[Linux下TI omap芯片 MUX 配置分析（以AM335X芯片为例）](http://blog.chinaunix.net/uid-20543672-id-3067021.html) 2012-02-07 15:15:23

分类： LINUX

    在移植内核的时候，通常会遇到引脚复用（MUX）的配置问题。在现在的Linux内核中，对于TI的ARM芯片，早已经有了比较通用的MUX配置框架。这对于许多TI的芯片都是通用的，这次看AM335X的代码顺手写一下分析，以备后用。

一、硬件

    对于许多TI的芯片来说，引脚复用的配置是在**Control Module（配置模块）的寄存器里配置**的，（这个和三星的CPU有点不同，三星的一般在GPIO的寄存器中配置）。所以当你需要配置这些寄存器的时候，请到数据手册的**Control Module的Pad Control Registers**查找。

1. TI的CPU芯片手册有两种：
2. 一种是**datasheet（DS：数据手册）**，较小，只是大概介绍下芯片的结构；
3. 另一种是**Technical Reference Manual（TRM:技术参考手册）**，较大，详细介绍芯片的各部分功能原理和寄存器定义。
4. 在开发过程中，这两个手册都需要参考，是互补的。

对于AM335X，关于引脚复用的列表及模式号与功能的对应可以在数据手册中找到：

2 Terminal Description：

2.2 Ball Characteristics

关于引脚复用寄存器定义及各引脚相应寄存器的偏移可以在TRM中找到：

9 Control Module

9.1 Control Module

9.1.3 Functional Description

9.1.3.2 Pad Control Registers （包含引脚复用寄存器定义）

9.1.5 Registers

9.1.5.1 CONTROL\_MODULE Registers （包含引脚相应寄存器的偏移）

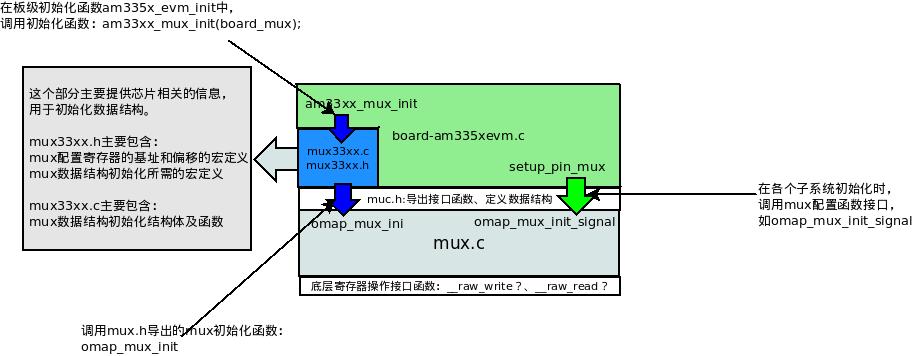
**二、软件**

    由于TI的芯片构架类似，对于Linux内核来说，早就已经为这个做好了一个软件上的框架，无论是在启动的初始化阶段还是在系统运行时，都可以通过这个框架提供的接口函数配置芯片的MUX。下面就来简要的分析一下。

以AM335X为例，相关代码位置：arch/arm/mach-omap2

1. **mux.h**
2. **mux.c**
3. **mux33xx.h**
4. **mux33xx.c**
5. **board-am335xevm.c**
6. (还有一些用到了：arch/arm/plat-omap/include/plat/omap\_hwmod.h）

其中他们的层次关系是：

[](http://blog.chinaunix.net/attachment/201202/7/20543672_1328599181Utcf.jpeg)

**（1）重要的数据结构**

1. /\*\*
2. \* struct mux\_partition - 包含分区相关信息
3. \* @name: 当前分区名
4. \* @flags: 本分区的特定标志
5. \* @phys: 物理地址
6. \* @size: 分区大小
7. \* @base: ioremap 映射过的虚拟地址
8. \* @muxmodes: 本分区mux节点链表头
9. \* @node: 分区链表头
10. \*/
11. struct omap\_mux\_partition {
12. const char        \*name;
13. u32            flags;
14. u32            phys;
15. u32            size;
16. void \_\_iomem        \*base;
17. struct list\_head    muxmodes;
18. struct list\_head    node;
19. };

    这个数据结构中包含了芯片中几乎所有定义好的mux的数据，它在mux数据初始化函数omap\_mux\_init中初始化，并添加到全局mux\_partitions链表中（通过node成员）。而其中的muxmodes是所有mux信息节点的链表头，用来链接以下数据结构：

1. /\*\*
2. \* struct omap\_mux\_entry - mux信息节点
3. \* @mux: omap\_mux结构体
4. \* @node: 链表节点
5. \*/
6. struct omap\_mux\_entry {
7. struct omap\_mux        mux;
8. struct list\_head    node;
9. };

而在以上数据结构中，struct omap\_mux是记录单个mux节点数据的结构体：

1. /\*\*
2. \* struct omap\_mux - omap mux 寄存器偏移和值的数据
3. \* @reg\_offset:    从Control Module寄存器基地址算起的mux寄存器偏移
4. \* @gpio:    GPIO 编号
5. \* @muxnames:    引脚可用的信号模式字符串指针数组
6. \* @balls:    封装中可用的引脚
7. \*/
8. struct omap\_mux {
9. u16    reg\_offset;
10. u16    gpio;
11. #ifdef CONFIG\_OMAP\_MUX
12. char    \*muxnames[OMAP\_MUX\_NR\_MODES];
13. #ifdef CONFIG\_DEBUG\_FS
14. char    \*balls[OMAP\_MUX\_NR\_SIDES];
15. #endif
16. #endif
17. };

     而struct mux\_partition中muxmodes链表及其节点数据的初始化都是在omap\_mux\_init初始化函数中（omap\_mux\_init\_list(partition, superset);），而struct omap\_mux节点数据中信息是由mux33xx.h和mux33xx.c中提供的。你可以在mux33xx.c中看到一个巨大的struct omap\_mux结构体数组初始化代码，这个代码一看就明了。不同的芯片只需要根据芯片资料修改这个结构体就好了，但是am33xx的这个结构体（当前）还不完善，gpio的数据还都是0。值得一提的是其中用到了一个宏：

1. #define \_AM33XX\_MUXENTRY(M0, g, m0, m1, m2, m3, m4, m5, m6, m7)        \
2. {                                    \
3. .reg\_offset    = (AM33XX\_CONTROL\_PADCONF\_##M0##\_OFFSET),    \
4. .gpio        = (g),                        \
5. .muxnames    = { m0, m1, m2, m3, m4, m5, m6, m7 },        \
6. }

这个宏使得这个结构体数组的初始化变得清晰明了。

以上的数据结构是在系统初始化的时候使用的，在struct omap\_mux\_partition完成初始化后，omap\_mux\_init初始化函数最后会根据不同的板子初始化部分mux寄存器（omap\_mux\_init\_signals(partition, board\_mux);），其中牵涉到了以下结构体：

1. /\*\*
2. \* struct omap\_board\_mux - 初始化mux寄存器的数据
3. \* @reg\_offset:    从Control Module寄存器基地址算起的mux寄存器偏移
4. \* @mux\_value:    希望设置的mux寄存器值
5. \*/
6. struct omap\_board\_mux {
7. u16    reg\_offset;
8. u16    value;
9. };

    在最上层的板级初始化文件（board-am335xevm.c）中会定义一个这样的结构体数组，确定所要初始化的引脚复用寄存器，交由omap\_mux\_init\_signals(partition, board\_mux);使用。例如：

1. #ifdef CONFIG\_OMAP\_MUX
2. static struct omap\_board\_mux board\_mux[] \_\_initdata = {
3. AM33XX\_MUX(I2C0\_SDA, OMAP\_MUX\_MODE0 | AM33XX\_SLEWCTRL\_SLOW |
4. AM33XX\_INPUT\_EN | AM33XX\_PIN\_OUTPUT),
5. AM33XX\_MUX(I2C0\_SCL, OMAP\_MUX\_MODE0 | AM33XX\_SLEWCTRL\_SLOW |
6. AM33XX\_INPUT\_EN | AM33XX\_PIN\_OUTPUT),
7. { .reg\_offset = OMAP\_MUX\_TERMINATOR },
8. };
9. #else
10. #define    board\_mux    NULL
11. #endif

其中用到了一个宏：

1. /\* 如果引脚没有定义为输入，拉动电阻将会被禁用
2. \* 如果定义为输入，所提供的标志位将确定拉动电阻的配置
3. \*/
4. #define AM33XX\_MUX(mode0, mux\_value)                    \
5. {                                    \
6. .reg\_offset    = (AM33XX\_CONTROL\_PADCONF\_##mode0##\_OFFSET),    \
7. .value        = (((mux\_value) & AM33XX\_INPUT\_EN) ? (mux\_value)\
8. : ((mux\_value) | AM33XX\_PULL\_DISA)),    \
9. }

**注意\_AM33XX\_MUXENTRY和AM33XX\_MUX这两个宏，前者是用于struct omap\_mux的；后者是用于struct omap\_board\_mux的。**

**（2）重要的接口函数**

1. /\*\*
2. \* omap\_mux\_init - MUX初始化的私有函数，请勿使用
3. \* 由各板级特定的MUX初始化函数调用
4. \*/
5. int omap\_mux\_init(const char \*name, u32 flags,
6. u32 mux\_pbase, u32 mux\_size,
7. struct omap\_mux \*superset,
8. struct omap\_mux \*package\_subset,
9. struct omap\_board\_mux \*board\_mux,
10. struct omap\_ball \*package\_balls);

这个函数是内部用于初始化struct mux\_partition的最总要的函数，但是这个函数并不作为接口函数使用，而是供各芯片初始化函数“\*\_mux\_init”所使用的。比如AM33XX芯片：

1. /\*\*
2. \* am33xx\_mux\_init() - 用板级特定的设置来初始化MUX系统
3. \* @board\_mux:        板级特定的MUX配置表
4. \*/
5. int \_\_init am33xx\_mux\_init(struct omap\_board\_mux \*board\_subset)
6. {
7. return omap\_mux\_init("core", 0, AM33XX\_CONTROL\_PADCONF\_MUX\_PBASE,
8. AM33XX\_CONTROL\_PADCONF\_MUX\_SIZE, am33xx\_muxmodes,
9. NULL, board\_subset, NULL);
10. }

    有了已经初始化好的struct mux\_partition结构体，我们可以利用mux.h提供的许多函数方便的初始化各mux寄存器：

1. /\*\*
2. \* omap\_mux\_init\_signal - 根据信号名字符串初始化一个引脚的mux
3. \* @muxname:        mode0\_name.signal\_name的格式的Mux名称
4. \* @val:        mux寄存器值
5. \*/
6. int omap\_mux\_init\_signal(const char \*muxname, int val);
7. /\*\*
8. \* omap\_mux\_get() - 通过名字返回一个mux分区
9. \* @name:        mux分区名
10. \*
11. \*/
12. struct omap\_mux\_partition \*omap\_mux\_get(const char \*name);
13. /\*\*
14. \* omap\_mux\_read() - 读取mux寄存器（通过分区结构体指针和寄存器偏移值）
15. \* @partition:        Mux分区
16. \* @mux\_offset:        mux寄存器偏移
17. \*
18. \*/
19. u16 omap\_mux\_read(struct omap\_mux\_partition \*p, u16 mux\_offset);
20. /\*\*
21. \* omap\_mux\_write() - 写mux寄存器（通过分区结构体指针和寄存器偏移值）
22. \* @partition:        Mux分区
23. \* @val:        新的mux寄存器值
24. \* @mux\_offset:        mux寄存器偏移
25. \*
26. \* 这个函数仅有在非GPIO信号的动态复用需要
27. \*/
28. void omap\_mux\_write(struct omap\_mux\_partition \*p, u16 val, u16 mux\_offset);
29. /\*\*
30. \* omap\_mux\_write\_array() - 写mux寄存器阵列
31. \* @partition:        Mux分区
32. \* @board\_mux:        mux寄存器阵列 （用MAP\_MUX\_TERMINATOR结尾）
33. \*
34. \* 这个函数仅有在非GPIO信号的动态复用需要
35. \*/
36. void omap\_mux\_write\_array(struct omap\_mux\_partition \*p,
37. struct omap\_board\_mux \*board\_mux);

在代码比较完备的芯片中，struct omap\_mux中的gpio成员有被初始化过，这样就可以使用以下接口函数：

1. /\*\*
2. \* omap\_mux\_init\_gpio - 根据GPIO编号初始化一个信号引脚
3. \* @gpio:        GPIO编号
4. \* @val:        mux寄存器值
5. \*/
6. int omap\_mux\_init\_gpio(int gpio, int val);
7. /\*\*
8. \* omap\_mux\_get\_gpio() - 根据GPIO编号获取一个mux寄存器值
9. \* @gpio:        GPIO编号
10. \*
11. \*/
12. u16 omap\_mux\_get\_gpio(int gpio);
13. /\*\*
14. \* omap\_mux\_set\_gpio() - 根据GPIO编号设定一个mux寄存器值
15. \* @val:        新的mux寄存器值
16. \* @gpio:        GPIO编号
17. \*
18. \*/
19. void omap\_mux\_set\_gpio(u16 val, int gpio);

     但是am33xx的gpio成员（当前）还都是0，所有这些函数没法使用。

     此外，在mux.h中还导出了其他的软件接口和数据结构，这些在am33xx中没有使用，有需要的时候再看。

     在板级初始化代码（比如board-am335xevm.c）运行完芯片特定的MUX初始化函数（am33xx\_mux\_init(board\_mux);）之后，也可以在各子系统初始化时通过上面的接口函数修改配置MUX，比如在am33xx中使用了自己封装的一个函数和结构体：

1. /\* 模块引脚复用结构体 \*/
2. struct pinmux\_config {
3. const char \*string\_name; /\* 信号名格式化字符串，“模式0字符串.目标模式字符串“ \*/
4. int val; /\* 其他mux寄存器可选配置值 \*/
5. };
6. /\*
7. \* @pin\_mux - 单个模块引脚复用结构体
8. \*            其中定义了本模块所有引脚复用细节.
9. \*/
10. static void setup\_pin\_mux(struct pinmux\_config \*pin\_mux)
11. {
12. int i;
13. for (i = 0; pin\_mux->string\_name != NULL; pin\_mux++)
14. omap\_mux\_init\_signal(pin\_mux->string\_name, pin\_mux->val);
15. }

你可以在board-am335xevm.c中看到如下的代码：

1. static struct pinmux\_config **d\_can\_ia\_pin\_mux**[] = {
2. {"uart0\_rxd.d\_can0\_tx", OMAP\_MUX\_MODE2 | AM33XX\_PULL\_ENBL},
3. {"uart0\_txd.d\_can0\_rx", OMAP\_MUX\_MODE2 | AM33XX\_PIN\_INPUT\_PULLUP},
4. {NULL, 0},
5. };
6. ......
7. static void d\_can\_init(int evm\_id, int profile)
8. {
9. switch (evm\_id) {
10. case IND\_AUT\_MTR\_EVM:
11. if ((profile == PROFILE\_0) || (profile == PROFILE\_1)) {
12. **setup\_pin\_mux(d\_can\_ia\_pin\_mux);**
13. /\* Instance Zero \*/
14. am33xx\_d\_can\_init(0);
15. }
16. break;
17. case GEN\_PURP\_EVM:
18. if (profile == PROFILE\_1) {
19. setup\_pin\_mux(d\_can\_gp\_pin\_mux);
20. /\* Instance One \*/
21. am33xx\_d\_can\_init(1);
22. }
23. break;
24. default:
25. break;
26. }
27. }

**三、使用注意**

**上面初始化过的结构体和接口函数的定义都是带有"\_\_init"和“\_\_initdata”的，所以这些都只能在内核初始化代码中使用，一旦系统初始化结束并进入了文件系统，这些定义都会被free。所有它们不能在内核模块（.ok）中被调用**，否则你就等着Oops吧。因为一个芯片的引脚复用一般是硬件设计的时候定死的，一般不可能在启动后更改。如果你是在要在模块中改变引脚复用配置，你只能通过自己ioremap相关寄存器再修改它们来实现。