kernel nor

simple_map_init

drivers/mtd/maps、map_funcs.c

设置读写函数

do_map_probe

drivers/mtd/chips/chiperg.c

get_mtd_chip_driver — 获取协议类型，总共有3种 cfi_probe、jedec_probe、map_ram

request_module — 如果没有获取到协议，则加载协议相关的模块

调用probe函数

```
struct mtd_info *do_map_probe(const char *name, struct map_info *map)
{
    struct mtd_chip_driver *drv;
    struct mtd_info *ret;

    //通过协议类型获取驱动
    drv = get_mtd_chip_driver(name);

    //获取失败则先加载协议类的驱动
    if (!drv && !request_module("%s", name))
        drv = get_mtd_chip_driver(name);

    if (!drv)
        return NULL;
    //调用probe函数
    ret = drv->probe(map);

    /* We decrease the use count here. It may have been a
       probe-only module, which is no longer required from this
       point, having given us a handle on (and increased the use
       count of) the actual driver code.
    */
    module_put(drv->module);

    if (ret)
        return ret;

    return NULL;
} /* end do_map_probe */
/*
```

module_put — 退出模块

调用register_mtd_chip_driver函数来添加协议

cfi_probe.c、jedec_probe.c、map_ram.c、map_rom.c

```
struct mtd_info *cfi_probe(struct map_info *map)
{
    /*
     * Just use the generic probe stuff to call our CFI-specific
     * chip_probe routine in all the possible permutations, etc.
     */
    return mtd_do_chip_probe(map, &cfi_chip_probe);
}

static struct mtd_chip_driver cfi_chipdrv = {
    .probe      = cfi_probe,
    .name       = "cfi_probe",
    .module     = THIS_MODULE
};

static int __init cfi_probe_init(void)
{
    register_mtd_chip_driver(&cfi_chipdrv);
    return 0;
}
```

cfi_probe.c

mtd_do_chip_probe

genprobe_ident_chips

产生新的cfi

调用probe_chip

```
static int genprobe_new_chip(struct map_info *map, struct chip_probe *cp,
                             struct cfi_private *cfi)
{
    int min_chips = (map_bankwidth(map)/4?:1); /* At most 4-bytes wide. */
    int max_chips = map_bankwidth(map); /* And minimum 1 */
    int nr_chips, type;

    for (nr_chips = max_chips; nr_chips >= min_chips; nr_chips >>= 1) {

        if (!cfi_interleave_supported(nr_chips))
            continue;

        cfi->interleave = nr_chips;

        /* Minimum device size. Don't look for one 8-bit device
           in a 16-bit bus, etc. */
        type = map_bankwidth(map) / nr_chips;

        for (; type <= CFI_DEVICETYPE_X32; type<<=1) {
            cfi->device_type = type;

            if (cp->probe_chip(map, 0, NULL, cfi))
                return 1;
        }
    }
    return 0;
} /* end genprobe_new_chip */
```

genprobe_new_chip

```
static struct chip_probe cfi_chip_probe = {
    .name       = "CFI",
    .probe_chip = cfi_probe_chip
};

struct mtd_info *cfi_probe(struct map_info *map)
{
    /*
     * Just use the generic probe stuff to call our CFI-specific
     * chip_probe routine in all the possible permutations, etc.
     */
    return mtd_do_chip_probe(map, &cfi_chip_probe);
}
```

cfi_probe_chip

qry_present — 判断能否读到QRY

cfi_chip_setup — 读取一些基本信息