版本：A

**CMD文件**

|  |  |
| --- | --- |
| 编制： | Lizy |
| 校对： | - |
| 2023.2.21 | |

更改记录

|  |  |
| --- | --- |
| **发布日期** | **更改描述** |
| 20230221 | 第一版发布 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# CMD文件是什么？

CMD：linker command files链接命令文件



CMD文件的作用：为**程序代码**和**数据**分配物理地址。

告诉链接程序怎么计算地址和分配空间；

## C语言生成的段

C语言生成的段分为**已初始化的段**和**未初始化的段**。已初始化的段含有真正的指令和数据，一般放在**程序存储空间**；未初始化的段只保留变量的地址空间，程序利用这些空间在运行时创建和存储变量，在DSP上电调用\_c\_init0初始化库前，没有真实内容，未初始化的段一般放在**数据存储空间**。

1. 已初始化的段

.text：编译后生成的**二进制指令代码段**；

.cinit：**全局和静态变量**初始化的常数，代码运行时拷贝到相应的变量空间中；

.const：**字符串常量**和全局、静态由**const**声明的变量；

.econst：**字符串常量**和全局、静态由**far** **const**声明的变量；

.pinit：全局**构造器**（C++）程序列表;

.switch：存放程序中**switch语句的跳转地址表**

1. 未初始化的段

.bss：存放**没有初始化的全局和静态变量**，为其保留的空间；程序上电时，数据从.cinit空间中复制出来并存放在.bss空间中；

.ebss：为使用**大寄存器模式**时预留出来的全局和局部变量；

.stack：C中的**栈**，用于保存返回地址、函数间的参数传递、存储局部变量和保存中间结果；

.system：为动态内存分配保留的空间（malloc、realloc）；

.esystem：为动态内存分配保留的空间（far malloc）；

## 自己定义段

### CMD格式

**MEMORY**

{

**PAGE 0:** /\*Program Memory\*/

RAML012 : origin = 0x008000, length = 0x003000

…..

**PAGE 1:** /\*Data Memory/

BOOT\_RSVD : origin = 0x000000, length = 0x000050

**……**

}

**SECTIONS**

{

PieVectTableFile: > PIE\_VECT, PAGE = 1

}

分为MEMORY和SECTIONS两个部分。

通过MEMORY伪指令来**指示物理存储**空间

通过SECTION伪指令来**分配段到物理存储**空间

（段类似容器的抽象概念，并非实际的物理空间）

MEMORY又分为**程序段PAGE 0**和**数据段PAGE 1。**

### 自定义段

#program CODE\_SECTION(symbol, “section name”); //程序段**PAGE 0**

#program DATA\_SECTION(symbol, “section name”); //数据段**PAGE 1**

symbol——符号，可以是函数名/变量名

section name——自己定义的段名

**eg:**

#pragma CODE\_SECTION(timer0\_isr, "ramfuncs");

#pragma DATA\_SECTION(AdcRegs,"AdcRegsFile");

将这句话放在对应.c的起始上；

**注：**

不能在函数体内声明#program

必须在符号被定义和使用前声明#program

#program定义

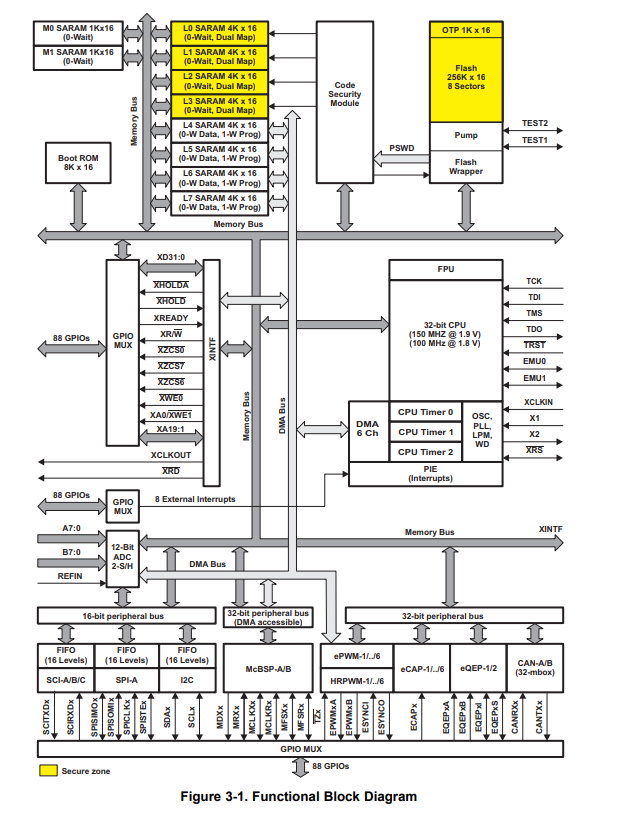
# CMD编写

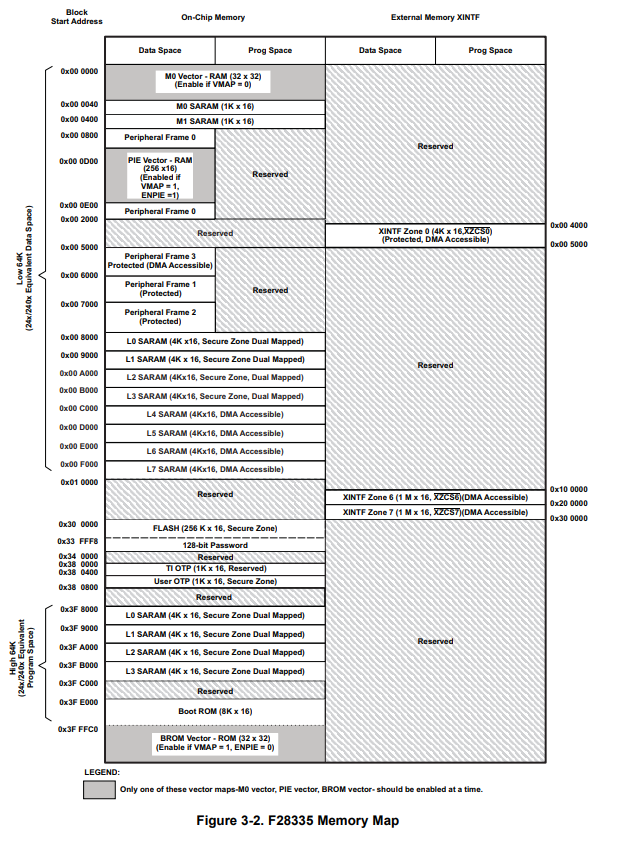
F28335中CMD文件一般有两个，RAM中调试时用的两个CMD文件分别为DSP2833x\_Headers\_nonBIOS.cmd和28335\_RAM\_lnk.cmd；烧写程序到FLASH中时用的两个CMD文件为DSP2833x\_Headers\_nonBIOS.cmd和Flash.cmd。

## F28335的存储映射

F28335片上存储器：FLASH:265Kx16bit;SARAM:34Kx16bit;

BOOT ROM:8Kx16bit;OTP ROM:2Kx16bit;





1. SARAM

SARAM是Single Acess RAM的缩写，即为单口随机读/写存储器，简称片内RAM。F28335片内RAM的大小为34Kx16bit，由10个块组成，M0,M1,L0-L7存储块。这些存储块都可以单独被访问，并且均可作为程序空间或数据空间。

1. FLASH

片内FLASH大小为265Kx16bit，这部分空间均可作为程序空间或数据空间。FLASH分为8个扇区，每一个扇区都可以单独操作。

1. 外设帧

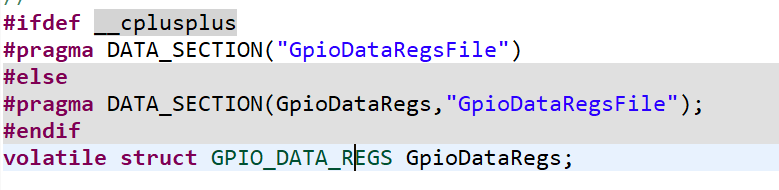
外设帧0、1、2、3只能是数据空间，不能是程序空间。这些空间存放着F28335外设寄存器，其中外设帧1、2、3空间是protectd，要对这3个空间中的寄存器操作，需要进行EALLOW声明，并以EDIS结束。

## DSP2833x\_Headers\_nonBIOS.cmd

DSP2833x\_Headers\_nonBIOS.cmd是通用的，主要作用是把**外设寄存器**产生的数据段映射到相应的存储空间，和DSP2833x\_GlobalVariableDefs.c文件对应。

寄存器和存储器本质上都是实际存在具有**存储功能的物理空间**，只是一个在CPU内部，一个在CPU外部。操作F28335片上外设其实就是对寄存器的存储空间进行写入或读出数据。以GPIO数据寄存器为例，看看如何利用CMD文件进行寄存器映射：

1. 在DSP2833x\_GlobalVariableDefs.c中定义一个数据段GpioDataRegsFile，将volatile struct GPIO\_DATA\_REGS类型变量GpioDataRegs映射到改段里



1. 在DSP2833x\_Headers\_nonBIOS.cmd文件的SECTION中给数据段GpioDataRegsFile指定其物理空间：



将GpioDataRegsFile分配到PAGE1中的GPIODAT空间中。

此时，建立好了一个与物理空间相对应的GPIO数据寄存器的数据段。

1. 在DSP2833x\_Headers\_nonBIOS.cmd文件MEMORY的PAGE 1数据空间中**定义名为**GPIODAT的**物理空间**，起始地址为0x006FC0，大小为0x000020：



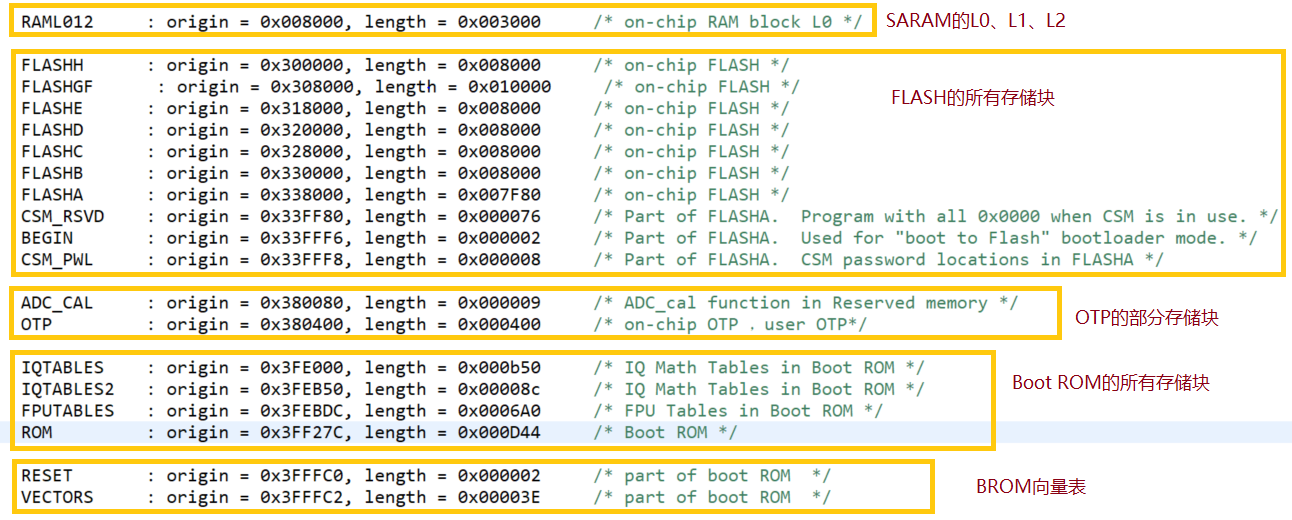
可以查阅F28335的数据手册可以看到，GPIO\_DATA\_REGS类型是一个大类，包括GPADAT- GPCTOGGLE 12个寄存器，大小均为2x16bit还有8x16bit的保留位，共32x16bit的空间。

|  |  |  |  |
| --- | --- | --- | --- |
| GPIO Data Registers | | | |
| Name | Address | Size(x16) | Register Description Bit Description |
| GPADAT | 0x6FC0 | 2 | GPIO A Data Register (GPIO0-GPIO31) |
| GPASET | 0x6FC2 | 2 | GPIO A Set Register (GPIO0-GPIO31) |
| GPACLEAR | 0x6FC4 | 2 | GPIO A Clear Register (GPIO0-GPIO31) |
| GPATOGGLE | 0x6FC6 | 2 | GPIO A Toggle Register (GPIO0-GPIO31) |
| GPBDAT | 0x6FC8 | 2 | GPIO B Data Register (GPIO32-GPIO63) |
| GPBSET | 0x6FCA | 2 | GPIO B Set Register (GPIO32-GPIO63) |
| GPBCLEAR | 0x6FCC | 2 | GPIO B Clear Register (GPIO32-GPIO63) |
| GPBTOGGLE | 0x6FCE | 2 | GPIO B Toggle Register (GPIO32-GPIO63) |
| GPCDAT | 0x6FD0 | 2 | GPIO C Data Register (GPIO64 - GPIO87) |
| GPCSET | 0x6FD2 | 2 | GPIO C Set Register (GPIO64 - GPIO87) |
| GPCCLEAR | 0x6FD4 | 2 | GPIO C Clear Register (GPIO64 - GPIO87) |
| GPCTOGGLE | 0x6FD6 | 2 | GPIO C Toggle Register (GPIO64 - GPIO87) |
| Reserved | 0x6FD8 0x6FDF | 8 | - |

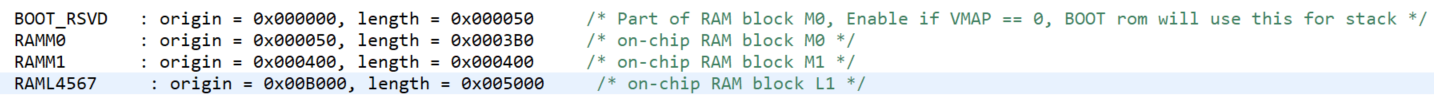
这样，每次操作GpioDataRegs变量时，相当于对0x006FC0-0x006FE0这段物理地址进行操作。

## FLASH.cmd/28335\_RAM\_lnk.cmd

1. 在FLASH.cmd的MEMORY中的程序空间PAGE 0用来指示RAM、FLASH、ROM的物理地址，与F28335存储器映像是一一对应的：

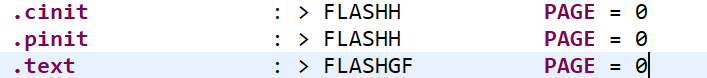


在FLASH.cmd的MEMORY中的数据空间PAGE 1用来指示RAM中M0、M1、L4-L7的物理地址，与F28335存储器映像是一一对应的：



28335\_RAM\_lnk.cmd中的MEMORY中主要是用来指示RAM的物理地址。

SECTIONS中将代码段和程序段分配给MEMORY物理地址，代码段和程序段包括C语言生成的段和自己定义的段：



## 修改cmd文件

实际中，一般先使用TI自带的CMD文件，需要修改CMD文件一般有2种情况：一是编译报错，空间不足，一种是程序运行不够快，想要加快运行时间。

### 存储空间不足

由于每个段的物理存储空间是有限的，如果程序需要用到占用空间很大的变量时，比如查找表之类的，编译器编译时可能会报存储空间不足，他会告诉你需要多大的空间，可以根据这个修改CMD文件。

1. 可以修改CMD文件的MEMORY部分的代码。比如。合并RAM或FLASH来创建更大的内存块。

例如，RAML4的空间不足，可以直接增大RAML4的空间，并减小相邻的RAML5的空间，甚至删掉RAML5，注意需要同时修改origin和length。

eg:

原：

RAML4 : origin = 0x00C000, length = 0x001000

RAML5 : origin = 0x00D000, length = 0x001000

修改为：

RAML4 : origin = 0x00C000, length = 0x001500

RAML5 : origin = 0x00D500, length = 0x000500

1. 可以修改CMD文件的SECTION部分的代码。可以将大的段分割开放到多个内存块中，使用以下语法，将.text分散到FLASHA、FLASHB、FLASHC中：

eg:

原：

. text : > FLASHA PAGE = 0

修改为：

. text **: >>** FLASHA **|** FLASHB **|** FLASHC PAGE = 0

还可以告诉编译器放到第一个能放下的段中，使用如下语法。这里的意思是，编译器首先去看第一个内存块FLASHA，如果他能放下，就放到FLASHA中，如果放不下，则去看第二个内存块FLASHB，以此类推。

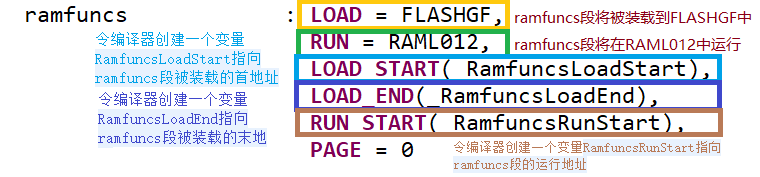
. text **: >** FLASHA **|** FLASHB **|** FLASHC PAGE = 0

### 加快程序运行速度

#### 搬运到RAM

当程序烧写到FLASH中运行时，速度大概降到RAM中运行的70%-80%，为了提高程序的运行速度，在上电后，将FLASH中的部分程序（对时间比较敏感或计算量大）可以复制到RAM中来运行。

1. 在FLASH.cmd中定义运行搬移程序的段ramfuncs



1. 在.c中定义要运行在ramfuncs段的函数



将函数timer0\_isr运行ramfuncs段

1. 还需要在.c中调用MemCopy（），将FLASH中的内容复制到RAM中：



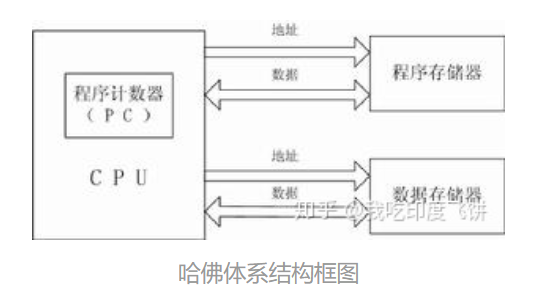
其中MemCopy（）函数定义如下：

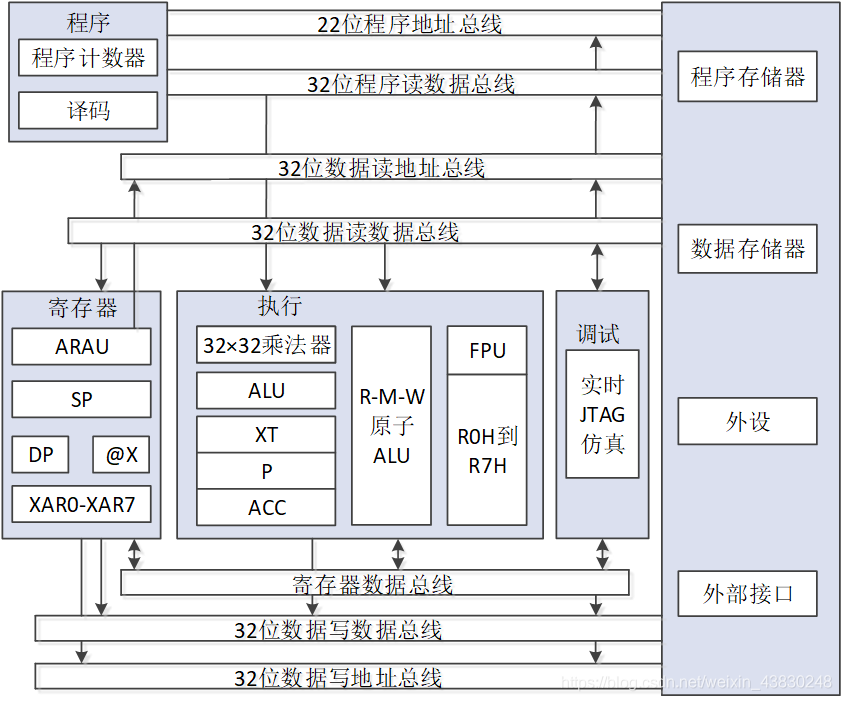


# 补充内容

### 为什么在CMD中分为程序空间和数据空间——哈佛结构

TMS320F28335的总线结构为哈佛结构，程序存储空间和数据存储空间是独立的，如下图所示：





程序读总线：22根地址总线和32根数据总线；

数据读/写总线：32根地址线和32根数据线；

（地址线用来寻址，即在计算机中找一个数据时，先访问其地址。数据线用来传输数据）

CMD主要是分配256K x 16bit的FLASH和34K x 16bit的随机访问存储器SARAM的空间给程序存储器和数据存储器；

哈佛结构的CPU可以在**一个时钟周期完成对数据存储器和程序存储器的访问**。

# 疑问：

1. flash.cmd中使用过的存储空间，ram.cmd中还可以用吗？//只有begin指向的起始不同；
2. 如果没有switch的段，switch语句放哪？
3. **大寄存器模式？**