基本概念

什么是EBD

定义

嵌入式系统就是将计算机的硬件或软件嵌入其它机、电设备或应用系统中去，构成的一种新系统。 end是以应用为中心，以计算机技术为基础，采用可剪裁软硬件，适用于对功能、可靠性、成本、体积、功耗等有严格要求的专用计算机系统，用于实现对其他设备的控制、监视或管理功能。

区别

1. 嵌入式系统中运行的任务是专用而确定的
2. 桌面通用系统需要支持大量的需求多样的应用程序。
3. 嵌入式系统对实时性有较高要求
4. 嵌入式：强实时：us-ms，一般：ms-s，弱：s+
5. 嵌入式中一般使用实时操作系统
6. 嵌入式需要高可靠性保障
7. 嵌入式系统需要长时间无人值守条件下的运行
8. 嵌入式系统有功耗约束
9. 嵌入式资源少
10. 嵌入式开发需要专用工具和特殊方法
11. 嵌入式是综合计算机应用技术

EBD开发的基本过程

并行的特性

嵌入式系统的并行是硬件/软件协同实现的

1. 始终运行
2. 必须相应连续和混合的事件
3. 实时系统对于相应有底线要求
4. 通常必须并行处理多个独立的任务

- 设计约束
  - 成本
  - 大小和重量限制
  - 功率和能耗限制

- 环境

软硬件的划分

决定什么部分用软件实现，什么部分用硬件实现

1. 硬件和软件具有双重性
2. 软硬件变动对于系统的决策造成影响
3. 划分和选择需要考虑多种因素
4. 硬件和软件的双重性是划分决策的前提

一般软件的部分：

- 操作系统功能：任务调度，资源管理，设备驱动
- 协议栈：TCP/IP
- 应用系统框架：除基本系统，物理接口，基本逻辑电路，许多由硬件实现的功能都可以由软件实现

还有双重性的部分。

通用件

- 标准构件
  - 已经产品化
  - 形成规模生产

- 标准构件+自行设计构件=用户系统
- 构件包括软件和硬件
- 标准硬构件：
  - IC：集成电路
  - PCB：核心板，接口板
  - IP：intellectual Property
    - 标准IC：CPU，DSP，RAM，ROM，ASIC
    - 标准IP：CPU核

  - 标准模块：GPRS，显示等
  - 标准计算平台：linux板子，安卓板子

- 标准软构件：
  - OS、RTOS
  - 协议栈：TCP/IP，路由，H.323
  - 图形开发包
  - 驱动程序

MCU

MCUvsCPU

1. MCU在CPU外带有外设
2. 至少有定时器和GPIO
3. 一般还有ADC、UART和PWM等 CPU：只有指令译码执行，没有外设，如x86 MCU：带有外设，至少有定时器和GPIO，一般还有UART、AD等，有的还有存储器 SoC：带有存储器和某个特定外设（如wifi，zigbee）的MCU

选择

intel8051：不再生产，广泛应用于低端消费类电子产品中，价格曾经是优势
Atmel的AVR：8位RISC，Havard架构，片上Flash，Sram，eeprom，CS出身常用
TI的MSP430：16位伪RISC，低功耗，针对仪表市场
Microchip PIC：8位RISC，hazard架构，内存模型复杂
DSP：TI和ADI的产品，支持傅里叶积分，已经在历史尘埃中
ARM：老大哥，第一名 ，分为Cotex-M和cotex-A，有MMU和没MMU

Cortex：

- Cortex-A：Application，有MMU跑操作系统，主要用于手机平板
- Cortex-R：real time
- Cortex-M： Micro-Controller 无MMU，不跑OS或跑RTOS

如何选择stm32：
满足资源要求的最小（封装、容量）型号 每件最终产品上省下的每一分钱都是你的利润 研制阶段采用相同封装下最大容量的型号 BGA？QFN？QFP？DIP？

产品价格组成： 元件物料成本，PCB成本，加工成本，检测成本，结构成本，包装成本

资源不足时： GPIO数量不够，通过移位寄存器（串并转换）扩展低速GPIO UART（通用异步收发传输器）、SPI（串行外设接口）、I2C（总线）：通过GPIO软件模拟 AD不够：外接I2C或SPI接口的AD芯片，采用CMOs开关矩阵扩展 没有DA：用阶梯电阻通过GPIO实现 定时器不够：软件扩展

性能不足时： 速度不够：改进算法 SRAM不够：改进算法 Flash不够：避免浮点，避免复杂函数，自己写库函数 选择：

1. 列出需要的接口
2. 查看软件架构
3. 选择架构
4. 确定内存要求

5. 开始找mc
6. 看看花费和能耗约束
7. 查看部件是否可用
8. 选择development kit
9. 调查编译器和工具
10. 实验

ARM

Load/Store架构，三地址指令，每条指令都可以条件执行，能在单周期移位，协处理器指令集可以扩展ARM指令集，包括在编程模式下增加了新的寄存器和数据类型，在Thumb体系结构中使用16位高压缩表示指令集

37个寄存器，31个通用32位，6个状态，每一种处理器模式中：可见15个通用，PC以及1-2个状态寄存器，

ARM工作模式：

- 模式切换方法：软件控制，外部中断，异常处理
- 处理器模式：用户，系统，快中断（快速中断信号直接送给ARM内核，无优先级仲裁，优先级最高，还有单独的快中断R8_fiq--R12_fiq），中断（有优先级仲裁），管理，中止，未定义

ISA

条件执行

所有arm指令都可以条件执行，而thumb只有B具有条件执行功能。如果不标明条件代码，默认为无条件执行
0010CS/HS无符号数大于或等于
CC/LO无符号数小于
MI负数
PL正数或0
VS溢出
VC没有溢出
HI无符号数大于
LS无符号数小于或等于
GE有符号数大于或等于
LT有符号数小于
GT有符号数大于
LE有符号数小于或等于
1110AL无条件执行

标志位：Z运算结果为0 C进位标志加法进位，减法借位，移位为移出的最后一位，N负数为1 V 符号位溢出
移位：

`ADD R1,R1,R1,LSL #3; SUB R1,R1,R2,LSR R3;`

基址加偏址寻址： `LDR R0,[R1,#4] LDR R0,[R1,#4]!` 自动会把R1=R1+4

多寄存器寻址：

一次可传送几个寄存器值，允许一条指令传送16个寄存器的任何子集或所有寄存器，多寄存器寻址指令：LDMIA R1!,{R2-R7,R12}R1指向的存储读出到R2-R7，R1自动加4，叹号回写 STMIA R0!,{R2-R7,R12}保存到R0指向的存储，R0自动加4

SWP指令用于将一个内存单元（地址放在Rn中）的内容读到Rd中，同时将Rm的内容写入到该内存单元。

SWP{cold}{B} Rd，Rm，[Rn]

Rn不能和Rd，Rm相同

函数调用规范

没有call/ret指令，bl指令将当前pc+4存入R14，返回时R14移入pc（R15）就可以返回了 使用栈的规范：满递减类型。 在函数调用之间传递/返回参数：

1. 4个以下参数：由R0~R3传递
2. 大于4个时，用堆栈
3. 返回结果在R0中
4. R4-R10用于本地变量或临时存储

THumb2兼容16位和32位

ARM用异常表示中断和异常，中断是异常的一部分，表示CPU外部来的异常，这与一般的术语体系相反。 初始化内存：

- 启动时一个中断，而不是从0开始执行
- 启动中断是中断向量表的第一项
- 在0地址的不是中断向量表的第一项
- 初始堆栈指针由0地址的值指定

地址	内容
0xFFFFFFFF-0xE0	系统层
0xDFFFFFFF-0xA0	外部设备
0x9FFFFFFF-0x60	外部ram
0x5FFFFFFF-0x40	peripherals
0x3FFFFFFF-0x20	SRAM
0x1fffffff-0x00	code

R0-R3不需要保存，用作参数，返回结果和临时变量 R4-R11需要子程序保存和恢复

cortex-m的中断向量： 中断向量表是编译连接时刻确定下来放在flash中的 不能再程序运行中来写入中断处理函数地址。 在启动代码文件中预置中断处理函数。 中断处理函数名要写对。

ICP、ISP、IAP 这三个术语的含义 ISP：In SYSTEM Programing，在系统编程，可以动态编程
ICP：In CIRCUIT Programing，在电路编程，整机烧录
IAP：In application Programing，在应用编程，可以使用接口来下载编程数据

- 由外部引脚在上电时的电平决定上电后进入用户程序还是片内bootloader
- boatload可以使用串口、USB、I2C、SPI来和上位机通信

轮询

整个程序就是一个循环，程序按顺序检查每一个需要关心的I/O设备，完成每一件需要完成的工作。

- 延时通过循环实现，IO操作通过循环等待来确定完成
    - 串口，ADC 程序逻辑简单直观，不适用中断，系统执行的时间是可预计的，系统执行所需的内存可以预计，没有共享数据冲突的风险。 缺点：无法对外部事件做出及时响应
- 如果一个设备需要的最长响应时间小于程序循环一次，无法满足要求

前后台

中断用来处理硬件的紧急请求，并设置相应的操作标识，主循环轮询这些标志，进行后续处理 延时通过定时器中断实现，在大小循环中查看定时标志。

优点：

- 紧急任务可以放在中断优先处理
- 大量计算的任务放在主循环，对其他任务的影响被减轻了
- 任务被分割在中断处理程序和主循环中，代码更加清晰

- 设备没有中断时不用查询和等待，节省了处理时间

AT指令集是从终端设备（Terminal Equipment，TE)或数据终端设备（Data Terminal Equipment，DTE)向终端适配器(Terminal Adapter，TA)或数据电路终端设备(Data Circuit Terminal Equipment，DCE)发送的。 其对所传输的数据包大小有定义：即对于AT指令的发送，除AT两个字符外，最多可以接收1056个字符的长度（包括最后的空字符）。 每个AT命令行中只能包含一条AT指令；对于由终端设备主动向PC端报告的URC指示或者response响应，也要求一行最多有一个，不允许上报的一行中有多条指示或者响应。AT指令以回车作为结尾，响应或上报以回车换行为结尾。

中断驱动

对于低功耗有用，任务分派机制，主函数配置好就去睡眠，只用中断来做正常的程序操作

- 没有主循环，功耗低
- 编程模型类似于事件驱动的GUI
- 通过中断优先级和中断嵌套实现不同任务之间的协调
- 适用于功能任务简单，任务之间冲突机会小，空闲时间多的产品，不适合任务间可能有冲突的产品

动态执行队列

将处理的各个功能编写成函数的形式 中断产生是，将相应的处理函数的指针放入队列中 主循环不断的读取队列，取出函数指针并调用该函数

优点：

- 主函数可以按照任何优先级策略来调用队列中的函数
    - 中断函数可以以任何策略来插入函数
- 高优先级的功能能得到更多的CPU资源
- 低优先级机会少可能饿死

动态调度

让调度可以按需执行，基于重要程度的调度
非抢占式（任务结束时决定需要运行啥）vs抢占式（任何时候都可以互相抢占）

BootLoader

上电后的第一段代码：

1. 程序本身：小规模单片机程序
2. bootloader
3. BIOS：PC

Boot loader 是在系统启动时在操作系统内核运行之前运行的一段程序。 1. 初始化硬件设备和建立内存空间的映射图 2. 将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境

- 芯片内的ISP程序
  - 不占用地址空间（特殊空间）
  - 二进制协议
  - 只能下载烧录程序

- 第三方boot loader
  - 在ROM/flash地址空间中
  - 需要由其他方式预先烧录
  - 文本协议
  - 可以做简单调试

嵌入式启动的方式：

- Flash启动方式
- 硬盘启动方式
  - 硬盘主引导区放置boot loader
  - 从文件系统中引导操作系统

- 网络启动方式
  - boot loader放置在EPROM或flash中
  - 通过以太网远程下载操作系统内核和文件系统
  - 开发板不需要配置大的存储介质

简单boot loader： 只有系统引导功能 具有监控功能的boot loader： 调试支持，内存读写，flash烧写，网络下载，环境变量配置。

boot loader阶段：
阶段1：实现依赖于CPU体系结构的代码
阶段2：实现一些复杂的功能

- 阶段1
  - 硬件设备初始化
    - 屏蔽所有的中断
    - 设置CPU的速度和时钟频率
    - RAM初始化
    - 初始化LED
    - 关闭CPU内部指令/数据Cache

  - 为加载阶段2准备RAM空间
    - 除了阶段2可执行映像的大小外，还必须把堆栈空间也考虑进来
    - 必须确保所安排的地址范围的确是可读写的RAM空间

        - 内存区域有效性方法
            - 保存指定内存区域
            - 写入预订数据
            - 读入数据并比较
    - 拷贝阶段2代码到RAM中
    - 设置堆栈指针SP
    - 跳转到阶段2的C语言入口点
- 阶段2
    - 初始化本阶段要使用到的硬件设备
        - 初始化至少一个串口，一边和终端用户进行IO输出信息
        - 初始化计时器等
    - 检测系统的内存映射
        - 内存映射的描述
        - 内存映射的检测
    - 加载内核映像和根文件系统映像
    - 设置内核的启动参数
    - 调用内核

RTOS

实时系统：指系统能够在限定的响应时间内提供所需水平的服务 嵌入式实时操作系统：VxWorks **Green Hills**公司是世界排名第二的嵌入式操作系统提供商 windows实时化：realtime extension

ucos：优先级抢占式，可移植，可裁剪的多任务实时操作系统。 ThreadX：强实时，内核小，实时性强，高可靠性，源代码开放

任务优先级：静态优先级，动态优先级 因为任务时间片运行完毕而引起的任务调度可以理解为时间片调度，而因为操作系统中最高就绪优先级的变化而引起的调度则为优先级调度。

基于优先级的系统：可抢占性调度是指操作系统可以剥夺正在运行任务的处理器使用权并交给拥有更高优先级的就绪任务，让别的任务运行

基于分时机制的系统：每个任务都能持续占用处理器一段时间，时间用完操作系统就切换任务。

不可抢占型：某任务必须运行完或者主动让出。

共享资源的竞争：访问共享资源，只能有一个同时访问

运行同步：任务间相互协作，按照规定的路线执行，拓扑。

数据通信：任务间的数据传输。

通信：任务间的数据传输，直接数据传输和间接方式

等待机制：（等待IPC（通信））直接返回结果，阻塞等待模式，时限等待模式。

优先级翻转，优先级继承，优先级天花板

中断机制：

- 外部中断：一般是系统外设
- 内部中断：处理器自身的原因引发的异常事件，非法指令，总线错误或运算出错。
- 软件中断：程序通过软件指令触发的，比如为了提升权限进入软件中断系统级

提升内核实时性：可抢占内核，内核关中断时间，存储管理机制（不支持虚拟存储，不支持动态内存分配），任务互斥、同步（资源有限等待，优先级逆转问题解决）

通讯：

- 共享数据结构，最直接的任务间通信方式，全局变量、现行缓冲区，循环缓冲区，链表，可以被不同上下文环境中运行的代码直接访问，需要互斥方法保护。
- 消息：内存空间中一段长度可变的缓冲区，任务间，ISR-任务之间的通讯机制，ISR可以写，不能读
- 常用消息分类：邮箱，消息队列。

同步、互斥：信号量，二元信号量，互斥信号量，计数信号量。

- 管道：管道是一个虚设备，提供了通过IO设备接口访问消息队列的一个界面，任务可以使用标准的IO接口open，read，write，以及ioctl调用。
- 时间
- 信号：

UC/OS

全称：micro Control OS 嵌入式领域研究人员Jean J.Labrosse与1992年在《嵌入式系统编程》杂志的5月，6月刊登上的文件连载，并发布源码。他创立了Micrium公司，提供嵌入式软件和解决方案，出售软件商业许可证 基于优先级抢占式，可移植，可裁剪，多任务实时操作系统。特征为短小精悍。源码用ansi c写的，移植性强。最小可达2kb，最小数据ram要求10kb。ucos可以再8位-64位，超过40中不同架构mcu上运行。

```
OS_STK userAppTaskStk1[1000];
OS_STK userAppTaskStk2[1000];

extern void userApp1(void *);

main

OSInit();

OSTaskCreate(userApp1,(void *)0,&userAppTaskStk1[1000-1],5);

OSStart();

OSTimeDly(int);//sleep

OS_ENTER_CRITICAL();
OS_EXIT_CRITICAL();//关中断，进临界区

OSTaskDel(OS_PRIO_SELF);//删除自己，但任务代码还在只是不调用
```

运行环境：

PC，SP，程序状态字寄存器（PSW），通用寄存器内容，函数调用信息（已存在于堆栈）,优先级，状态等，保存在任务控制块TCB中。

OSUnMapTbl[] OSMapTbl[]两张映射表

OSRdyGrp OSRdyTbl[]两个变量

统计任务：OSTaskStat()，每秒计算一次CPU在单位时间内被使用的时间，并把计算结果以百分比的形式存放在变量OSCPUsage中，一遍应用程序通过访问它来了解CPU的利用率。

UCOS：64个优先级别，数字越大优先级越低。

应用程序创建新任务时，需要把初始数据（任务指针，任务堆栈指针，程序状态字等）实现存放在任务的堆栈中。在OSTaskCreate()中调用OSTaskStkInit()来完成任务堆栈初始化工作。

任务就绪表：ucos在内存中设立了一个记录表，系统中的每个任务都在这个表中占据一个位置，使用该位置的状态（1或0）表示是否就绪。

OSCtxSw完成任务上下文切换。

进入中断时，中断嵌套+1，OSIntEnter（），离开时调用OSIntExit（）

在中断服务程序中调用的负责任务切换工作的函数是OSIntCtxSw（）

时钟节拍（TimeTick），调用OSTimeTick（）完成系统每个时钟节拍需要做的工作

同步与通信：

事件控制块(ECB):用来描述信号量，邮箱和消息队列这些时间。事件的总数：OS*MAX*EVENTS，OSEventPtr把空事件都联成一个单向链表，每次创建一个事件块时，系统从链表中取出一个空块，删除事件时归还一个块。

OSSemCreate（）创建信号量OSSemPend（）请求信号量，可以设置timeout

OSSemPost（）释放信号量，先判断有没有等待该信号量的任务，有的话用OS*Sched去调用它，没有的话信号量加1，成功后返回OS*ON_ERR

OSSemDel删除信号量

互斥性信号量：用OSMutexCreate()创建，OSMutexPend（）请求，OSMutexPost()发送信号量，OSMutexDel()删除。

消息邮箱：通过传递数据缓冲区指针的方法来通信。OS_EVENT_TYPE_MBOX，OSEVENPTR指向数据缓冲区。

消息队列：在任务间传递多条消息，分为三个部分：事件控制块，消息队列，消息。 信号量还是看ppt吧。
信号量集：有等待任务链表，使用OSFlagCreate（）创建信号量集，同样的使用pend和post来请求和发送。

内存动态分配

改进了malloc和free，是执行时间成为确定的。 ucos对内存进行两级管理，大片内存变若干个分区，分区分为内存块，操作系统以分区为单位来管理，任务以内存块为单位来获得和释放。使用情况由内存控制块记录。ucos使用空内存控制块链表管理空内存控制块

移植步骤

要让ucos正常运行，必须：

- c编译器能产生可重入代码
- 处理器支持中断，并且能产生定时中断，通常在10-100hz
- 提供打开和关闭中断的指令
- 处理器支持能够容纳一定量数据的堆栈
- 处理器有将堆栈指针，以及寄存器读出，存储到堆栈，或内存中的指令

Linux

ARM有7种运行状态：用户，中断，快中断，监管，中止，无定义，系统

虚拟内存，使用MMU，其余是操作系统VM。

S	R	CPU特权	CPU用户
0	0	NO Access	No
1	0	Readonly	N
0	1	R	R
1	1	Not sure(可读可写)	Not sure（可读可写）

Linux的启动

1. head.S是linux运行的第一个文件
2. 内核的入口是text，在arch/arm/kernel/vmlinux.lds.S中定义-ENTRY(text)
3. vmlinux.lds.S是ld script文件

启动主线：

- 确定process type
- 确定machine type
- 创建页表
- 调用平台特定的_*CPU*FLUSH函数（清除Cache，清除Dcache，清除Writebuffer，清除TLB）
- 开启mum
- 切换数据

ARM-linux系统调用

- LIBC和直接调用
- arm处理器有自陷指令SWI
- CPU遇到自陷指令之后，跳转到内核态
- 操作系统保存当前运行的信息，根据系统调用号查找相应的函数去执行
- 执行完了返回，恢复信息。

arch/arm/kernel下写.c系统函数
arch/arm/kernel/call.S中添加新的系统调用调用号和函数名

设备驱动程序的重要性

设备驱动程序往往工作与系统内核状态，因而运行性能，可靠性制约着应用系统的性能和可靠性

Linux的设备驱动

由接口充当应用程序和实际硬件之间的桥梁。

设备驱动程序： 是直接控制设备操作的那部分程序，是设备上层的一个软件接口。 实际上对于软件角度来说，设备驱动程序就是负责完成对IO端口地址进行读写操作。 设备驱动程序的功能是对对IO进行操作，且只能被调用。
与操作系统内核的接口：
与系统引导的接口：完成对设备的初始化。
与设备的接口，完成对设备的交互操作功能
与操作系统内核的接口：include/linxu/fs.h中的file_operations数据结构 与系统引导的接口：字符设备初始化在drivers/char/mem.c中的chr_dev_init()函数，块设备初始化在drivers/block/ll_rw_blk.c中的blk_dev_init()函数 与设备的接口：完成对设备的交互操作功能

外部设备：

1. 字符设备（像字节流一样被访问的设备，不允许来回读写，以字符为单位传输，可以通过文件的方式访问）
2. 块设备（以数据“块”为单位）
3. 网络设备（能和其他主机进行网络同学的设备，基于BSD套接口访问）

设备驱动程序表：描述了系统中所有设备驱动程序，以及设备驱动程序所支持的操作的入口地址。

当应用程序打开/dev下的文件，并进行各种操作时，OS将应用程序的调用及参数转给注册过的对应的对应的设备驱动程序的对应函数。

对设备文件进行系统调用时，将从用户态进入到内核态。内核根据该设备文件的设备类型和主设备查询相应的设备驱动程序，有驱动程序判断设备号，完成相应的操作。

接口与外设

GPIO就是可以由CPU直接操纵的引脚，可以写，可以读，可以接通信单元。

如果要使用GPIO那么首先要include,并注意在gpio的使用上使用HIGH和LOW来代表输出的信号，而不是直接使用1或0（因为在不同的平台上对HIGH和LOW的定义可能会不一样）。
对于测试GPIO引脚的数字是否可用，需要使用

```
    int gpio_is_valid(int number);
```

对于申请GPIO要使用：

```
    int gpio_request(unsigned gpio,const char* label);
```

使用完之后释放GPIO需要：

```
    void gpio_free(unsigned gpio);
```

对于激活GPIO为输入或输出，需要使用以下的两个函数：

```
int gpio_direction_input(unsigned gpio);//初始化为输入
int gpio_direction_output(unsigned gpio,int value)//初始化为输出
```

返回值为0为成功，负值为错误。对于初始化为输出，value则是初始输出值。

读取值或者设置输出值使用：

```
int gpio_get_value(unsigned gpio);//读取值
void gpio_set_value(unsigned gpio,int value)//设置输出值
```

返回值1为HIGH，0为LOW。这两个函数没有错误返回，因为在一开始错误的引脚就不能被初始化。

对于使用I2C或者SPI这种方式的设备来说，需要等待对方的起始信号。使用

```
int gpio_cansleep(unsigned gpio);
```

随后使用：

```
//返回值为0-low，1-high，可能会进入等待
int gpio_get_value_cansleep(unsigned gpio);
void gpio_set_value_cansleep(unsigned gpio,int value);
```

来读取引脚或设置引脚电平。

将GPIO映射为IRQ中断：

```
//将GPIO映射到对应的IRQ编号
int gpio_to_irq(unsigned gpio);
//IRQ->GPIO
int irq_to_gpio(unsigned irq);
```

/sys的局限（文件用于直接修改gpio）

这个级别的程序会受到两种额外的延时和延迟的影响。一种是调度延迟，即切换的时间，一种是系统负载，其他更重要的应用程序需要使用处理器，系统不会让你的程序使用处理器。

linux访问GPIO

内核级：使用设备驱动程序等 虚拟文件系统级： 应用程序： 让内存映射到AHB上，访问/dev/这种？？？（不确定）

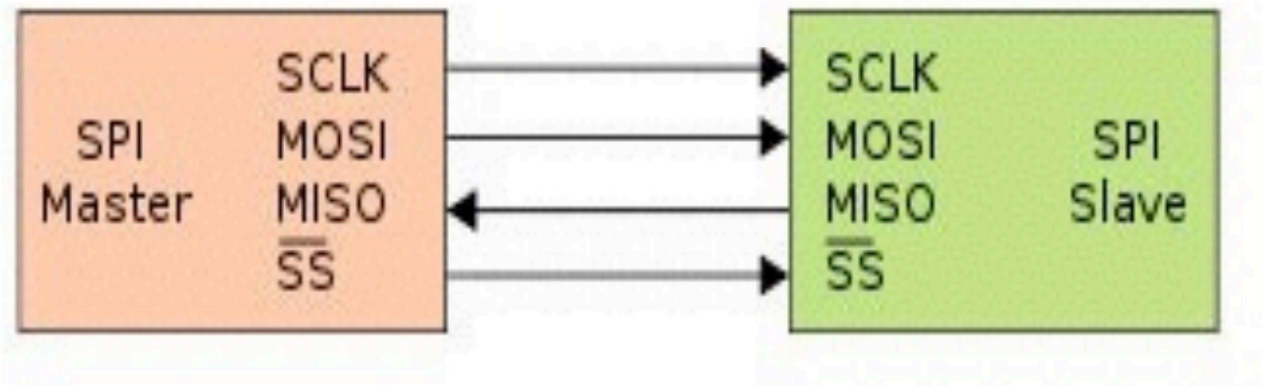
通信

异步通信是指通信的发送和接收设备使用各自的时钟控制发送和接受。
接口信号
DST数据装置准备好，表明通信装置可用。DTR数据终端可以使用。
接收线信号检出（RLSD），用来告知DTE准备接受数据。振铃提示（RI），通知终端已经被呼叫。

RS232 TxD和RxD上：
逻辑1（MARK）=-3V~-15V
逻辑0（SPACE）=+3V~+15V
RTS、CTS、DSR、DTR和DCD等控制线上：
信号有效（接通，ON状态，正电压）：+3V~+15V 信号无效（断开，OFF状态，负电压）：-3~-15

SPI

- Serial Peripherals Interface
- 应用于外部移位寄存器，D/A，A/D，等外部设备进行扩展
- 接线：
  - MOSI-SPI从输入，主输出
  - MISO-SPI主输入，从输出
  - SPICLK-SPI时钟
  - SPICE-SPI从发送使能



SPI的优点

- Fast and easy
- Everyone supports it

Multiple receivers do not require separate select lines as in SPI

- At start of each I2C transaction a 7-bit device address is sent

- Each device listens – if device address matches internal address, then device responds
- SDA (data line) is bidirectional, communication is half duplex
- SDA, SCLK are open-drain, require external pullups
- Allows multiple bus masters

```
Speed Comparison
• Assume a 400 Khz I2C bus, 2.5 us clock period
(2.5 e-6)
• Random write:
– 9 bit transmission = 2.5 us * 9 = 22.5 us
– 5 ms + 22.5 us* 4 (control,addhi,addlo,data) =5.09 ms
– For 64 bytes = 325 ms approximately, not counting software
overhead.
• Page Write
– 67 bytes total (control, addhi, addlo, data)
– 5 ms + 67 * 22.5 us = 6.5 ms!!!
```

文件系统

- ROM文件系统
  - ROMFS是一种只读文件系统
- 磁盘文件系统
  - FAT16，FAT32，Ext2FS，NTFS
- Flash文件系统
  - TrueFFS、MFFS
  - JFFS/JFFS2，YAFFS/YAFFS2
  - FAT16,FAT32,NTFS,exFAT
- 内存文件系统

NORvsNAND

- NOR型闪存的特点：
  - 具有独立的地址线，数据线，支持快速随机访问，容量较小
  - 具有芯片内执行（execute in place）的功能，按照字节为单位进行随机写
  - NOR适合用来存储少量的可执行代码
- NAND：
  - 地址线、数据线公用，单元尺寸比NOR小，更高的价格容量比，高存储密度和大容量
  - 读写采用512字节的页面
  - NAND更适合作为高密度数据存储

性能参数	NAND型	NOR型
读操作的时间	3.5MB/sec	15MB/sec
写操作的时间	0.65MB/sec	0.15MB/sec
擦除操作的时间	2ms	1sec
擦除大小	8-32KB	64-128KB
擦除次数限制	1,000,000 次	100,000 次

- 与NOR型器件相比，NAND型器件的写入、擦除速度较快。
- NOR闪存带有SRAM接口，可以实现随机写。
- NAND器件使用I/O口串行存取数据，操作单元为512字节，可取代硬盘或其他块设备，需要Memory Technology Devices(MTD)驱动。

10

写前需要擦除，擦除单元（block）> 读写单元（Page）：

- 损耗均衡问题：闪存上的每个块都具有擦除次数的上限，被称为擦除周期计数（erase cycle count）。
- 为了提高闪存寿命，并减少某些块提前损坏的概率，闪存管理算法设计时应尽可能减少擦除次数，并将擦除操作均布于整个芯片。
- 位交换问题：所有闪速存储器都受到位交换现象的困扰，表现为一个bit位发生反转。当存储器用于敏感信息存储时，需使用错误探测/更正（EDC/ECC）算法提供可靠性支持。
- 坏块处理问题：NAND器件中的坏块是随机分布的，由于消除坏块的代价太高，因而使用NAND器件的初始化阶段进行扫描以发现坏块，并将坏块标记为不可用。

12

- 掉电保护问题：文件系统应能保证在系统突然断电时，最大限度地保护数据，使文件恢复到掉电前的一个一致性状态。

文件系统分类

- 硬盘模拟法：将闪存设备模拟成具有每个扇区512字节的标准块设备，在此基础上用成熟的磁盘文件系统进行管理。
  1. 读入整个擦除快，修改，重写整个块
  2. 不考虑擦除的均衡性，某些块可能很快被写坏
  3. 系统一致性没有安全保证，内存可能随时断电，这种丢失无法恢复
  4. 为了提供均衡性和可靠的操作， 模拟块设备的山区存放在物理介质的不同位置上，地址转化用于记录当前每个扇区在模拟块设备上的位置。

- 系统实现层次：
  - 设备驱动程序层
  - 地址转化层
  - 文件系统管理层

- TrueFFS Flash模拟硬盘程序包，M-system公司

  - 动态和静态的损耗级别判定算法
  - 安全算法保证在突然断电时数据完整性
  - 解决为交换问题的Reed-Solomon纠错算法
  - 自动的坏块管理
  - 优化处理功能，减少擦除次数，优化垃圾回收操作
  - 容量*总擦写次数*0.75/每天的写入字节

- 直接实现法：直接对闪存设备进行操作，建立日志文件系统，避免模拟转化工作

  - 去掉FTL这一层，对性能有很大提高
  - 日志文件系统：特点是对数据的更新采用前向写入。更新的数据写入空白快。

- 损耗均衡问题：闪存上的每个块都具有擦除次数的上限，被称为擦除周期计数（**erase cycle count**）。
- 为了提高闪存寿命，并减少某些块提前损坏的概率，闪存管理算法设计时应尽可能减少擦除次数，并将擦除操作均布于整个芯片。
- 位交换问题：所有闪速存储器都受到位交换现象的困扰，表现为一个**bit**位发生反转。当存储器用于敏感信息存储时，需使用错误探测/更正（**EDC/ECC**）算法提供可靠性支持。
- 坏块处理问题：**NAND**器件中的坏块是随机分布的，由于消除坏块的代价太高，因而使用**NAND**器件的初始化阶段进行扫描以发现坏块，并将坏块标记为不可用。
- 掉电保护问题：文件系统应能保证在系统突然断电时，最大限度地保护数据，使文件恢复到掉电前的一个一致性状态。

嵌入式复习

1 基本概念

1.1 什么是 EDB? ——嵌入式系统

a. 定义：有 CPU，上面有程序在跑

特点：功能、可靠性、成本、体积小、功耗

b. EBD 与其它系统的区别

区别于 PC：PC 的资源更足->CPU、内存、硬盘、电源

区别于逻辑硬件电路：硬件上面没有 CPU

1.2 EBD 开发的基本过程

a. 并行的特性

b. 设计决定什么

c. 软件硬件划分

软件：操作系统、编程模型

硬件：CPU

d. 通用件的好处

2 MCU

2.1 MCU vs CPU

MCU 是微控制器/单片机

2.2 如何选择？

资源性能不足时，封装尺寸相同时选容量最大的，实际做的时候可以换成容量小的。

选择单片机的基本条件是什么，运算能力，片上的，SDRAM。如果资源不足，该算法。如果 flash 不足怎么办，如果接口不足巴拉巴拉的不足怎么办，如何扩展等。AD 扩展，1 切 4 芯片，成本是不是划得来。通常情况下集成的片内 AD 最

便宜，但是有时候需要用片外的，精度需求。

2.3 ARM

a. ARM：32 位

Thumb：16 位

Thumb2——Cortex{

Cortex A application 高端 手机

Cortex R Realtime

Cortex M Microcontrol 低端单片机 }

树莓派 arm 没有 Thumb2（混合 16 和 32 位），

b. 工作模式（不考 x86）

ARM：特权模式、用户模式、中断模式、快中断模式（牵扯到寄存器的问题，某些寄存器有不同的 bank）

MIPS：没有快中断，快中断关系到寄存器

工作模式怎么切换？

如用户模式怎么切换到系统模式：软中断

c. ISA（多条件执行 多寄存器寻址）

d. RAM 的指令集，32 位的都带条件执行。

S 标志：要不要改寄存器，带 S 标志修改标志寄存器会影响后一条跳转指令的结果。后一条指令如果是跳转，并且前面修改了标志位就要 stall

e. 函数调用规范

f. 中断向量表

中断向量表的第一项是一个初始值，reset 启动地址，Cortex 启动地址不为 0，ARM 启动地址为 0

startup.s 里面做了啥，拷贝和清零。Main 函数是在启动程序先做完后才做的。

g. 内存模型

h. 启动过程

startup.s 拷贝有全局变量的寄存器，清零 xxxxxx 寄存器

main 不是上电后第一个执行，启动程序之后才做

3 软件模型

3.1 轮询：没有任何中断，无法处理紧急的事情

3.2 前后台（一般不考）

主循环要做的事情是固定的，有中断（置标志）和循环（读标志），把数据放到共享内存，在大循环里读取共享内存，在大循环里要不断关中断开中断

3.3 中断驱动

没有前台的大循环，只有中断，中断来就做事情，没有就休眠

3.4 动态队列

在中断里做基础工作，其他工作放队列里，主循环要做什么由队列决定

VS 前后台主循环要做的事跟中断有关系

VS 中断中断来了之后事情不是在中断中做的，而是在主循环里面做的

4 Bootloader

4.1 什么是 Bootloader

上电到第一行代码之间有 Bootloader

4.2 Bootloader 的功能

两种：一种为了 linux，一种为了 RTOS

一种 SDRAM

一种 Flash（本身）

4.3 两阶段工作

把 CPU 放在核里，初始化 CPU、SDRAM，调整 CPU 频率等跟操作系统无关的事情

把操作系统可执行代码放在 SDRAM，然后跳到 SDRAM，开始系统

不具有第二阶段工作能力的 bootloader，能把 cpu 初始化，然后跳到 flash 中开始操作系统。

5 RTOS

5.1 实时的概念

规定时间内完成，ddl

强实时

弱实时

ns us ms

5.2 抢占式调度的实现

调度时机

在就绪队列里出现了比在 CPU 中优先级更高的任务。

还在 CPU 上不会被拉下来，要么自己发起：新任务（唤醒一个线程）、优先 级发生变化、IO 等待（间接主动等待，线程主动发起）。要么中断

调度实现

几张图，任务控制表 tcb

堆栈指针、栈顶指针、寄存器，切出来做什么，切进去做什么

一个任务新建时要不要在寄存器里放假的什么东西（？）

5.3 优先级反转

提高优先级，火星车

5.4 任务间同步与通信（不重要）

任务间通不通讯，信号量等

5.5 uC/OS 基本常识

uC/OS 对 RTOS 的强化，函数库的形态出现，和自己的程序编译形成一个整体，从 main 中调用系统的函数。如何从优先队列里面找出任务，查找表，不需要便利搜索。

6 Linux

6.1 ARM Linux 的启动过程

6.2 系统调用的路径

6.3 内核模块的加载与卸载

设备驱动程序，作为内核模块的方式加载，内核模块在加载的时候两个必须的函数，in out

每个设备在 dev 里面的虚拟文件，主设备号，类别，从设备号，传给具体的操作函数，从而

知道现在在用哪个设备。

Bootload

6.4 设备驱动程序的接口

7 接口与外设

7.1 GPIO 基本概念

可以由 CPU 直接操纵的引脚，可以写，可以读，可以接通信单元。

GPIO=通用输入输出

- 输入：程序可以判断输入信号是 1 还是 0
- 输出：程序可以设置输出 1 或 0

可以以此来连接外部器件

- 输入：开关/按钮
- 输出：LED、扬声器

复用：有多个功能，输入输出不能同时进行，不需要关心太 EE 的事情

7.2 Linux 下访问 GPIO 的方法

内核级

文件系统级

应用程序级：writingpi 的库

在 ARM 上运行的速度远远低于 CPU 上的速度

外设实际上是一个地址

Linux sys 目录下有专门管 GPIO 的文件，可以跨语言使用；应用程序都做了库。

7.3 串行通信

异步

UART 通信原理——通用异步收发传输器（Universal Asynchronous Receiver/Transmitter)

UART：9600（波特率） 8（bit） N（停止位） 1（校验位 1 位）

奇校验 1 的个数是奇数个还是偶数个

RS232 电平规范(倒过来的)

RS232

- EIA-RS-232C对电器特性、逻辑电平和各种信号线功能都作了规定。
- 在TxD和RxD上：
  - 逻辑1(MARK) =-3V～-15V
  - 逻辑0(SPACE)=+3～＋15V
- 在RTS、CTS、DSR、DTR和DCD等控制线上：
  - 信号有效（接通，ON状态，正电压）＝+3V～+15V
  - 信号无效（断开，OFF状态，负电压) = -3V～-15V

同步

SPI（serial peripheral interface）ptotocol：双工 4 线

应用于外部移位寄存器，D/A，A/D、串行 EEPROM、LED 显示驱动器等外部设备进行扩展。

I2C：半双工 2 线

片选信号（地址和连线）

8 文件系统

8.1 NOR 和 NAND 区别

NOR：可以给出地址直接读，可以运行程序，速度比 SRAM 慢，大容量数据储存器

NAND：不能给出地址读一个值，一次读一个 block，写入、擦除速度慢，适合做 Bootloader

8.2 Flash 特点

按块擦写

擦写均衡问题

不能随意写入，写入之前先要擦除，读-擦除-写，一次写一个 block，有写入次数限制，支撑时间很长

8.3 Flash 文件系统

模拟块设备型　TrueFFS

Flash 专用的日志文件系统：往下写，不擦除

根文件系统：虚拟文件系统

模拟磁盘文件系统（实际上在 NAND 上，转换表）

JFFS

YAFFS

9 扩展

开发过程

上位机：PC

下位机：嵌入式板卡

中间怎么连的：下载调试一条线，通信一条线

Makefile 目标依赖动作

仿真器：好处是具有完全的硬件能力，虚拟机的所有 GPIO 也是假的，对复杂脉冲会不好处理。

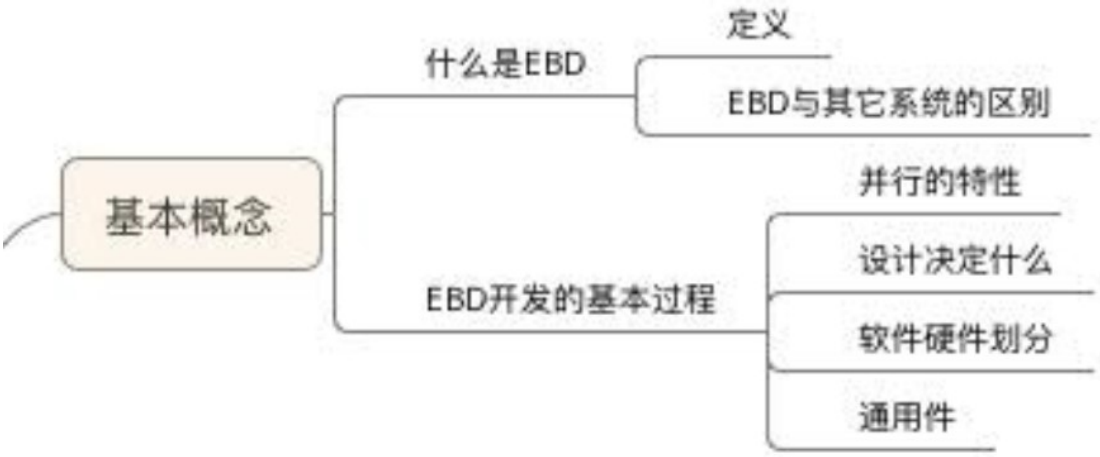
虚拟机：程序上有软件模拟运行你的 CPU

embeded system复习课

[illegible]： 我的第一个笔记本
[illegible]： 2016/6/21 8:08 [illegible]： 2016/6/21 9:24
[illegible]： huangminzh@gmail.com

embeded system复习课

思维导图：



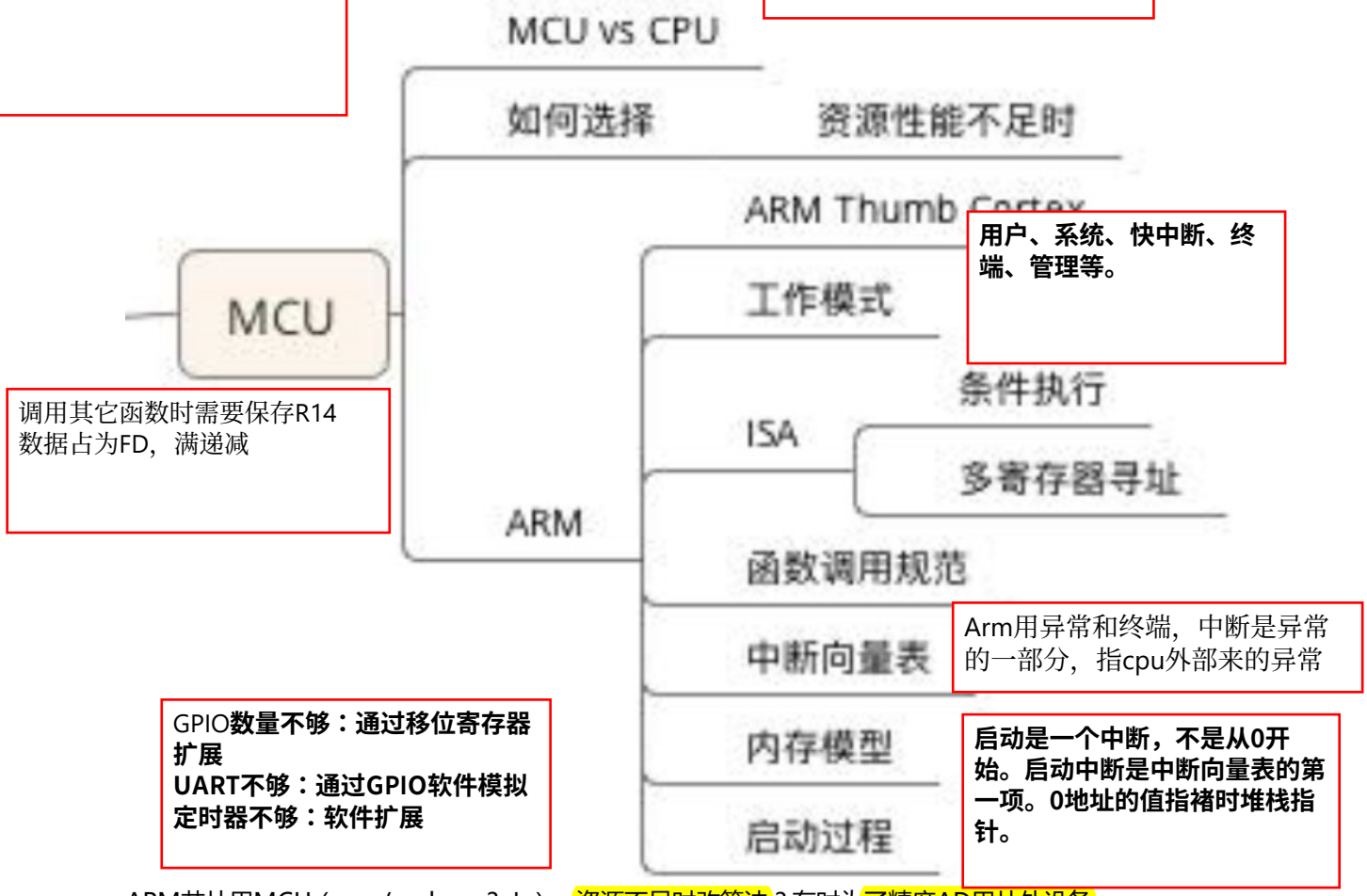
IEEE定义；自己也定义了；
必须有软件（cpu跑程序，这是底线），不能纯硬件；但是又和PC不同，资源不足（内存硬盘等），但不代表他会比PC差；

承认嵌入式并行（硬件软件都有并行）使你的嵌入式健壮；硬件上要选择哪种硬件，软件选择编程模型软件库等（有设计的题，硬件软件该有什么，不需要说型号，要有电池，RTOS设备驱动怎么实现，中断怎么实现）

通用件！

嵌入式中，cpu指mcu；
用mpu强调不带外设的cpu。
mcu是cpu之外带有外设；
至少有定时器，gpio；
一般有adc，pwm。

ARM 32，支持16Thumb已过时
Corte为增强16Thumb，有部分32。
no MMU：cortex-M
with MMU: cortex-A



调用其它函数时需要保存R14
数据占为FD，满递减

Arm用异常和终端，中断是异常的一部分，指cpu外部来的异常

GPIO**数量不够：通过移位寄存器扩展**
UART不够：通过GPIO软件模拟
定时器不够：软件扩展

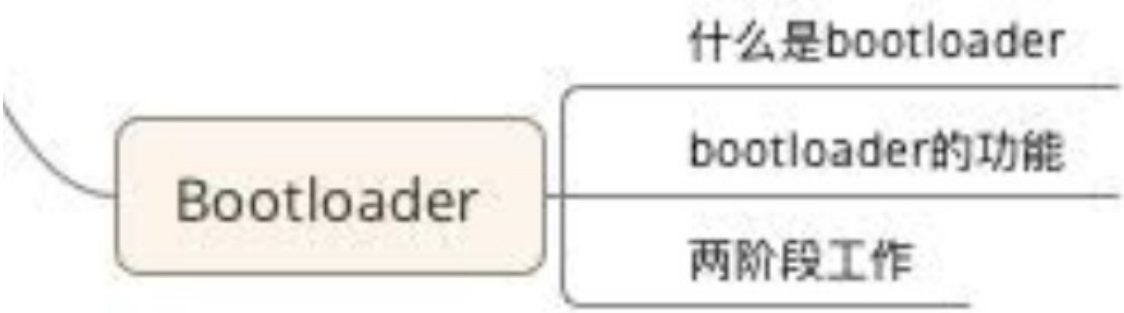
启动是一个中断，不是从0开始。启动中断是中断向量表的第一项。0地址的值指褚时堆栈指针。

ARM芯片用MCU（mer/perhero？！）；资源不足时改算法？有时为了精度AD用片外设备；

Cortex A R M的意思要知道；ARM的快中断别人没有的（寄存器会换一组），工作模式切换（软中断）；所有的arm isa指令都带有条件指令（这代表什么？flag寄存器，x86是判断某个寄存器）；可以同时对多个寄存器load或store；swap？！

cortex 中断向量表前有堆栈初始值；reset 启动地址（不是0）；

轮询无法解决紧急的事；前后台，要注意共享数据的保护；中断驱动没有大循环，只有中断；动态队列，中断程序不能太长，假如中断要做的事太多，就只执行一部分，把大部分事扔到队列中；



上电到第一行之间有bootloader （PC是bols？！嵌入式是bootloader）；设置pll，让操作系统进入可以工作的状态，，把可执行代码放到SDRAM中，bootloader本身在flash中；PEEK POKE那个实验没有第二阶段，没有在SDRAM中的可执行程序；SDRAM和flash

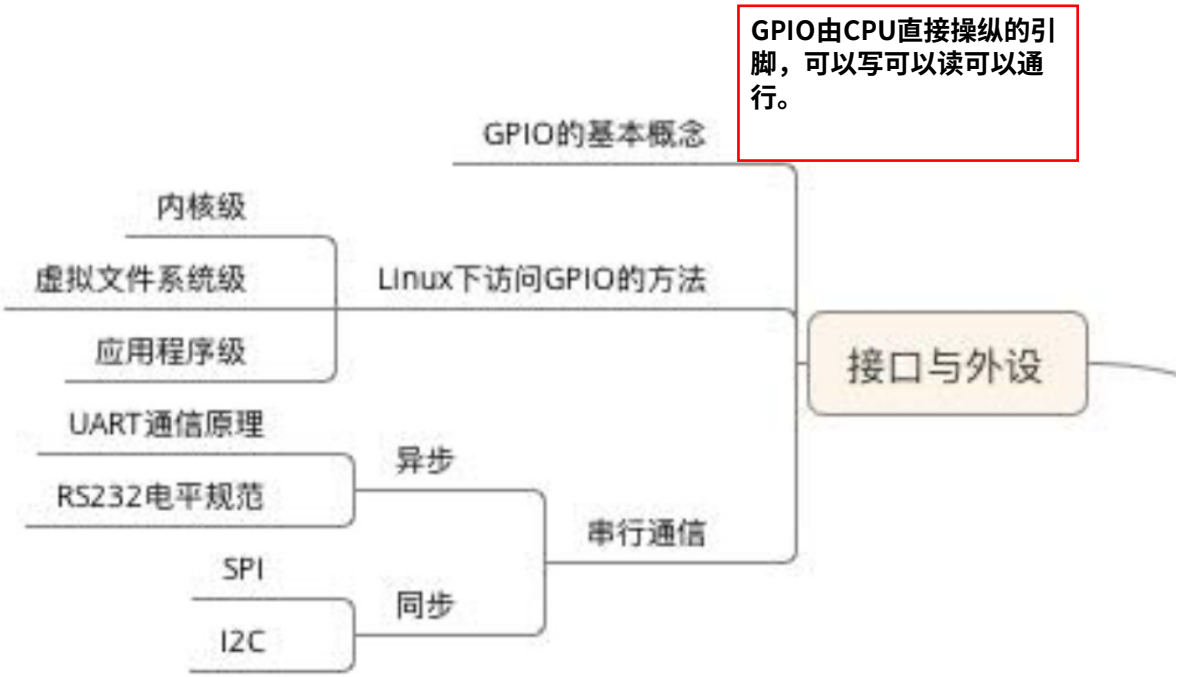
实时不是快，ddl前完成（强实时，弱实时的指标确实是时间，要记住）；动态队列不会强占，抢占式调度强实时性；时机：程序自己发生的（主动等待），或者是外部中断；调度实现：看ppt里的图，要不要放假的寄存器的值wtf？！；

火星车，优先级反转；

RTOS以函数库的形式出现；任务堆栈就是全局变量wtf？！；

ismod两个函数，启动和删除的；虚拟文件（主设备号，从设备号？！）；

操作系统内核的接口。
与系统引导的接口：完成对设备的初始化。
与设备的接口，完成对设备的交互操作功能。



一个管脚都是复用的；输入输出不能同时做；上拉下拉不考；wtf；abp控制器wtf？；sys目录下有对gpio的操作；应用级别可以用别人的库控制GPIO；

UART9600 8 n 1什么意思 8o1奇校验（1的个数奇数） 10个bit；算buffer溢出；

SPI四线（一根线做片选）；I2C两线，半双工（地址做片选，简单）

NOR可以执行程序，（什么东西太小，在flash执行了代码）；NOR放bootloader，把NAND中的执行程序放到SDRAM中；0地址放NOR，1个G的地方放NAND；

flash写入前要按块擦除（读->擦除->写），有寿命，擦写均衡；磁盘文件系统下再虚拟出what？TrueFFS；

虚拟机，仿真机？仿真机有硬件的能力，虚拟机都是假的（对外来的脉冲处理不同）

判断最后一题应该是错的

根文件系统就是虚拟文件系统

数据库设计，要有private key；

写伪代码，流程图都可以

1. 项目开发报告：
a 技术方案
b 实施方案
c 人员分工
d 心路历程

2. 尽可能多的资源
   全部代码
   照片
   PPT
   视频
   会议/通信（邮件沟通，wtf？！）