高手进阶Linux系统下MTD_CFI驱动介绍 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/268/2021_2022__E9_AB_98_E6_89_8B_E8_BF_9B_E9_c67_268226.htm 因为前不久作了些关于FLASH编程方面的东西，加上看了Linux下MTD/CFI代码，感觉收获不小，就准备记个笔记，没想到竟然花了半天时间才写好。 某些Intel的FLASH芯片（如StrataFlash系列）支持多分区，也就是各个分区可以同时进行操作。应该说这是不错的特性，但是也会带来些问题。记得当初移植Linux-2.4.21，挂JFFS2文件系统的时候，经常会报一些"Magic bitmask not found"之类的错误，跟进去发现FLASH读出来的都是些0x80之类的数据，查看资料发现该款FLASH有分区的特性，而Linux的FLASH驱动只用一个状态变量表示整个FLASH的状态，这就会造成某个分区的实际状态和系统记录的不符,从而导致读FLASH的时候该点实际上不处在读状态。当时的解决办法是，每次读的时候，不管记录的状态是什么，先进入读状态再说，当然这会带来性能的下降，具体损失多少个时钟周期就不算了。 话说进入Linux-2.6.x的时代（具体是2.6.13）,除了Lock/Unlock（Linux在擦/写的时候不先Unlock，解决办法就是初始化的时候先全部Unlock）这个老问题外，竟然多分区的错误没有出现，惊讶之下决定好好研究下Linux的MTD/FLASH驱动。 说驱动之前，先明确几个编程要点：1：读写，要按照总线位宽读写，注意不是FLASH芯片位宽（例如背靠背）。 2：寻址，程序要访问的地址和FLASH芯片地址引脚得到的值是不一样的，例如16位的FLASH芯片，对于CPU，0x00和0x01表示2个不同的字节，但是到了FLASH引

脚得到的都是0，也就是都指向FLASH的第一个WORD。可以认为地址总线的bit0悬空，或者认为转换总线, bit0上实际输出的是bit1。这个解释了要点1。 3：芯片手册提到偏移量都是基于WORD的，而WORD的位宽取决于芯片的位宽，因此在下命令的时候，实际偏移=手册偏移*buswidth/8。 4：芯片手册提到的变量长度（典型如CFI信息）例如2，指的是，变量是个16bit数，但是读的时候，要读2个WORD，然后把每个WORD的低8位拼成1个16bit数。读WORD再拼凑确实挺麻烦，尤其是读取大结构的时候，不过参照cfi_util.c的cfi_read_pri函数的做法就简单了。 5：背靠背，也就是比方说2块16位的芯片一起接在32位的总线上。带来的就是寻址的问题，很显然，首先要按32位读写；其次就是下命令的地址，实际偏移=手册偏移*interleave*device_type/8，device_type=buswidth/interleave，而buswidth这个时候是32(总线位宽)。另外就是背靠背的时候，命令和返回的状态码是“双份的”，例如2块16位背靠背，读命令是0x00ff00ff。 如果不是想写像Linux那么灵活的代码（考虑各种接法/位宽/CFI获取信息等），那事情就简单很多，只要考虑要点1以及擦除块的大小就好了，当然如果是背靠背接法，擦除块的实际大小要乘个interleave。 下面就进入Linux代码，不过关于CHIP/MAP/MTD之间绕来绕去的关系现在还糊涂着呢，因此下面只是简单的跟一下脉络和各个编程要点。 1：构造map_info结构，指定基址/位宽/大小等信息以及"cfi_probe"限定，然后调用do_map_probe()。 2：do_map_probe()根据名字"cfi_probe"找到芯片驱动"cfi_probe.c"直接调用cfi_probe()。 3：cfi_probe()直接调用mtd_do_chip_probe()，传

入cfi_probe_chip()函数指针。 4：mtd_do_chip_probe()分2步，先调用genprobe_ident_chips()探测芯片信息，后调用check_cmd_set()获取和初始化芯片命令集（多分区初始化就在里面）。 5：genprobe_ident_chips()函数如果不考虑多芯片串连的情况，那只需看前面的genprobe_new_chip()调用，完成后cfi.chipshift=cfi.cfiq->DevSize， 2 ^chipshift=FLASH大小。 6：genprobe_new_chip()枚举各种不同的芯片位宽和背靠背数量，结合配置设定依次调用步骤 3 的cfi_probe_chip()，注意cfi->device_type=bankwidth/nr_chips，bankwidth是总线位宽，device_type是芯片位宽。这里我们只需要注意有限复杂情况即可，所谓有限复杂指的是编译时确定的复杂连接。这样，cfi_probe_chip()只有第1次调用才成功，如果考虑32位宽的FLASH插在16bit总线上的情况，那第2次调用成功。 7：cfi_probe_chip()，由于步骤6的原因，函数就在cfi_chip_setup()直接返回，后面的代码就不用考虑了。 8：cfi_chip_setup()读取CFI信息，可以留意下Linux是怎么实现要点4的。 9：回到步骤4的check_cmd_set()阶段，进入cfi_cmdset_0001()函数，先调用read_pri_intelext()读取Intel的扩展信息，然后调用cfi_intelext_setup()初始化自身结构。 10：read_pri_intelext()函数，可以留意下怎么读取变长结构的技巧，也就是"need_more"的用法。这里说明下一些变量的含义，例如对于StrataFlash 128Mb Bottom类型的的FLASH芯片，块结构是4*32KB 127*128KB=16MB，一共16个分区，每个分区1MB。nb_parts=2。 第1部分 NumIdentPartitions=1 // 有1个重复的分区 NumBlockTypes=2 // 分区内有2种不同的Block类型 第1类型 NumIdentBlocks=3 // 有4个Block(3 1)

BlockSize=0x80 // 32KB(0x80*256) 第2类型 NumIdentBlocks=6 // 有7个Block(6 1) BlockSize=0x200 // 128KB(0x200*256) 第2部分 NumIdentPartitions=15// 有15个重复的分区 NumBlockTypes=1 // 分区内有1种Block类型 第1类型 NumIdentBlocks=7 // 有8个Block(7 1) BlockSize=0x200 // 128KB(0x200*256) 11
：cfi_intelext_setup()函数首先根据CFI建立mtd_erase_region_info信息，然后调用cfi_intelext_partition_fixup()来支持分区。 12
：cfi_intelext_partition_fixup()用来建立虚拟Chip，每个分区对应1个Chip，不过并没有完全根据CFI扩展信息来建立，而是假定每个分区的大小都一致。cfi->chipshift调整为partshift，各个虚拟chip->start调整为各分区的基址。将来访问FLASH的入口函数cfi_varsize_frob()就根据ofs得到chipnum(chipnum=ofs>>cfi->chipshift)，这也是为什么要假定分区一致的原因。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com