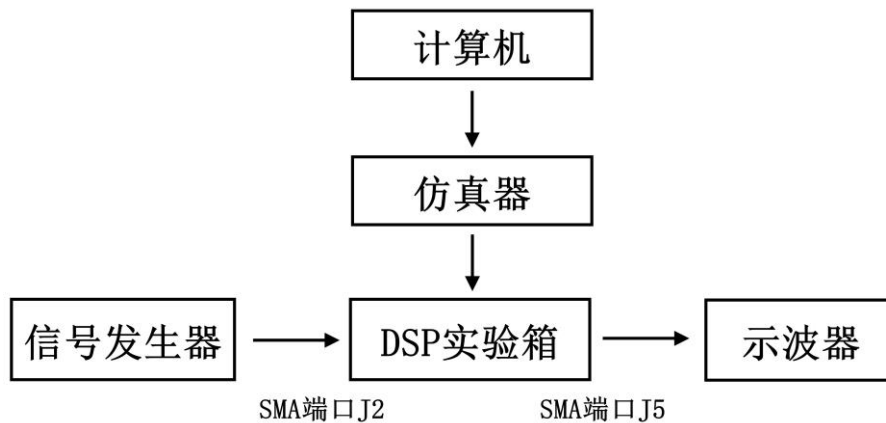


图 2.1 硬件连接示意图

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

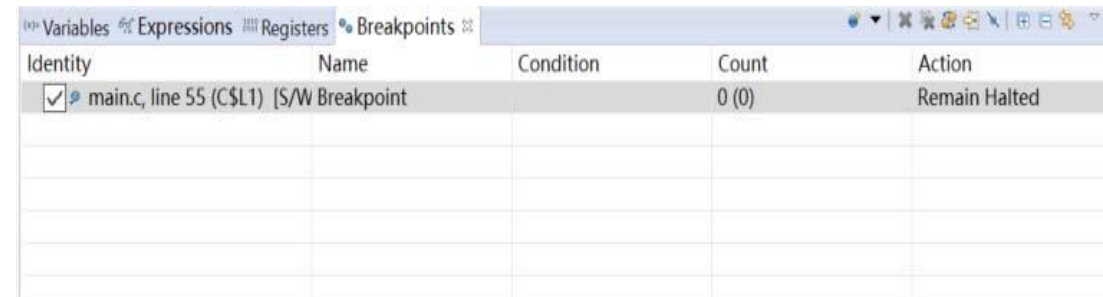
在 CCS Debug 环境界面的主菜单中选择“Run → Resume”可以让 DSP 从 main 函数的第一条语句开始执行程序。由于 DSP 程序输出并不具备 GUI 界面，由此执行结果只有依赖外部硬件或者查看寄存器、存储器的数值加以验证。在主菜单中选择“Run → Suspend”，可以暂停程序的执行。DSP 指令的执行严格按照指令流的顺序。当想再次运行程序，可以执行菜单命令“Run → Restart”，使程序指针 PC 重新指向_c_int00，也可以重新加载程序（“Run → Load → Reload Program”）。当执行菜单命令“Run → Reset”时，DSP 复位，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。

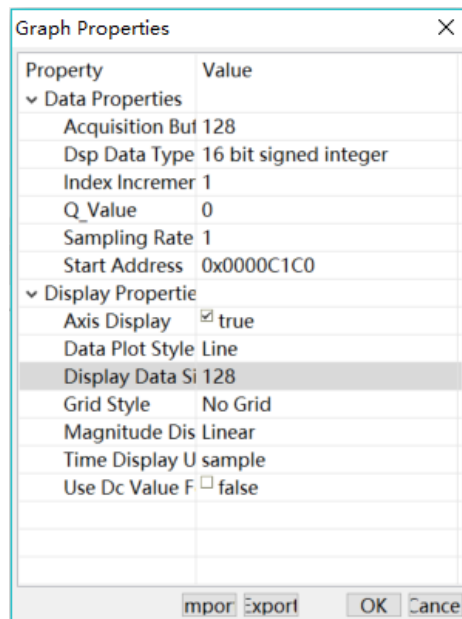
在 dataIO()处设立断点，在断点属性中关联输入文件 sine.dat，设置数据加载的起始地址为 currentBuffer.input，长度为 128。



Identity	Name	Condition	Count	Action
<input checked="" type="checkbox"/> main.c, line 55 (C\$L1) [S/W Breakpoint]			0 (0)	Remain Halted

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。



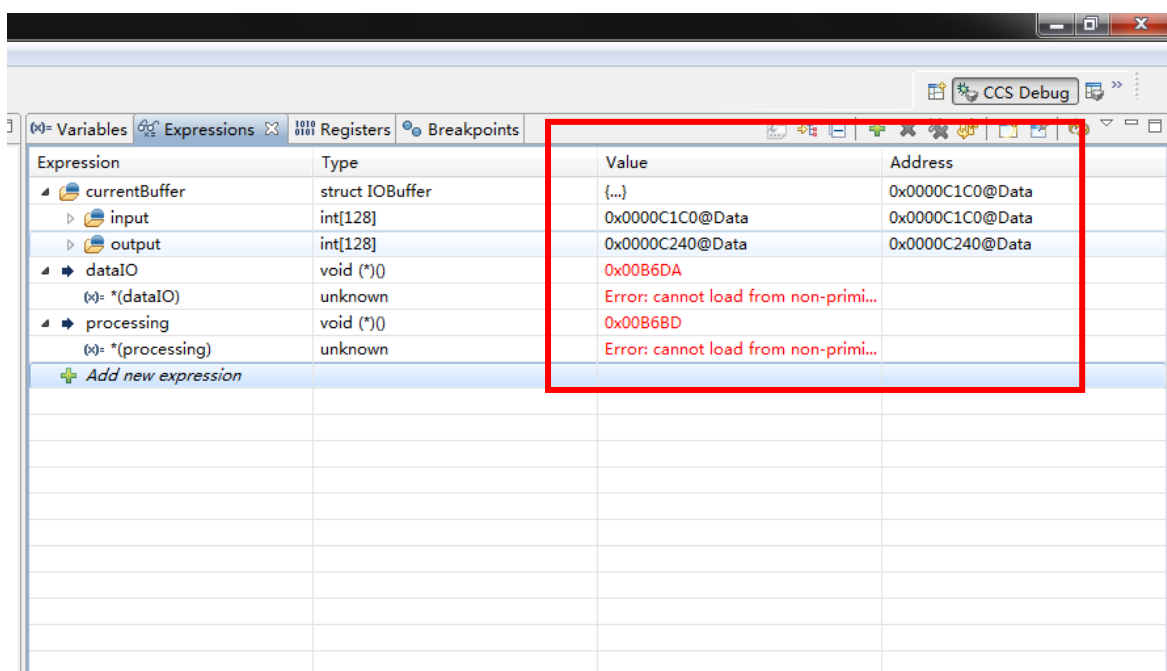
查看存储空间的数值在程序相关语句执行前后的变化。

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別。

4 实验结果汇总及问题回答

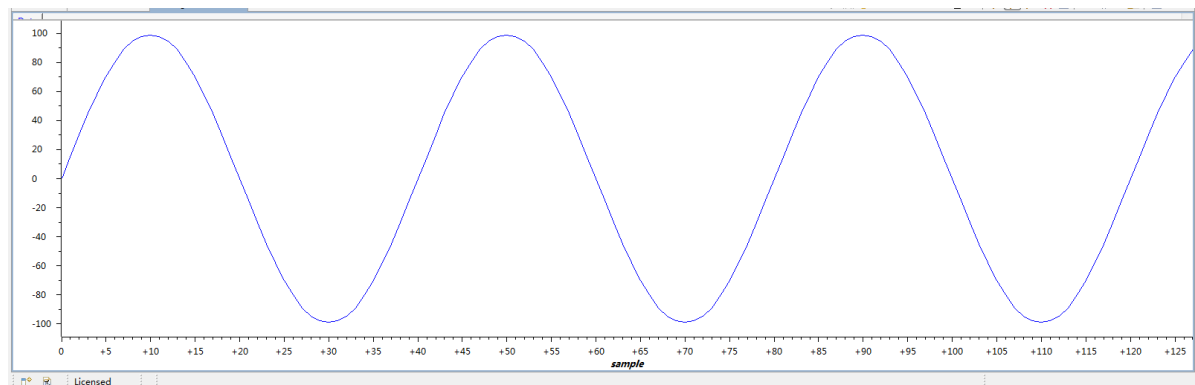
4.1 子程序入口地址与结构体存储地址

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



Expression	Type	Value	Address
currentBuffer	struct IOBuffer	{...}	0x0000C1C0@Data
input	int[128]	0x0000C1C0@Data	0x0000C1C0@Data
input[0 ... 99]			
input[0]	int	0	0x0000C1C0@Data
input[1]	int	15	0x0000C1C1@Data
input[2]	int	30	0x0000C1C2@Data
input[3]	int	45	0x0000C1C3@Data
input[4]	int	58	0x0000C1C4@Data
input[5]	int	70	0x0000C1C5@Data
input[6]	int	80	0x0000C1C6@Data
input[7]	int	89	0x0000C1C7@Data
input[8]	int	95	0x0000C1C8@Data
input[9]	int	98	0x0000C1C9@Data
input[10]	int	99	0x0000C1CA@Data
input[11]	int	98	0x0000C1CB@Data
input[12]	int	95	0x0000C1CC@Data
input[13]	int	89	0x0000C1CD@Data
input[14]	int	80	0x0000C1CE@Data
input[15]	int	70	0x0000C1CF@Data
input[16]	int	58	0x0000C1D0@Data
input[17]	int	45	0x0000C1D1@Data
input[18]	int	30	0x0000C1D2@Data
input[19]	int	15	0x0000C1D3@Data
input[20]	int	0	0x0000C1D4@Data
input[21]	int	-15	0x0000C1D5@Data
input[22]	int	-30	0x0000C1D6@Data
input[23]	int	-45	0x0000C1D7@Data
input[24]	int	-58	0x0000C1D8@Data
input[25]	int	-70	0x0000C1D9@Data
input[26]	int	-80	0x0000C1DA@Data
input[27]	int	-89	0x0000C1DB@Data

4.2 显示缓冲存储器中的波形



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

```

文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)
*****
TMS320C2000 Linker PC v6.1.0
*****
>> Linked Tue Jul 28 02:28:05 2015

OUTPUT FILE NAME:  <e99.out>
ENTRY POINT SYMBOL: "_c_int00" address: 0000b621

MEMORY CONFIGURATION

      name          origin      length      used      unused      attr      fill
-----
PAGE 0:
BEGIN              00000000      00000002      00000002      00000000      RWIX
RAMM0              00000050      000003b0      00000000      000003b0      RWIX
RAML0              00008000      00001000      000000af      00000f51      RWIX
RAML1              00009000      00003000      00002a2e      000005d2      RWIX
ZONE7A            00200000      0000fc00      00000000      0000fc00      RWIX
CSM_RSVD          0033ff80      00000076      00000000      00000076      RWIX
CSM_PWL           0033fff8      00000008      00000000      00000008      RWIX
ADC_CAL           00380080      00000009      00000007      00000002      RWIX
IQTABLES          003fe000      00000b50      00000000      00000b50      RWIX
IQTABLES2         003feb50      0000008c      00000000      0000008c      RWIX
FPUTABLES         003febd0      000006a0      00000000      000006a0      RWIX
BOOTROM           003ff27c      00000d44      00000000      00000d44      RWIX
RESET             003fffc0      00000002      00000000      00000002      RWIX

PAGE 1:
BOOT_RSVD         00000002      0000004e      00000000      0000004e      RWIX
RAMM1             00000400      00000400      00000400      00000000      RWIX
DEV_EMU           00000880      00000180      000000d0      000000b0      RWIX
FLASH_REGS        00000a80      00000060      00000008      00000058      RWIX
CSM               00000ae0      00000010      00000010      00000000      RWIX
ADC_MIRROR        00000b00      00000010      00000010      00000000      RWIX
XINTF             00000b20      00000020      0000001e      00000002      RWIX
CPU_TIMER0        00000c00      00000008      00000008      00000000      RWIX
CPU_TIMER1        00000c08      00000008      00000008      00000000      RWIX
CPU_TIMER2        00000c10      00000008      00000008      00000000      RWIX
PIE_CTRL          00000ce0      00000020      0000001a      00000006      RWIX
PIE_VECT          00000d00      00000100      00000100      00000000      RWIX
DMA               00001000      00000200      000000e0      00000120      RWIX
MCBSPA            00005000      00000040      00000025      0000001b      RWIX
MCBSPB            00005040      00000040      00000025      0000001b      RWIX
ECANA             00006000      00000040      00000034      0000000c      RWIX
ECANA_LAM         00006040      00000040      00000040      00000000      RWIX
ECANA_MOTS        00006080      00000040      00000040      00000000      RWIX
ECANA_MOTO        000060c0      00000040      00000040      00000000      RWIX
ECANA_MBOX        00006100      00000100      00000100      00000000      RWIX

```

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。

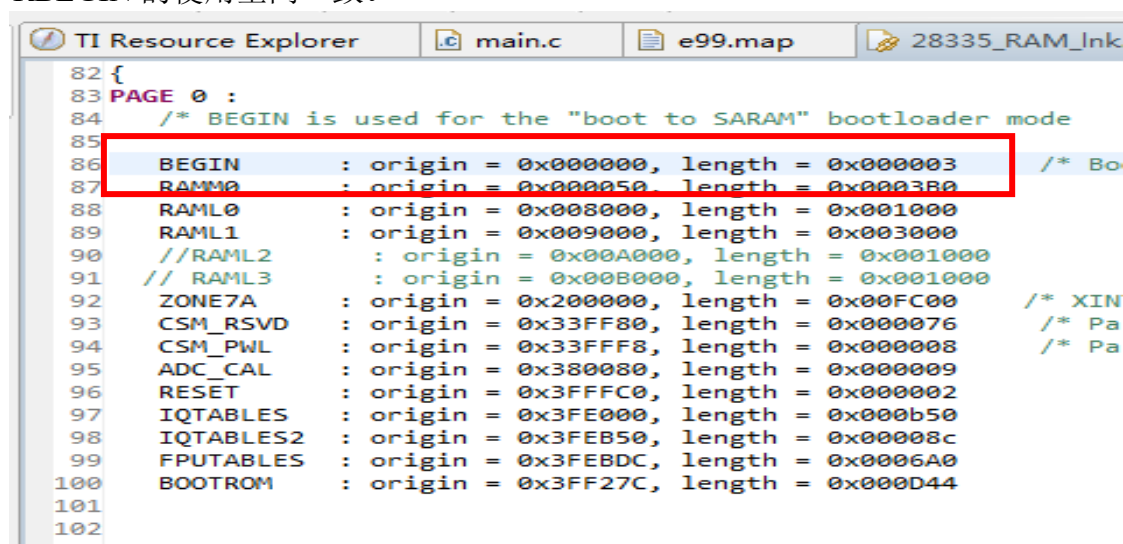
表 4.1 .map 文件各段信息

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
.systemem	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。



```

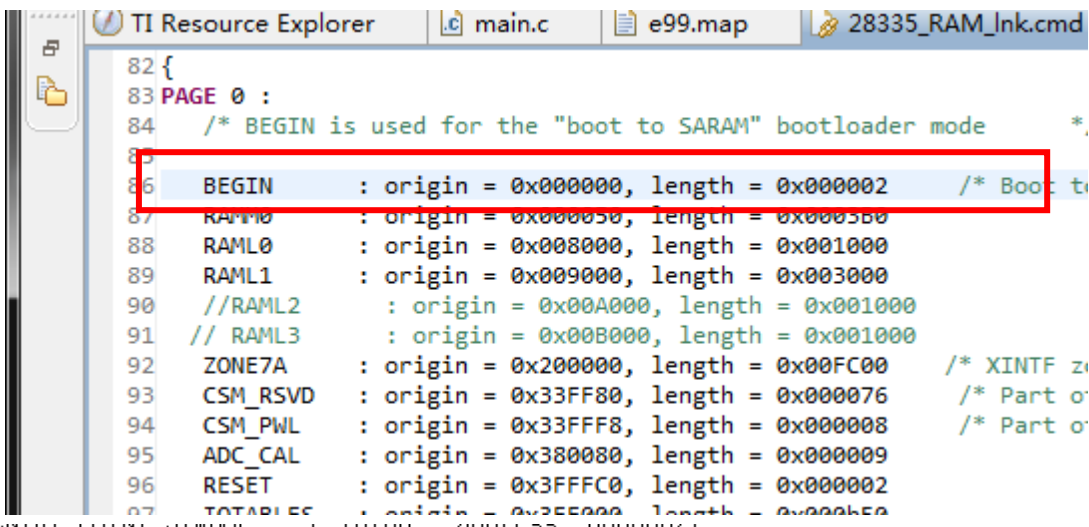
82 {
83 PAGE 0 :
84 /* BEGIN is used for the "boot to SARAM" bootloader mode
85
86 BEGIN : origin = 0x000000, length = 0x000003 /* Bo
87 RAMM0 : origin = 0x000050, length = 0x000380
88 RAML0 : origin = 0x008000, length = 0x001000
89 RAML1 : origin = 0x009000, length = 0x003000
90 //RAML2 : origin = 0x00A000, length = 0x001000
91 // RAML3 : origin = 0x00B000, length = 0x001000
92 ZONE7A : origin = 0x200000, length = 0x00FC00 /* XIN
93 CSM_RSVD : origin = 0x33FF80, length = 0x000076 /* Pa
94 CSM_PWL : origin = 0x33FFF8, length = 0x000008 /* Pa
95 ADC_CAL : origin = 0x380080, length = 0x000009
96 RESET : origin = 0x3FFFC0, length = 0x000002
97 IQTABLES : origin = 0x3FE000, length = 0x000b50
98 IQTABLES2 : origin = 0x3FEB50, length = 0x00008c
99 FPUTABLES : origin = 0x3FEBDC, length = 0x00006A0
100 BOOTROM : origin = 0x3FF27C, length = 0x000D44
101
102
103 PAGE 1 :

```

name	origin	length	used	unused	attr	fill
PAGE 0:						
BEGIN	00000000	00000003	00000002	00000001	RWIX	
RAMM0	00000050	000003b0	00000000	000003b0	RWIX	
RAML0	00008000	00001000	000000af	00000f51	RWIX	
RAML1	00009000	00003000	00002a2e	000005d2	RWIX	
ZONE7A	00200000	0000fc00	00000000	0000fc00	RWIX	
CSM_RSVD	0033ff80	00000076	00000000	00000076	RWIX	
CSM_PWL	0033fff8	00000008	00000000	00000008	RWIX	
ADC_CAL	00380080	00000009	00000007	00000002	RWIX	
IQTABLES	003fe000	00000b50	00000000	00000b50	RWIX	
IQTABLES2	003feb50	0000008c	00000000	0000008c	RWIX	
FPUTABLES	003febd0	000006a0	00000000	000006a0	RWIX	
BOOTROM	003ff27c	00000d44	00000000	00000d44	RWIX	
RESET	003fffc0	00000002	00000000	00000002	RWIX	

TABLE 4

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。



TI Resource Explorer | main.c | e99.map | 28335_RAM_Ink.cmd

```

82 {
83 PAGE 0 :
84 /* BEGIN is used for the "boot to SARAM" bootloader mode */
85
86 BEGIN : origin = 0x000000, length = 0x000002 /* Boot to
87
88 RAMM0 : origin = 0x000050, length = 0x0003b0
89 RAML0 : origin = 0x008000, length = 0x001000
90 RAML1 : origin = 0x009000, length = 0x003000
91 //RAML2 : origin = 0x00A000, length = 0x001000
92 // RAML3 : origin = 0x00B000, length = 0x001000
93 ZONE7A : origin = 0x200000, length = 0x00FC00 /* XINTF z
94 CSM_RSVD : origin = 0x33FF80, length = 0x000076 /* Part o
95 CSM_PWL : origin = 0x33FFF8, length = 0x000008 /* Part o
96 ADC_CAL : origin = 0x380080, length = 0x000009
97 RESET : origin = 0x3FFFC0, length = 0x000002
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

MEMORY CONFIGURATION

name	origin	length	used	unused	attr	fill
PAGE 0:						
BEGIN	00000000	00000002	00000002	00000000	RWIX	
RAMM0	00000050	000003b0	00000000	000003b0	RWIX	
RAML0	00008000	00001000	000000af	00000f51	RWIX	
RAML1	00009000	00003000	00002a2e	000005d2	RWIX	
ZONE7A	00200000	0000fc00	00000000	0000fc00	RWIX	
CSM_RSVD	0033ff80	00000076	00000000	00000076	RWIX	
CSM_PWL	0033fff8	00000008	00000000	00000008	RWIX	
ADC_CAL	00380080	00000009	00000007	00000002	RWIX	
IQTABLES	003fe000	00000b50	00000000	00000b50	RWIX	
IQTABLES2	003feb50	0000008c	00000000	0000008c	RWIX	
FPUTABLES	003febd0	000006a0	00000000	000006a0	RWIX	
BOOTROM	003ff27c	00000d44	00000000	00000d44	RWIX	
RESET	003fffc0	00000002	00000000	00000002	RWIX	

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

2. build 时会出现编译错误，显示未找到 student.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。

3. 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。

4. 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。

5. 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，原因是病毒的出现，在重启电脑后解决了此问题。

6. 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， $\text{bufferinput}(127)=70$ ，乘以-4 后为-280，正好等于 $\text{bufferoutput}(127)$ 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。