设备驱动架构设计与实现

ZJUNIX

目录

1. 设备驱动架构
2. 显示驱动(VGA)
3. 存储驱动(SD卡)
4. 键盘驱动(PS/2键盘)
5. 其他设备
6. 基础驱动综合应用

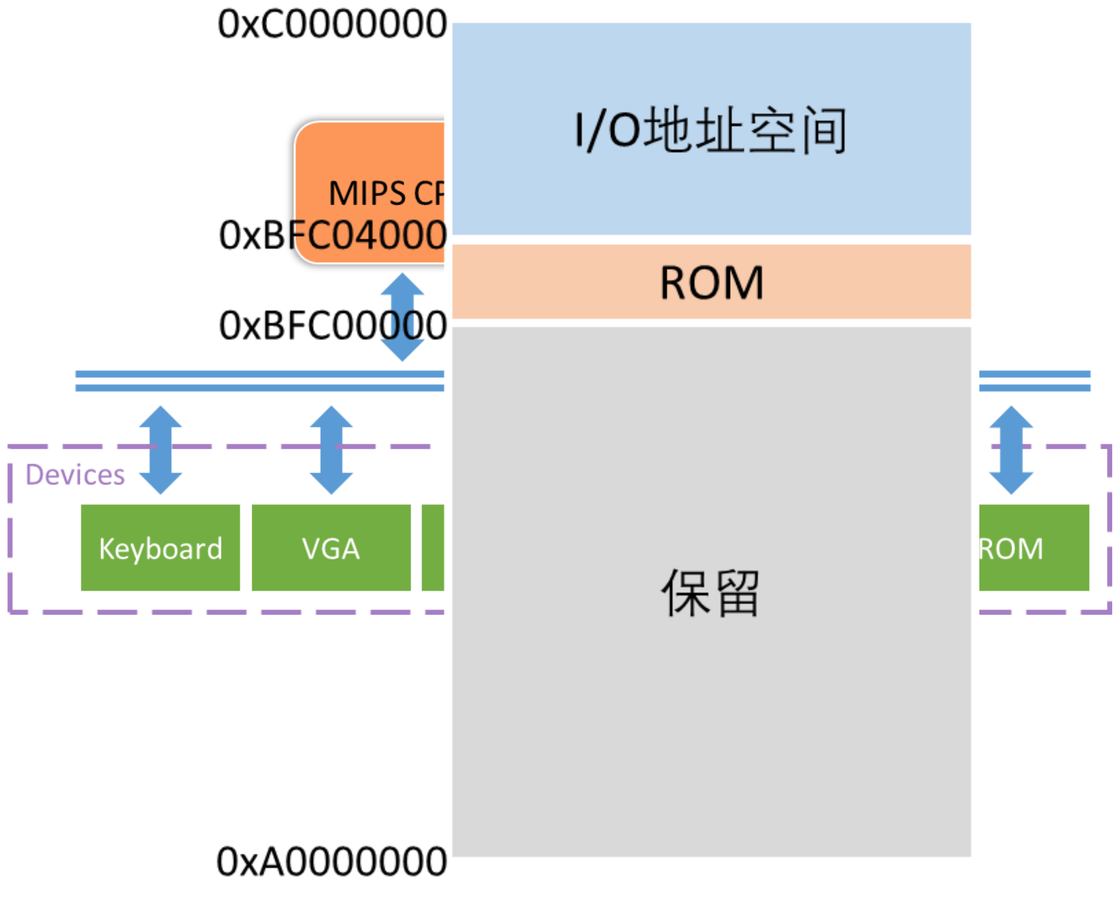
设备驱动架构

- 设备驱动层次
- 硬件接口层
- 设备驱动层
- 应用接口层

设备驱动层次

硬件接口层

- CPU与各外设的总线连接提供硬件与软件间的接口
- 每个外设均映射到地址空间里的某一段地址



设备驱动层

- 根据设备的硬件实现以及应用程序需求，实现对设备进行操作的底层函数

kernel_init_vga() kernel_putchar_at()	kernel_init_ps2() ps2_handler()	sd_send_cmd_blocking()
VGA显示设备	PS/2键盘设备	SD存储设备

应用接口层

- 按照应用程序需求，将设备驱动层的底层函数封装成易于使用的应用接口函数

kernel_putchar() kernel_puts() kernel_printf()	kernel_getkey() kernel_getchar()	sd_read_sector_blocking() sd_write_sector_blocking()
VGA驱动层	PS/2驱动层	SD驱动层

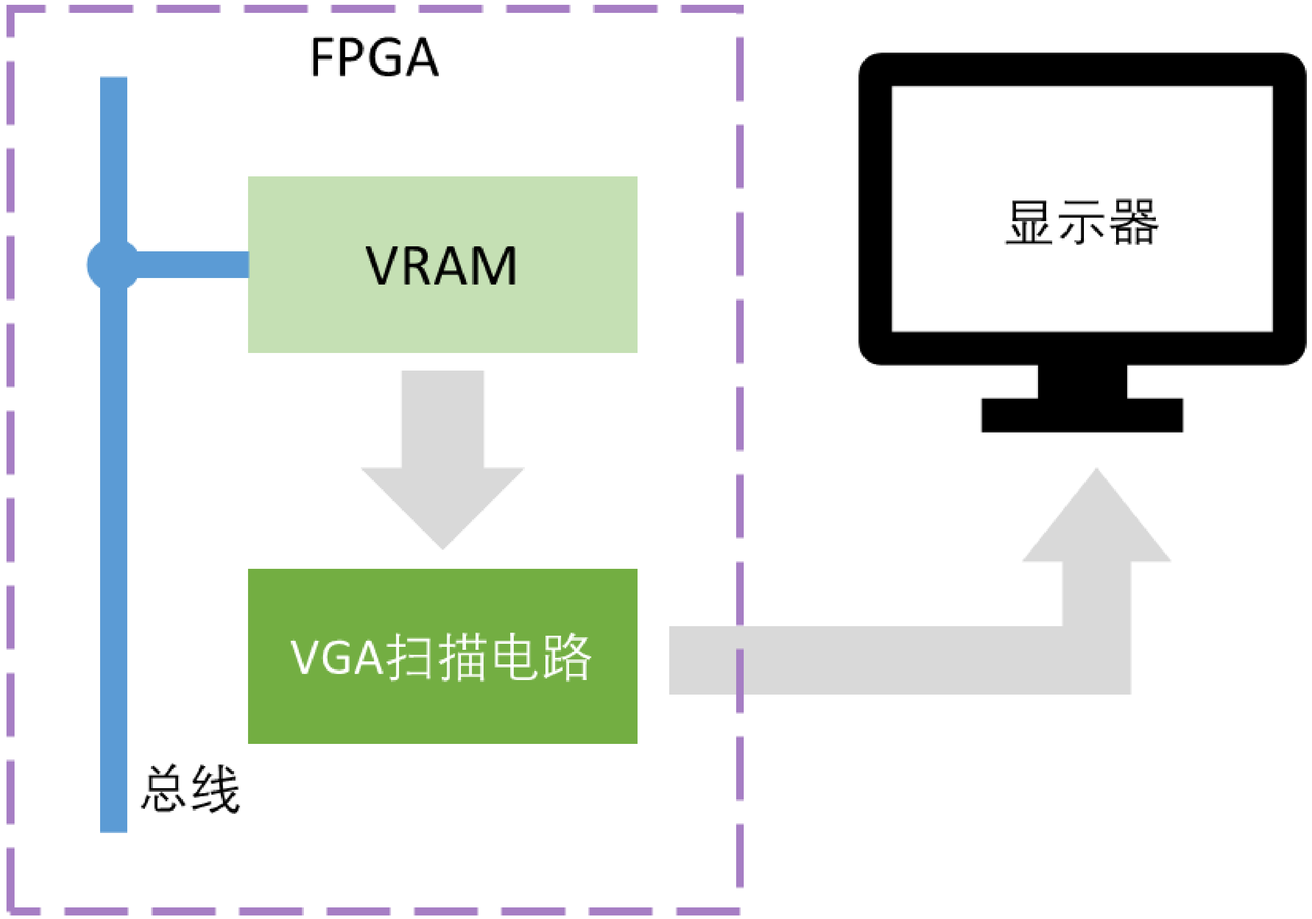
显示驱动

- 显示原理
- 驱动实现
- 应用接口

显示原理

- 显存(VRAM)是软件与显示硬件之间的接口
- VGA扫描电路从VRAM中读出数据，经过处理后产生VGA颜色和行场同步信号
- 本系统提供字符显存，没有图形显存
- VRAM地址范围: 0xbfc04000-0xbfc08000
- VRAM大小: 128字符×32字符，每字符4字节
  - 实际显示时取左上角80×30字符
  - 将显存大小对齐到2的整数次幂可简化由坐标计算显存地址的操作

显示原理



显示原理

- 显存单元：每个字符对应4字节（1个字）
- CPU是小端的
  - 所以，字符编码所在字节是这个字里地址最小的字节

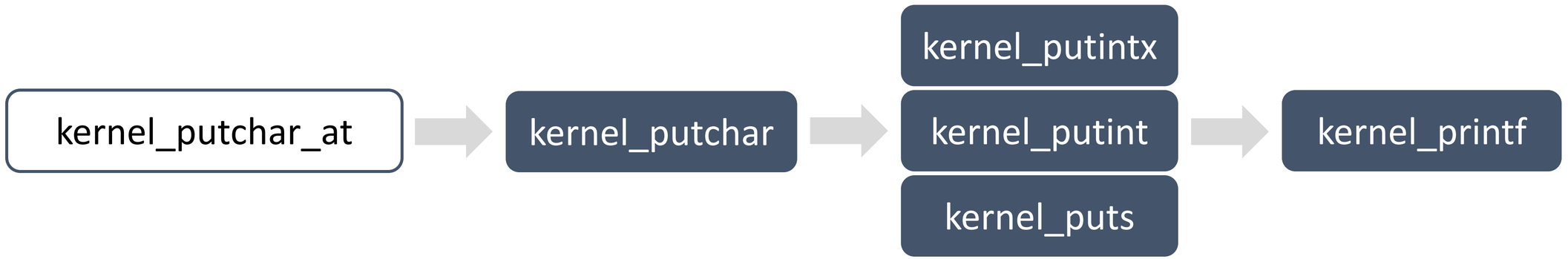
驱动实现

- 驱动程序核心函数: kernel_putchar_at
- 在指定位置写入字符，可指定字符颜色和背景色

```
unsigned int* const CHAR_VRAM = (unsigned int*)0xbfc04000;
const int VGA_CHAR_MAX_COL = 128;
...
void kernel_putchar_at(int ch, int fc, int bg, int row, int col) {
    unsigned int *p;
    row = row & 31;
    col = col & 127;
    p = CHAR_VRAM + row * VGA_CHAR_MAX_COL + col;
    *p = ((bg & 0xfff) << 20) + ((fc & 0xfff) << 8) + (ch & 0xff);
}
```

应用接口

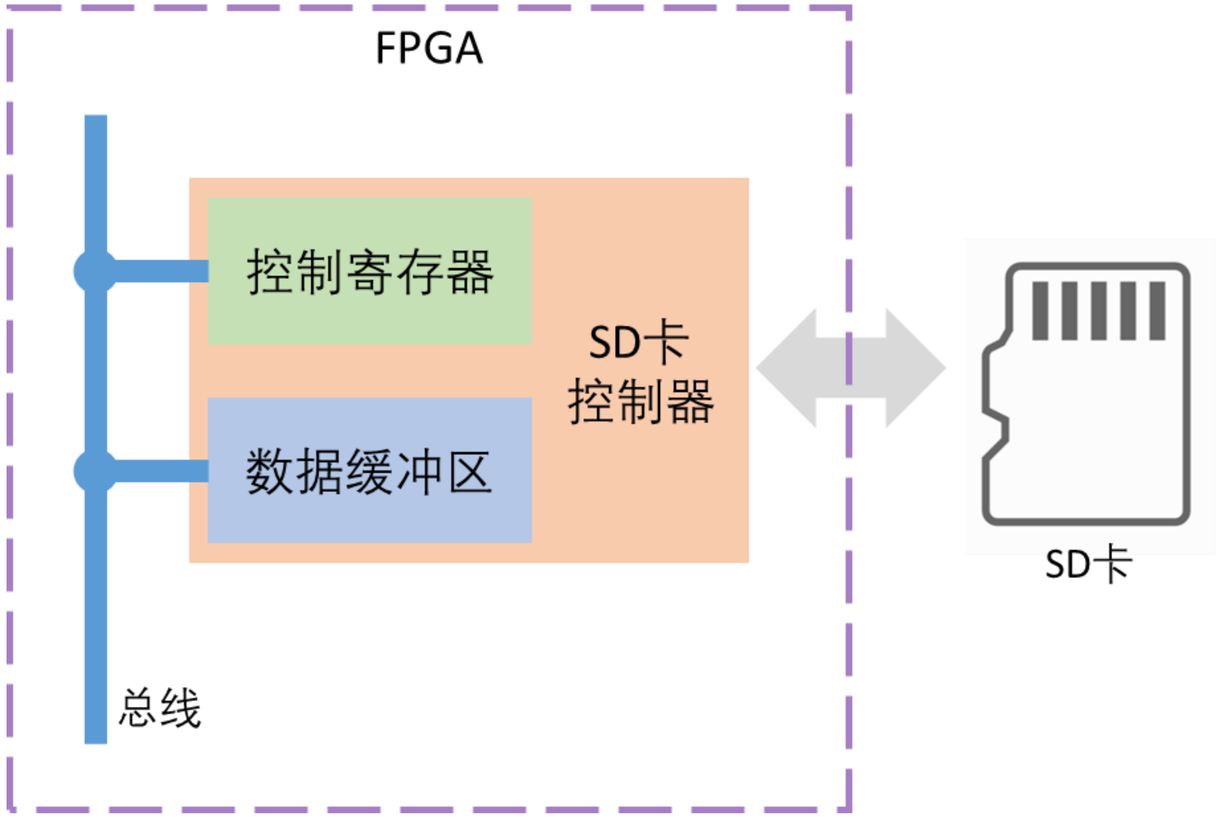
- 在上述核心驱动的基础上，可以实现一系列在屏幕上显示字符的函数



存储驱动

- 总线接口
- 驱动实现
- 应用接口

总线接口



总线接口

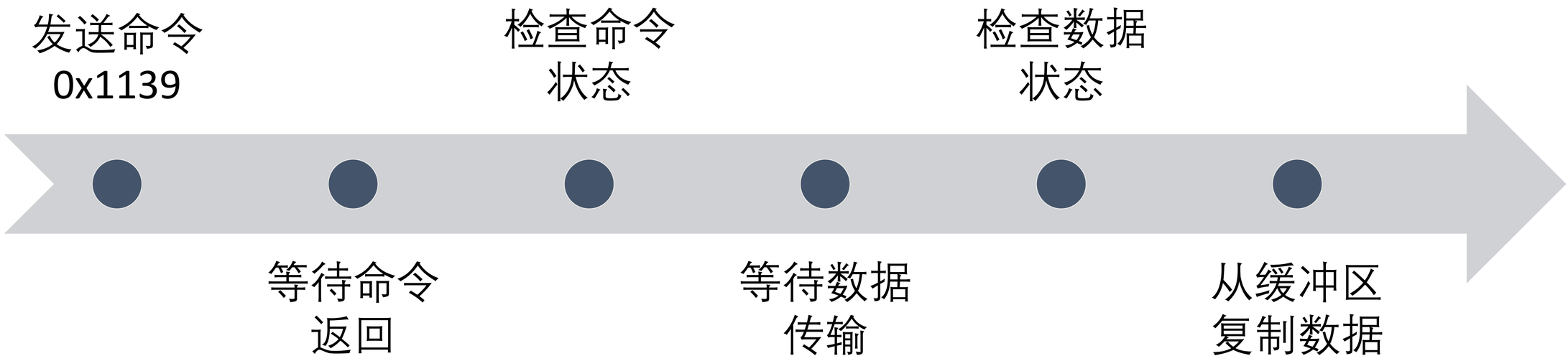
- SD卡控制器使用GitHub上的开源项目 https://github.com/mczerski/SD-card-controller
  - 详细资料可参阅其文档
- SD卡控制器共有20个寄存器
- 存储驱动需要用到控制器内5个寄存器
  - SD卡的初始化操作已经由Bootloader完成，因此其余寄存器无需使用

总线接口

- 需要操作的寄存器：
  - Command(命令): 0xBFC09104
  - Argument(命令参数): 0xBFC09100
  - Command transaction event(命令状态): 0xBFC09134
  - Data transaction event(数据状态): 0xBFC0913C
  - DMA address(DMA地址): 0xBFC09160

驱动实现

- 目前实现阻塞地对SD卡进行读写的驱动程序函数
- 读扇区流程：



驱动实现

- 目前实现阻塞地对SD卡进行读写的驱动程序函数
- 写扇区流程：

应用接口

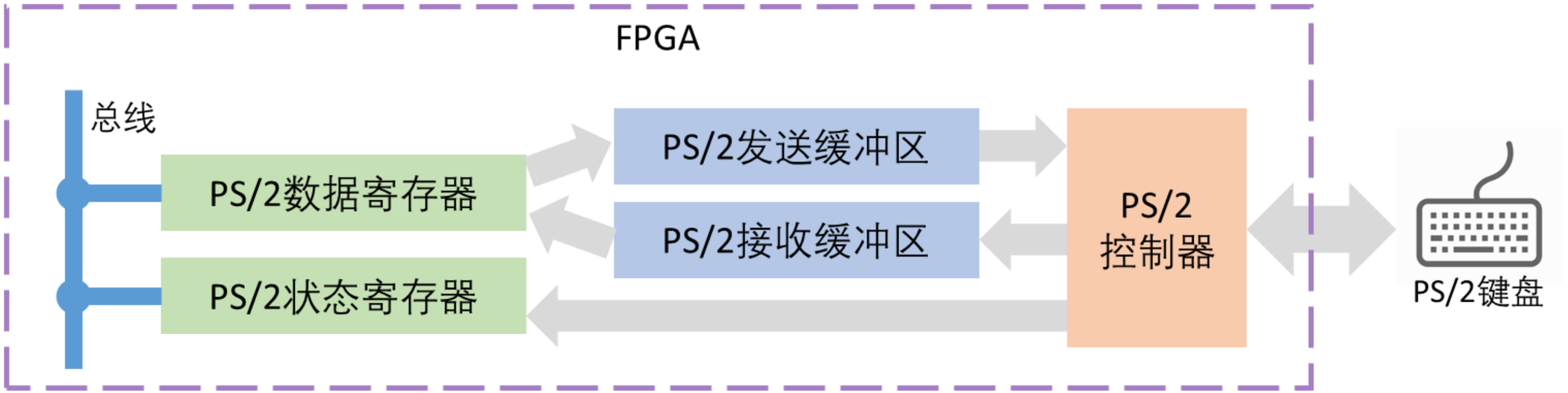
- 提供对SD卡进行阻塞读写的函数
- sd_read_sector_blocking
- sd_write_sector_blocking

键盘驱动

- 总线接口
- 驱动实现
- 应用接口

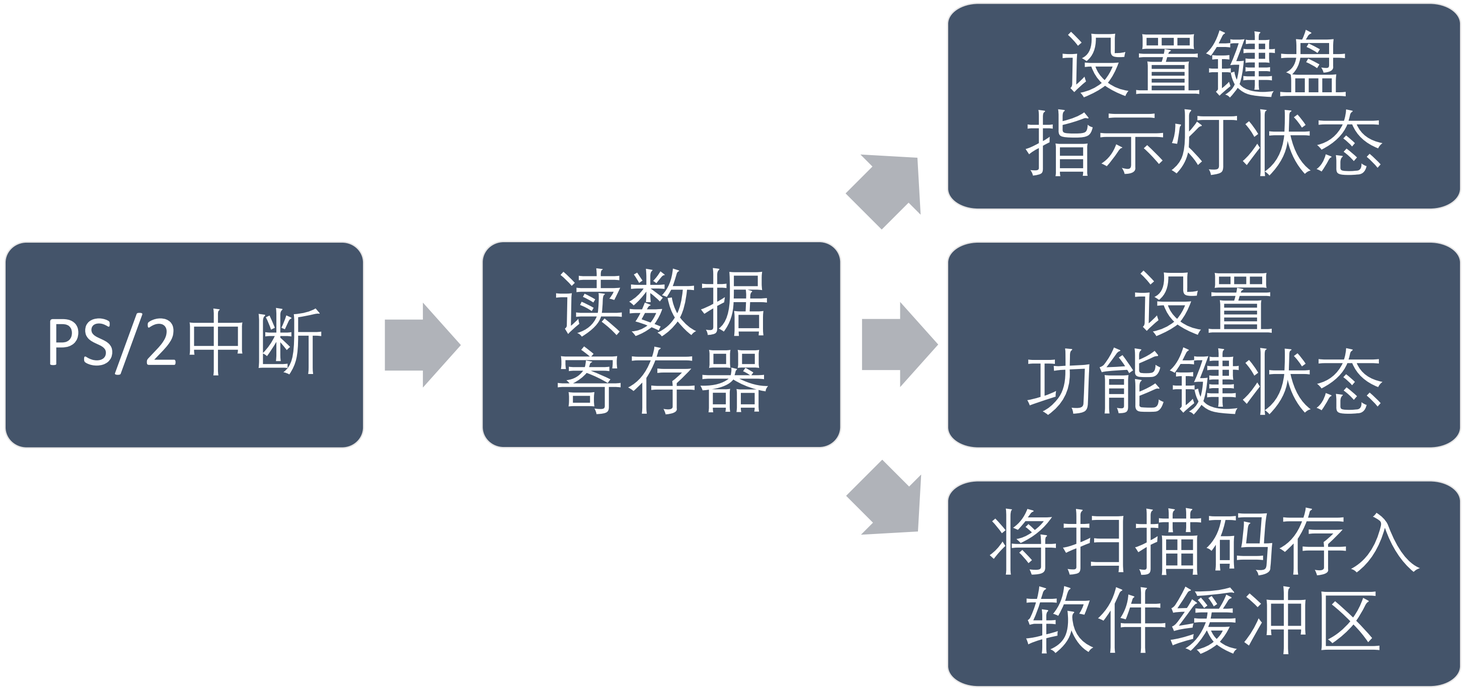
总线接口

- PS/2数据寄存器：发送/接收缓冲区与总线的接口
  - 低8位代表接收的按键扫描码或发送给键盘的指令
- PS/2状态寄存器：记录缓冲区用量、控制器状态等信息



驱动实现

- 由于PS/2设备的速度远低于CPU的速度，因此PS/2驱动依赖于中断实现



应用接口

- kernel_getkey()
  - 获取按键扫描码
  - 如果缓冲区为空，则返回0xfff
- kernel_scantoascii(int)
  - 将扫描码转换为ASCII码
  - 如果扫描码无法转换为ASCII码，则返回-1
- kernel_getchar()
  - 持续调用上述两个函数，直到获取到一个有效的扫描码

5

其他设备

- GPIO
- 高精度计时器

GPIO

- 大多数GPIO直接映射到某个内存地址
  - 例如，要将16个LED全部点亮，则向0xBFC0 900C写入0xFFFF：

```
*(unsigned int *)0xbfc0900c = 0xffff;
```

  - arch.c内列出了GPIO设备与地址的映射关系

高精度计时器

- COP0(协处理器0)内有一个64位高精度计时器
  - 该计时器非MIPS32规范所规定
  - 通过9.6和9.7号COP0寄存器读取计数值
  - 计时器的值不受系统复位以外的任何事件影响
  - 可通过该计时器获取系统启动后经历的时间
- 基于此计时器实现获取系统时间的函数
  - get_time(char *buf)
  - 将当前计时器的值转换为表示时间的字符串

6

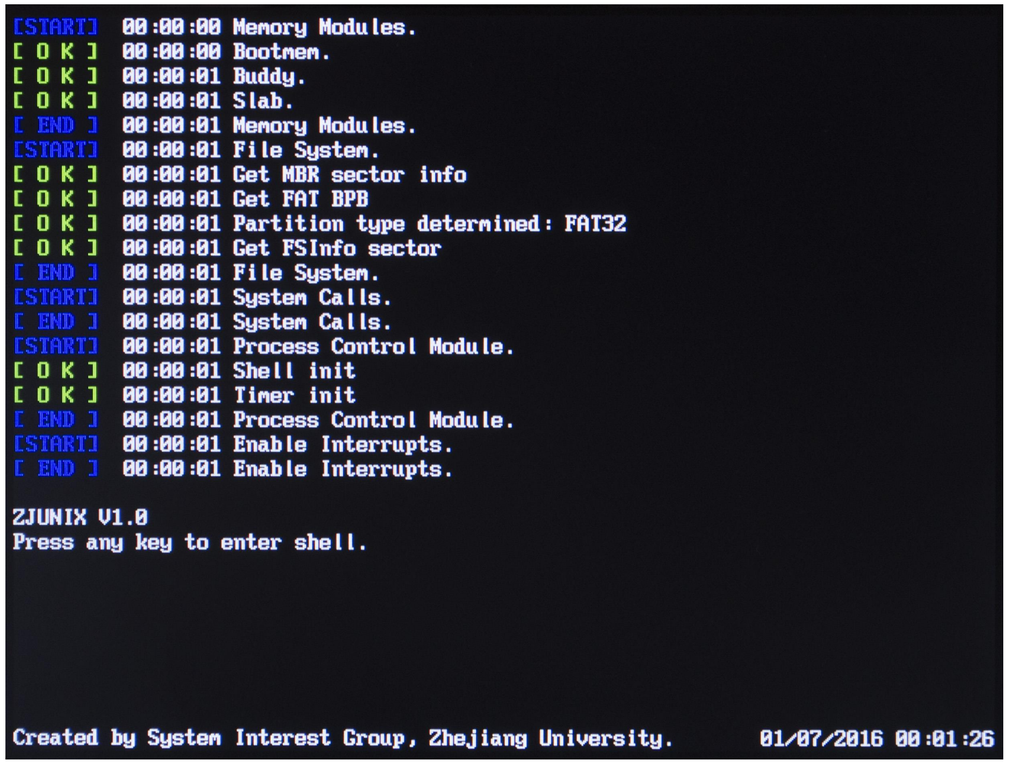
基础驱动综合应用

- 启动日志输出
- 时间显示
- 简单Shell实现

启动日志输出

- 结合内部计时器与显示驱动，在内核初始化阶段显示各模块初始化信息
- 有助于调试内核初始化过程中可能产生的各种问题
- 相关函数：

```
void log(int status,
    const char *format, ...);
```



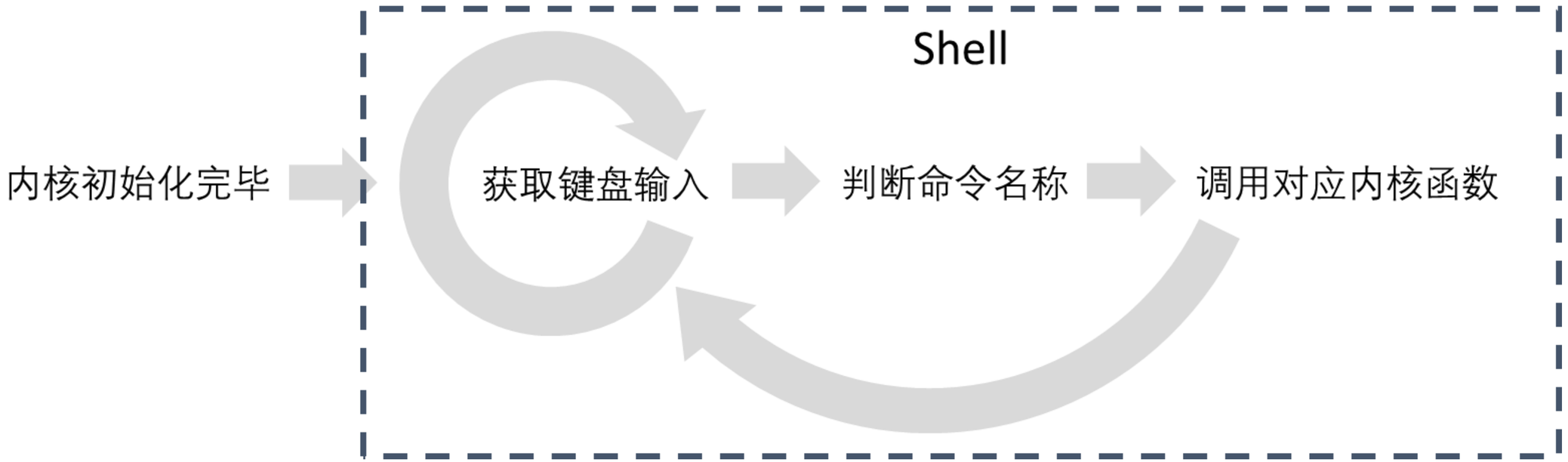
时间显示

- 通过配置定时器中断，定期更新屏幕右下角显示的时间
- 时间信息的来源为内部高精度计时器

简单Shell实现

- 我们实现一个简单的内核态Shell程序，主要用途为：
  - 基本人机交互界面
  - 操作系统内核调试工具
- 新添加的内核功能模块都将通过Shell命令进行测试
- 内核初始化完成后按任意键进入Shell
- 此Shell执行的所有命令都是内置命令
  - 通过调用内核中相应函数实现命令功能；
  - 传统的Shell则是从指定目录搜索程序并执行

简单Shell实现



THANK YOU