引导装载程序

翁恺

2016春

什么是bootloader?

- 上电后的第一段代码可能是：
  - 程序本身：小规模单片机程序
  - bootloader：
  - BIOS：PC
- 采用bootloader是为了方便程序的下载、启动和最初的调试

Bootloader程序基本概念

- Boot Loader是在系统启动时激活，在操作系统内核运行之前运行的一段程序
- 初始化硬件设备和建立内存空间的映射图
- 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

如何把程序放到嵌入式机器里?

- 预先烧录:
  - ROM
  - flash
    - 将整个MCU当作flash来烧录
- 专用硬件接口:
  - JTAG
  - SWI
- ISP:
  - 芯片厂家实现的bootloader

bootloader

- 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bootloader
  - 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

ISP or ICP

- 不同厂家的不同名词而已
- 有ISP的好处是不需要特殊的编程（烧录）硬件，有串口就可以
- IAP＝？

程序可以在

- ROM
- flash
- SDIO
- IDE
- SATA
- 但是只有ROM或flash的可以启动时直接访问

程序可以运行在

- ROM
- flash
- SRAM
- DRAM
- 上电时的第一条指令只能在ROM或flash中
- DRAM需要初始化控制器之后才能访问

RESET

- x86: 0x0FFF FFF0，16位实模式
- ARM：0x0000 0000或0xFFFF 0000
- PPC：0x0000 0100或0xFFFF 0100

- 启动地址是启动代码的地址
- 启动地址是中断向量表，所以reset是一种中断

- Bootloader所支持的硬件环境
  - 每种不同的CPU体系结构都有不同的Boot Loader。
  - 也依赖于具体的嵌入式板级设备的配置。

- Bootloader相关的设备和机制
  - 主机和目标机之间通过串口建立连接，通过串口进行I/O。

- Bootloader的启动过程
  - 启动过程可以分为阶段1和阶段2两部分。

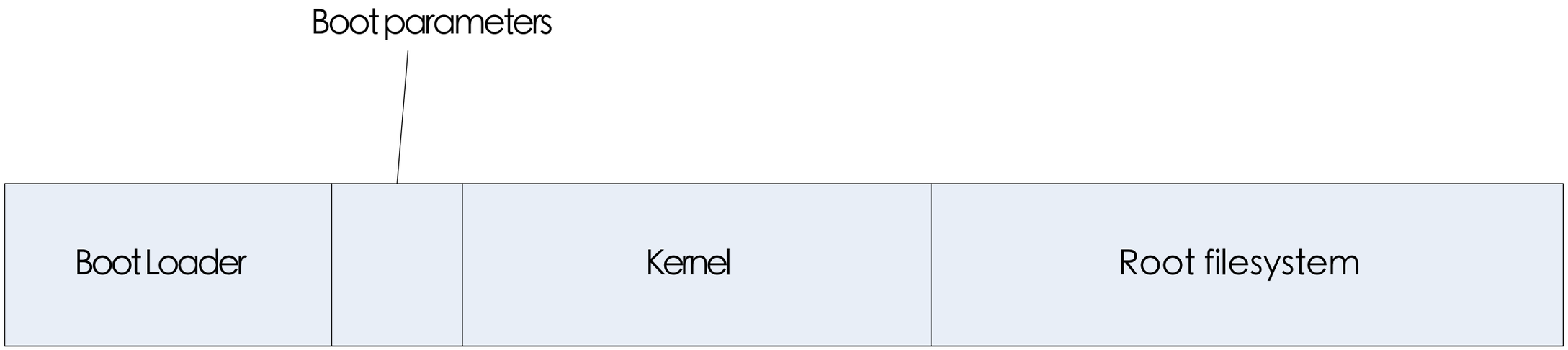
- Bootloader的操作模式
  - 启动加载模式
  - 下载模式
- Bootloader与主机之间的通信设备及协议
  - 串口协议：xmodem/ymodem/zmodem
  - 网络协议：TFTP

- Bootloader的操作模式
  - 启动加载
  - 下载模式
- Bootloader
  - 串口协议：xmodem/ymodem/zmodem
  - 网络协议：TFTP

1、xmodem是最早的协议之一，一种由几乎所有通讯程序支持的文件传送协议, 传送128个字节信息块；
ymodem和zmodem都是它的改进协议，
2、ymodem传送1024字节长的信息块，快于xmodem并且可送多个文件；
3、zmodem速度快于ymodem和xmodem，且可以更好地在断开后恢复传输。

嵌入式Linux的引导装载程序

嵌入式Linux的典型空间分配结构



固态存储设备指ROM、EEPROM或FLASH等

由Bootloader启动操作系统的方式

- Flash启动方式
- 硬盘启动方式
  - 在硬盘主引导区放置bootloader
  - 从文件系统中引导操作系统
- 网络启动方式
  - Bootloader放置在EPROM或Flash中
  - 通过以太网远程下载操作系统内核或文件系统
  - 开发板不需配置大的存储介质

Bootloader的功能种类

- 简单Bootloader
  - 只具有系统引导功能
- 具有监控功能（Monitor）的Bootloader
  - 调试支持
  - 内存读写
  - Flash烧写
  - 网络下载
  - 环境变量配置

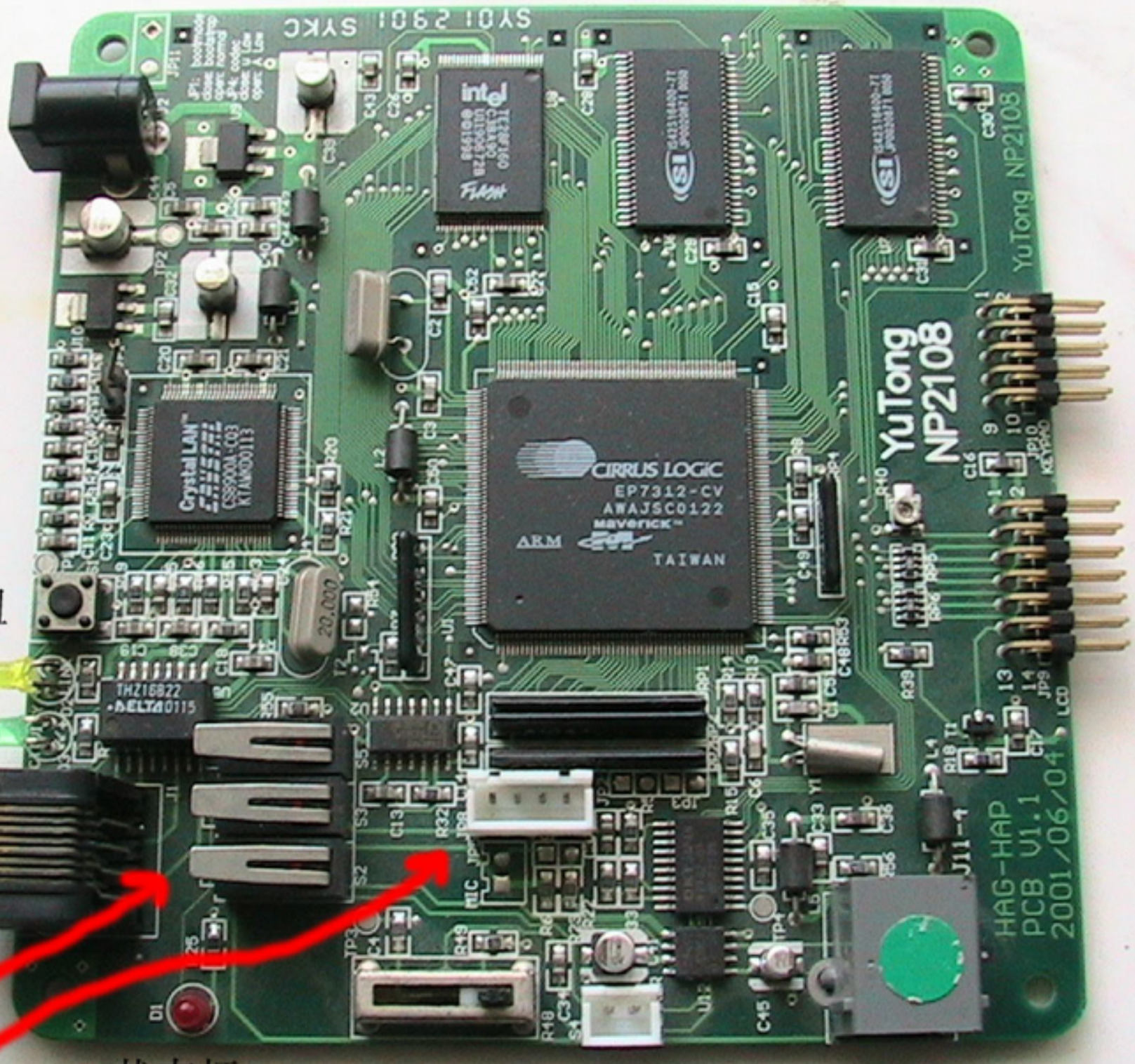
开源的Bootloader程序

名称	支持体系结构	是否Monitor
LILO	X86	否
GRUB	X86	否
BLOB	ARM	否
Etherboot	X86	否
U—boot	X86、ARM、PPC	是
RedBoot	X86、ARM、PPC	是
VIVI	ARM	是

引导装载程序的例子

NP2108的例子

- http://fm.zju.edu.cn/~wengkai/projects/2108/
- Cirrus Logic 的EP7312为CPU，片外具有4MB的flash，16MB的SDRAM，一个10M的以太网口
- 2108上电以后，flash里的bootloader启动，把flash里的linux的kernel的压缩包解开释放到SDRAM中，再把flash里的ram disk的包解开到SDRAM中，最后启动Linux。整个过程大约需要90秒



电源5V,内正

RESET按钮

网络灯

以太网

电话叉簧

接串口

状态灯

输出音量调节

接扬声器

接电话手柄

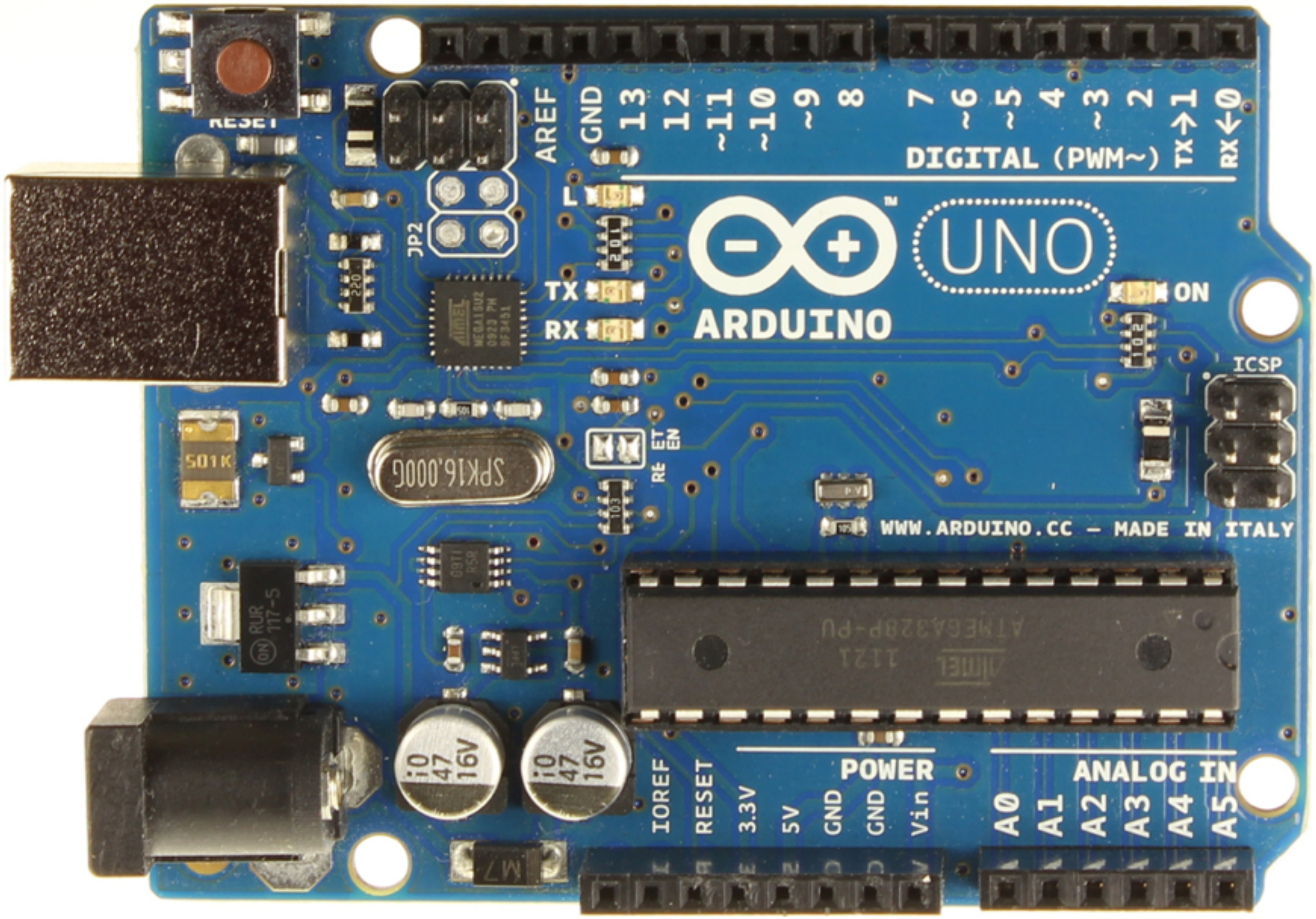
接键盘

接LCD

EP7312

- 具有片内bootloader，通过跳线选择
- 这个bootloader通过串口接受一个很小的程序ramloader，放在片内SRAM执行
- 这个片内SRAM的程序初始化片外SDRAM，通过串口接受较大的程序（hermit），放在SDRAM执行
- hermit具有shell界面，可以查看/修改内存，接收程序烧录flash
- 由hermit烧录一个bootloader到flash，下次启动时跳线选择进flash
- 为了直接启动Linux，有两个版本的bootloader

Arduino


RESET
AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
1
0
DIGITAL (PWM~)
TX
RX
L
JP2
ARDUINO
UNO
ON
ICSP
WWW.ARDUINO.CC – MADE IN ITALY
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5

Arduino

- AVR单片机，8位，16MHz，片内flash/SRAM
- 不具有片内bootloader，需用ICP方式烧录程序
- Arduino bootloader是第三方bootloader，采用ICP烧录后，上电首先进入这个bootloader
- bootloader在串口等待2s，如果没有收到特殊字符，就转入用户程序，否则进入bootloader

Arduino

- 必须由上位机程序负责重启MCU，否则来不及进入bootloader
  - 串口的DTR信号经过一个电容接到MCU的reset引脚，DTR的变化形成一次reset

- 由于bootloader占据用户程序空间，链接时必须指定特殊的程序起始地址

引导装载程序的实现

Bootloader的典型结构

- Boot Loader的启动过程通常是多阶段的
  - 提供复杂的功能：如突破引导扇区512字节限制
  - 提高代码移植性：高阶段代码采用高级语言
  - 提高运行速度 ： 高阶段代码在内存中执行
- 大多数Boot Loader可分为阶段1和阶段2两大部分
  - 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

Bootloader阶段1

- Boot Loader的阶段1通常包括以下步骤：
  - 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

Bootloader阶段1

- 2）为加载阶段2准备RAM空间
  - 除了阶段2可执行映象的大小外，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
    - 内存区域有效性检测方法
      - 保存指定内存区域
      - 写入预定数据
      - 读入数据并比较
      - 恢复内存数据

Bootloader阶段1

- 3）拷贝阶段2代码到RAM中
- 4）设置堆栈指针sp
- 5）跳转到阶段2的C语言入口点

Bootloader阶段2

- 1）初始化本阶段要使用到的硬件设备
  - 初始化至少一个串口，以便和终端用户进行I/O输出信息
  - 初始化计时器等

Bootloader阶段2

- 2）检测系统的内存映射
- 内存映射的描述
- 可以用如下数据结构来描述RAM地址空间中的一段连续的地址范围：
-

```
typedef struct memory_area_struct {
        u32 start; /* 内存空间的基址 */
        u32 size;  /* 内存空间的大小 */
        int used;
} memory_area_t;
```

- 内存映射的检测

Bootloader阶段2

- 连续内存区域探测——X86系统内存探测算法举例

```
for (p = (char *)0x100000; (int)p < 0x40000000; p += delta)
    {
     for (ix = 0; ix < N_TIMES; ix++)  /* 保存内存原有信息 */
    {  temp[ix] = *((int *)p + ix);
       *((int *)p + ix) = TEST_PATTERN;  /*TEST_PATTERN 0x12345678*/
     }
     cacheFlush (DATA_CACHE, p, 4 * sizeof(int));
     if (*(int *)p != TEST_PATTERN)/* 测试内存单元有效性 */
       p -= delta;
     for (ix = 0; ix < N_TIMES; ix++) /* 恢复内存原有信息 */
       *((int *)p + ix) = temp[ix];
}
```

Bootloader阶段2

- 3）加载内核映像和根文件系统映像
  - 规划内存占用的布局
    - 内核映像所占用的内存范围
      - MEM_START + 0X8000
    - 根文件系统所占用的内存范围
      - MEM_START + 0X00100000
- 从Flash上拷贝
  - While循环

Bootloader阶段2

- 4）设置内核的启动参数

  - 标记列表(tagged list)的形式来传递启动参数，启动参数标记列表以标记ATAG_CORE开始，以标记ATAG_NONE结束

  - 嵌入式Linux系统中，通常需要由Boot Loader设置。常见启动参数有：ATAG_CORE、ATAG_MEM、ATAG_CMDLINE、ATAG_RAMDISK、ATAG_INITRD，以ATAG_NONE结束。

Bootloader阶段2

- 5）调用内核
  - CPU寄存器的设置：
    - R0＝0；
    - R1＝机器类型ID；关于机器类型号，可以参见：
    - linux/arch/arm/tools/mach-types。
    - R2＝启动参数标记列表在RAM中起始基地址；
  - CPU 模式：
    - 必须禁止中断（IRQs和FIQs）；
    - CPU必须SVC模式；
  - Cache和MMU的设置：
    - MMU必须关闭；
    - 指令Cache可以打开也可以关闭；
    - 数据Cache必须关闭。

关于串口终端

- 向串口终端打印信息也是一个非常重要而又有效的调试手段
- 如果碰到串口终端显示乱码或根本没有显示的问题，可能是因为：
  - Bootloader 对串口的初始化设置不正确
  - 运行在host 端的终端仿真程序对串口的设置不正确

- Bootloader 启动内核后却无法看到内核的启动输出信息：
  - 确认内核在编译时是否配置了对串口终端的支持，并配置了正确的串口驱动程序
  - Bootloader 对串口的初始化设置是否和内核对串口的初始化设置一致
  - 还要确认 Bootloader 所用的内核基地址必须和内核映像在编译时所用的运行基地址一致

U-Boot

U-Boot介绍

- U-Boot全称Universal Bootloader，遵循GPL协议，是由德国的工程师Wolfgang Denk从8XXROM代码发展而来的。

- 不仅支持Linux系统的引导，还支持NetBSD、VxWorks、QNX、RTEMS以及LynxOS嵌入式操作系统。

- 它支持很多处理器，比如PowerPC、ARM、MIPS和x86。

U-Boot特点（1）

- 支持SCC/FEC以太网、OOTP/TFTP引导、IP和MAC的预置功能。
- 在线读写Flash、DOC、IDE、IIC、EEROM、RTC。
- 支持串行口kermit和S-record下载代码。工具可以把ELF32格式的可执行文件转换成为S-record格式，直接从串口下载并执行。
- 识别二进制、ELF32、uImage格式的Image，对Linux引导有特别的支持。

U-Boot特点（2）

- 单任务软件运行环境。U-Boot可以动态加载和运行独立的应用程序。
- 监控命令集：读写I/O、内存、寄存器、内存、外设测试功能等。
- 脚本语言支持（类似bash脚本）。
- 支持WatchDog、LCD logo和状态指示功能等。
- 支持MTD和文件系统。
- 支持中断。
- 详细的开发文档。

代码结构分析（1）

- Board：和一些已有开发板相关的文件，比如Makefile和u-boot.lds等都和具体硬件有关。
- common：与体系结构无关的文件，实现各种命令的C文件。
- cpu：CPU相关文件，其子目录都是以所支持的CPU为名，比如arm926ejs、mips、mpc8260和nios等，每个子目录中都包括cpu.c和interrupt.c，start.S。
- disk：disk驱动的分区处理代码。
- doc：文档。
- drivers：通用设备驱动程序。

代码结构分析（2）

- dtt：数字温度测量器或传感器的驱动。
- examples：一些独立运行的应用程序例子。
- fs:支持文件系统的文件，U-Boot现在支持cramfs、fat、fdos、jffs2和registerfs。
- include：头文件，还有对各种硬件平台支持的汇编文件，系统的配置文件和对文件系统支持的文件。
- net：与网络有关的代码，BOOTP协议、TFTP协议、RARP协议和NFS文件系统的实现。
- lib_xxx：与处理器体系结构相关的文件，如lib_mips目录与MIPS体系结构相关，lib_arm目录与ARM相关。
- tools：创建S-Record格式文件 和U-Bootimages的工具。