DSP应用丛书

TMS 320 F28x 源码解读

任润柏　周荔丹　姚　钢　编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

DSP 应用丛书

TMS 320 F28x 源码解读

任润柏　周荔丹　姚　钢　编著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

这是一本 F28x 系统的入门书。书中提供的 F28x 外设驱动源码文件是 TMS 320 F28x 系统的底层文件，适合所有 F28x 硬件系统。实践证明，从可操作调试的外设驱动源码出发，是进入 F28x 领域的一条捷径。

本书通过解读德州仪器提供的 DSP2000 系列外设驱动源码（文档名 SPRC097）的方式，介绍了 TMS 320 F28x 各种外设的驱动机制、运行要领及与系统之间的关系，并对建立 SPRC097 文件体系的位域结构体方法给出详细的说明。书中源代码均通过实际运行验证。

本书可作为电气、自动控制和电子类专业本科生和研究生的教科书或参考书，也可作为相关领域的工程技术人员的参考书。

书中的所有源码、支持源码的头文件和共享文件均可在电子工业出版社网站（http://www.phei.com.cn/“资源下载”）免费下载。

图书在版编目（CIP）数据

TMS 320 F28x 源码解读/任润柏，周荔丹，姚钢编著. —北京：电子工业出版社，2010.7
（DSP 应用丛书）
ISBN 978-7-121-11329-1

Ⅰ. ①T… Ⅱ. ①任… ②周… ③姚… Ⅲ. ①数字信号－信号处理－数字通信系统，TMS320F28x
Ⅳ. ①TN911.72

中国版本图书馆 CIP 数据核字（2010）第 131756 号

责任编辑：万子芬
印　　刷：北京市天竺颖华印刷厂
装　　订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1 092　1/16　印张：24.5　字数：624 千字
印　　次：2010 年 7 月第 1 次印刷
印　　数：4 000 册　　定价：49.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

序

在四年前认识了任老先生，他十几年潜心于各种 MCU 应用的开发，功力深厚。任老师在年已花甲之时开始接触和使用 C2000，而且兴致盎然，尤其当他孜孜不倦地和我们研讨一些技术细节的时候，其严谨的态度让人敬佩。依然清晰地记得在这一过程中他对 C2000 的执著和兴奋之情。当他谈到希望做些工作，用 MCU 的方式把 C2000 介绍给更多的已经习惯了 MCU 应用的朋友时，我和大学部门的沈洁女士都觉得无论对业界还是对 TI 公司这都是一件极其有益的事。

C2000 是最早融合和吸收 MCU 优点并向传统的 MCU 应用渗透的 DSP 芯片。有一段时间，业界很热门地讨论如何界定 DSP 和 MCU，最后对 C2000 这一类芯片给出了“数字信号控制器”（Digital Signal Controller）这样一个名头。对于 MCU 工程师来说，DSP 通常意味着高端兼复杂，虽然改名叫了 DSC，很多人还是对它望而却步，一是因为开发环境的不同，二是它与信号处理技术相关。这两方面都与开发设计理念有些关系，也容易给熟悉 MCU 的朋友带来困扰和困难。TI 一直不懈地努力，发布了很多参考代码。这些参考代码在业界得到了很高评价，甚至成为很多工程师软件模块化设计的教科书。

C2000 如今已经深入传统的 MCU 应用领域，并成为极具特点的一个分支，为电力电子的数字化控制和各种高性能的工业应用提供了一个强大的引擎。任先生花费数年之力，用系统的方式在书中详细地解读了这些代码，并一一注释，经过高校老师多次的研讨，终而集册成书。

下面把宝贵的时间留给各位，以尽情品味书中精华。

谭　徽

2010 年 6 月，上海

前　言

随着能源日趋紧张，对电能进行有效的控制从而节约能源越发显得重要。TI 推出的 F28x 系统芯片是一款在电动机及电力控制中被业界广泛应用的控制类芯片，但目前该系列芯片在国内的应用远远没有达到 51 系列单片机那样的程度。

五六年前，笔者自恃有十几年从事 8 位单片机开发的经验，曾几次捧起 DSP2000 系列的原理书想了解一个大概，然而几次不得不驻足兴叹。事实上，即使是一个比较简单的 51 系统，其入门也不能仅靠原理书来完成，必须有示范板及相关的驱动源码，初学者在逐一调试这些驱动源码的基础上才能慢慢入门和渐入佳境。

F28x（32 位机）是一个较 51（8 位机）系统更复杂、更难学的系统，然而，相关中文书籍除翻译外版的原理书之外，基本上没有行之有效的驱动源码的介绍。事实上，TI 早在 2003 年 9 月就颁布了一个 V100 版本的 SPRC097 文件，之后，在 2007 年 9 月将该文件更新为 V111 版本，并在 2009 年 7 月又更新为 V120 版本。该文件以 C/C++语言为基础，通过位域结构体的方法为 F28x 提供了完整的一个头文件体系，并且针对 F28x 的外围设备给出了 20 个外设驱动源码（V100 之后的版本增加了 5 个源码）。这是 F28x 入门的一个有效文件。

2006 年初夏，笔者从国内第三方获得 eZdsp 翻版的示范板，发现与示范板配套的很少几个代码都取自 SPRC097 文件中的部分内容，但缺少了前后关联及 SPRC097 文件自身的系统性。因此，笔者尝试直接在 SPRC097 文件系统上运行该文件中的源码，多数源码竟然可以直接调试运行！之后，笔者潜心阅读、调试，并将点滴领悟记录下来。到 2007 年年初，这些记录（或称为解读）已有相当篇幅，笔者设计的 DSP2812 示范板也于 2006 年年底调试完毕。

2007 年 5 月，经上海交通大学陈健老师（德州仪器 DSP 大学计划的国内早期发起人和组织者之一）的引荐，认识了德州仪器亚洲区大学计划部沈洁经理及半导体事业部经理谭徽博士，当年他们对笔者所做的事情以及打算将 SPRC097 文件源码解读成书的愿望的褒扬至今历历在目。

TI 大学计划部与上海交通大学电气工程系连续两年（2008—2009 年）在上海交通大学举办 DSP2000 青年教师暑期培训班，均采用 2008 年 6 月由笔者编写的“DSP281x 外设驱动源码解读”讲义（308 页）作为教材。

本书以“DSP281x 外设驱动源码解读”讲义为雏形，对 TI 提供的外设驱动源码进行了更深入的解读，具有以下特点：

（1）对构建 SPRC097 文件体系的位域结构体方法及文件中出现的陌生的指令进行详细的注解；

（2）对尚未公示的采用 SPRC097 文件风格建立一个简捷且容易管理的新项目的方法进行解析和引用；

（3）对每条外设示例源码设置专用寄存器的指令，具体到对每一个控制位进行注释；

（4）对每一个函数（子程序），在题头说明该函数的实现功能及相关要点；

（5）说明怎样通过示波器观察关键变量，以及怎样通过 CCS 视窗观察动态数据及动态波形；

（6）对 F28x 所有外设的相关原理的叙述，围绕实际应用展开并通过图示进行说明；

（7）对反映在专用控制寄存器中非显式表示的配置表达式，都用规范的数学公式表示；

（8）通过实例归纳出相应外设驱动的设置要领。

除了原讲义关于 F28x 所有外设的章节外，本书还增加了“第 11 章　CMD 文件的运用”及“第 12 章　IQmath 方法概述”。其中“CMD 文件的运用”是从 C 语言进入 F28x 必须掌握的概念。

本书每章后面列出的外设驱动源码来自 SPRC097 文件，部分源码增加了一些存储类的指令并开辟了相应的存储区，目地是为了在内存中观察数据，或在图形窗观察波形，以便获得直观的感受。本书源码均通过实际运行，适用于所有 F28x 硬件系统。

本书旨在叙述各个外设的原理，并力求讲述清楚明了，而没有徘徊于原理的各个细节，对各种寄存器的介绍也进行了取舍。简言之，本书并非一本严格意义上的原理书。读者若要进行更深入的探究可查阅相关的原理书籍。

感谢上海交通大学陈健老师的推荐，使得笔者所做的一点事情得到了 TI 亚洲大学计划部的肯定。感谢上海交通大学电气工程系姜建民老师提供的教学平台，更重要的是由于他的介绍，笔者才结识了本书的合作伙伴周荔丹和姚钢两位博士。

特别要感谢 TI 亚洲大学计划部沈洁经理，她的大学计划部对上海交通大学电气工程系 DSP 2000 系列实验中心给予了大力的支持，同时给予笔者莫大的信任和期待。

本书的第 1、2、3、6 章由姚钢编写；第 4、9、11、12 章由周荔丹编写；任润柏编写了第 5、7、8、10 章，并最后统稿。

由于功力有限，错误难免。只是祈望本书少些给读者特别是自学者带来困惑，并对可能给读者造成的困惑深深致歉，请不吝赐教。联系方式：renrunbai@hotmail.com。

任润柏

2010 年 6 月于上海交通大学

目　录

第 1 章　DSP F28x 使用入门

2003 年 9 月 TI 公司公布了一个 SPRC097 1.00 版本的文件，该文件以 C/C++语言为基础，通过位域结构体的方法为 F28x 提供了一个完整的头文件体系，并且针对 F28x 的外围设备给出了 20 个外设示例，这是 DSP 控制类芯片在软件领域的一大进步。

SPRC097 提供的代码可作为学习 F28x 的有力工具，也可作为用户基本的开发平台。

1）学习工具

20 个外设示例项目需要在 F2812 eZdsp 平台上调试运行，目前国内现有的 DSP28 系列试验板基本上是 eZdsp 的翻版，因此，eZdsp 平台完成的实验，在国内的试验板上也基本能做通。

SPRC097 提供的例子是对器件的初始化所需的步骤及使用片上外设的一个示范。这些例子可以复制和修改，使用户快速建立针对不同的外设配置的实验平台。

2）开发平台

用户可以容易地将外设头文件合并到一个新的或者已经存在的项目中，以便提供一个采用 C 或者 C++代码访问片上外设的平台。另外，需要的话，用户可以从示范代码中选取一些函数，而将其余的舍去。

SPRC097 说明文档提供以下资料：

（1）用于 F28x C/C++外设头文件的位域结构体方法纵览；

（2）外设示例项目纵览；

（3）将外设头文件组合到一个新的或者已有项目中的步骤。

最后，需要说明的是，这个文档不是采用 CCS 或者采用 F28x 编译器和汇编程序，编写 C 代码的指南。使用这个文档的用户需要有一个硬件平台，并将这个平台连接到安装了 CCS 的主机上。另外，用户对完成基本的调试操作及如何使用 CCS 通过 JTAG 下载代码必须有一个基本的了解。

1.1　项目文件的目录结构

SPRC097 压缩文件释放安装后，形成一个 V100 的目录，如图 1.1 所示，该目录下有定义明确的 4 个子目录：

- DSP281x_headers，DSP281x 外设头文件夹；
- DSP281x_common，DSP281x 共享文件夹；
- DSP281x_examples，DSP281x 外设示例文件夹；
- DOC 自述文件夹。

本书对 SPRC097 说明文档进行适当的裁减和补充。

V1.00 版本将原来外设示例及共享源文件中的头文件清晰地分离出来，更新为 V1.00 版本的头文件。

这样一种新的文件结构使得对文件的定位，以及将或

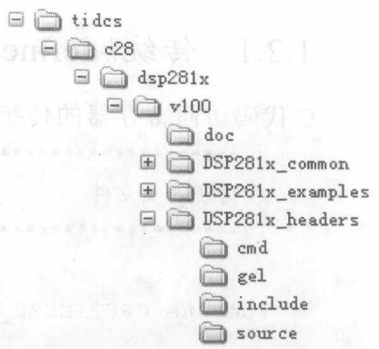

图 1.1　项目文件的目录结构

多或少的文件合并成想要得到的一个新的或已经存在的项目变得容易了。

表 1.1 为 DSP281x V1.00 版本主目录的内容。

表 1.1 DSP281x V1.00 版本主目录结构

目 录	说 明
<base>	安装根目录默认为 c:\tidcs\c28\DSP281x\V100。这个目录内其他的子目录，采用<base>默认路径名
<base>\DOC	说明文档，包括对以前版本修订的记录
<base>\DSP281x_headers	项目中所需的外设头文件； 头文件采用 1.2 节描述的位域结构体的方法构成； 1.3 节描述了如何将头文件结合到一个新的或者已有的项目中
<base>\DSP281x_examples	基于 DSP281x 头文件的 CCS 项目示例。这些项目示例用于阐明如何配置多数 281x 片上外设
<base>DSP281x_common	公共源文件被多数 DSP281x 项目示例交叉共享，用于阐明如何采用 DSP281x 头文件的方法执行任务，这些文件是可选的，但在新建项目中是有用的

在 DSP281x_headers 及 DSP281x_common 目录内，源文件可进一步往下分解成标明文件类型的子目录。表 1.2 列出了子目录，同时列出每个子目录中所建立的文件类型。

表 1.2 DSP281x 子目录结构

子 目 录	说 明
DSP281x_headers\cmd	分配位域结构体的链接器命令文件
DSP281x_headers\source	将头文件组合到一个新的或已有的项目中所需的源文件
DSP281x_headers\include	281x 每个片上外设所需的头文件
DSP281x_common\cmd	示例内存链接器命令文件，用于分配 281x 器件的内存
DSP281x_common\include	公共*.h 文件，用于 DSP281x 外设示例
DSP281x_common\source	公共*.c 文件，用于 DSP281x 外设示例

1.2 外设位域结构体方法综述

DSP281x 头文件和外设示例使用位域结构体方法，映射和访问基于 F28x 外设寄存器。本节将介绍这种方法，并把它和传统的#define 方法加以比较。

1.2.1 传统#define 方法

C 代码访问寄存器的传统方法是使用#define 宏为每一个寄存器分配一个地址。例如：

```
//*****************************************************************************
//传统的头文件
//*****************************************************************************
                                          //存储器映像地址寄存器
#define CPUTIMER0_TIM (volatile unsigned long *)0x0C00
                                          //0xC00 定时器 0 计数器低位
                                          //0xC01 定时器 0 计数器高位
```

```
#define CPUTIMER0_PRD (volatile unsigned long *)0x0C02
                                                    //0xC02 定时器 0 周期寄存器低位
                                                    //0xC03 定时器 0 周期寄存器高位
#define CPUTIMER0_TCR (volatile unsigned int *)0x0C04
                                                    //0xC04 定时器 0 控制寄存器
                                                    //0xC05 保留
#define CPUTIMER0_TPR (volatile unsigned int *)0x0C06
                                                    //0xC06 定时器 0 预定标寄存器低位
#define CPUTIMER0_TPRH (volatile unsigned int *)0x0C07
                                                    //0xC07 定时器 0 预定标寄存器高位
```

同样的#define 方法将在每个外设寄存器上不断重复。甚至对于诸如 SCI-A 和 SCI-B 这样完全相同的外设，每个寄存器都必须被一一分配地址。传统#define 方法有以下显著弊端：

（1）不容易访问寄存器中的位域部分；

（2）不容易在 CCS 观察窗内显示位域的值；

（3）不能利用 CCS 的自动完成功能；

（4）对于重复的外设，头文件开发者不能获得重复使用的便利。

1.2.2　位域及结构体方法

位域及结构体方法采用 C 代码结构体方式，将属于某个指定外设的所有寄存器组成一个集合。通过链接器，每个 C 代码结构体就是外设寄存器的内存映射。这一映射允许编译器通过使用 CPU 数据页指针（DP）直接访问外设寄存器。另外，多数寄存器都定义了位域，从而使编译器能读取或者操作某个寄存器中的单个位域。

1）外设寄存器结构体

在 1.2.1 节中，我们使用传统#define 方法定义了 CPU 定时器 0 寄存器（CPU -Timer0）。本节改为采用 C 代码结构体方法将 CPU 定时器的寄存器集合在一起，定义同一个 CPU 定时器 0 寄存器。通过使用链接器，将结构体映射到内存 CPU-Timer0 寄存器上。

以下代码示例是一个与 DSP281x CPU 定时器外设对应的 C 代码结构体类型：

```
//*****************************************************************************
//采用结构体形式的 CPU 定时器头文件
//*****************************************************************************
struct CPUTIMER_REGS      //仅仅定义了一个结构体类型 CPUTIMER_REGS，还没有
                          //定义变量
{   Uint32 TIM;           //定时器计数寄存器
    Uint32 PRD;           //定时器周期寄存器
    Uint16 TCR;           //定时器控制寄存器
    Uint16 rsvd1;         //保留
    Uint16 TPR;           //定时器预定标寄存器低位
    Uint16 TPRH;          //定时器预定标寄存器高位
};
```

该结构体类型由 6 个成员组成，前后顺序与它们在内存中的顺序相同。“CPUTIMER_REGS”代表了一个结构体类型。即它和系统已经定义的标准类型（如 int，char 等）一样可以用来作为定义变量的类型。

注意以下几点：

（1）寄存器名出现的顺序必须与它们在内存中被安排的顺序相同；

（2）在结构体中，通过使用保留变量（rsvd1，rsvd2 等）来预留内存中的保留位置。这种保留结构仅仅用以预留内存中的空间；

（3）Uint16 和 Uint32 分别是无符号 16 位或者 32 位数的类型定义，在 DSP281x 中，则用来定义无符号整型和无符号长整型。这样使用起来就方便一些。相应的类型定义声明由 DSP281x_Device.h 文件建立。

2）声明可访问寄存器的变量

寄存器结构体类型可被用于声明一个可访问寄存器的变量，对器件的每个外设都采用这一相同的做法，同一种外设的复用外设可以采用同样的结构体类型定义。例如，如果一个器件上有 3 个 CPU-Timers，可以创建如下所示的 3 个具有“struct CPUTIMER_REGS”结构体类型的变量。

```
//*****************************************************************************
//采用结构体形式的 CPU 定时器头文件
//*****************************************************************************
volatile struct CPUTIMER_REGS CpuTimer0Regs; //定义 CpuTimer0Regs 是
                                             //一个具有 CPUTIMER_REGS
                                             //类型的变量，下同
volatile struct CPUTIMER_REGS CpuTimer1Regs;
volatile struct CPUTIMER_REGS CpuTimer2Regs;
```

这里，关键字 volatile 在变量声明中十分重要。它告诉编译器，这些变量的内容可由硬件改变，并且编译器无须优化使用 volatile 变量的代码。

注意：定义结构体类型变量都需要关键字 struct。变量 CpuTimer0Regs，CpuTimer1-Regs，CpuTimer2Regs 都是具有 CPUTIMER_REGS 结构体类型的结构体变量。这里把 CPUTIMER_REGS 看做是 CPU 定时器的同一种外设，而把 CpuTimer0Regs，CpuTimer1-Regs，CpuTimer2Regs 看做是同一外设的 3 次复用。每一次复用均包含前一个代码框定义的那些专用寄存器变量。这种定义方式与 F28x 外设寄存器的分配相对应。

SPRC097 1.00 版本用头文件方式为 F28x 的所有外设提供了一个完整的位域结构体体系。对接触 C 语言时间不长的读者，花点时间把它弄懂是非常必要的。

单有上面两个定义是不够的，还必须为 3 个 CPU 定时器分配数据区。分配给某个定时器在数据区的起始地址必须与系统定义的该定时器的内存地址一致。这可通过#pragma DATA_SECTION 等指令完成。否则，编译器将其视为普通的结构体类型变量，统一分配数据区，导致对 CPU 定时器不能有效地访问。

3）分配专用的数据区

在 DSP281x_GlobalVariableDefs.c 文件中（该文件位于 DSP281x_Headers\common\source 目录内），通过使用编译器的#pragma DATA_SECTION 指令，与外设寄存器结构

体类型相对应的每一个变量都将被分配一个专用的数据区。在下面所示的代码中，变量 CpuTimer0Regs 被分配到 CpuTimer0RegsFile 的数据区。

```
//*************************************************************************
//DSP281x_headers\source\DSP281x_GlobalVariableDefs.c
//*************************************************************************
//采用#pragma 编译器声明,将 CpuTimer0Regs 变量分配到 CpuTimer0RegsFile 数据
//区。C 或 C++采用不同的#pragma 声明方式。当对一个 C++程序进行编译时，编译器自
//动定义__cplusplus。
#ifdef __cplusplus                                    //用于 C++代码
#pragma DATA_SECTION("CpuTimer0RegsFile");
#else                                                 //用于 C 代码
#pragma DATA_SECTION(CpuTimer0Regs,"CpuTimer0RegsFile");
#endif
Volatile structCPUTIMER_REGS CpuTimer0Regs;  //定义 CpuTimer0Regs 是一
                                             //个具有 CPUTIMER_REGS 类
                                             //型的变量
//#ifdef、#else 及#endif 为预处理器条件编译指令。其中#ifdef 和#else 格式类似
//C 中的 if 和 else。主要差异为预处理器不能识别标记代码块的大括号"{}"，因此使用
//#else（如果需要）和#end if（必须存在）来标记指令块。上面指令的含义为：如果采
//用的是 C++，则执行语句#pragma DATA_SECTION("CpuTimer0RegsFile");
//倘若采用的是 C，则执行语句#pragma DATA_SECTION(CpuTimer0Regs,
//"CpuTimer0RegsFile");
```

对器件的每个外设寄存器结构体变量，都会重复这一数据区分配操作。

4）映射到外设寄存器

当每个结构体都分配到自身的数据区之后，通过使用链接命令文件 DSP281x_Headers_nonBIOS.cmd，每个数据区都将被直接映射到外设内存映射寄存器上，如以下代码所示：

```
//*************************************************************************
//DSP281x_headers\include\DSP281x_Headers_nonBIOS.cmd
//*************************************************************************
MEMORY                  /* 定义存储区域。注意：cmd 文件不可以用“//”注释符 */
{
PAGE 1:                 /* PAGE 1 在 cmd 文件中表示数据区 */
    CPU_TIMER0 : origin = 0x000C00, length = 0x000008
                        /* 将起始地址为 0x0C00，长度为 8 个单元的内存区域 */
                        /* 定义为 CPU 定时器 0 寄存器存储区 */
}
SECTIONS                /* 分配存储区域 */
{
    CpuTimer0RegsFile : > CPU_TIMER0, PAGE = 1
```