#define module\_init(x) \_\_initcall(x);

#define \_\_initcall(fn) device\_initcall(fn)

#define device\_initcall(fn) \_\_define\_initcall("6",fn,6)

#define \_\_define\_initcall(level,fn,id) static initcall\_t \_\_initcall\_##fn##id \_\_used \_\_attribute\_\_((\_\_section\_\_(".initcall" level ".init"))) = fn

#define module\_init(x) \

static initcall\_t \_\_initcall\_##fn##id \_\_used \_\_attribute\_\_((\_\_section\_\_(".initcall" level ".init"))) = fn

module\_init(**ft5x06\_ts**) ==

static initcall\_t \_\_initcall\_**ft5x06\_ts**\_6 \_\_used \_\_attribute\_\_((\_\_section\_\_(".initcall6.init"))) = **ft5x06\_ts;**

关于\_\_attribute\_\_[《C30编译器用户指南》](https://www.baidu.com/s?wd=%E3%80%8AC30%E7%BC%96%E8%AF%91%E5%99%A8%E7%94%A8%E6%88%B7%E6%8C%87%E5%8D%97%E3%80%8B&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvPvDvPjmkujNbnH0vPARY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjnLPHmYrjDv)中有介绍   
2.3.1 指定变量的属性  
编译器的关键字 \_\_attribute\_\_ 用来指定 **变量或结构 位域** 的特殊属性。关键字后的  
双括弧中的内容是属性说明。下面是目前支持的变量属性：  
• address (addr)  
• aligned (alignment)  
• boot   
• deprecated  
• fillupper  
• far  
• mode (mode)  
• near  
• noload  
• packed  
• persistent  
• reverse (alignment)  
• section ("section-name")  
• secure   
• sfr (address)  
• space (space)  
• transparent\_union  
• unordered  
• unused  
• weak

也可以通过在关键字前后使用 \_\_（双下划线）来指定属性（例如，用\_\_aligned\_\_代替aligned）。这样将使你在头文件中使用它们时不必考虑会出现与宏同名的情况。要指定多个属性，可在双括弧内用逗号将属性分隔开，例如：  
 **\_\_attribute\_\_** ((**aligned (16)**, **packed**))。

所以

static initcall\_t \_\_initcall\_**ft5x06\_ts**\_6 \_\_used \_\_attribute\_\_((\_\_section\_\_(".initcall6.init"))) = **ft5x06\_ts;**

相当于

static initcall\_t \_\_initcall\_**ft5x06\_ts**\_6 = **ft5x06\_ts;**

//initcall\_t是一个函数指针 typedef int (\*initcall\_t)(void)

vmlinux.lds中有如下

.initcall.init : AT(ADDR(.initcall.init) – (0xc0000000 – 0x00000000)) {

\_\_initcall\_start = .;

\*(.initcallearly.init) \_\_early\_initcall\_end = .;

\*(.initcall0.init) \*(.initcall0s.init) \*(.initcall1.init) \*(.initcall1s.init) \*(.initcall2.init) \*(.initcall2s.init) \*(.initcall3.init) \*(.initcall3s.init) \*(.initcall4.init) \*(.initcall4s.init) \*(.initcall5.init) \*(.initcall5s.init) \*(.initcallrootfs.init) \*(.initcall6.init) \*(.initcall6s.init) \*(.initcall7.init) \*(.initcall7s.init)  
\_\_initcall\_end = .;

}

模块中的module\_init中的初始化函数何时被调用了：

在系统启动过程中start\_kernel()->rest\_init()->kernel\_init()->do\_basic\_setup()->do\_initcalls()。

static void \_\_init do\_initcalls(void)  
{  
initcall\_t \*call;  
for (call = \_\_initcall\_start; call < \_\_initcall\_end; call++)  
do\_one\_initcall(\*call);  
/\* Make sure there is no pending stuff from the initcall sequence \*/  
flush\_scheduled\_work();  
}

关于模块初始化的优先级

在init.h 中有如下定义

#define pure\_initcall(fn)  \_\_define\_initcall("0",fn,1)

#define core\_initcall(fn)  \_\_define\_initcall("1",fn,1)  
#define core\_initcall\_sync(fn)  \_\_define\_initcall("1s",fn,1s)  
#define postcore\_initcall(fn)  \_\_define\_initcall("2",fn,2)  
#define postcore\_initcall\_sync(fn) \_\_define\_initcall("2s",fn,2s)  
#define arch\_initcall(fn)  \_\_define\_initcall("3",fn,3)  
#define arch\_initcall\_sync(fn)  \_\_define\_initcall("3s",fn,3s)  
#define subsys\_initcall(fn)  \_\_define\_initcall("4",fn,4)  
#define subsys\_initcall\_sync(fn) \_\_define\_initcall("4s",fn,4s)  
#define fs\_initcall(fn)   \_\_define\_initcall("5",fn,5)  
#define fs\_initcall\_sync(fn)  \_\_define\_initcall("5s",fn,5s)  
#define rootfs\_initcall(fn)  \_\_define\_initcall("rootfs",fn,rootfs)  
#define **device\_initcall**(fn)  \_\_define\_initcall("6",fn,6)  
#define device\_initcall\_sync(fn) \_\_define\_initcall("6s",fn,6s)  
#define **late\_initcall**(fn)  \_\_define\_initcall("7",fn,7)  
#define late\_initcall\_sync(fn)  \_\_define\_initcall("7s",fn,7s)

#define module\_init(x) \_\_initcall(x);  
#define \_\_initcall(fn) **device\_initcall**(fn)

所有的\_\_init函数在区段.init.text区段中，同时还在.initcall.init中还保存了一份函数指针，  
在初始化时内核会通过这些函数指针调用这些\_\_init函数指针，  
并在**整个初始化完成后，释放整个init区段**（包括.init.text，.initcall.init等）。  
   
这些函数在内核初始化过程中的调用顺序只和这里的函数指针的顺序有关

在2.6内核中，initcall.init区段又分成7个子区段

不同的区段，调用的顺序不一样，数字越小的优先级越高。

也就是说 late\_initcall 还要在 module\_init的后面。

驱动中用：   
module\_init(**i2c\_dev\_init**);   
module\_init(**as352x\_afe\_init**);   
module\_init(**as352x\_afe\_i2c\_init**);

大家都是同一个优先级去初始化，   
最后这些驱动加载的顺序，可以查看在根目录下，   
生成的system.map：

。。。   
c00197d8 t \_\_initcall\_alignment\_init5   
。。。。。   
c00197f4 t \_\_initcall\_default\_rootfsrootfs   
c00197f8 t \_\_initcall\_timer\_init\_sysfs6   
c00197fc t \_\_initcall\_clock\_dev\_init6   
。。。   
c00198d8 t \_\_initcall\_loop\_init6   
c00198dc t \_\_initcall\_net\_olddevs\_init6   
c00198e0 t \_\_initcall\_loopback\_init6   
c00198e4 t \_\_initcall\_enc28j60\_init6   
。。。   
c0019900 t \_\_initcall\_as352x\_spi\_init6   
c0019904 t \_\_initcall\_spidev\_init6   
。。。   
c0019920 **t \_\_initcall\_i2c\_dev\_init6**   
c0019924 **t \_\_initcall\_as352x\_afe\_i2c\_init6**   
c0019928 **t \_\_initcall\_as352x\_afe\_init6**