F2812 与 F28335 的区别

最大的区别就是 28335 是浮点的，而 2812 是定点的。

F2812 主要指标

150 MHz/EMIF /128KB Flash/36KB RAM/GPIO 56 路/McBSP/176-pin Package

F28335 主要指标

300 MFLOPS/独立 DMA/512KB Flash/68 KB RAM/32-bit EMI/GPIO 88 路

还有一点 28335 将 2812 的 EV 分解成了相互独立的 epwm，ecap，eq 三个模块互相之间互不干扰，因此可以比较方便地实现复杂的信号输出。尤其是 epwm 相对于 EV 中的 pwm 输出功能，有了很大的提高。

F28335 比 F2812 多了一个 MAC 单元，也就是速度增加了一倍。

F28335 是带浮点运算的，动态范围更大。

F2833x 的执行速度，比相同时钟频率的 F28xx 系列定点芯片，快 50%。处理数学运算性能提升 2.45 倍，控制算法性能提升 1.57 倍，DSP 性能提升 1.38 倍。总体性能提升近 2 倍。

TMS320F28335 的 ADC

TMS320F28335 上有 16 通道、12 位的模数转换器 ADC。他可以被配置为两个独立的 8 通道输入模式，也可以通过配置 AdcRegs.ADCTRL1.bit.SEQ_CASC=1，将其设置为一个 16 通道的级联输入模式。输入的方式可以通过配置 AdcRegs.ADCTRL1.bit.ACQ_PS=1，将其设置为顺序采集。即从低通道开始到高通道结束。

值得注意的是片上 ADC 的输入电压范围为 0--3V，一旦超过 3V，片上的 ADC 模块将会被烧掉。TI 上的 DATASHEET 介绍其 ADC 的精度可达到 12 位，实际上达到不了。经测试，我们估计最好的时候可以达到 11 位就不错了。下面我们来简单介绍一下 ADC 模块的原理。其数字值由下面公司来计算，其中公式中的 3 为片内参考电压 Digital Value=4096*（Input Analog Voltage-ADCLO）/3 ；（when 0 V < input < 3 V）。ADC 可以分为 SEQ1 和 SEQ2 两个模块，其中 SEQ1 包括 ADCIN00--ADCIN07;SEQ2 包括 ADCIN08--ADCIN15。SEQ1 模块可以通过软件、PWM、外部中断引脚来启动，而 SEQ2 不可以通过外部中断引脚来启动。另外就是 ADC 可以与 DMA 进行数据交换。

TMS320F28335 的时钟介绍

TMS320F28335 上有一个基于 PLL 电路的片上时钟模块，为 CPU 及外设提供时钟有两种方式：一种是用外部的时钟源，将其连接到 X1 引脚上或者 XCLKIN 引脚上，X2 接地；另一种是使用振荡器产生时钟，用 30MHz 的晶体和两个 20PF 的电容组成的电路分别连接到 X1 和 X2 引脚上，XCLKIN 引脚接地。我们常用第二种来产生时钟。此时钟将通过一个内部 PLL 锁相环电路，进行倍频。由于 F28335 的最大工作频率是 150M，所以倍频值最大是 5。其中倍频值由 PLLCR 的低四位和 PLLSTS 的第 7、8 位来决定。其详细的倍频值可以参照 TMS320F28335 的 Datasheet。下面是 F28335 的时钟设置：

```
void InitPll(Uint16 val, Uint16 divsel)
{

// Make sure the PLL is not running in limp mode
    if (SysCtrlRegs.PLLSTS.bit.MCLKSTS != 0)
    {
    // Missing external clock has been detected
  // Replace this line with a call to an appropriate
    // SystemShutdown(); function.
        asm("        ESTOP0");
    }

// DIVSEL MUST be 0 before PLLCR can be changed from
// 0x0000. It is set to 0 by an external reset XRSn
    // This puts us in 1/4
    if (SysCtrlRegs.PLLSTS.bit.DIVSEL != 0)
    {
        EALLOW;
        SysCtrlRegs.PLLSTS.bit.DIVSEL = 0;
        EDIS;
    }

    // Change the PLLCR
    if (SysCtrlRegs.PLLCR.bit.DIV != val)
    {
```

```
EALLOW;
// Before setting PLLCR turn off missing clock detect logic
SysCtrlRegs.PLLSTS.bit.MCLKOFF = 1;
SysCtrlRegs.PLLCR.bit.DIV = val;
EDIS;

// Optional: Wait for PLL to lock.
// During this time the CPU will switch to OSCCLK/2 until
// the PLL is stable.  Once the PLL is stable the CPU will
// switch to the new PLL value.
//
// This time-to-lock is monitored by a PLL lock counter.
//
// Code is not required to sit and wait for the PLL to lock.
// However, if the code does anything that is timing critical,
// and requires the correct clock be locked, then it is best to
// wait until this switching has completed.

// Wait for the PLL lock bit to be set.

// The watchdog should be disabled before this loop, or fed within
// the loop via ServiceDog().

// Uncomment to disable the watchdog
DisableDog();

while(SysCtrlRegs.PLLSTS.bit.PLLLOCKS != 1)
{
// Uncomment to service the watchdog
// ServiceDog();
}

EALLOW;
SysCtrlRegs.PLLSTS.bit.MCLKOFF = 0;
EDIS;
```

```
}

// If switching to 1/2
if((divsel == 1)||(divsel == 2))
{
EALLOW;
SysCtrlRegs.PLLSTS.bit.DIVSEL = divsel;
EDIS;
}

// If switching to 1/1
// * First go to 1/2 and let the power settle
//   The time required will depend on the system, this is only an example
// * Then switch to 1/1
if(divsel == 3)
{
EALLOW;
SysCtrlRegs.PLLSTS.bit.DIVSEL = 2;
DELAY_US(50L);
SysCtrlRegs.PLLSTS.bit.DIVSEL = 3;
EDIS;
  }
}
```

如果我们希望 DSP 工作在某一个频率下，我们就可以对 Uint16 val, Uint16 divsel 两个参数进行设定。

TMS320F28335 外部中断总结

在这里我们要十分清楚 DSP 的中断系统。C28XX 一共有 16 个中断源，其中有 2 个不可屏蔽的中断 RESET 和 NMI、定时器 1 和定时器 2 分别使用中断 13 和 14。这样还有 12 个中断都直接连接到外设中断扩展模块 PIE 上。说的简单一点就是 PIE 通过 12 根线与 28335 核的 12 个中断线相连。而 PIE 的另外一侧有 12*8 根线分别连接到外设，如 AD、SPI、EXINT 等等。这样 PIE 共管理 12*8=96 个外部中断。这 12 组大中断由 28335 核的中断寄存器 IER 来控制，即 IER 确定每个中断到底属于哪一组大中断（如 IER |= M_INT12;说明我们要用第 12 组的

中断，但是第 12 组里面的什么中断 CPU 并不知道需要再由 PIEIER 确定 ）。接下来再由 PIE 模块中的寄存器 PIEIER 中的低 8 确定该中断是这一组的第几个中断,这些配置都要告诉 CPU（我们不难想象到 PIEIER 共有 12 总即从 PIEIER1-PIEIER12）。另外，PIE 模块还有中断标志寄存器 PIEIFR，同样它的低 8 位是来自外部中断的 8 个标志位，同样 CPU 的 IFR 寄存器是中断组的标志寄存器。由此看来，CPU 的所有中断寄存器控制 12 组的中断，PIE 的所有中断寄存器控制每组内 8 个的中断。除此之外，我们用到哪一个外部中断，相应的还有外部中断的寄存器，需要注意的就是外部中断的标志要自己通过软件来清零。而 PIE 和 CPU 的中断标志寄存器由硬件来清零。

```
EALLOW;  // This is needed to write to EALLOW protected registers

PieVectTable.XINT2 = &ISRExint; //告诉中断入口地址

EDIS;    // This is needed to disable write to EALLOW protected registers


PieCtrlRegs.PIECTRL.bit.ENPIE = 1;          // Enable the PIE block 使能 PIE

PieCtrlRegs.PIEIER1.bit.INTx5= 1;   //使能第一组中的中断 5

IER |= M_INT1;                              // Enable CPU 第一组中断


EINT;   // Enable Global interrupt INTM

ERTM;   // Enable Global realtime interrupt DBGM
```

也就是说，12 组中的每个中断都要完成上面的相同配置，剩下的才是去配置自己的中断。如我们提到的 EXINT，即外面来个低电平我们就进入中断，完成我们的程序。在这里要介绍一下，DSP 的 GPIO 口都可以配置为外部中断口，其配置方法如下：

```
GpioCtrlRegs.GPBMUX2.bit.GPIO54 = 0; //选择他们是 GPIO 口
```

```
GpioCtrlRegs.GPBMUX2.bit.GPIO55 = 0;

GpioCtrlRegs.GPBMUX2.bit.GPIO56 = 0;

GpioCtrlRegs.GPBMUX2.bit.GPIO57 = 0;

GpioCtrlRegs.GPBDIR.bit.GPIO54 = 0;//选择他们都是输入口

GpioCtrlRegs.GPBDIR.bit.GPIO55 = 0;

GpioCtrlRegs.GPBDIR.bit.GPIO56 = 0;

GpioCtrlRegs.GPBDIR.bit.GPIO57 = 0;

GpioCtrlRegs.GPBQSEL2.bit.GPIO54= 0;//GPIO 时钟和系统时钟一样且支持 GPIO

GpioCtrlRegs.GPBQSEL2.bit.GPIO55= 0;

GpioCtrlRegs.GPBQSEL2.bit.GPIO56= 0;

GpioCtrlRegs.GPBQSEL2.bit.GPIO57= 0;

GpioIntRegs.GPIOXINT3SEL.bit.GPIOSEL = 54;//中断 3 选择 GPIO

GpioIntRegs.GPIOXINT4SEL.bit.GPIOSEL = 55;

GpioIntRegs.GPIOXINT5SEL.bit.GPIOSEL = 56;

GpioIntRegs.GPIOXINT6SEL.bit.GPIOSEL = 57;

XIntruptRegs.XINT3CR.bit.POLARITY= 0;//触发模式为下降沿触发

XIntruptRegs.XINT4CR.bit.POLARITY= 0;

XIntruptRegs.XINT5CR.bit.POLARITY= 0;
```

XIntruptRegs.XINT6CR.bit.POLARITY= 0;

XIntruptRegs.XINT3CR.bit.ENABLE = 1;//使能中断

XIntruptRegs.XINT4CR.bit.ENABLE = 1;

XIntruptRegs.XINT5CR.bit.ENABLE = 1;

XIntruptRegs.XINT6CR.bit.ENABLE = 1;

注意一点就是外部中断 1 和 2 只能对 GPIO0—GPIO31 配置；外部中断 3 和 4、5、6、7 只对 GPIO32—GPIO63 配置。

F28335 的 McBSP

McBSP 是 Multichannel Buffered Serial Port 的缩写，即多通道缓冲型串行接口，是一种多功能的同步串行接口，它具有很强的可编程能力，可以配置为多种同步串口标准，直接与各种器件高速接口：

*T1/E1 标准：通信器件

*MVIP 和 ST-BUS 标准：通信器件

* IOM-2 标准：ISDN 器件

*AC97 标准：PC Audio Codec 器件

* IIS 标准：Codec 器件

*SPI：串行 A/D、D/A，串行存储器等器件

McBSP 由发送器和接收器构成，各有 3 个信号：位-时钟、帧同步和串行数据，所以一个 McBSP 有下列信号：FSR、CLKR、DR 和 FSX、CLKX、DX。 帧同步信号：FSR、FSX 位时钟：CLKR、CLKX 和 串行数据流：DR、DX 。同步串行通信协议包含：

*串行数据流起始时刻称为帧同步事件。帧同步事件由位-时钟采样帧同步信号给出。

*串行数据流长度：串行传输的数据流位数达到设定的长度后，结束本次传输，等下一个帧同步信号达到，再发起另一次串行传输。

*串行数据流传输速度：即每一个串行位的持续时间，由位-时钟决定。

*FSR（FSX）、CLKR（CLKX）、DR（DX）三者之间的关系即如何取得帧同步事件、何时采样串行数据位流、或何时输出串行数据位流，是可以通过 McBSP 的寄存器进行配置的。

TMS320F28335 的 GPIO

F28335 有三种 32 位的 I/O 口，依次 PORTA(GPIO0-GPIO31)， PORTB(GPIO32-GPIO63)， PORTC(GPIO64-GPIO87)，这些口都可以配置为普通的数字 IO 口同样也能被配置为外部接口。这样涉及到了 IO 的寄存器，IO 口共有三类寄存器：控制寄存器、数据寄存器和中断控制寄存器。下面依次介绍这些寄存器。

```
struct GPIO_CTRL_REGS {

   union  GPACTRL_REG  GPACTRL;    // GPIO A Control Register (GPIO0 to 31)
union  GPA1_REG     GPAQSEL1;  // GPIO A Qualifier Select 1 Register (GPIO0 to 15)
union  GPA2_REG     GPAQSEL2;  // GPIO A Qualifier Select 2 Register (GPIO16 to 31)
   union  GPA1_REG      GPAMUX1;    // GPIO A Mux 1 Register (GPIO0 to 15)
   union  GPA2_REG      GPAMUX2;    // GPIO A Mux 2 Register (GPIO16 to 31)
  union  GPADAT_REG   GPADIR;     // GPIO A Direction Register (GPIO0 to 31)
union  GPADAT_REG   GPAPUD;     // GPIO A Pull Up Disable Register (GPIO0 to 31)
                    Uint32                  rsvd1;
  union  GPBCTRL_REG  GPBCTRL;    // GPIO B Control Register (GPIO32 to 63)
union  GPB1_REG   GPBQSEL1;  // GPIO B Qualifier Select 1 Register (GPIO32 to 47)
union  GPB2_REG   GPBQSEL2;  // GPIO B Qualifier Select 2 Register (GPIO48 to 63)
   union  GPB1_REG     GPBMUX1;    // GPIO B Mux 1 Register (GPIO32 to 47)
   union  GPB2_REG     GPBMUX2;    // GPIO B Mux 2 Register (GPIO48 to 63)
```

union GPBDAT_REG GPBDIR; // GPIO B Direction Register (GPIO32 to 63)
union GPBDAT_REG GPBPUD; // GPIO B Pull Up Disable Register (GPIO32 to 63)
Uint16 rsvd2[8];
union GPC1_REG GPCMUX1; // GPIO C Mux 1 Register (GPIO64 to 79)
union GPC2_REG GPCMUX2; // GPIO C Mux 2 Register (GPIO80 to 95)
union GPCDAT_REG GPCDIR; // GPIO C Direction Register (GPIO64 to 95)

union GPCDAT_REG GPCPUD; // GPIO C Pull Up Disable Register (GPIO64 to 95)
};其中 GPXCTRL 寄存器的作用是设置采样窗周期 T=2*GPXCTRL*Tsysclk；GPxQSEL 中每两位控制一个引脚，确定是 3 周期采样还是 6 周期采样或者不用采样；GPxMUX 的功能很简单就是配置各个引脚的功能，或者是数字 IO 口，或者是外部接口。同样是两位控制一个引脚；GPxDIR 是控制每个引脚的输入或是输出，0 是输入，1 是输出。还有一个 GPxPUD 寄存器，是用来使能或禁止指定接口的内部上拉。以上这些就是属于 IO 口的控制寄存器。下面的是 IO 数据类寄存器所有集合 struct GPIO_DATA_REGS {
union GPADAT_REG GPADAT; // GPIO Data Register (GPIO0 to 31)
union GPADAT_REG GPASET; // GPIO Data Set Register (GPIO0 to 31)
union GPADAT_REG GPACLEAR; // GPIO Data Clear Register (GPIO0 to 31)
union GPADAT_REG GPATOGGLE; // GPIO Data Toggle Register (GPIO0 to 31)
union GPBDAT_REG GPBDAT; // GPIO Data Register (GPIO32 to 63)
union GPBDAT_REG GPBSET; // GPIO Data Set Register (GPIO32 to 63)
union GPBDAT_REG GPBCLEAR; // GPIO Data Clear Register (GPIO32 to 63)
union GPBDAT_REG GPBTOGGLE; // GPIO Data Toggle Register (GPIO32 to 63)
union GPCDAT_REG GPCDAT; // GPIO Data Register (GPIO64 to 95)
union GPCDAT_REG GPCSET; // GPIO Data Set Register (GPIO64 to 95)
union GPCDAT_REG GPCCLEAR; // GPIO Data Clear Register (GPIO64 to 95)
union GPCDAT_REG GPCTOGGLE; // GPIO Data Toggle Register (GPIO64 to 95)
Uint16 rsvd1[8];};这些寄存器就是为了写数据，读数据用的。再下面就是中断寄存器 struct GPIO_INT_REGS {
union GPIOXINT_REG GPIOXINT1SEL; // XINT1 GPIO Input Selection
union GPIOXINT_REG GPIOXINT2SEL; // XINT2 GPIO Input Selection
union GPIOXINT_REG GPIOXNMISEL; // XNMI_Xint13 GPIO Input Selection
union GPIOXINT_REG GPIOXINT3SEL; // XINT3 GPIO Input Selection
union GPIOXINT_REG GPIOXINT4SEL; // XINT4 GPIO Input Selection
union GPIOXINT_REG GPIOXINT5SEL; // XINT5 GPIO Input Selection
union GPIOXINT_REG GPIOXINT6SEL; // XINT6 GPIO Input Selection
union GPIOXINT_REG GPIOXINT7SEL; // XINT7 GPIO Input Selection

union GPADAT_REG GPIOLPMSEL; // Low power modes GP I/O input select
}; 以上就是 IO 口的外部中断寄存器，我们可以对上面 GPIO0-GPIO63 进行外部中断配置。

TMS320F28335 的存储空间

TMS320F28335 为哈佛结构的 DSP，在逻辑上有 4M×16 位的程序空间和 4M×16 位点的数据空间，但在物理上已将程序空间和数据空间统一成一个 4M×16 位的空间。TMS320F28335 片上有 256K×16 位的 FLASH，34K×16 位的 SRAM，8K×16 位的 BOOT ROM，2K×16 位的 OPT ROM。

1、 TMS320F28335 片上 SARAM

TMS320F28335 片内共有 34K×16 位单周期单次访问随机存储器的 SARAM，分成 10 个块，他们分别称为 M0、M1、L0-L7。

M0 和 M1 块 SARAM 的大小均为 1K×16 位，当复位后，堆栈指针指向 M1 块的起始地址，堆栈指针向上生长。M0 和 M1 段都可以映射到程序区和数据区。

L0-L7 块 SARAM 的大小均为 4K×16 位，既可映射到程序空间，也可映射到数据空间，其中 L0-L3 可映射到两块不同的地址空间并且受片上的 FLASH 中的密码保护，以免存在上面的程序或数据，被他人非法拷贝。

2、 TMS320F28335 片上 FLASH 和 OTP

TMS320F28335 片上有 256K×16 位嵌入式 FLASH 存储器和 1K×16 位一次可编程 EEPROM 存储器，他们均受片上 FLASH 中的密码保护。FLASH 存储器由 8 个 32K×16 位扇区组成，用户可以对其中任何一个扇区进行擦除、编程和校验，而其他扇区不变。但是，不能在其中一个扇区上执行程序来擦除和编程其他的扇区。

3、 TMS320F28335 外部存储器接口

TMS320F28335 的外部存储器接口包括：20 位地址线，16（最大 32）位数据线，3 个片选控制线及读写控制线。这 3 个片选线映射到 3 个存储区域，Zone0, Zone6 和 Zone7。这 3 个存储器可分别设置不同的等待周期。

Zone0 存储区域： 0X004000—0X004FFF, 4K×16 位 可编程最少一个等待周期

Zone6 存储区域： 0X100000—0X1FFFFF, 1M×16 位 10ns 最少一个等待周期

Zone7 存储区域： 0X200000—0X2FFFFF, 1M×16 位 70ns 最少一个等待周期

eCAP 模块

1.介绍

eCAP 模块包括以下的资源：

*可分配的输入引脚。

*32-bit 时间基准（计数器）。

*4 个 32bit 时间窗捕获控制寄存器。

*独立的边缘极性选择。

*输入信号分频（2~62）。

*4Capture event 均可引起中断。

2. eCAP 模块功能分析

eCAP 模块可以设置为 event capture 模式或者是 APWM 模式，一般而言前者比较常用，在这里我们只对第一种进行介绍。在 event capture 模式下，一般可以将 eCAP 模块分为以下几个模块：事件分频、边沿极性选择与验证、中断控制。

2.1 事件分频

输入事件信号可通过分频器分频处理（分频系数 2~62），或直接跳过分频器。这个功能通常针对输入事件信号频率很高的情况下。

2.2 边沿极性选择与验证

1）4 个独立的边沿极性（上升沿/下降沿）选择通道。

2）Modulo4 序列发生器对 Each edge（共 4 路）进行事件验证。

3）CAPx 通过 Mod4 对事件边沿计数。CAPx 寄存器在下降沿时被装载。

2.3 32-bit 计数器（TSCTR）

此计数器为 event capture 提供事件基准，而时钟的计数则是基于系统时钟的。当此计数器计数超过范围时，则会产生相应的溢出标志，若溢出中断使能，则产生中断。此计数器在计算事件周期时非常有效。详细的资料请参看 spru807 应用部分。

2.4 中断控制

中断能够被 capture events（CEVT1-CEVT4,CTROVF）触发。计数溢出同样会提供中断。事件单独地被极性选择部分以及序列验证部分审核。这些事件中的一个被选择用来作为中断源送入 PIE。

设置中断的 Proper 过程

1） Disable global interrupts.

2） 停止 eCAP 计数。

3） Disable eCAP interrupts。

4） 设置外设寄存器。

5） 清除 eCAP 中断标志位。

6） Enable eCAP 中断。

7） Start ecap 计数器。

8） Enable global 中断。

3. eCAP 模块的理解

配置好 eCAP 模块的引脚后，外部事件由引脚输入，首先通过模块的分频部分，分频系数

为 2~62，也可以选择跳过分频部分。经过分频部分后的信号（通常频率会降低），送至边沿及序列审核部分，边沿审核即设置为上升沿或下降沿有效，序列审核则是分配当前对哪个寄存器（CAP1～CAP4）作用的问题，之后就是中断部分了。引起中断的中断源有 7 个，event capture 模式下有五个，分别是审核后各路的事件以及溢出中断。

CMD 配置文件介绍

CMD 文件的编写

1. COFF 格式

1> 通用目标文件格式（Common Object File Format）是一种流行的二进制可执行文件格式，二进制可执行文件包括库文件（lib），目标文件（obj）最终可执行文件（out）。，现今 PC 机上的 Windows95 和 NT4.0 以后的操作系统的二进制文件格式（PE）就是在 COFF 格式基础上的进一步扩充。

2> COFF 格式：详细的 COFF 文件格式包括段头，可执行代码和初始化数据，可重定位信息，行号入口，符号表，字符串表等，这些属于编写操作系统和编译器人员关心范畴。而对于 C 只需要了解定义段和给段分配空间就可以了。

3> 采用 COFF 更有利于模块化编程，程序员可以自由决定愿意把哪些代码归属到哪些段，然后加以不同的处理。

2. Section 目标文件中最小单位称为块。一个块就是最终在存储器映象中占据连续空间的一段代码或数据。

1> COFF 目标文件包含三个默认的块：

.text 可执行代码

.data 已初始化数据

.bss 为未初始化数据保留的空间

2> 汇编器对块的处理

未初始化块

.bss 变量存放空间

.usect 用户自定义的未初始化段

初始化块

.text 汇编指令代码

.data 常数数据（比如对变量的初始化数据）

.sect 用户自定义的已初始化段

.asect 通.sect，多了绝对地址定位功能，一般不用

3>C 语言的段

未初始化块（data）

.bss 存放全局和静态变量

.ebss 长调用的.bss(超过了 64K 地址限制)

.stack 存放 C 语言的栈

.sysmem 存放 C 语言的堆

.esysmem 长调用的.sysmem(超过了 64K 地址限制)

初始化块

.text 可执行代码和常数(program)

.switch switch 语句产生的常数表格 (program/低 64K 数据空间)

.pinit Tables for global constructors (C++)(program)

.cinit 用来存放对全局和静态变量的初始化常数值(program)

.const 全局和静态的 const 变量初始化值和字符串常数，（data）

.econst 长.const（可定位到任何地方）（data）

3>　　自定义段（C 语言）

#pragma DATA_SECTION(函数名或全局变量名,"用户自定义在数据空间的段名");

#pragma CODE_SECTION(函数名或全局变量名,"用户自定义在程序空间的段名");

不能在函数体内声明。

必须在定义和使用前声明

#pragma 可以阻止对未调用的函数的优化

3. 连接命令文件（CMD）

1> MEMORY 指定存储空间

MEMORY

{

PAGE 0:

name 0 [attr] : origin = constant, length = constant

PAGE n:

name n [attr] : origin = constant, length = constant

}

PAGE n:标示存储空间，n<255；PAGE 0 为程序存储空间；PAGE 1 为程序存储空间

name:存储空间名称

attr:存储空间属性：只读 R，只写 W，可包含可执行代码 X,可以被初始化 I。

orgin:用来定义存储空间的起始地址

Lenth:用来定义存储空间的长度

2>　　SECTIONS 分配段

SECTIONS

{

name　　: [property,property,……]

}

name:输出段的名称

property：输出段的属性：

load＝allocation（强制地址或存储空间名称）同>allocation：定义输出段将会被装载到哪里。

run= allocation（强制地址或存储空间名称）同>allocation：定义输出段将会在哪里运行。

注：CMD 文件中只出现一个关键字 load 或 run 时，表示两者的地址时表示两者的地址时重合的。

PAGE = n，段位于那个存储页面空间。

例：ramfuncs : LOAD = FLASHD,

RUN = RAML0,

LOAD_START(_RamfuncsLoadStart),

LOAD_END(_RamfuncsLoadEnd),

RUN_START(_RamfuncsRunStart),

PAGE = 0

3>　　直接写编译命令

-l rts2800_ml.lib 连接系统文件 rts2800_ml.lib

-o filename.out　　最终生成的二进制文件命名为 filename.out

-m filename.map 生成映射文件 filename.map

-stack 0x200　　堆栈为 512 字

关于 F28335 bootmode 的说明

F28335　bootmode 与 F2407 及 F2812 不同，有一些网友有疑问，现做说明如下：

1. ″当 28335 判断 boot mode 时会检查 gpio 87 86 85 84 ，然后读取各个引脚上的值来判断用哪种 bootmode。问题是之前这几个引脚是如何定义的呢？″

回复：CPU 上电复位后，GPIO84~87 缺省设置为 IO 引脚（输入）而且上拉，CPU 读跳线状态决定 bootmode 去向！！

2. “而且这几个脚和地址线 XA15 14 13 12 是复用的，如果使用外接跳线的话，当需要这几个地址引线时，又是如何解决这个矛盾？总不能手动把跳线还原为原来的状态吧！新手实在不知怎么弄，望请各位高手赐教！”

回复：虽然 GPIO84~87 与地址线 XA15 14 13 12 是复用的，但不影响寻址！因为当配置为地址线时，变为输出，原来的上拉（1 电平）及下拉（0 电平）均由电阻实现，并不影响输出状态。（一般上拉电阻 20K,下拉电阻 2K）

3. 为实现 bootmode 的改变，需接跳线器及上拉、下拉电阻。

4. TI F28335 datasheet 上有 Boot Mode Selection