南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

DSP 应用技术

——DSP 开发基础实验

作　者：	杨文歆
学　号：	9151040G0212
学　院：	电子工程与光电技术学院
专　业：	电子信息工程
指导老师：	李彧晟
实验日期：	2018 年 11 月

目录

实验九 DSP 开发基础实验

一、 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。

二、实验仪器

计算机，C2000DSP 教学实验箱，XDS510USB 仿真器

三、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。

四、实验准备

CCS 2(C2000)这一集成开发环境,不仅支持汇编的编译、链接,还支持对C/C++汇编、编译、链接以及优化。同时强大的 IDE 开发环境也为代码的调试提供了强大的功能支持,已经成为 TI 各 DSP 系列的程序设计、制作、调试、优化的主流工具。

TMS320C28x 软件开发流程如下图所示：

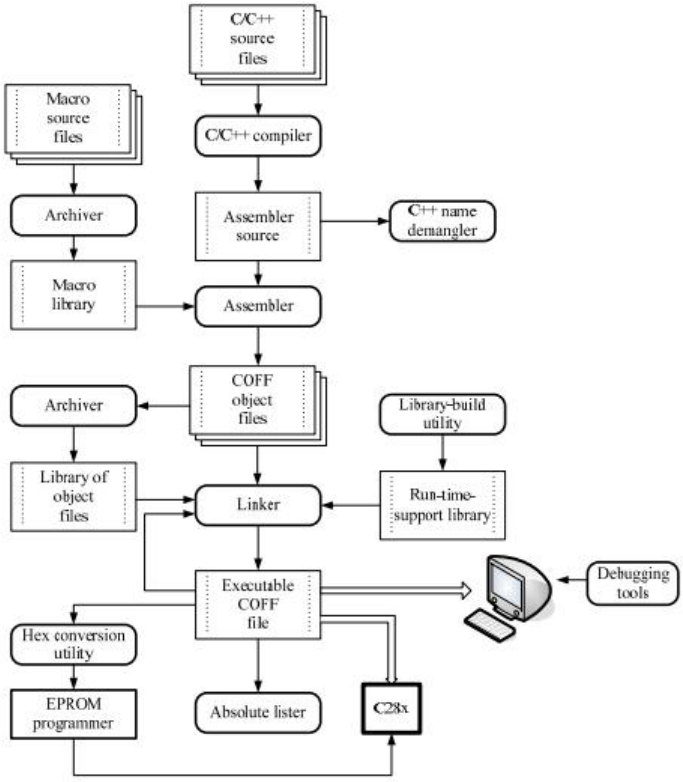

图 1　TMS320C28x 软件开发流程

下面简单介绍各主要模块功能：

（1）C/C++ Compiler(C/C++编译器)

C/C++编译器把 C/C++程序自动转换成 C28x 的汇编语言源程序。这种转换并非一一对应,甚至会产生冗余的汇编代码,在某些场合需要使用优化器(Optimizer)来提高转换的效率,使得汇编代码长度尽可能的短小,程序所使用的资源尽可能的少。优化器是编译器的一部分。

（2）Assembler(汇编器)

汇编器负责将汇编源程序转换为符合公共目标格式(COFF)的机器目标代码,这种转换是一一对应的,每一条汇编指令都对应了唯一的机器代码。源文件中还包括汇编指令、伪指令和宏指令。

（3）Linker(链接器)

链接器负责把可重定位的多个目标文件和目标库文件转换为一个 DSP 可执行程序。链接器必须依赖配置命令文件(CMD)的指令,实现对目标文件中各段的定位。

（4）Run-time-support library(运行支持库)

函数运行支持库包含有 ANSI/ISO C 的标准运行支持库函数、编译器功能函数、浮点算术函数和系统初始化子程序(这些函数都集成在汇编源文件 rts.src 中)。

五、实验步骤

（1）设备检查
（2）启动集成开发环境
（3）新建工程
（4）添加工程文件
（5）查阅代码
（6）建立工程(Build 工程)
（7）加载程序
（8）程序的运行
（9）程序的调试

六、实验结果

1、实验箱测试

（1）打开示波器，信号发生器，调节信号源输出，幅度控制在±0.5V 以内。

（2）将信号源输出端接至实验箱 INPUT1，将实验箱 OUT3、OUT2 分别连接至示波器。

（3）打开实验箱电源，检查电源指示灯是否点亮。

（4）点击桌面 CCS2（C2000），进入 F2812 的集成开发环境。

（5）加载 test.out 程序，并运行（Run），查看数码管显示、LED 闪烁、示波器两路输出信号。

最终用 CCS2（C2000）运行测试程序，观察到数码管显示数字“12345678”，LED 灯不断闪烁，同时示波器上显示出由信号源传来的两路输出信号，实验结果是实验箱工作完全正常，无异常现象。

2、C 程序调试

（1）记录 dataIO()、processing()子程序的入口地址,记录 currentBuffer.input

和 currentBuffer.output 所在存储器地址。

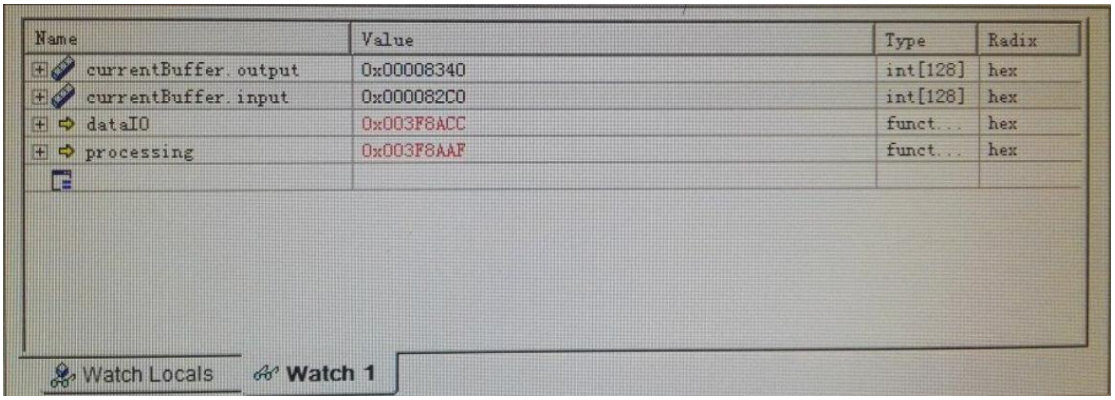

图 2 dataIO()、processing()、currentBuffer 变量观察窗口

由 CCS 中的变量观察窗口可以清楚看到 dataIO()、processing()子程序的入口地址分别为 0x003F8ACC 和 0x003F8AAF，currentBuffer.input 和 currentBuffer.output 所在存储器地址分别为 0x00008340 和 0x000082C0。

（2）记录增益控制处理后，以图形方式显示数据空间 currentBuffer.input 和 currentBuffer.output 缓冲存储器中的波形。

数据空间 currentBuffer.input 和 currentBuffer.output 缓冲存储器中的波形，通过 CCS 程序显示如下图所示：

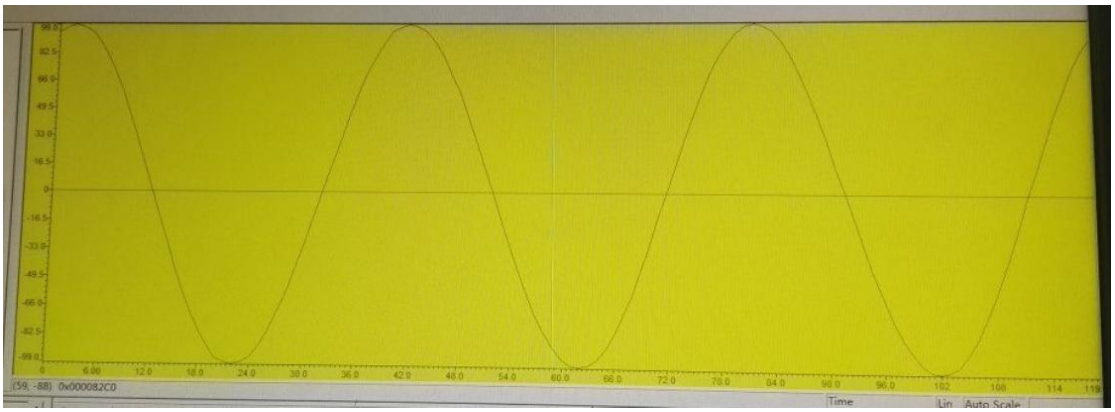

图 3 currentBuffer.input 缓冲存储器波形

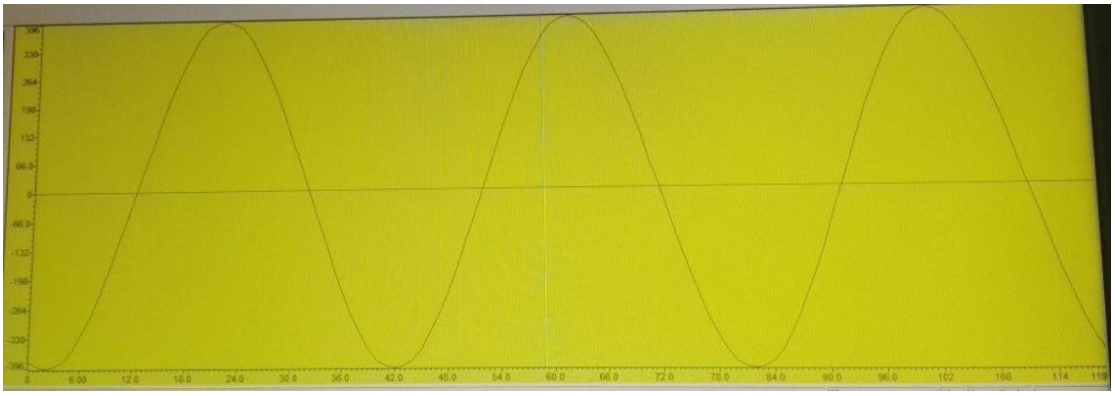

图 4 currentBuffer.output 缓冲存储器波形

可以看到，input 波形和 output 波形是反相的。这是因为输入的 currentBuffer.input 缓冲存储器波形是从.dat 文件中读来的，而 currentBuffer.output

缓冲存储器波形是处理函数处理的最后一组数据。

（3）打开工程的.map 文件,查看.text、.data、.bss 段在存储空间的地址和长度,指出分别位于 TMS320F2812 的什么存储空间以及物理存储块名称。

SECTION ALLOCATION MAP

output section	page	origin	length	attributes/ input sections
.pinit	0	003f8080	00000000	UNINITIALIZED
.text	0	003f8080	00000adc	
		003f8080	00000232	rts2800_ml.lib : lowlev.obj (.text)
		003f82b2	000001f8	: trgdrv.obj (.text)
		003f84aa	000001d5	: memory.obj (.text)
		003f867f	000000dd	: fopen.obj (.text)
		003f875c	0000009a	: fputs.obj (.text)
		003f87f6	0000007c	: ankmsg.obj (.text)
		003f8872	0000006e	: _io_perm.obj (.text)
		003f88e0	00000061	: setvbuf.obj (.text)
		003f8941	0000004e	: fflush.obj (.text)
		003f898f	0000004b	: exit.obj (.text)
		003f89da	00000046	: boot.obj (.text)
		003f8a20	00000040	: memcpy.obj (.text)
		003f8a60	0000003b	: fclose.obj (.text)
		003f8a9b	00000032	sine.obj (.text)
		003f8acd	00000030	rts2800_ml.lib : fseek.obj (.text)
		003f8afd	00000018	: strncpy.obj (.text)
		003f8b15	0000000e	: memchr.obj (.text)
		003f8b23	0000000a	: strlen.obj (.text)
		003f8b2d	00000009	: _lock.obj (.text)
		003f8b36	00000009	: memset.obj (.text)
		003f8b3f	00000009	: strchr.obj (.text)
		003f8b48	00000009	: strcpy.obj (.text)
		003f8b51	00000008	: strcmp.obj (.text)
		003f8b59	00000003	: remove.obj (.text)

```
.cinit     0    003f8b5c    00000146
                003f8b5c    000000f7    rts2800_ml.lib : defs.obj (.cinit)
                003f8c53    0000002d                   : lowlev.obj (.cinit)
                003f8c80    0000000e                   : exit.obj (.cinit)
                003f8c8e    0000000a                   : _lock.obj (.cinit)
                003f8c98    00000004                   : memory.obj (.cinit)
                003f8c9c    00000004    sine.obj (.cinit)
                003f8ca0    00000002    --HOLE-- [fill = 0]

.reset     0    003fffc0    00000002    DSECT
                003fffc0    00000002    rts2800_ml.lib : boot.obj (.reset)

.sysmem    1    00000000    00000400    UNINITIALIZED
                00000000    00000400    --HOLE--

.bss       1    00000400    00000000    UNINITIALIZED

.stack     1    00000400    00000400    UNINITIALIZED
                00000400    00000400    --HOLE--

.ebss      1    00008000    000006c0    UNINITIALIZED
                00008000    00000280    rts2800_ml.lib : defs.obj (.ebss)
                00008280    00000140    sine.obj (.ebss)
                000083c0    00000110    rts2800_ml.lib : lowlev.obj (.ebss)
                000084d0    00000008                   : memory.obj (.ebss)
                000084d8    00000004                   : _lock.obj (.ebss)
                000084dc    00000024    --HOLE--
                00008500    00000108                   : trgdrv.obj (.ebss)
                00008608    00000038    --HOLE--
                00008640    00000080                   : exit.obj (.ebss)


.econst    1    000086c0    0000001c
                000086c0    0000001a    sine.obj (.econst:.string)
                000086da    00000002    rts2800_ml.lib : fputs.obj (.econst)

.cio       1    00008700    00000120    UNINITIALIZED
                00008700    00000120    rts2800_ml.lib : ankmsg.obj (.cio)
```

图 5 .map 文件信息

Section	page	origin	length	物理存储块
.text	0	003f8080	00000adc	PROG
.data	1	00000000	00000000	M0RAM
.bss	1	00000400	00000000	M1RAM

另外，由于实验时间限制没有对其他块进行实验。通过网上查找资料，得到其他段的信息如下表：

Section	page	origin	length	作用	物理存储块
.pinit	0	003f8090	00000000	调用全局对象构造函数的表	FLASH
.cinit	0	003f8b6c	00000146	变量和常数初始化表	FLASH
.reset	0	003fffc0	00000002		RAM1
.sysmem	1	00000000	00000400	为动态存储分配保留的空间	RAM
.stack	1	00000400	00000400	分配系统堆栈空间	Lower 64Kw RAM
.ebss	1	00008000	000006c0	为使用大寄存器模式时的全局变量和静态变量预留的空间	RAM
.econst	1	000086c0	0000001c	字符串常量和数据（不包括 volatile）	FLASH
.cio	1	00008700	00000120	用于 stdio 函数	RAM

（4）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。

sinewave.cmd - 记事本

```
/*****************************************************************************/

MEMORY
{
   PAGE 0 : BOOT(R)     : origin = 0x3f8000, length = 0x80
   PAGE 0 : PROG(R)     : origin = 0x3f8080, length = 0x1f80
   PAGE 0 : RESET(R)    : origin = 0x3fffc0, length = 0x2


   PAGE 1 : M0RAM(RW)   : origin = 0x000000, length = 0x400
   PAGE 1 : M1RAM(RW)   : origin = 0x000400, length = 0x400
   PAGE 1 : L0L1RAM(RW) : origin = 0x008000, length = 0x2000

}

SECTIONS
{
   /* 22-bit program sections */
   .reset   : > RESET, PAGE = 0, TYPE = DSECT
```

shiyan1.map - 记事本

```
MEMORY CONFIGURATION

         name            origin    length     used     unused   attr    fill
----------------------  --------  ---------  --------  --------  ----  --------
PAGE 0:
  BOOT                  003f8000  00000080  00000000  00000080  R
  PROG                  003f8080  00001f80  00000c22  0000135e  R
  RESET                 003fffc0  00000002  00000000  00000002  R

PAGE 1:
  M0RAM                 00000000  00000400  00000400  00000000  RW
  M1RAM                 00000400  00000400  00000400  00000000  RW
  L0L1RAM               00008000  00002000  000007fc  00001804  RW


SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin     length       input sections
--------  ----  ---------- ----------   ----------------
```

sinewave.cmd - 记事本

```
SECTIONS
{
   /* 22-bit program sections */
   .reset   : > RESET, PAGE = 0, TYPE = DSECT
   .pinit   : > PROG,  PAGE = 0
   .cinit   : > PROG,  PAGE = 0
   .text    : > PROG,  PAGE = 0


   /* 16-Bit data sections */
   .const   : > M0RAM, PAGE = 1
   .bss     : > M1RAM, PAGE = 1
   .stack   : > M1RAM, PAGE = 1
   .sysmem  : > M0RAM, PAGE = 1



   /* 32-bit data sections */
   .ebss    : > L0L1RAM, PAGE = 1
```

shiyan1.map - 记事本

```
.cinit    0   003f8b5c   00000146
              003f8b5c   000000f7    rts2800_ml.lib : defs.obj (.cinit)
              003f8c53   0000002d                   : lowlev.obj (.cinit)
              003f8c80   0000000e                   : exit.obj (.cinit)
              003f8c8e   0000000a                   : _lock.obj (.cinit)
              003f8c98   00000004                   : memory.obj (.cinit)
              003f8c9c   00000004    sine.obj (.cinit)
              003f8ca0   00000002    --HOLE-- [fill = 0]

.reset    0   003fffc0   00000002    DSECT
              003fffc0   00000002    rts2800_ml.lib : boot.obj (.reset)

.sysmem   1   00000000   00000400    UNINITIALIZED
              00000000   00000400    --HOLE--

.bss      1   00000400   00000000    UNINITIALIZED

.stack    1   00000400   00000400    UNINITIALIZED
              00000400   00000400    --HOLE--
```

图 6 .cmd 和.map 地址对比

在记事本中打开.cmd 文件和.map 文件，可以观察到地址有一一映射的关系，如上图所示。

进一步的，如果在.cmd 文件中更改 PROG 的起始地址为 0x3f8090，运行程序后，.map 文件中 PROG 的地址也发生相应的变化，即变为 0x3f8090：

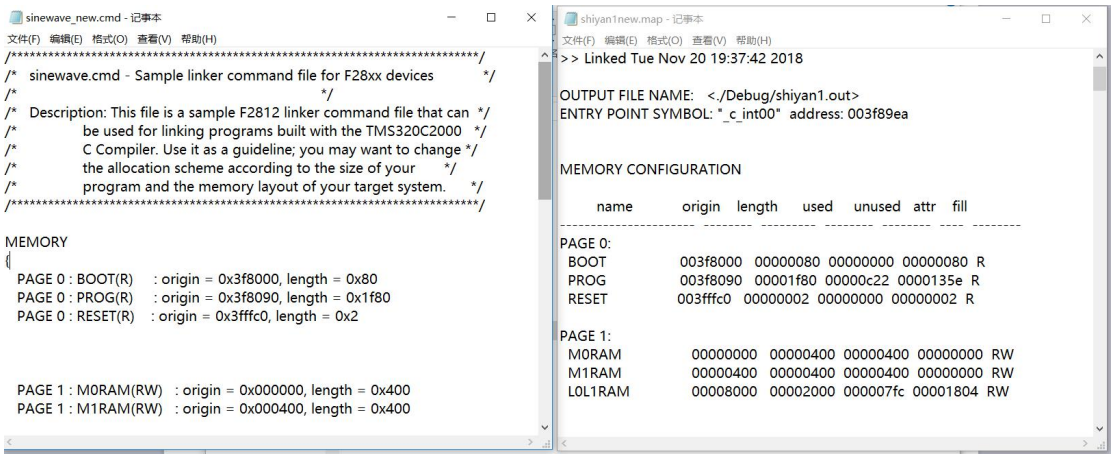

图 7 .cmd 和.map 地址对比

在此基础上修改.cmd 文件中的一些地址，则能得出以下结论：

（1）在.cmd 文件中，

PAGE 1：M1RAM（RW） origin=0x000400，length=0x400

将上面的 M1RAM（RW）的值改为 origin=0x000500 时，.bss 的起始位置变为如下所示：

.bss 1 00000500 00000000 UNINITIALIZED

这说明.bss 的存储空间应该在 M1RAM（RW）内。

（2）由.cmd 命令文件中的语句：

PAGE0：PROG（R）： origin=0x3f8080，length=0x1f80

再由.map 文件中

.text 0 003f8080 00000adc

根据其地址及长度推断，.text 的存储空间为 PROG（R）。

七、实验总结

通过本次 DSP 实验我学会了 DSP 的基本配置。和《电子信息工程课程设计》不同，《电子信息工程课程设计》侧重于使用 MATLAB 进行软件上的处理，对实验箱的操作只有配置、编译和下载等。而本次 DSP 实验则侧重于配置，并且熟悉 DSP 集成开发环境（CCS），掌握 C 语言开发的基本流程，熟悉代码调试的基本方法。从基础开始学习 DSP 的开发环境，了解它的特性，这是一个想要掌握 DSP 必不可少的阶段。在 C 程序的调试中，我们学会了查看子程序的入口地址和相关存储器的地址，学习了进行一些参数的设置然后将一段存储器中连贯地址的数据用图形显示功能显示。在助教的帮助下我们学会了对比.cmd 文件和.map 文件，看懂了.cmd 文件和.map 文件中一些地址的映射关系，并且能够找出其中.text、.data、.bss 段在存储空间的地址和长度。最后我们通过修改.cmd 文件中的地址，再次编译链接后，可以观察到.map 文件中的对应地址也发生变化。

在 DSP 实验中我们遇到了不少问题。往往从无到有的过程是最艰难的，最开始的配置就使我们举步维艰。还有“connect”、“添加文件”等现在看来十分简单的操作，在第一节 DSP 实验课上耗费了我们不少时间。我们发现.cmd 文件和.map 文件中的 length 不是一致的，在助教的帮助下我们明白了.cmd 的某个块会划分成许多部分。另外，我们因为对实验要求理解不准确，做了很多无意义的操作。现在已经结束第二节 DSP 实验课，随着实验进度的增加，我们实验的内容逐渐从基本操作转向对程序的修改。相信以后会学到更多实用的知识。