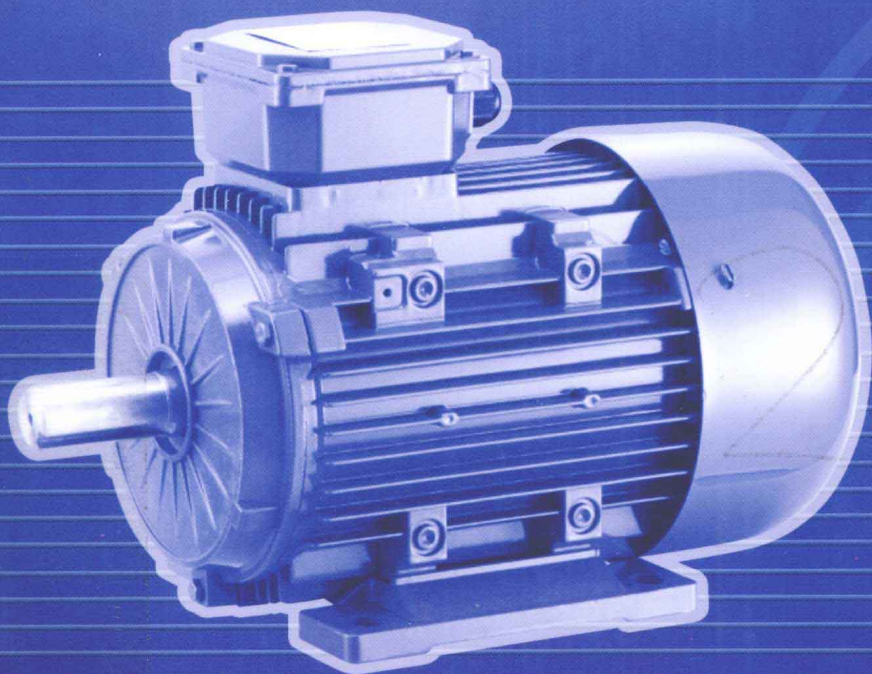


电动机的 DSP控制技术与实践

任志斌 编著



中国电力出版社
CHINA ELECTRIC POWER PRESS

电动机的 DSP控制技术与实践

任志斌 编著



中国电力出版社
CHINA ELECTRIC POWER PRESS

内 容 提 要

本书主要以 TI 公司 TMS320F2812 的 DSP 为对象,以电动机控制技术为重点,介绍了 TI 公司的 TMS320F2812 芯片在电动机控制系统中的应用。全书共分为 6 章,第 1 章介绍了 TMS320F2812 的主要原理,包括 DSP 的结构及性能、存储空间及时钟、中断系统、事件管理器及 A/D 转换器;第 2 章围绕电动机控制技术方面的编程需要,重点介绍了数据 Q 格式、带有死区的 PWM 波形、数字 PI 调节器、数字测速及电压空间矢量 PWM (SVPWM) 控制技术的 DSP 实现方法;第 3 章~第 6 章分别对直流无刷电动机、异步电动机 SPWM 控制、基于坐标变换的异步电动机的矢量控制技术及永磁同步电动机的 DSP 控制技术作了详细介绍。

本书以 DSP 的电动机控制实践技术为主要内容,并且各个环节都有作者在实际工作中的实例,使读者通过实例加深对内容的理解,全书讲解通俗易懂、深入浅出。

本书适合作为电动机与电器、电气工程及其自动化、电力电子与电力传动专业及其他相关专业的高年级本科生和研究生教材,也可作为工程技术人员研究、开发电动机 DSP 控制系统的参考书。

图书在版编目 (CIP) 数据

电动机的 DSP 控制技术与实践 / 任志斌编著. —北京: 中国电力出版社, 2012.3

ISBN 978-7-5123-2810-5

I. ①电… II. ①任… III. ①数字信号处理—应用—电动机—控制 IV. ①TM320.12

中国版本图书馆 CIP 数据核字 (2012) 第 043720 号

中国电力出版社出版、发行

(北京市东城区北京站西街 19 号 100005 <http://www.cepp.sgcc.com.cn>)

北京市同江印刷厂印刷

各地新华书店经售

*

2012 年 6 月第一版 2012 年 6 月北京第一次印刷
787 毫米×1092 毫米 16 开本 12.5 印张 290 千字
印数 0001—3000 册 定价 28.00 元

敬告读者

本书封面贴有防伪标签,加热后中心图案消失
本书如有印装质量问题,我社发行部负责退换

版权专有 翻印必究

TMS320F2812 是 TI 公司最新推出的 DSP 芯片, 是目前国际市场上最先进、功能最强大的 32 位定点 DSP 芯片。它既具有数字信号处理能力, 又具有强大的事件管理能力和嵌入式控制功能, 特别适用于有大量数据处理的测控场合, 如工业自动化控制、电力电子技术应用、智能化仪器仪表及电动机伺服控制系统等。为了帮助广大工程技术人员及教学人员尽快掌握 TMS320F2812 的 DSP 编程技术及其在电动机控制中应用, 我们特编写了本书。

本书介绍 DSP 芯片的结构、功能和接口原理, 深入浅出地阐述了电动机控制的各种基本原理和方法, 以及电动机控制所必需的常用信号检测元件, 使读者对电动机及其数字控制有较为系统的了解, 以便为进一步解决实际问题打下理论基础; 另外, 本书较为系统地介绍了电动机控制系统的硬件和软件设计方法, 并提供大量的范例给读者参考, 有助于读者快速地了解整个电动机的 DSP 控制系统的框架、需要设计的重点及难点。本书共分为 6 章, 第 1 章为 TMS320F2812 编程技术基础; 第 2 章为电动机控制的编程技术; 第 3 章为无刷直流电动机的 DSP 控制技术; 第 4 章为异步电动机的 SPWM 控制技术; 第 5 章为基于坐标变换的异步电动机矢量控制技术; 第 6 章为永磁同步电动机的 DSP 控制技术。

本书由江西理工大学任志斌教授编写, 参加编写及程序调试的还有梁建伟、龙宇涛、刘森、王中科、王业占等。

我们在编写过程中虽然花了不少精力, 仍难免有错误与不足之处, 殷切期望广大读者批评指正。

编 者

2012 年 4 月

前言

第 1 章 TMS320F2812 编程技术基础	1
1.1 TMS320F28x 系列芯片的结构及性能.....	1
1.2 TMS320F2812 的存储空间及时钟.....	4
1.2.1 存储空间.....	4
1.2.2 CMD 文件.....	6
1.2.3 时钟.....	10
1.3 TMS320F2812 的中断系统.....	12
1.3.1 外设中断介绍.....	13
1.3.2 PIE 中断向量及其映射方式.....	14
1.3.3 TMS320F2812 的 3 级中断机制.....	17
1.3.4 CCS 对中断的定义、初始化及使用.....	20
1.3.5 TMS320F2812 中断处理过程举例.....	21
1.4 事件管理器 (EV).....	22
1.4.1 事件管理器功能.....	25
1.4.2 事件管理器的寄存器地址.....	27
1.4.3 通用目的 (GP) 定时器.....	29
1.4.4 全比较单元电路.....	40
1.4.5 QEP 电路.....	44
1.4.6 捕获单元.....	46
1.5 DSP 的 A/D 转换器.....	48
1.5.1 ADC 模块结构.....	48
1.5.2 TMS320F2812 内部 A/D 的工作方式.....	50
1.5.3 举例.....	51
第 2 章 电动机控制的编程技术	55
2.1 定点 DSP 的数据 Q 格式.....	55
2.1.1 Q 格式说明.....	55
2.1.2 电动机控制中电流采样值的 Q 格式处理.....	56
2.2 带有死区的 PWM 波形.....	58
2.3 数字 PI 调节器的 DSP 实现方法.....	60

2.3.1	模拟 PI 调节器的数字化	60
2.3.2	改进的数字 PI 算法	61
2.3.3	数字 PI 调节器的举例	62
2.4	数字测速	63
2.4.1	旋转编码器	63
2.4.2	数字测速方法的精度指标	64
2.4.3	M 法测速	64
2.4.4	T 法测速	66
2.4.5	M/T 法测速	67
2.4.6	速度测量的实现	69
2.4.7	例程	70
2.5	电压空间矢量 PWM (SVPWM) 控制技术	74
2.5.1	空间矢量的定义	74
2.5.2	电压与磁链空间矢量的关系	75
2.5.3	PWM 逆变器基本输出电压矢量	76
2.5.4	正六边形空间旋转磁场	77
2.5.5	期望电压空间矢量的合成与实现	78
2.5.6	SVPWM 的三个关键问题解决	79
2.5.7	SVPWM 编程实现举例	89

第 3 章 无刷直流电动机的 DSP 控制技术 94

3.1	无刷直流电动机的组成结构和工作原理	94
3.1.1	无刷直流电动机的结构	94
3.1.2	无刷直流电动机的霍尔传感器位置检测	95
3.1.3	无刷直流电动机的工作原理	99
3.2	无刷直流电动机的基本公式	101
3.3	无刷直流电动机的 DSP 控制	102
3.3.1	一般交流传动控制系统结构	102
3.3.2	无刷直流电动机控制框图	102
3.3.3	系统硬件结构	103
3.3.4	控制程序设计	108
3.4	无刷直流电动机相序测定方法	114

第 4 章 异步电动机的 SPWM 控制技术 116

4.1	异步电动机调速基本原理	116
4.1.1	基频以下调速	116
4.1.2	基频以上调速	117

4.2	异步电动机变压变频控制时的稳态特性	117
4.3	SPWM 变频技术	119
4.3.1	交流 PWM 变频技术	119
4.3.2	正弦脉宽调制原理	119
4.3.3	脉宽调制的约束条件	122
4.3.4	同步调制与异步调制	123
4.3.5	SPWM 控制方法	124
4.4	异步电动机 SPWM 调速系统实现	126
4.4.1	异步电动机 SPWM 控制框图	126
4.4.2	规则采样在 DSP 上的实现	127

第 5 章 基于坐标变换的异步电动机矢量控制技术 133

5.1	异步电动机动态数学模型	133
5.1.1	异步电动机动态数学模型的性质	133
5.1.2	异步电动机三相原始数学模型	134
5.2	坐标变换控制的基本思想	138
5.3	坐标变换	139
5.4	异步电动机在两相坐标系上的动态数学模型	143
5.5	异步电动机在两相坐标系上的状态方程	145
5.6	异步电动机按转子磁链定向的矢量控制系统	149
5.6.1	按转子磁链定向同步旋转坐标系 MT 中的状态方程	149
5.6.2	按转子磁链定向矢量控制的基本思想	150
5.6.3	按转子磁链定向矢量控制系统的实现	151
5.6.4	磁链开环转差型矢量控制系统——间接定向	151
5.7	DSP 在异步电动机矢量控制系统中的实现	153
5.7.1	控制系统总体设计	153
5.7.2	编程分析	153
5.8	常见问题分析	165

第 6 章 永磁同步电动机的 DSP 控制技术 172

6.1	同步电动机的一些基本概念	172
6.1.1	同步电动机与异步电动机的比较	172
6.1.2	同步电动机的分类	173
6.1.3	同步电动机的矩角特性	174
6.1.4	同步电动机的稳定运行	175
6.1.5	同步电动机的调速	176
6.2	同步电动机数学模型与磁场定向控制原理	176

6.2.1	永磁同步电动机的数学模型	176
6.2.2	永磁同步电动机矢量控制原理	179
6.3	DSP 在永磁同步电动机矢量控制系统中的实现	180
6.3.1	永磁同步电动机矢量控制框图	180
6.3.2	编程分析	181

TMS320F2812 编程技术基础

TMS320F28x 系列是 TI 公司最新推出的数字信号处理器 (DSP) 芯片, 是目前国际市场上最先进、功能最强大的 32 位定点 DSP 芯片。它既具有数字信号处理能力, 又具有强大的事件管理能力和嵌入式控制功能, 特别适用于有大量数据处理的测控场合, 如工业自动化控制、电力电子技术应用、智能化仪器仪表及电机伺服控制系统等。本章将介绍 TMS320F28x 系列芯片的结构、性能、特点及相关编程技术。

1.1 TMS320F28x 系列芯片的结构及性能

F28x 系列的主要片种为 TMS320F2810 和 TMS320F2812。两种芯片的差别是: F2812 内含 128K×16 位的片内 Flash 存储器, 有外部存储器接口, 而 F2810 仅有 64K×16 位的片内 Flash 存储器, 且无外部存储器接口。其硬件特征如表 1-1 所示。

表 1-1 硬 件 特 征

特 征	F2810	F2812
指令周期 (150MHz)	6.67ns	6.67ns
SRAM (16 位/字)	18K	18K
3.3V 片内 Flash (16 位/字)	64K	128K
片内 Flash/SRAM 的密钥	有	有
Boot ROM	有	有
掩膜 ROM	有	有
外部存储器接口	无	有
事件管理器 A 和 B (EVA 和 EVB)	EVA、EVB	EVA、EVB
通用定时器	4	4
比较寄存器/脉宽调制	16	16
捕获/正交解码脉冲电路	6/2	6/2
看门狗定时器	有	有
12 位的 ADC	有	有
通道数	16	16
32 位的 CPU 定时器	3	3
串行外围接口	有	有
串行通信接口 (SCI) A 和 B	SCIA、SCIB	SCIA、SCIB
控制器局域网络	有	有

续表

特 征	F2810	F2812
多通道缓冲串行接口	有	有
数字输入/输出引脚（共享）	有	有
外部中断源	3	3
供电电压	核心电压 1.8V, I/O 电压 3.3V	核心电压 1.8V, I/O 电压 3.3V
封装	128 针 PBK	179 针 GHH, 176 针 PGF
温度选择*A: -40~+85℃ S: -40~+125℃	PBK, 仅适用于 TMS	PGF 和 GHH, 仅适用于 TMS
产品状况** 产品预览 (PP) 高级信息 (AI) 产品数据 (PD)	AI (TMP) ***	AI (TMP) ***

* “S”是温度选择（-40~+125℃）的特征化数据，仅对 TMS 是适用的。

** 产品预览 (PP): 在开发阶段的形成和设计中与产品有关的信息，特征数据和其他规格是设计的目标。TI 保留了正确的东西，更换或者终止了一些没有注意到的产品。

高级信息 (AI): 在开发阶段的取样和试制中与新产品有关的信息，特征数据和其他规格用以改变那些没有注意到的东西。

产品数据 (PD): 是当前公布的数据信息，产品遵守 TI 的每项标准保修规格，但产品加工不包括对所有参数的测试。

*** TMP: 最终的硅电路小片，它与器件的电气特性相一致，但是没有进行全部的品质和可靠性检测。

本章重点以 TMS320F2812 为主（以后文中简称为 2812），芯片的主要性能如下：

- (1) 高性能静态 CMOS (Static CMOS) 技术。
 - 1) 150MHz (时钟周期 6.67ns)。
 - 2) 低功耗 (核心电压 1.8V, I/O 口电压 3.3V)。
 - 3) Flash 编程电压 3.3V。
- (2) JTAG 边界扫描 (Boundary Scan) 支持。
- (3) 高性能的 32 位中央处理器。
 - 1) 16 位×16 位和 32 位×32 位乘且累加操作。
 - 2) 16 位×16 位的两个乘且累加。
 - 3) 哈佛总线结构 (Harvard Bus Architecture)。
 - 4) 强大的操作能力。
 - 5) 迅速的中断响应和处理。
 - 6) 统一的寄存器编程模式。
 - 7) 可达 4 兆字的线性程序地址。
 - 8) 可达 4 兆字的数据地址。
 - 9) 代码高效 (用 C/C++ 或汇编语言)。
 - 10) 与 TMS320F24x/LF240x 处理器的源代码兼容。
- (4) 片内存储器。
 - 1) 8K×16 位的 Flash 存储器。

- 2) 1K×16 位的 OTP 型只读存储器。
- 3) L0 和 L1: 两块 4K×16 位的单口随机存储器 (SARAM)。
- 4) H0: 一块 8K×16 位的单口随机存储器。
- 5) M0 和 M1: 两块 1K×16 位的单口随机存储器。
- (5) 引导只读存储器 (Boot ROM) 4K×16 位。
 - 1) 带有软件的 Boot 模式。
 - 2) 标准的数学表。
- (6) 外部存储器接口 (仅 F2812 有)。
 - 1) 有多达 1MB 的存储器。
 - 2) 可编程等待状态数。
 - 3) 可编程读/写选通计数器 (Strobe Timing)。
 - 4) 三个独立的片选端。
- (7) 时钟与系统控制。
 - 1) 支持动态的改变锁相环的频率。
 - 2) 片内振荡器。
 - 3) 看门狗定时器模块。
- (8) 三个外部中断引脚。
- (9) 外部中断扩展 (PIE) 模块。可支持 96 个外部中断, 当前仅使用了 45 个外部中断。
- (10) 128 位的密钥 (Security Key/Lock)。
 - 1) 保护 Flash/OTP 和 L0/L1 SARAM。
 - 2) 防止 ROM 中的程序被盗。
- (11) 3 个 32 位的 CPU 定时器。
- (12) 电动机控制外围设备。
 - 1) 两个事件管理器 (EVA、EVB)。
 - 2) 与 240 兼容的指令和代码。
- (13) 串口外围设备。
 - 1) 串行外围接口 (SPI)。
 - 2) 两个串行通信接口 (SCIs), 标准的 UART。
 - 3) 改进的局域网络 (eCAN)。
 - 4) 多通道缓冲串行接口 (McBSP) 和串行外围接口模式。
- (14) 12 位的 ADC, 16 通道。
 - 1) 2×8 通道的输入多路选择器。
 - 2) 两个采样保持器。
 - 3) 单个的转换时间为 200ns。
 - 4) 单路转换时间为 60ns。
- (15) 最多有 56 个独立的可编程、多用途通用输入/输出 (GPIO) 引脚。
- (16) 高级的仿真特性。
 - 1) 分析和设置断点的功能。

- 2) 实时的硬件调试。
 - (17) 开发工具。
 - 1) ANSI C/C++编译器/汇编程序/连接器。
 - 2) 支持 TMS320C24x/240x 的指令。
 - 3) 代码编辑集成环境。
 - 4) DSP/BIOS。
 - 5) JTAG 扫描控制器 (TI 或第三方的)。
 - 6) 硬件评估板。
 - (18) 低功耗模式和节能模式。
 - 1) 支持空闲模式、等待模式、挂起模式。
 - 2) 停止单个外围的时钟。
 - (19) 封装方式。
 - 1) 带外部存储器接口的 179 球形触点 BGA 封装。
 - 2) 带外部存储器接口的 176 引脚低剖面四芯线扁平 LQFP 封装。
 - 3) 没有外部存储器接口的 128 引脚贴片正方扁平 PBK 封装。
 - (20) 温度选择。
 - 1) A: $-40\sim+85^{\circ}\text{C}$ 。
 - 2) S: $-40\sim+125^{\circ}\text{C}$ 。
- 28x 芯片的功能框图如图 1-1 所示。

TI 公司生产的还有 TMS320C2812, 称为 C 系列, 与 TMS320F2812 的 F 系列的差别在于: F 系列带有 Flash 存储器, 而 C 系列不带 Flash 存储器; F 系列的一次性可编程 (OTP) ROM 在 C 系列中改成 ROM。若采用 C 系列编写程序, 则必须在软件开发完后将代码交付给生产厂商, 在出厂前将程序固化进 ROM, 而 F 系列程序存储器可重复编程、反复擦写, 在产品开发阶段用起来比较方便。

1.2 TMS320F2812 的存储空间及时钟

1.2.1 存储空间

2812 片内集成了 RAM、ROM 和 FLASH, 具体的存储器资源如表 1-2 所示。

表 1-2 片内存储器资源

名称	大小	名称	大小
FLASH	128K×16 位	M0 (RAM)	1K×16 位
H0 (RAM)	8K×16 位	M1 (RAM)	1K×16 位
L0 (RAM)	4K×16 位	Boot Rom	4K×16 位
L1 (RAM)	4K×16 位	OTP (One time Programmable ROM)	1K×16 位

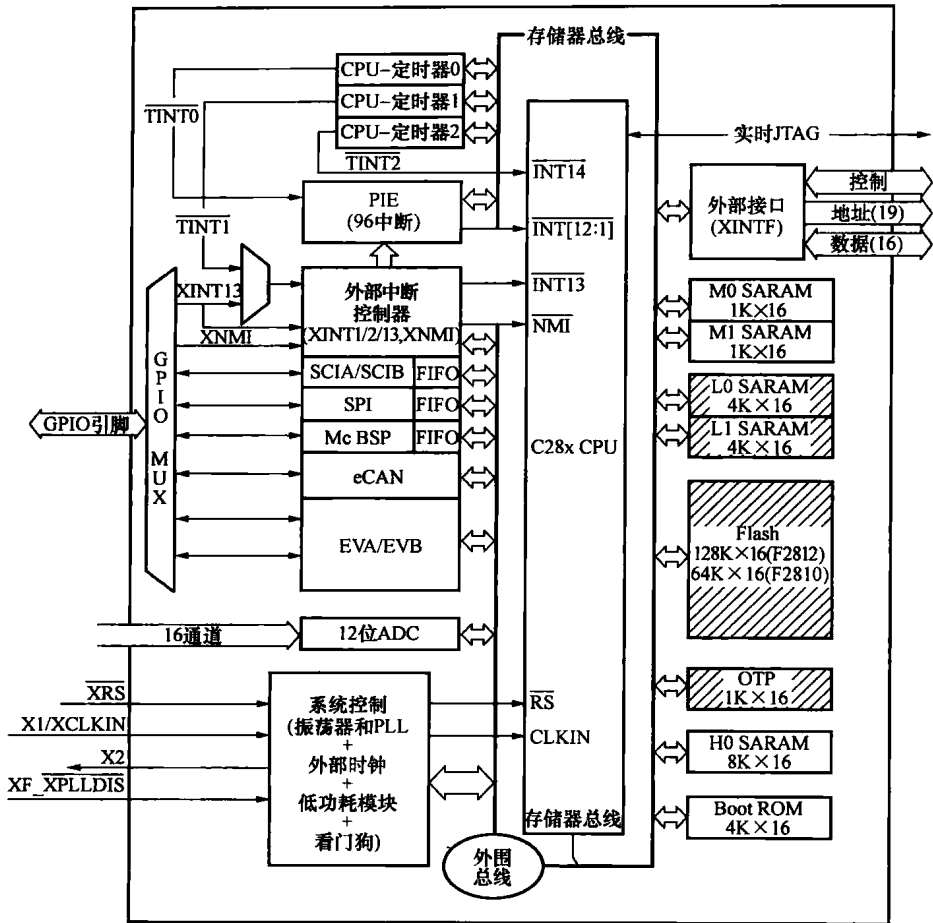


图 1-1 28x 功能框图

注：1. 图中有斜线方框是代码保护的模块。

2. +器件上提供 96 个中断，45 个可用；+XINTF 在 F2810 上不可用。

存储器就像一个仓库，仓库用来存放很多的货物，只不过存储器是用来存放指令和数据的。2812 的存储器被划分成了下面的几个部分：

(1) 程序空间和数据空间。2812 所具有的 RAM、ROM 和 FLASH 都被统一编址，映射到了程序空间和数据空间，这些空间的作用就是存放指令代码和数据变量。

(2) 保留区。数据空间里面某些地址被保留了，作为 CPU 的仿真寄存器使用，这些地址是不向用户开放的。

(3) CPU 中断向量。在程序空间里也保留了 64 个地址作为 CPU 的 32 个中断向量。通过 CPU 的一个寄存器 ST1 中的 VMAP 位来将这一段地址映射到程序空间的底部或者顶部。

各个存储空间就像物流公司的仓库一样，有的是存放程序代码的，有的是用来存放数据的。而且我们对各个存储单元进行了统一的编址，确定了各个存储单元所在的位置，在放置代码或者数据的时候，根据它们的类型进行分配究竟放在哪个区域，并记录下了它们的地址，这样需要用的时候只要根据这些地址就能很方便地找到我们所需要的内容。而记录下如何

分配存储空间内容的就是我们的 CMD 文件了。我们会在稍后详细介绍 CMD 文件的内容以及编写方法。其存储空间分布如表 1-3 所示。

表 1-3 存储空间分布

低地址空间		高地址空间	
0x0000 0000	M0 矢量 RAM (VMAP=0)	0x003D 7800	OTP (1K×16 并保留 1K)
0x0000 0040	M0 SARAM (1K×16)	0x003D 8000	FLASH (128K×16)
0x0000 0400	M1 SARAM (1K×16)	0x003F 7FF8	128 位密钥
0x0000 0800	外设帧 Frame0 (2K×16)	0x003F 8000	H0 SARAM (8K×16)
0x0000 0D00	PIE 向量 (256×16, VMAP=1)	0x003F A000	保留空间
0x0000 1000	保留空间	0x003F F000	Boot Rom(4K×16 MP/MC=0)
0x0000 6000	外设帧 Frame1 (4K×16)	0x003F FFC0	BROM 向量 (VMAP=1 MP/MC=0 ENPIE=0)
0x0000 7000	外设帧 Frame2 (4K×16)		
0x0000 8000	L0 SARAM (4K×16)		
0x0000 9000	L1 SARAM (4K×16)		
0x0000 A000	保留空间		

例如，我们的程序可放在以 00X3F800 开始的 RAM 区内，也可放在以 0X3D800 开始的 FLASH 内，究竟放在什么地方以及其他数据的存放地址，这些都是由 CMD 文件通过链接程序分配的。

1.2.2 CMD 文件

CMD 是 command 命令的缩写，顾名思义就是命令文件，它是用来分配 ROM 和 RAM 空间用的，告诉链接程序怎样计算地址和分配空间。所以不同的芯片就有不同大小的 ROM 和 RAM，存放用户程序的地方也不尽相同，所以要根据芯片进行修改。CMD 文件分两部分，分别是 MEMORY 和 SECTIONS。2812 的 CMD 采用的是分页制，其中 PAGE0 用于存放程序空间，而 PAGE1 用于存放数据空间。我们以一个常用的 SRAM.CMD 文件为例来说明程序如何来分配地址的。

SRAM.CMD 文件如下：

```
MEMORY
{
    PAGE 0 :
    PRAMH0 : origin = 0x3f8000, length = 0x001000
    PAGE 1 : /* SARAM */
    RAMM0 : origin = 0x000000, length = 0x000400
    RAMM1 : origin = 0x000400, length = 0x000400
    /* Peripheral Frame 0: */
    DEV_EMU : origin = 0x000880, length = 0x000180
    FLASH_REGS : origin = 0x000A80, length = 0x000060
    CSM : origin = 0x000AE0, length = 0x000010
```

```
XINTF : origin = 0x000B20, length = 0x000020
CPU_TIMER0 : origin = 0x000C00, length = 0x000008
CPU_TIMER1 : origin = 0x000C08, length = 0x000008
CPU_TIMER2 : origin = 0x000C10, length = 0x000008
PIE_CTRL : origin = 0x000CE0, length = 0x000020
PIE_VECT : origin = 0x000D00, length = 0x000100
/* Peripheral Frame 1: */
ECAN_A : origin = 0x006000, length = 0x000100
ECAN_AMBOX : origin = 0x006100, length = 0x000100
/* Peripheral Frame 2: */
SYSTEM : origin = 0x007010, length = 0x000020
SPI_A : origin = 0x007040, length = 0x000010
SCI_A : origin = 0x007050, length = 0x000010
XINTRUPT : origin = 0x007070, length = 0x000010
GPIOMUX : origin = 0x0070C0, length = 0x000020
GPIODAT : origin = 0x0070E0, length = 0x000020
ADC : origin = 0x007100, length = 0x000020
EV_A : origin = 0x007400, length = 0x000040
EV_B : origin = 0x007500, length = 0x000040
SPI_B : origin = 0x007740, length = 0x000010
SCI_B : origin = 0x007750, length = 0x000010
MCBSP_A : origin = 0x007800, length = 0x000040
/* CSM Password Locations */
CSM_PWL : origin = 0x3F7FF8, length = 0x000008
/* SARAM */
DRAMH0 : origin = 0x3f9000, length = 0x001000
}
SECTIONS
{
/* Allocate program areas: */
.reset : > PRAMH0, PAGE = 0
.text : > PRAMH0, PAGE = 0
.cinit : > PRAMH0, PAGE = 0
/* Allocate data areas: */
.stack : > RAMM1, PAGE = 1
.bss : > DRAMH0, PAGE = 1
.ebss : > DRAMH0, PAGE = 1
.const : > DRAMH0, PAGE = 1
.econst : > DRAMH0, PAGE = 1
.systemem : > DRAMH0, PAGE = 1
/* Allocate Peripheral Frame 0 Register Structures: */
DevEmuRegsFile : > DEV_EMU, PAGE = 1
FlashRegsFile : > FLASH_REGS, PAGE = 1
CsmRegsFile : > CSM, PAGE = 1
XintfRegsFile : > XINTF, PAGE = 1
CpuTimer0RegsFile : > CPU_TIMER0, PAGE = 1
CpuTimer1RegsFile : > CPU_TIMER1, PAGE = 1
CpuTimer2RegsFile : > CPU_TIMER2, PAGE = 1
PieCtrlRegsFile : > PIE_CTRL, PAGE = 1
```



```

PieVectTable : > PIE_VECT, PAGE = 1
/* Allocate Peripheral Frame 2 Register Structures: */
ECanaRegsFile : > ECAN_A, PAGE = 1
ECanaMboxesFile : > ECAN_AMBOX PAGE = 1
/* Allocate Peripheral Frame 1 Register Structures: */
SysCtrlRegsFile : > SYSTEM, PAGE = 1
SpiaRegsFile : > SPI_A, PAGE = 1
SciaRegsFile : > SCI_A, PAGE = 1
XIntruptRegsFile : > XINTRUPT, PAGE = 1
GpioMuxRegsFile : > GPIOMUX, PAGE = 1
GpioDataRegsFile : > GPIODAT PAGE = 1
AdcRegsFile : > ADC, PAGE = 1
EvaRegsFile : > EV_A, PAGE = 1
EvbRegsFile : > EV_B, PAGE = 1
ScibRegsFile : > SCI_B, PAGE = 1
McbspaRegsFile : > MCBSP_A, PAGE = 1
/* CSM Password Locations */
CsmPwlFile : > CSM_PWL, PAGE = 1
}

```

上述文件中，MEMORY 和 SECTIONS 是命令文件中最常用的两伪指令。MEMORY 伪指令用来表示实际存在目标系统中的可以使用的存储器范围，在这里每个存储器都有自己的名字、起始地址和长度。SECTIONS 伪指令是用来描述输入端是如何组合到输出端内的。什么意思呢？下面我们来具体分析。

在 MEMORY 伪指令中：

```

PAGE 0 :
PRAMH0 : origin = 0x3f8000, length = 0x001000
PAGE 1 : /* SARAM */
RAMM0 : origin = 0x000000, length = 0x000400
RAMM1 : origin = 0x000400, length = 0x000400
/* Peripheral Frame 2: */
.....
EV_A : origin = 0x007400, length = 0x000040
.....
DRAMH0 : origin = 0x3f9000, length = 0x001000

```

定义存储器 PRAM0 在 PAGE0，起始地址和长度分别为 0x3f8000 和 0x001000。定义 RAMM0、RAMM1 及 EV_A 等在 PAGE1，以及各自的起始地址和长度。

在 SECTION 伪指令中：

```

/* Allocate program areas: */
.reset : > PRAMH0, PAGE = 0
.text : > PRAMH0, PAGE = 0
.cinit : > PRAMH0, PAGE = 0

```

其中，.text: 所有可以执行的代码和常量；.cinit:全局变量和静态变量的 C 初始化记录；.reset: 复位中断向量，这些都放在 origin = 0x3f8000, length = 0x001000 中。

```

/* Allocate data areas: */

```

```
.stack : > RAMM1, PAGE = 1
.bss : > DRAMH0, PAGE = 1
.ebss : > DRAMH0, PAGE = 1
.const : > DRAMH0, PAGE = 1
.econst : > DRAMH0, PAGE = 1
.systemem : > DRAMH0, PAGE = 1
/* Allocate Peripheral Frame 1 Register Structures: */
EvaRegsFile : > EV_A, PAGE = 1
```

其中，`.stack`:为系统堆栈保留的空间，主要用于和函数传递变量或为局部变量分配空间；`.bss`:为全局变量和局部变量保留的空间，在程序上电时，`cinit`空间中的数据复制出来并存储在`.bss`空间中；`.ebss`: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，在程序上电时，`cinit`空间中的数据复制出来并存储在`.ebss`中；`.const`:包含字符串常量和初始化的全局变量和静态变量（由`const`）的初始化和说明；`.econst`:包含字符串常量和初始化的全局变量和静态变量（由`far const`）的初始化和说明；`.systemem`:为动态存储分配保留的空间，如果有宏函数，此空间被宏函数占用，如果没有的话，此空间保留为0。

我们再来看 `EVA_REGS` 如何来指定在 `0x7400~0x7431`，在 `MEMORY` 中 `EV_A:origin = 0x007400, length = 0x000040`，表明 `EV_A` 处在 `PAGE1` 起始地址 `0X7400`，长度 `0X40`。在 `SECTIONS` 中 `EvaRegsFile :> EV_A, PAGE = 1` 表明 `EvaRegsFile` 映射在 `EV_A` 中。而在工程中的 `DSP281x_GlobalVariableDefs.c` 文件中定义：

```
#pragma DATA_SECTION(EvaRegs, "EvaRegsFile");
volatile struct EVA_REGS EvaRegs;
```

其中：`EVA_REGS` 是结构体，定义如下：

```
struct EVA_REGS {
union GPTCONA_REG GPTCONA; //0x7400
Uint16 T1CNT; //0x7401
Uint16 T1CMPR; //0x7402
Uint16 T1PR; //0x7403
union T1CON_REG T1CON; //0x7404
Uint16 T2CNT; //0x7405
Uint16 T2CMPR; //0x7406
Uint16 T2PR; //0x7407
union T2CON_REG T2CON; //0x7408
union EXTCONA_REG EXTCONA; //0x7409
Uint16 rsvd1[7]; //0x740A-10
union COMCONA_REG COMCONA; //0x7411
Uint16 rsvd2; //0x7412
union ACTRA_REG ACTRA; //0x7413
Uint16 rsvd3; //0x7414
union DBTCON_REG DBTCONA; //0x7415
Uint16 rsvd4; //0x7416
Uint16 CMPR1; //0x7417
Uint16 CMPR2; //0x7418
Uint16 CMPR3; //0x7419
Uint16 rsvd5[6]; //0x741A-F
union CAPCONA_REG CAPCONA; //0x7420
```