1. the module is built-in driver, **MODULE** macro is not defined, the module will be embedded in kernel

```
/**
 * module_init() - driver initialization entry point
 * @x: function to be run at kernel boot time or module insertion
 *
 * module_init() will either be called during do_initcalls() (if
 * builtin) or at module insertion time (if a module).  There can only
 * be one per module.
 */
#define module_init(x)     __initcall(x);

/**
 * module_exit() - driver exit entry point
 * @x: function to be run when driver is removed
 *
 * module_exit() will wrap the driver clean-up code
 * with cleanup_module() when used with rmmod when
 * the driver is a module.  If the driver is statically
 * compiled into the kernel, module_exit() has no effect.
 * There can only be one per module.
 */
#define module_exit(x)    __exitcall(x);
```

```
#define __initcall(fn) device_initcall(fn)

#define device_initcall(fn)             __define_initcall(fn, 6)
```

也就是built-in module的fn在initcall的**level 6**被调用。

```
#define __exitcall(fn) \
        static exitcall_t __exitcall_##fn __exit_call = fn

#define __exit_call   __used __section(.exitcall.exit)
```

即built-in module的exit function被安排在section ".exitcall.exit".

in vmlinux.lds

```
SECTIONS
{
 /*
        * XXX: The linker does not define how output sections are
        * assigned to input sections when there are multiple statements
        * matching the same input section name.  There is no documented
        * order of matching.
        *
        * unwind exit sections must be discarded before the rest of the
```

```
        * unwind sections get included.
        */
 /DISCARD/ : {
  *(.ARM.exidx.exit.text)
  *(.ARM.extab.exit.text)
  *(.exitcall.exit)
  *(.discard)
  *(.discard.*)
 }
```

也就是说实际上exit function pointer会被discard.因为由于是built-in module，所以不会有调用exit function的机会。

这里丢弃的是exit function pointer,而不是exit function本身！

2. the module is dynamic loadable driver, MODULE macro is defined.

```
/* Each module must use one module_init(). */
#define module_init(initfn)                                     \
        static inline initcall_t __inittest(void)               \
        { return initfn; }                                      \
        int init_module(void) __attribute__((alias(#initfn)));
```

```
/* This is only required if you want to be unloadable. */
#define module_exit(exitfn)                    \
        static inline exitcall_t __exittest(void)        \
        { return exitfn; }                    \
        void cleanup_module(void) __attribute__((alias(#exitfn)));
```

__attribute__((alias(#initfn)))是gcc的属性，表示这里

module_init(initfn)中的initfn还有一个alias(别名)为init_module.

比如

```
static int __init edt_ft5x06_ts_driver_init(void)
{
        return      i2c_add_driver(&(edt_ft5x06_ts_driver))
}
module_init(edt_ft5x06_ts_driver_init);
```

即edt_ft5x06_ts_driver_init()还有一个名字叫init_module。这样init_module()就等同与edt_ft5x06_ts_driver_init()。

这主要是为了载入module的insmod，因为它总认为载入的module的初始化函数名只有一个名字init_module，而unload module的函数名叫cleanup_module。