24-平台设备总线

1、总线概念
2、设备端框架
3、驱动端框架

一、总线概念

裸机：IIC、SPI、485、USB、CAN，真实存在，相应的设备挂接在总线上，通信有指定的时序，由总线进行匹配完成通信。

IIC：时序 + 器件地址

linux：虚拟出一个总线，将驱动分为设备端和驱动端，加载到内核，匹配由总线完成。--平台设备总线 --- platform总线

驱动：

设备端：硬件的资源(IO、中断等)

驱动端：驱动的框架(设备注册、函数接口、其他的内核操作等)

优点：

1、可移植性强

2、资源统一交由内核管理、安全

3、驱动统一格式编写

二、设备端框架

注册：platform_device_register(struct platform_device * pdev);

卸载：platform_device_unregister(struct platform_device * pdev);

struct platform_device mypdev;//linux-3.5/include/linux/platform_device.h

```
struct platform_device{
 const char  * name;//用于匹配驱动端的名字
 int  id;//一般为-1 ，表示总线上就一对设备
 struct device  dev;// 填充里面的 .release  资源释放函数
 u32     num_resources;//资源数目
 struct resource * resource;//资源
  const struct platform_device_id *id_entry;
```

```
/* MFD cell pointer */
 struct mfd_cell *mfd_cell;
/* arch specific additions */
  struct pdev_archdata    archdata;
};
struct resource * resource;//描述设备资源   linux-
3.5/include/linux/ioport.h
struct resource {
    resource_size_t start;//资源的起始地址
    resource_size_t end;//资源的结束地址
    const char *name;//名字
    unsigned long flags;  //资源的分类
    struct resource *parent, *sibling, *child;
};
#define IORESOURCE_TYPE_BITS    0x00001f00  /* Resource type
*/
#define IORESOURCE_IO       0x00000100
#define IORESOURCE_MEM      0x00000200
#define IORESOURCE_IRQ      0x00000400
#define IORESOURCE_DMA      0x00000800
#define IORESOURCE_BUS      0x00001000
#define IORESOURCE_PREFETCH 0x00002000  /* No side effects
*/
#define IORESOURCE_READONLY 0x00004000
#define IORESOURCE_CACHEABLE    0x00008000
#define IORESOURCE_RANGELENGTH  0x00010000
#define IORESOURCE_SHADOWABLE   0x00020000
```

三、驱动端的编写框架

1、注册：platform_driver_register(struct platform_driver * drv)

2、卸载：platform_driver_unregister(struct platform_driver * drv)

```
struct platform_driver{
   int (*probe)(struct platform_device *);//驱动端的探测函数
   int (*remove)(struct platform_device *);//驱动端卸载函数(与探测
函数做相反的工作)
   void (*shutdown)(struct platform_device *);
   int (*suspend)(struct platform_device *, pm_message_t state);
   int (*resume)(struct platform_device *);
   struct device_driver driver;
```

```
    const struct platform_device_id *id_table;
};
```
探测函数：驱动端和设备端匹配成功后，探测函数就会被调用。
(获取设备端的资源，写框架性的东西)

```
struct resource *platform_get_resource(struct platform_device
*dev,unsigned int type, unsigned int num)
```

参数dev：探测函数的形参
参数type：资源类型
参数num：资源占第几个
利用平台设备总线 写led、key驱动

串口驱动：
1395
ULCONn：字长
UCONn：发送和接收的状态

```
UTRSTATn：    while(!(a & (1<<1)));
UTXHn   UTXH =ch;
URXHn   ch=URXH;
UBRDIVn  0x35
UFRACVALn   4
```

参考：

struct platform_device * pdev：linux-3.5/include/linux/platform_device.h

struct platform_driver * drv：linux-3.5/include/linux/platform_device.h

struct resource * resource：linux-3.5/include/linux/ioport.h