DSP应用实验实验报告

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

学号：9171040G0501

姓名：白鸽

指导老师：李彧晟

2020年11月28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

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，打开计算机和实验箱电源。

2.点击桌面 CCS5 快捷方式，启动 CCS 集成开发环境。

3.按照实验讲义，首先熟悉 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：

将“LAB 9”工程文件添加至目录。

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。

在 CCS Debug 环境界面的主菜单中选择“Run→Resume”运行程序。

5.添加结构体变量至观察窗口：

选中变量 currentBuffer.input，点击右键，在下拉菜单中点击“Add Watch

Expression…”即可添加变量至观察窗口来观察变量的类型、内容和地址。同样操作，将变量 currentBuffer.output 添加至观察窗口，同时将 dataIO( )到变量窗口，查看该子程序的入口地址。观察窗口截图如图 1 所示：

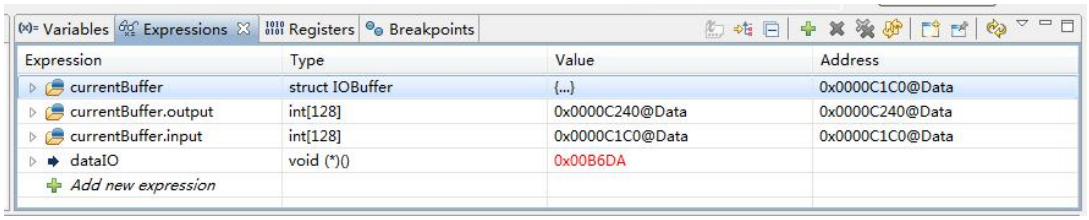


图 1 变量观察窗口

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。记录下两个变量和子程序的地址，以供后续操作查看数据绘制图像使用。

6.设置断点，关联输入文件

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 2 所示：

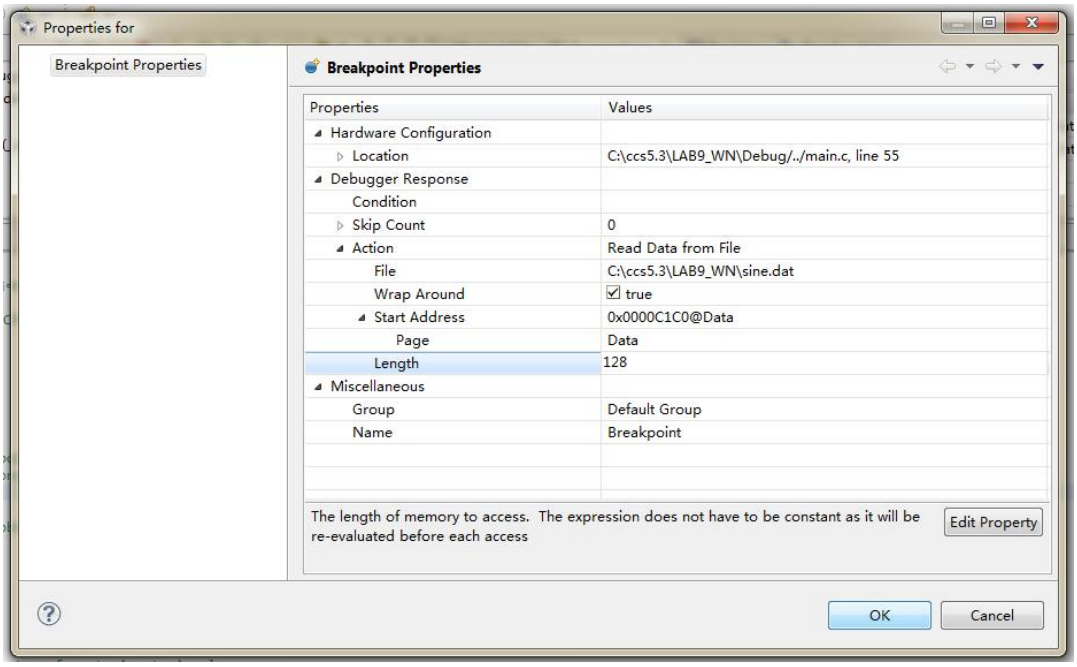


图 2 断点及关联数据设置

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”。设置各项参数如表 1 所示：

Property	Value
Data Properties	
Acquisition Buffer Size	128
Dsp Data Type	16 bit signed integer
Idex Increment	1
Q_Value	0
Sampling Rate Hz	1
Start Address	0x0000C1C0@Data
Display Properties	
Axis Display	√true
Data Plot Style	Line
Display Data Size	128
Grid Style	No Grid
Mognitude Display Sc.	Linear
Time Display Unit	sample
Use De Value For Grap	false

表 1 图形绘制参数设置

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：

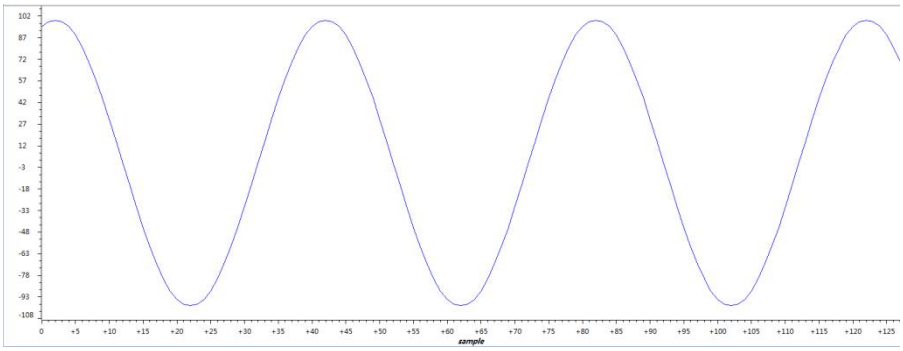


图 4 数据空间 currentBuffer.input 缓冲存储器中的波形

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：

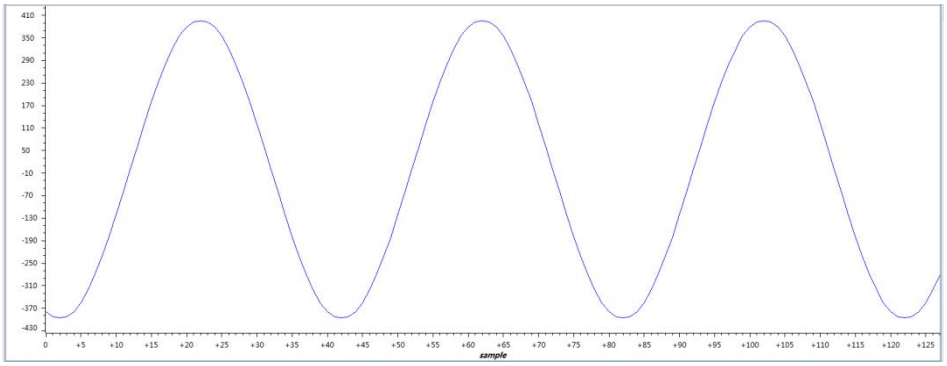


图 5 数据空间 currentBuffer.output 缓冲存储器中的波形

可见，数据空间 currentBuffer.input 和 currentBuffer.output 缓冲存储器中的波形都为正弦波，只是 currentBuffer.output 相比 currentBuffer.input 中波形产生了一定的时延。

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 5 所示。联系 cmd 文件中 section 指令即可了解该段所在地址。如图 6 所示：

```
86
87
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)
116                   00009911    000003f4     DSP2833x_DMA.obj (.text)
117                   00009d05    00000323     DSP2833x_DefaultIsr.obj (.text:retain)
118                   0000a028    00000274     DSP2833x_Lcd.obj (.text)
119                   0000a29c    0000024b     rts2800_fpu32.lib : lowlev.obj (.text)
120                   0000a4e7    00000201                       : trgdrv.obj (.text)
```

图 5 section location map

```
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

   /* Uncomment the section below if calling the IQNexp() or IQexp()
      functions from the IQMath.lib library in order to utilize the
      relevant IQ Math table in Boot ROM (This saves space and Boot ROM
      is 1 wait-state). If this section is not uncommented, IQmathTables2
      will be loaded into other memory (SARAM, Flash, etc.) and will take
      up space, but 0 wait-state is possible.
   */
```

图 6 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：

由上文易知 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 7 所示：

```
79  I2CA                 00007900   00000040   00000022   0000001e   RWIX
80  RAML4                0000c000   00001000   00000483   00000b7d   RWIX
81  RAML5                0000d000   00001000   0000025a   00000da6   RWIX
82  RAML6                0000e000   00001000   00000300   00000d00   RWIX
83  RAML7                0000f000   00001000   00000000   00001000   RWIX
```

图 7 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，即可得出 currentBuffer 变量在 RAML4 中。如图 8 所示：

```
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
```

图 8 cmd 文件 section

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两图所示。

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。

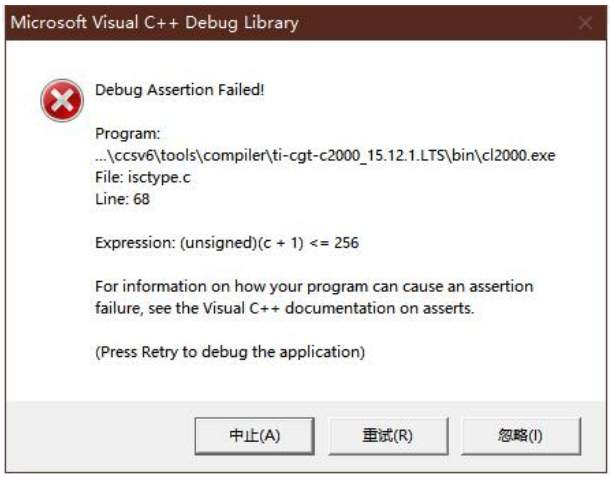
MAP 文件大概分为文件头、内存配置、段映射、全局符号四部分。内存配置与 CMD 文件中的 MEMORY 指令关联，在 CMD 文件中定义的程序与数据区间定义，在该部分均可以找到对应，与 CMD 文件不同的时，在 MAP 文件中加入了一个实际使用的区间，即在程序中实际用到的空间长度。段映射部分与 CMD 文件中的 SECTION 指令关联，在该部分程序中所有的段实际映射的起始地址与实际长度均有详细说明。可以具体到程序中#pragma 指定的段和各个单独文件产生的 OBJ 文件。

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 ebss 段命令志向 RAML5 空间，该段的起始地址将变为 0x00D000。

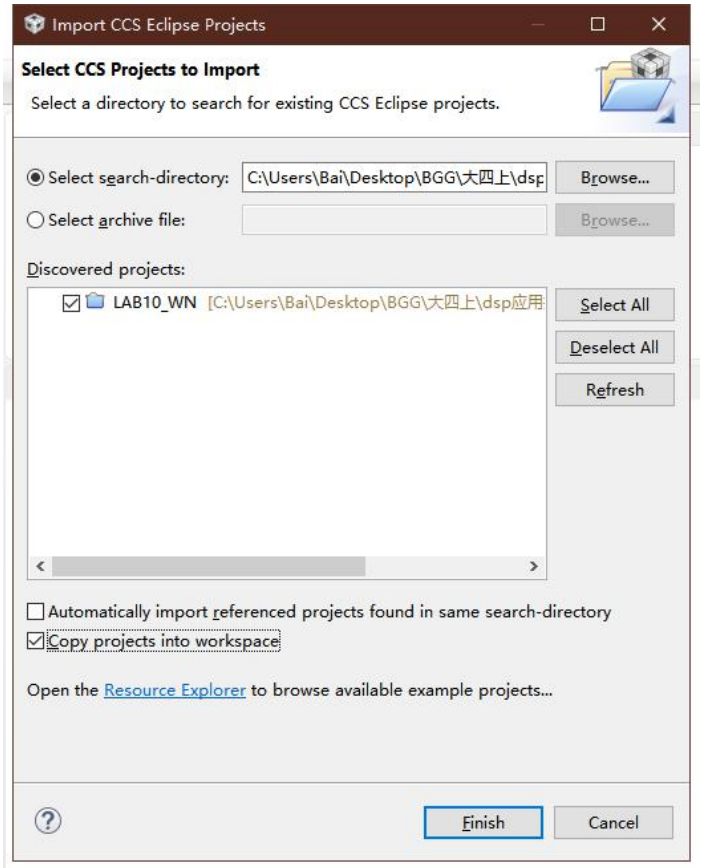
9.5 实验总结

通过本次实验，首次真正接触到了 DSP 开发板和 CCS 开发软件。在刚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.在添加工程文件时，出现了如下报错



在一次尝试中，如下图，添加工程文件时勾选“copy projrcts into workspace”就解决了该问题。



2. 在刚连上 DSP 开发板编译时会报错，但由于本次实验主程序代码是由老师提供的正确代码，故不会是程序编写的问题，而只能是之前操作步骤出了问题。仔细检查回忆此前操作步骤，认为并无错误，于是决定上网查询报错原因及解决方法，他人提供的方案是尝试重新启动软件，重启几次之后就解决了该问题，程序可以正常运行。

3. 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，检查 sine.data 文件，观察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，是在 debug 之后忘记进行 Resume 操作，程序没有完全执行，改正之后就得到了正确波形。

通过本次实验，熟悉了 CCS 的操作环境及调试方法，为接下来几次实验奠定了基础。总体来说，本次实验不涉及程序编写，只是简单的操作过程，因此较为简单。希望在接下来几次实验中能够继续深入学习 DSP 芯片开发过程中的其他方面。