第六章 嵌入式系统开发

6.1 嵌入式系统开发环境构建

课程大纲

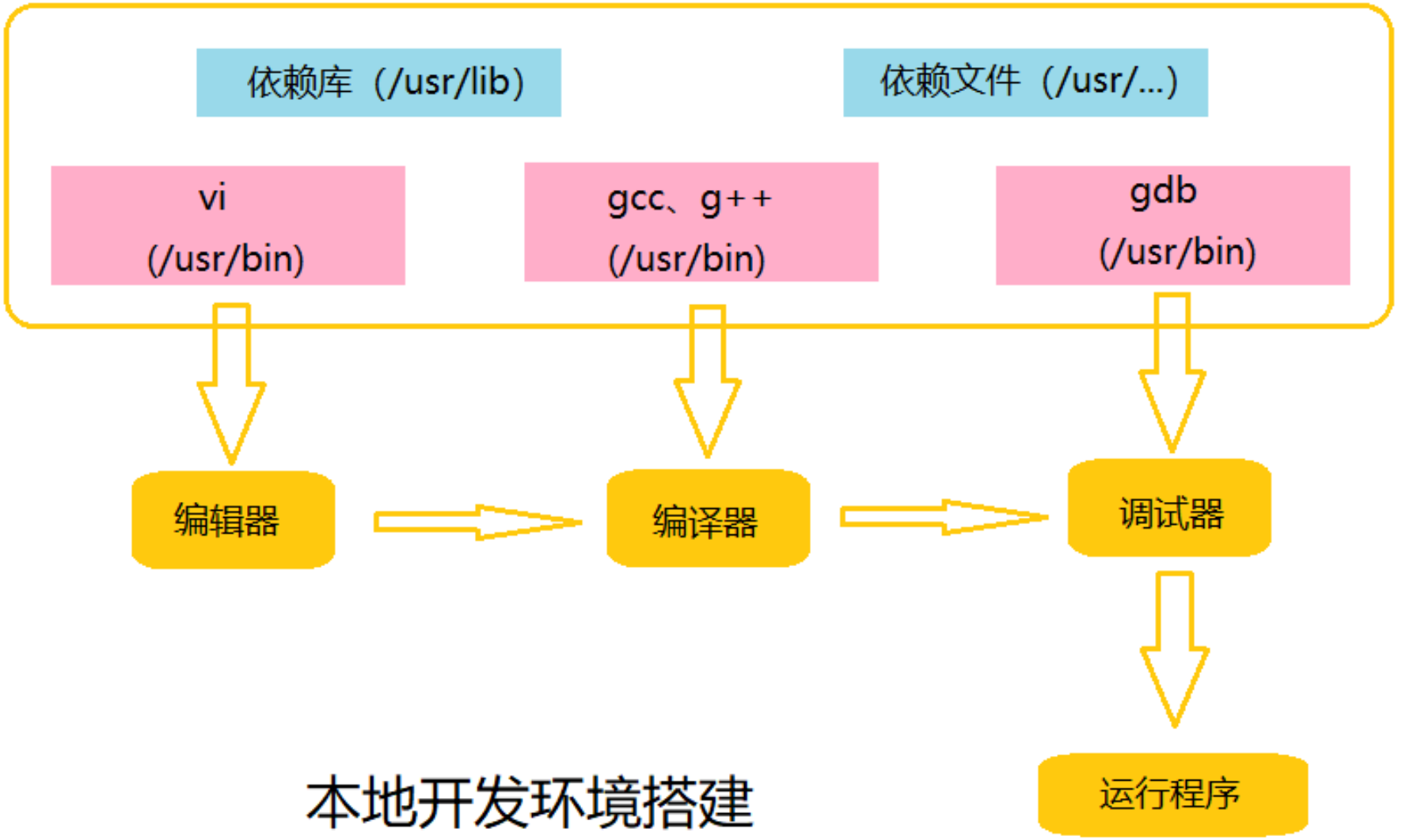
- 嵌入式软件系统开发概述
- 开发环境构建中的仿真技术
- 嵌入式 Linux 开发环境构建

6.1.1 嵌入式软件系统开发概述

Linux 本地软件开发模式

- 1. 程序编辑
- #vi debug.c
- 2. 程序编译
- 3. 程序运行
- 4. 程序调试

Linux 本地软件开发环境



本地开发环境搭建

嵌入式系统不具备自举开发能力

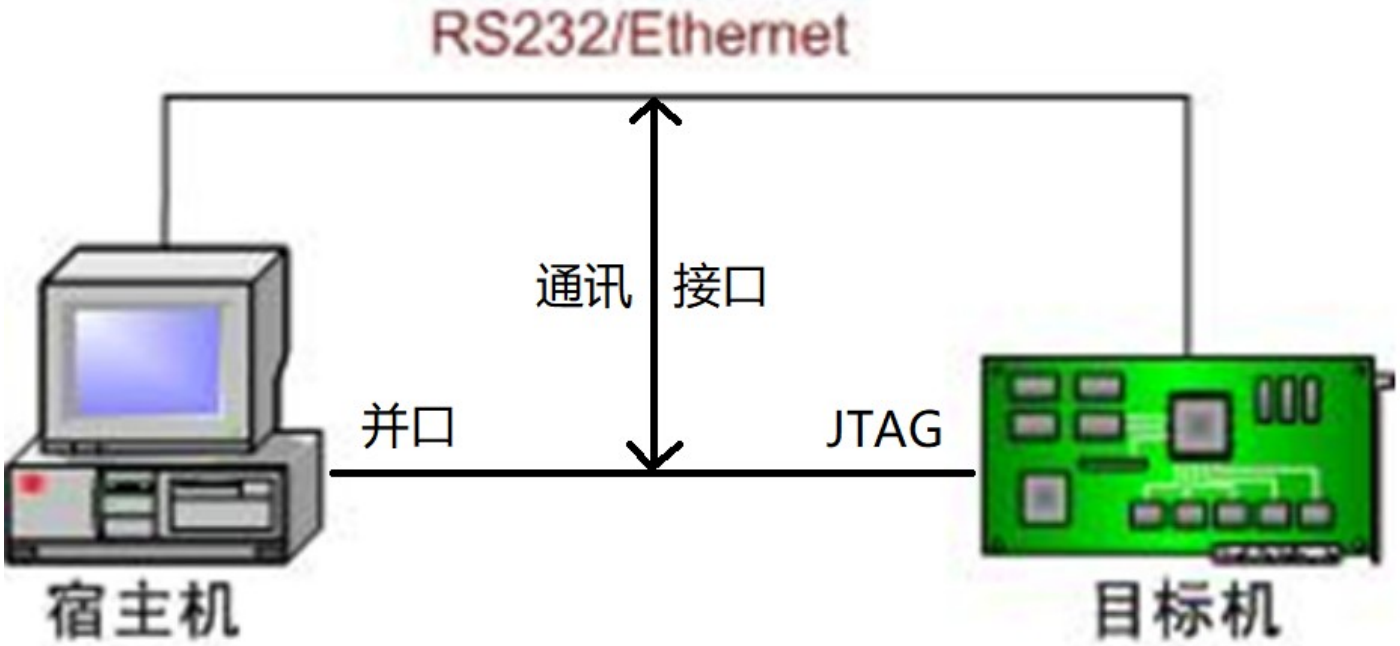
- 由于计算、存储、显示等资源受限，嵌入式系统无法完成 自举开发。

嵌入式软件开发

- 在 PC 机上去开发嵌入式系统，编写嵌入式软件 ，然后通过本地或者交叉编译生成目标代码，
- 将目标代码下载到目标平台上去运行。

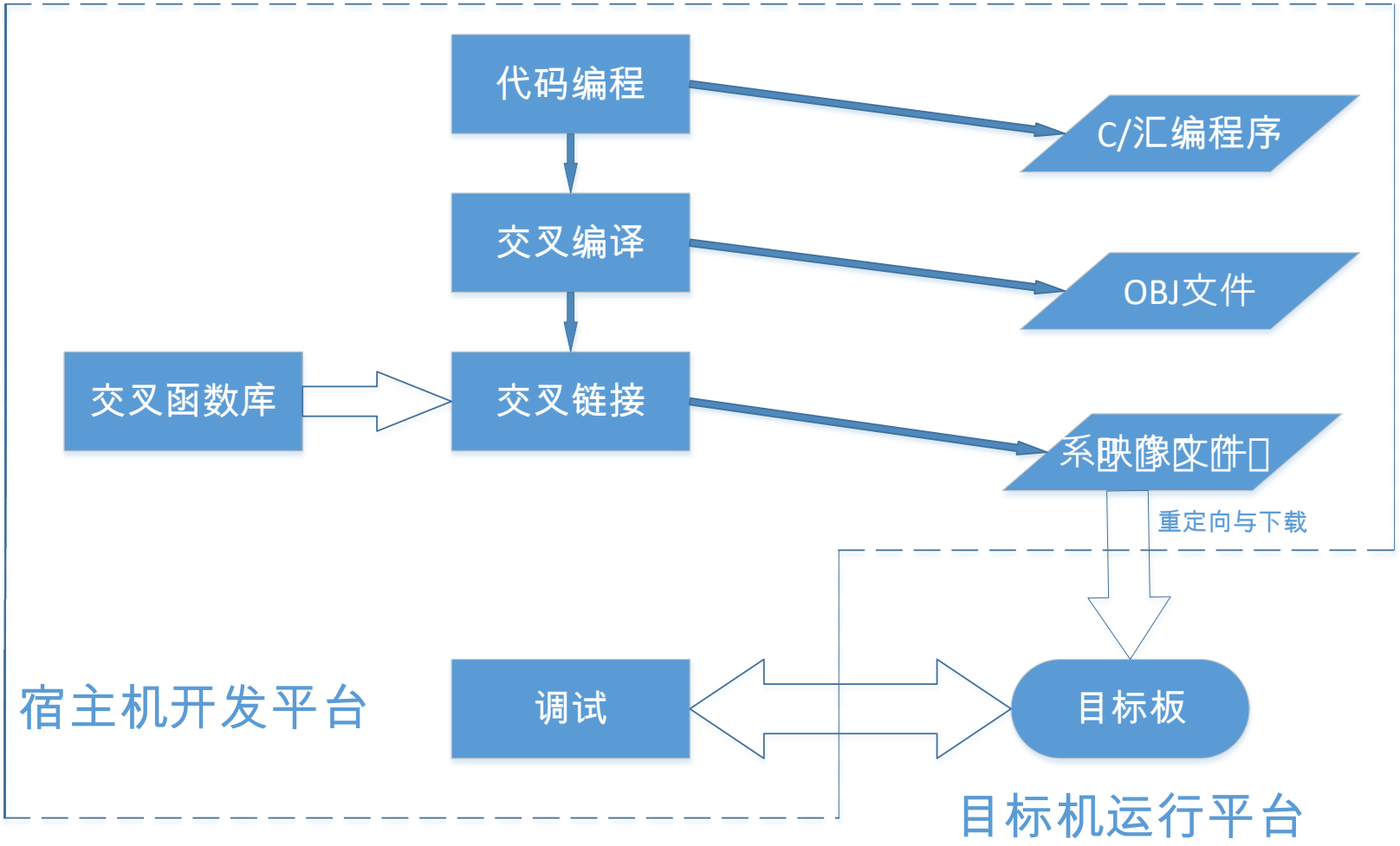
宿主机 - 目标机开发模式

- 嵌入式系统采用双机开发模式：宿主机 - 目标机开发模式，利用资源丰富的 PC 机来开发嵌入式软件。



宿主机：资源丰富　　目标机：资源受限

嵌入式软件开发流程

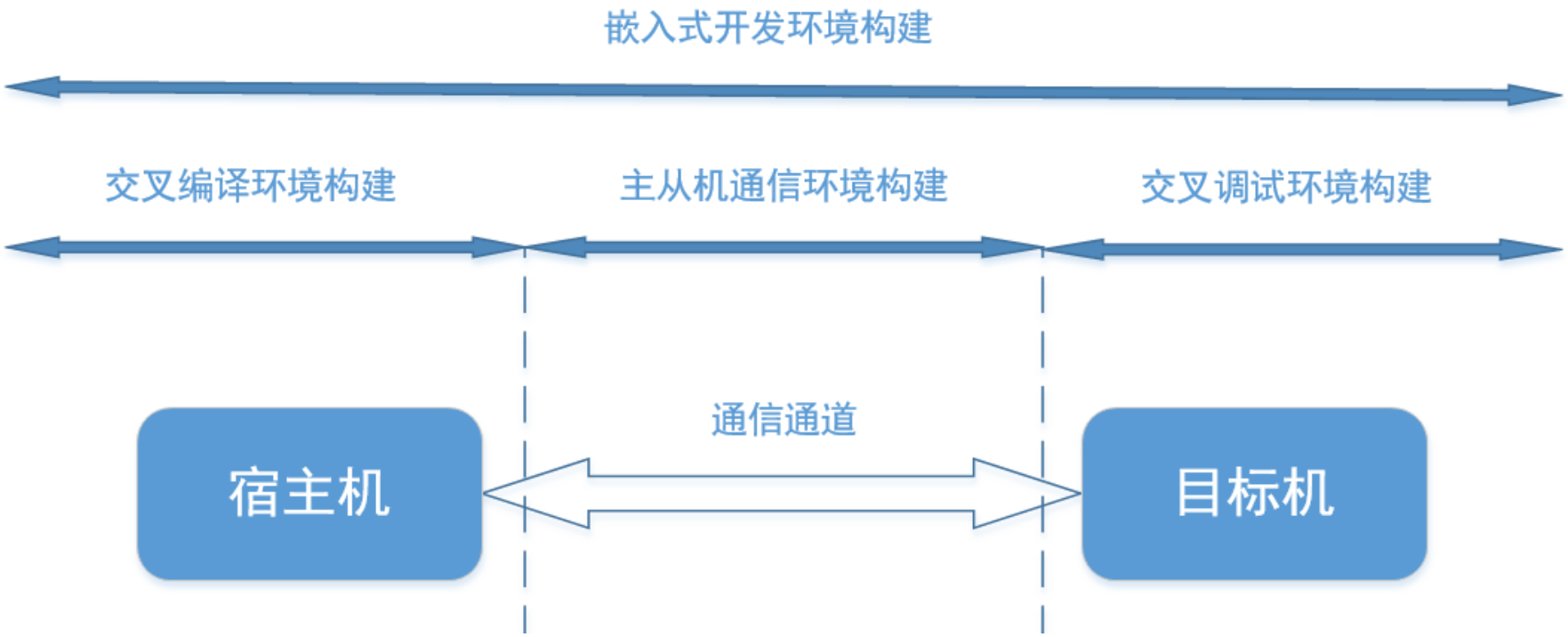

代码编程
C/汇编程序
交叉编译
OBJ文件
交叉函数库
交叉链接
系统映像文件
重定向与下载
宿主机开发平台
调试
目标板
目标机运行平台

交叉开发环境

- 需要交叉开发环境（ Cross Development Environment ）的支持，是嵌入式软件开发的一个显著特点。
- 交叉编译器只是交叉开发环境的一部分，完整的交叉开发环境是指包含交叉编译、交叉链接、交叉调试在内的嵌入式应用软件开发环境。

嵌入式开发环境构建

- 嵌入式开发环境构建



6.2 Boot Loador

Bootloader 概述

- 一个嵌入式 Linux 系统从软件的角度看通常分为 4 个层次：
- Bootloader
- Linux 内核
- 文件系统
- 用户应用程序

- 简单地说 ,Boot loader 就是在操作系统运行之前运行的一段小程序。通过这段小程序，可以初始化硬件设备、建立系统的内存空间映射图，从而将系统的软件硬件环境带到 - 一个合适的状态，以便为调用操作系统内核准备好正确的环境。
- 最终 ,Bootloader 把操作系统内核映像加载到 RAM 中，并将系统控制权交给它

- BootLoader 的特点
  - Bootloader 不属于操作系统，一般采用汇编语言和 C 语言开发。需要针对特定的硬件平台编写。
  - 在移植系统时，首先为开发板移植 Bootloader 。
  - Bootloader 不但依赖于 CPU 的体系结构，而且依赖于嵌入式系统板级设备的配置。

Boot loader 的操作模式

大多数 Bootloader 都包含两种不同的操作模式，即启动加载模式和下载模式，二者的区别仅对开发人员才有意义，从最终用户的角度看， Boot loader 的作用就是加载操作系统。

- 启动加载模式：在这种模式下，Bootloader 从目标机上的某个固态存储设备上将操作系统加载到 RAM 中运行，整个过程并没有用户的介入。
- 下载模式：在这种模式下，目标机上的 Bootloader 将通过串口或网络等通信手段从开发主机（Host）上下载内核映像和根文件系统映像等到 RAM 中。可以被 Bootloader 写到目标机上的固态存储媒质中，或者直接进行系统的引导。也可以通过串口接收用户的命令

常用的 bootloader

- U-Boot

U-Boot (Universal Boot Loader) 是德国 DENX 小组开发的用于多种嵌入式 CPU 的 BootLoader 程序，它可以运行于 PowerPC 、ARM 、 MIPS 等多种嵌入式开发板上。

- vivi

vivi 是由韩国 MIZI 公司开发的专门用于 SAMSUNG ARM 架构的 - 一种 Bootloader

- RedBoot

RedHa t 公司开发的，针对 eCos 操作系统

- BLOB

基于 LART 硬件平台，目前它主要支持 INTEL 的 StrongARM 体系结构和 XScale 结构的 ARM 芯片。

- LILO

Linux 磁盘引导程序

- GRUB

GNU 的 LILO 替代程序

U-boot 特点

- 代码结构清晰、易于移植《见目录结构）。
- 支持多种处理器体系结构《见 arch 目录）
- 支持众多开发板《目前官方包中有 200 多种，见 board 目录）
- 命令丰富、有监控功能
- 支持网络协议、 USB 、 SD 等多种协议和设备
- 支持文件系统
- 更新较活跃，使用者多，有助于解决问题

Boot loader 的启动过程分析

- Boot loader 的启动过程大多是分两个阶段，即 stagel 和 stage2 。
- 依赖 CPU 体系结构的代码通常放在 stage1 中，而且通常用汇编语言实现。
- stage2 中的代码通常用 C 语言实现，这样可以实现更复杂的功能，而且代码会具有更好的可读性和可移植性。

Boot Loader 的 stage1

- 基本的硬件初始化
  - 这是 Boot Loader 一开始就执行的操作，其目的是为 stage2 的执行以及随后的 kernel 的执行准备好一些基本的硬件环境。它通常包括以下步骤（以执行的先后顺序）:
  - 1．屏蔽所有的中断。为中断提供服务通常是 OS 设备驱动程序的责任，因此在 BootLoader 的执行全过程中可以不必响应任何中断。中断屏蔽可以通过写 CPU 的中断屏蔽寄存器或状态寄存器（比如 ARM 的 CPSR 寄存器）来完成。
  - 2．设置 CPU 的速度和时钟频率。
  - 3．RAM 初始化。包括正确地设置系统的内存控制器的功能寄存器以及各内存库控制寄存器等。
  - 4．初始化 LED。典型地，通过 GPIO 来驱动 LED，其目的是表明系统的状态是 OK 还是 Error。如果板子上没有 LED, 那么也可以通过初始化 UART 向串口打印 Boot Loader 的 Logo 字符信息来完成这一点。
  - 5．关闭 CPU 内部指令 / 数据 cache。

Boot Loader 的 stage1

- 为加载 stage2 准备 RAM 空间
  - 为了获得更快的执行速度，通常把 stage2 加载到 RAM 空间中来执行，因此必须为加载 Boot Loader 的 stage2 准备好一段可用的 RAM 空间范围。
  - 由于 stage2 通常是 C 语言执行代码，因此在考虑空间大小时，除了 stage2 可执行映象的大小外，还必须把堆栈空间也考虑进来。一般而言，1M 的 RAM 空间已经足够了。
- 拷贝 stage2 到 RAM 中
  - 拷贝时要确定两点：(1) stage2 的可执行映象在固态存储设备的存放起始地址和终止地址；(2)RAM 空间的起始地址。

Boot Loader 的 stage1

- 设置堆栈指针 sp
  - 堆栈指针的设置是为了执行 C 语言代码作好准备。通常我们可以把 sp 的值设置那个 1MB 的 RAM 空间的最顶端 (堆栈向下生长) 。
- 跳转到 stage2 的 C 入口点
  - 在上述一切都就绪后，就可以跳转到 Boot Loader 的 stage2 去执行了。比如，在 ARM 系统中，这可以通过修改 PC 寄存器为合适的地址来实现。

Boot Loader 的 stage2

- Bootloader 的 stage2 通常包括以下步骤：
  - 1、初始化本阶段要用到的硬件设备
  - 2、检测系统内存映射
  - 3、将 kernel 映像和根文件系统映像从 Flash 上读到 RAM 空间
  - 4、为内核设置启动参数
  - 5、调用内核

6.2.4 常用 Bootloader 简介

任务：了解常用的 Bootloader

表 7-1 开放源码的 Linux 引导程序

Bootloader	Monitor	描述	X86	ARM	PowerPC
LILO	否	Linux 磁盘引导程序	是	否	否
GRUB	否	GUN 的 LILO 替代程序	是	否	否
Loadlin	否	从 DOS 引导 Linux	是	否	否
ROLO	否	从 ROM 引导 Linux 而不需要 BIOS	是	否	否
Etherboot	否	通过以太网卡启动 Linux 系统的固件	是	否	否
LinuxBIOS	否	完全替代 BUIS 的 Linux 引导程序	是	否	否
BLOB	是	LART 等硬件平台的引导程序	否	是	否
U-Boot	是	通用引导程序	是	是	是
RedBoot	是	基于 eCos 的引导程序	是	是	是
Vivi	是	Mizi 公司针对 SAMSUNG 的 ARM CPU 设计的引导程序	否	是	否

6.3.1 嵌入式系统软件

- 嵌入式操作系统 EOS （ Embedded Operating System ）是一种支持嵌入式系统应用的操作系统，它是嵌入式系统的重要组成部分。通常包括与硬件相关的底层驱动软件、系统内核、设备驱动接口、通信协议、图形界面、标准化浏览器等。

- EOS 是相对于一般操作系统而言的，它除具备了一般操作系统最基本的功能，如任务调度、同步机制、中断处理、文件处理等外，还有以下特点：
- ① 可装卸性。开放性、可伸缩性的体系结构。
- ② 强实时性。 EOS 实时性一般较强，可用于各种设备控制当中。
- ③ 统一的接口。提供各种设备驱动接口。
- ④ 操作方便、简单。提供友好的图形界面，追求易学易用。
- ⑤ 提供强大的网络功能。支持 TCP/IP 协议及其它协议，提供 TCP/UDP/IP/PPP 协议支持及统一的 MAC 访问层接口，为各种移动计算设备预留接口。
- ⑥ 强稳定性，弱交互性。嵌入式系统一旦开始运行就不需要用户过多的干预，这就要负责系统管理的 EOS 具有较强的稳定性。嵌入式操作系统的用户接口一般不提供操作命令，它通过系统的调用命令向用户程序提供服务。

- ⑦ 固化代码。在嵌入式系统中，嵌入式操作系统和应用软件被固化在嵌入式系统计算机的ROM 中。辅助存储器在嵌入式系统中很少使用，因此，嵌入式操作系统的文件管理功能应该能够很容易地拆卸，而用各种内存文件系统。
- ⑧ 更好的硬件适应性，也就是良好的移植性。

6.1.2 嵌入式 Linux

- 嵌入式 Linux （ Embedded Linux ）是指对 Linux 经过小型化裁剪后，能够固化在容量只有几百 K 字节或几兆字节的存储芯片或单片机中，应用于特定嵌入式场合的专用 Linux 操作系统。
- 嵌入式 Linux ，免费、软件版权灵活、支持硬件多、开发资源多。
- Linux 最初设计为通用操作系统，其目标是追求整体最佳性能，所以不能很好处理实时任务；其次，如何精简 Linux 适用于嵌入式环境也是一个难题。

6.3.2 嵌入式应用软件开发

6.3.2 嵌入式应用软件开发

- 确定需求
- 确定方案
- 程序编码
- 代码调试
- 交叉编译
- 联调测试
- 打包程序
- 现场试用
- 稳定性测试

确定需求

- 硬件资源：首先确定目标板的硬件资源，包括各种操作和设备，这对内核编译和驱动的开发至关重要。
- 软件资源：然后应确定我们能使用的有哪些软件 资源，如工具链、 IDE 、第三方库等。
- 功能需求：用户都需要哪些功能，在现有的硬件和软件资源情况下能否实现。
- 界面需求：用户有没有界面需求，是彩色液晶还是单色液晶，操作键盘有多少个键，各个键的功能等。

确定需求

- 性能需求：稳定性、响应速度、运行速度、容错性等性能需求。
- 接口需求：对外有多少个硬件接口和软件接口，都有哪些需求。
- 其它需求：如工期、交付物形态等。

确定方案

- 开发工具：我们将使用哪种或哪几种语言进行编程，编程时会用到哪些工具等。
- 开发方法：我们将采用哪种思想或哪种模式进行开发，开发过程需要哪些资源等。
- 程序架构：程序的哪些内容可以做成平台，哪些内容应当实现成接口，哪些内容可以抽象等。
- 程序详细方案：程序功能时序顺序、详细类结构和数据库结构等。

程序编码

- 程序编码其实通俗易懂，就是将详细设计方案实现成代码的过程。但我们仍然应当注意以下几点：
- 切记只使用标准 C/C++ 的库，因为太多的第三方库会让你的程序变得臃肿，这违背了嵌入式的宗旨。
- 面向对象在嵌入式 Linux 开发中可用，但不是到处都非用不可，有些地方不用反而更好。
- 时刻牢记嵌入式软件的可扩展、可裁剪特性。
- 时刻牢记软件的异常处理，如可能发生的指针错误等，但你不可以抛异常，而就忘记录异常
- 你就好熟悉甚至精通 STL 和系统的 API 函数训，并将其用到你的实际工作中，否则你的工作将会事倍功半。

代码调试

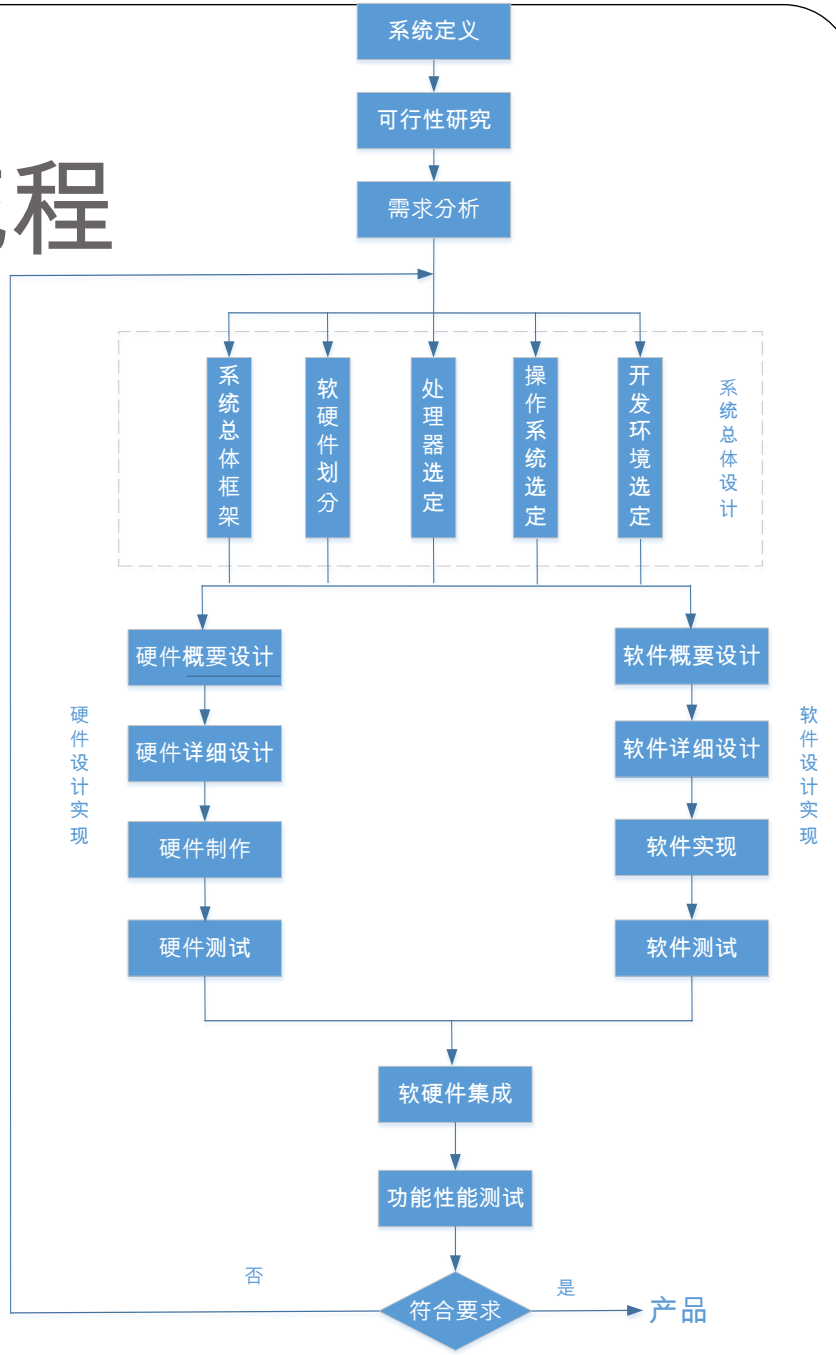
- 代码编写完成后，我们应当进行调试，而且是100% 覆盖率的逐行调度。
- 由于我们只使用标准 C/C++ 进行代码编写，所以我们一般会在开发主机中先对代码进行调度，来排查和编译器无关的问题。
- 使用 IDE ，会让代码调度更加轻松，这比直接使用 GDB 进行调度要友好得多。
- 原则：不放过一个警告，因为警告是潜在的错误。

6.4 嵌入式硬件开发

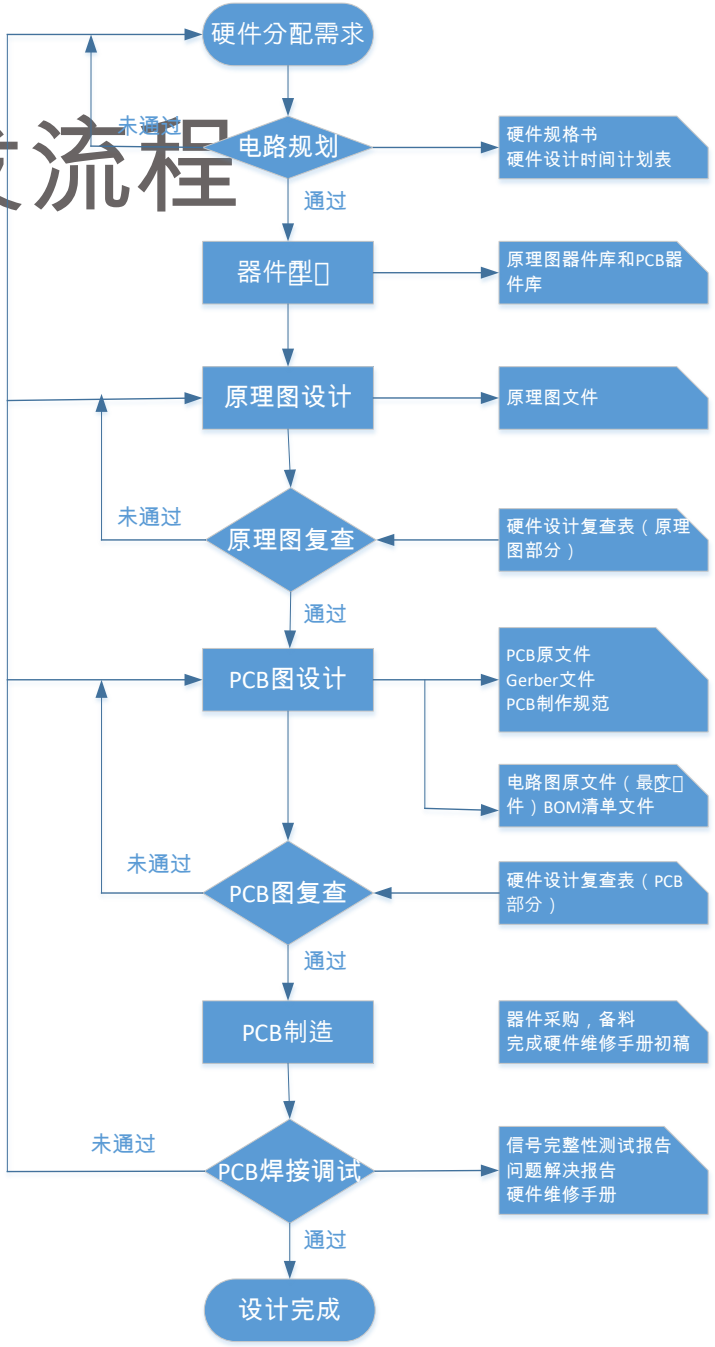
概述

- 嵌入式系统产品都是软硬件的结合体，嵌入式系统开发的最大特点就是需要软硬件综合开发，并且嵌入式软件是针对相应的嵌入式硬件开发的，是专用的。因此嵌入式系统硬件的设计开发是嵌入式系统开发中的一个环节（或组成部分），需要在各个阶段与系统的总体设计相结合，综合考虑，并处理好与软件开发的关系。

嵌入式系统开发流程


系统定义
可行性研究
需求分析
系统总体框架
软硬件划分
处理器选定
操作系统选定
开发环境选定
系统总体设计
硬件概要设计
硬件详细设计
硬件制作
硬件测试
硬件设计实现
软件概要设计
软件详细设计
软件实现
软件测试
软件设计实现
软硬件集成
功能性能测试
符合要求
否
是
产品

嵌入式硬件系统开发流程



总结：硬件系统设计原则

- 要有明确的需求，采用自上而下的方法进行设计
- 采用模块化设计方法，对功能模块进行划分
- 选择器件需要从性能、可靠性、成本等方面进行考虑
- 可靠性设计可根据需要，采用器件参数降额使用或者采用冗余设计等方法来提高和保证。

交叉编译

- 首先编写交叉编译脚本 Makefile ，定义 CPU 架构、代码目录和编译选项。
- 执行 make ，编译程序。
- 排除出现错误，不放过一个警告。因为，警告是隐含的错误，也许将来警告会变成错误。

联调测试

- 这一步又叫集成测试，有两种集成，内部集成和系统集成。
- 内部集成：多进程、多线程之间的联动测试，系统模块之间的联调测试。
- 系统集成：和其它子系统的联调测试，比如自动路测仪应当和近端调度软件 、远端控制软件进行联调测试，保证软件功能和其它 子系统没有歧义。

打包程序

- 准备需要打包的资源，如主程序、动态库、配置文件、脚本和第三方库等。
- 编写打包脚本（开发主机）
- 编写安装脚本（开发主机）
- 测试打包脚本（开发主机）。
- 测试安装脚本（目标板）
- 打包并发布程序和安装 脚本 。
- 有些场合你可能还需要编写一个专用于刷写用户程序的 PC 程序。

现场试用

- 原因及其重要性：前面的过程基本上都是研发人员闭门造车的结果，能否满足用户需求，适应现声情况，还不确定。嵌入式设备一般都需求无故障运行很长一段时间，或者出现故障能够自动恢复，而且现场试用能够发现我们在实验室发现不了的问题。
- 方法：假如现场已有工程点，则挑选一定数量的工程点进行试用。假如 没有工程点，则应模拟实际现场情况进行测试。

稳定性测试

- 稳定性测试是研发人员在进入维护阶段后应该认真执行的事，目的是发现一些细微的问题，或者是测试人员没有发现的问题，并确定程序能否满足长期运行的要求。
- 嵌入式软件的稳定性要求非常之高，想象一下，假如程序不稳定，那么我们的技服人员可能需要每天疲于奔命的去追着公交车跑维护，那会是我们花不起的成本。