

嵌入式系统

An Introduction to Embedded System

第二课 嵌入式系统开发环境构建

教师：蔡铭

cm@zju.edu.cn

浙江大学计算机学院人工智能研究所
航天科技—浙江大学基础软件研发中心

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

Linux本地软件开发模式

1、程序编辑

vi debug.c

```
1 #include <stdio.h>
2 int func(int n)
3 {
4     int sum = 0, i;
5     for(i=0; i<n; i++)
6     {
7         sum += i;
8     }
9     return sum;
10 }
11
12 main()
13 {
14     int i;
15     long result=0;
16     for(i=1; i<=100; i++)
17     {
18         result+=i;
19     }
20     printf("result[1-100]=%d \n", result);
21     printf("result[1-250]=%d \n", func(250));
22 }
```

2、程序编译

gcc debug.c -o debug -g

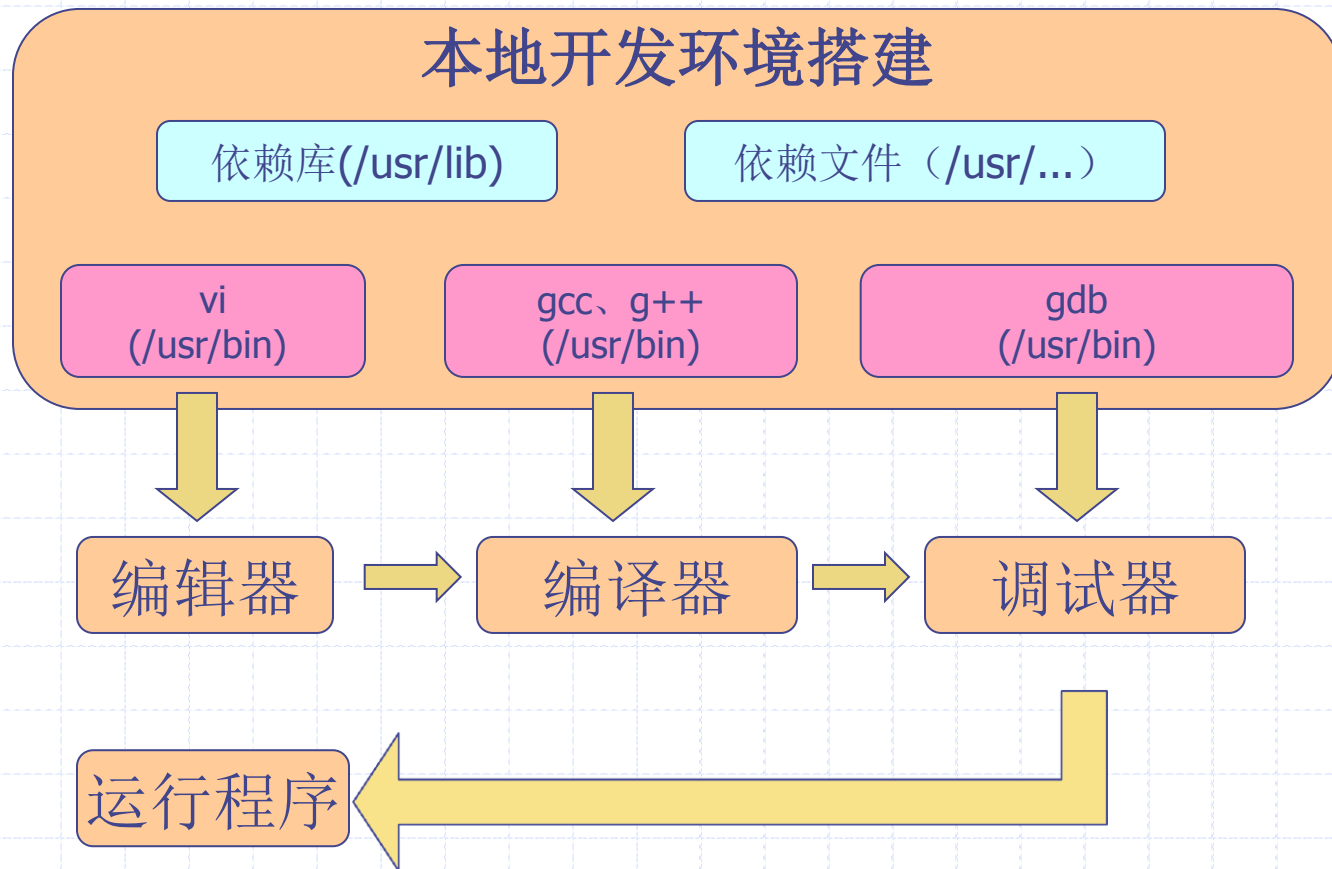
3、程序运行

./debug

4、程序调试

gdb debug

Linux本地软件开发环境



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。

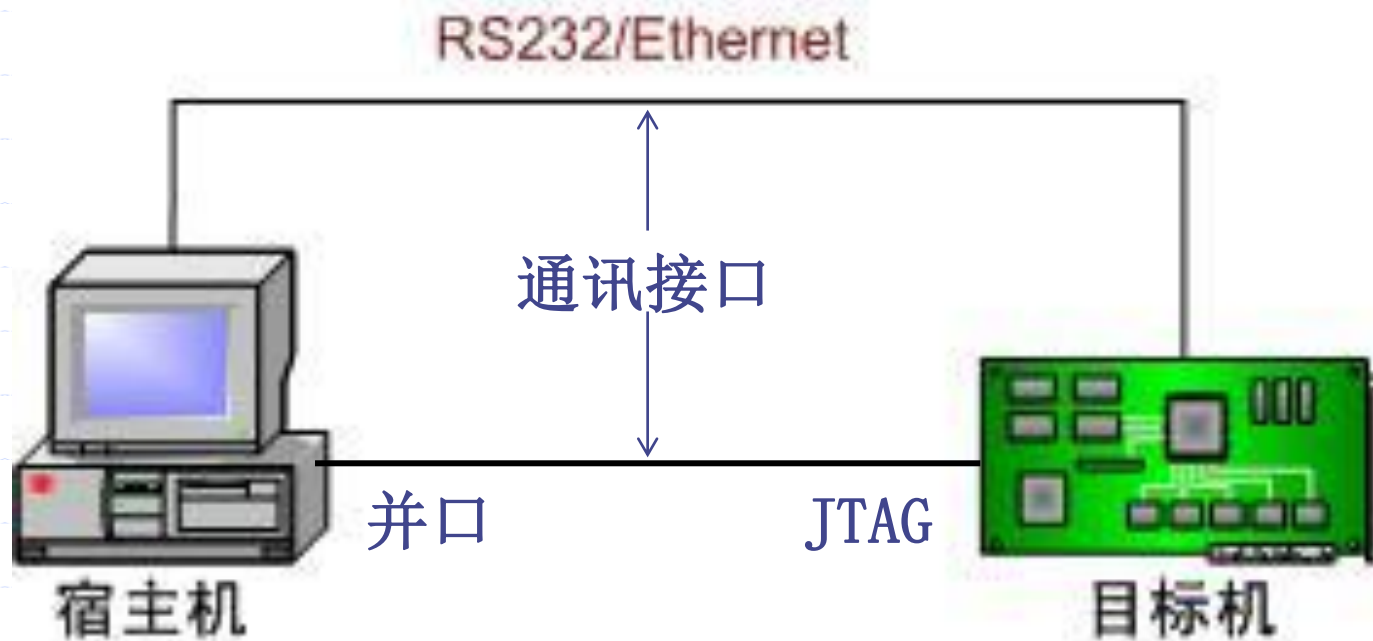


嵌入式软件开发模式

- ◆ 嵌入式系统资源受限，直接在嵌入式系统硬件平台上编写软件较为困难。
- ◆ 解决方法
 - 首先在通用计算机上编写软件
 - 然后通过本地编译或者交叉编译生成目标平台上可以运行的二进制代码格式
 - 最后再下载到目标平台上运行

宿主机—目标机开发模式

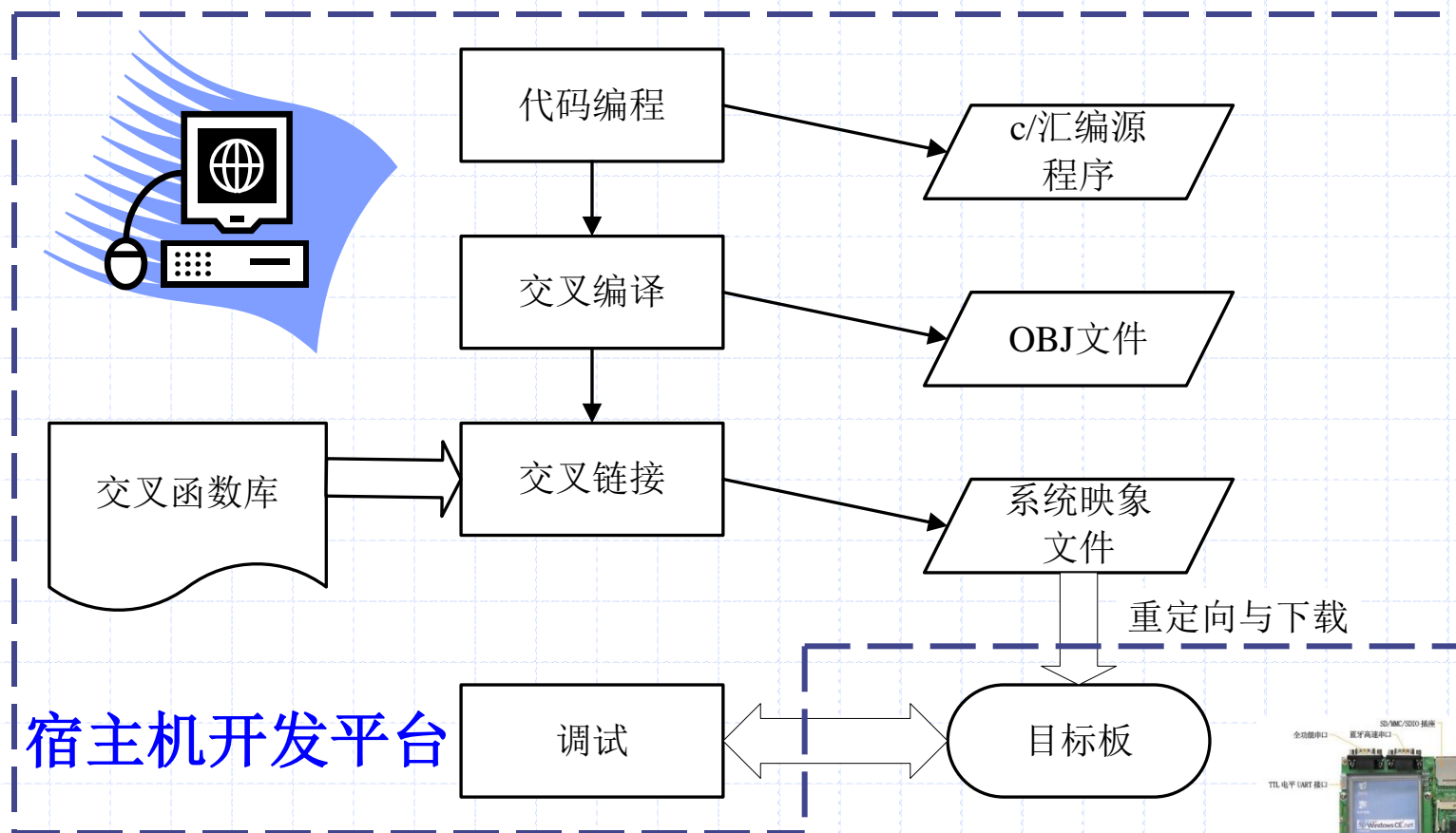
- ◆ 嵌入式系统采用双机开发模式：宿主机—目标机开发模式，利用资源丰富的PC机来开发嵌入式软件。



宿主机：资源丰富

目标机：资源受限

嵌入式软件开发流程



目标机运行平台

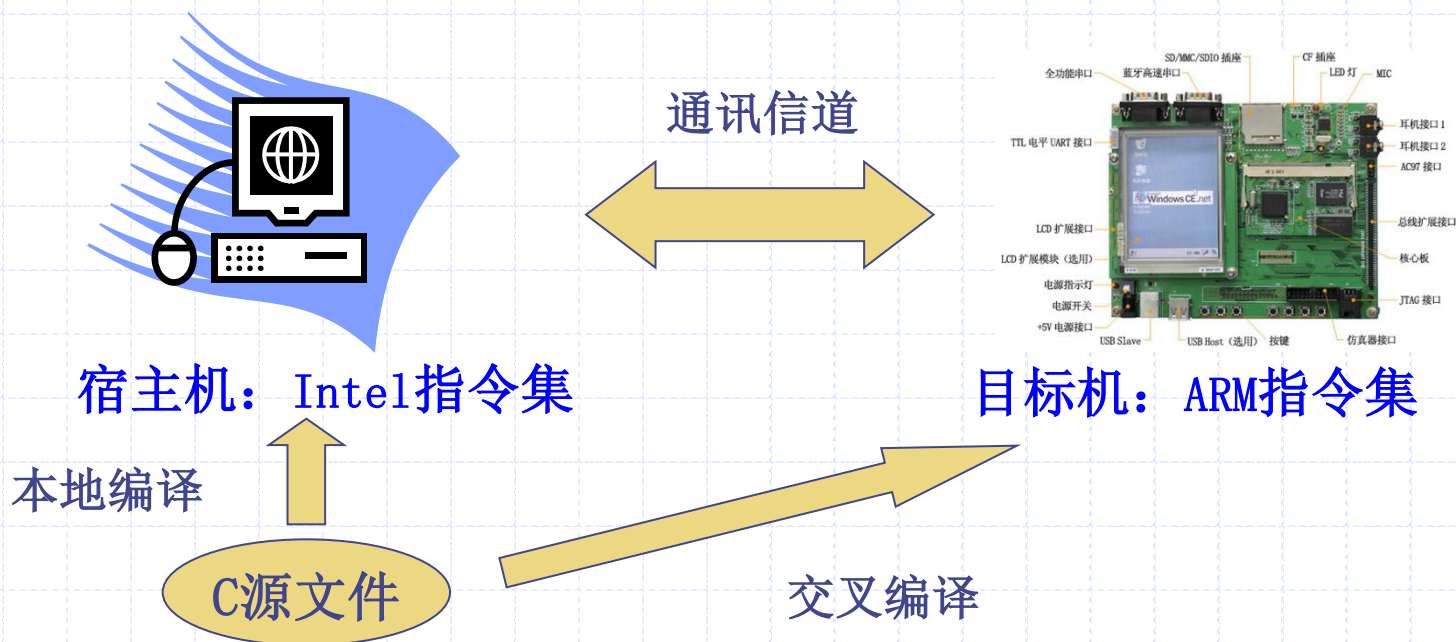
关于交叉编译

◆ 什么是交叉编译

- 在一种平台上编译出能在另一种平台（体系结构不同）上运行的程序；
- 在**PC平台(X86)**上编译出能运行在**ARM**平台上的程序，即编译得到的程序在**X86**平台上不能运行，必须放到**ARM**平台上才能运行；
- 用来编译这种程序的编译器就叫交叉编译器；
- 为了不与本地编译器混淆，交叉编译器的名字一般都有前缀，例如：**arm-linux-gcc**。

交叉编译 VS 本地编译

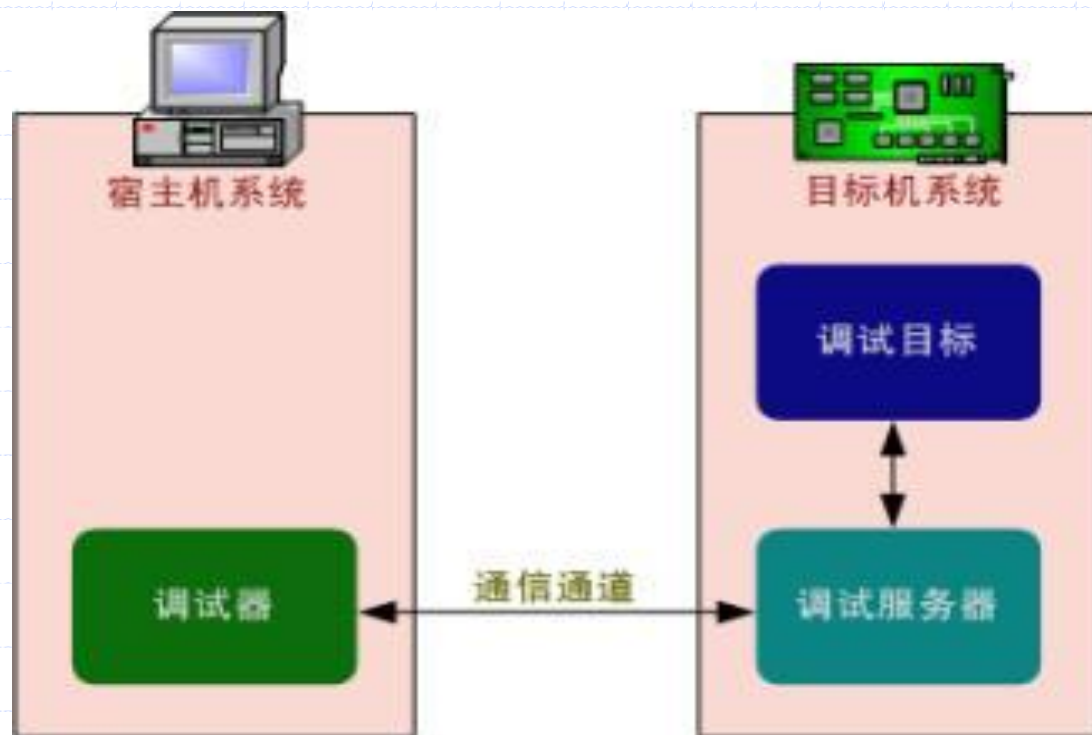
- 交叉编译器和交叉链接器是指能够在宿主机上安装，但是能够生成在目标机上直接运行的二进制代码的编译器和链接器



- 基于ARM体系结构的gcc交叉开发环境中，arm-linux-gcc是交叉编译器，arm-linux-ld是交叉链接器

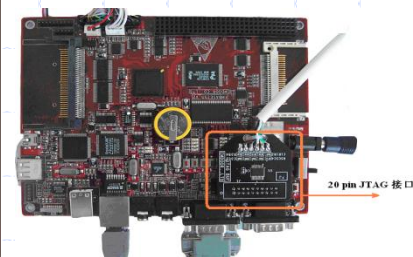
交叉调试概述

◆ 一般而言，嵌入式软件需要交叉调试。



宿主机

目标机



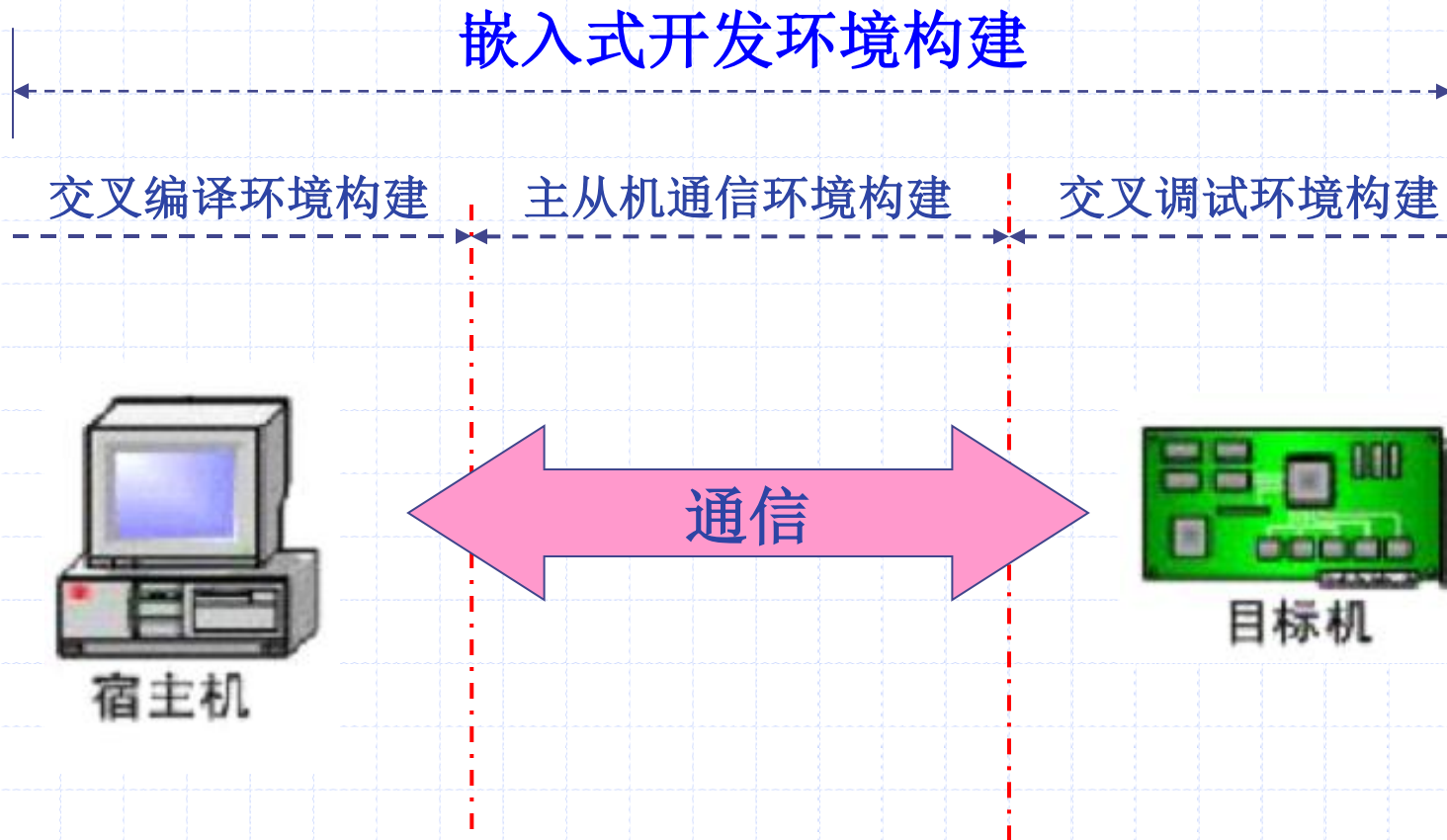
交叉调试 VS 本地调试

| 交叉调试 | 本地调试 |
|---------------------|----------------------|
| 调试器和被调试程序运行在不同的计算机上 | 调试器和被调试程序运行在同一台计算机上 |
| 可独立运行 | 需要操作系统的支持 |
| 被调试程序的装载由调试器完成 | 被调试程序的装载由Loader程序完成 |
| 需要通过外部通信的方式来控制被调试程序 | 不需要通过外部通信的方式来控制被调试程序 |
| 可以调试不同指令集的程序 | 只能调试相同指令集的程序 |




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



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

宿主机端的仿真

◆ 嵌入式应用开发中会出现宿主机操作系统（如**Windows**）与交叉开发环境中要求的宿主机操作系统（如**Linux**）不一致，因此，需要利用虚拟化、仿真化手段建立开发环境，包括：

- **API仿真器：Cygwin、MinGW**
- **虚拟机：Virtual PC、VMWare、Virtualbox**



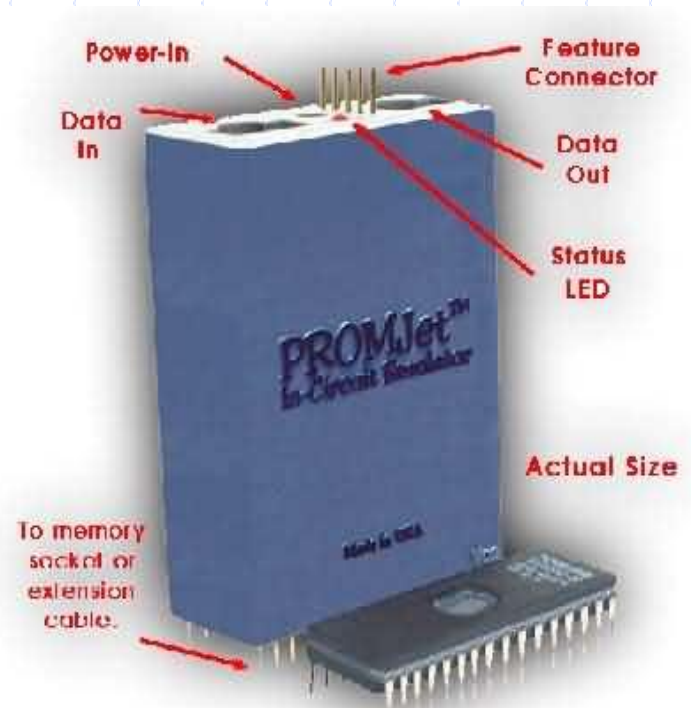
目标机端的仿真

- ◆ 嵌入式应用的开发经常会遭遇缺少目标机环境、缺乏目标机芯片等资源的问题，因此提出了根据不同的应用需要，利用仿真器件、仿真环境进行开发的方法，包括：
- ◆ 硬件仿真开发
 - ROM Emulator
 - ICE
 - OCD
- ◆ 软件仿真开发



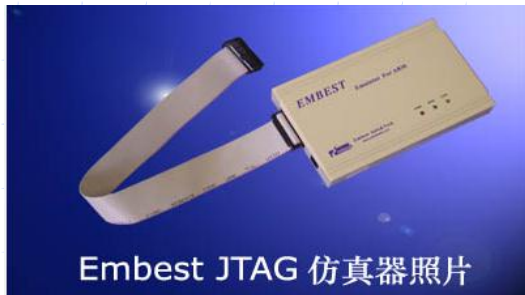
硬件仿真开发——ROM Emulator

- ◆ **ROM Emulator**可用于替代目标机上的**ROM**芯片。利用该设备，目标机可没有**ROM**，但目标机的**CPU**可读取**ROM Emulator**上**ROM**的内容。
- ◆ **ROM Emulator**的**ROM**地址可实时映射到目标机**ROM**地址空间，从而仿真目标机的**ROM**。



硬件仿真开发——ICE

- ◆ **ICE (In-Circuit Emulator)** 是一种用于替代目标机上**CPU**的设备，即在线仿真器。
- ◆ 它比一般的**CPU**有更多的引出线，能够将内部的信号输出到被控制的目标机。
- ◆ **ICE**上的**Memory**也可以被映射到用户的程序空间，即使目标机不存在，也可以进行代码的调试。
- ◆ **ICE**可支持软断点和硬件断点的设置、设置各种复杂的断点和触发器、实时跟踪目标程序的运行等。



Embest JTAG 仿真器照片



硬件仿真开发——OCD

- ◆ **OCD (On Chip Debugging)** 是**CPU**芯片提供的一种调试功能（片上调试），可以认为是一种廉价的**ICE**功能。
- ◆ **OCD**不占用目标机资源，调试环境和最终目标机运行环境基本一致，支持软硬断点、**Trace**功能，可提供精确计量程序的执行时间、时序分析等功能。



VITRA shown here with both debug and trace connections to ARM[™] Integrator/CM-966E-S development board.



软件仿真开发

◆以软件仿真的方式在宿主机上创建一个虚拟的目标机环境，再将应用系统下载到这个虚拟目标机上运行 / 调试

◆仿真精度

- 指令级
- 周期级
- 时序（节拍）级

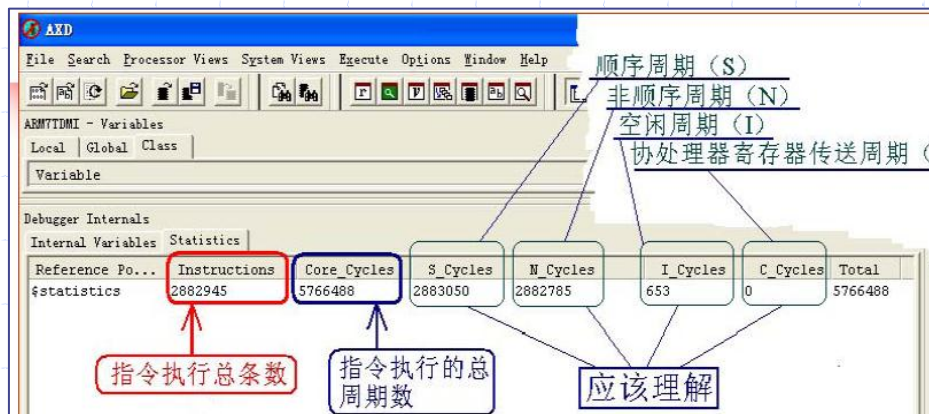
◆仿真能力

- 指令集仿真
- 全系统仿真

How To Write a Computer Emulator:

<http://fms.komkon.org/EMUL8/HOWTO.html>

软件仿真器举例






ARM仿真器Armulator



Android仿真器

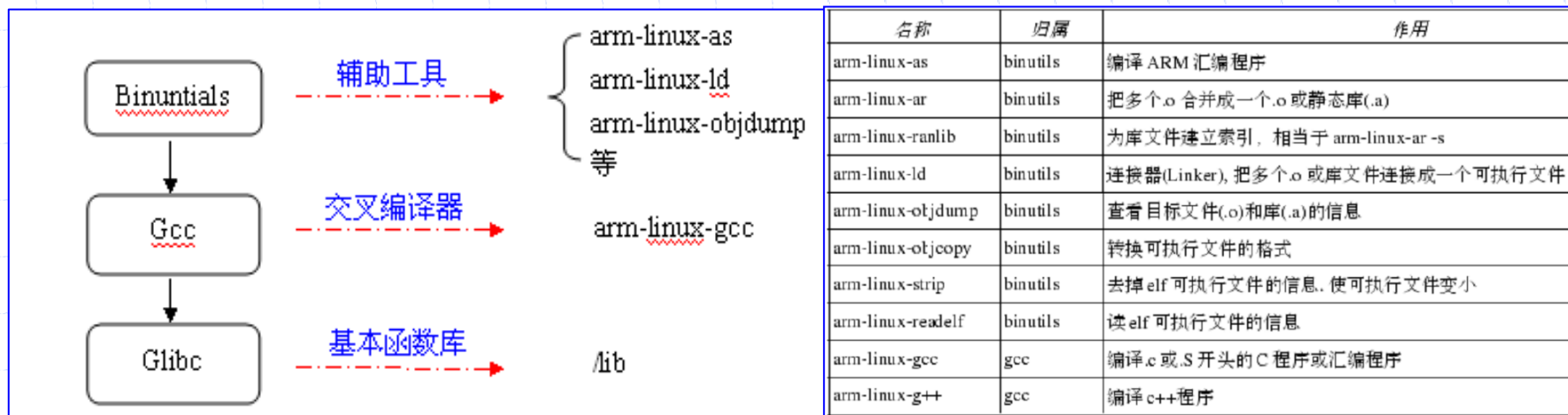
课程大纲

-  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

构建交叉编译环境

◆ 构建交叉编译环境所需的工具链主要包括：

- 交叉编译器，例如**arm-linux-gcc**
- 交叉汇编器，例如**arm-linux-as**
- 交叉链接器，例如**arm-linux-ld**
- 用于处理可执行程序 and 库的一些基本工具，例如**arm-linux-strip**



生成交叉编译器（1/3）

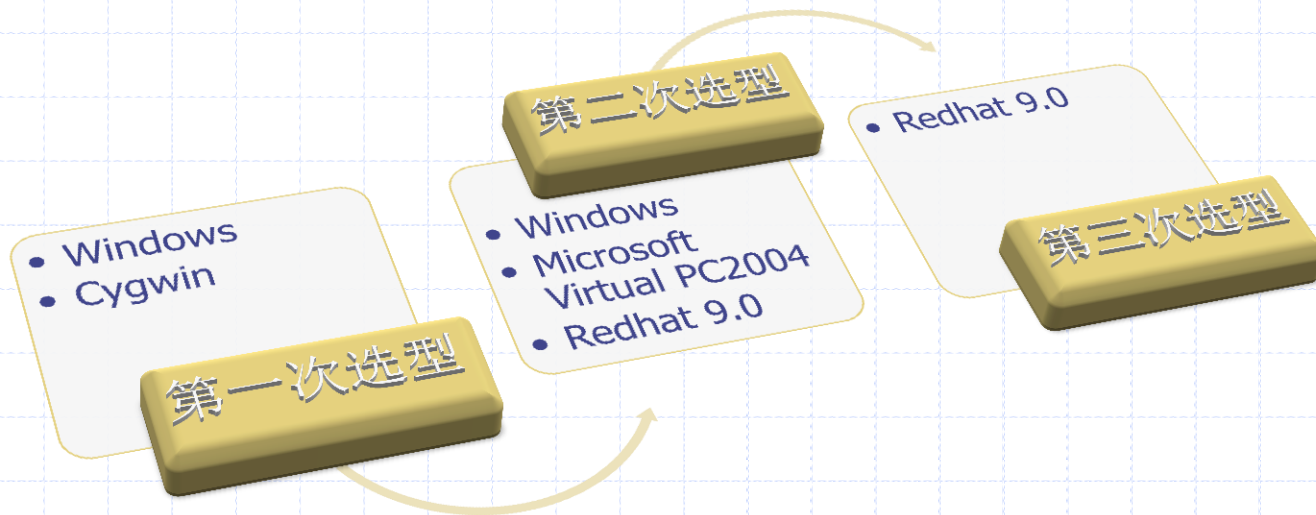
◆ 为什么需要自行生成交叉编译器？

针对特定的嵌入式体系结构，并不一定有现成的交叉编译器，因而，我们不得不使用现有的GCC代码来生成交叉编译器！

生成交叉编译器 (2/3)

◆ 宿主机平台如何进行选择？

- 仿真环境
- 真机环境



SPARC V8交叉编译宿主平台选型实例

生成交叉编译器（3/3）

◆ 交叉编译器的生成过程

- 制作交叉的**binutils**二进制工具
- 制作不带库的**gcc**交叉编译器
- 用制作好的**gcc**交叉编译器，生成所需要的C库（**glibc**、**newlib**、**uclibc**等）
- 重新编译带库的**gcc**，生成完整的交叉编译器

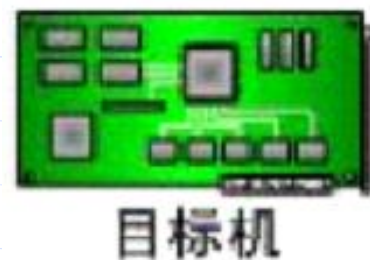
主从机通信环境构建



如何建立通信连接?

如何下载程序?

如何访问宿主机?



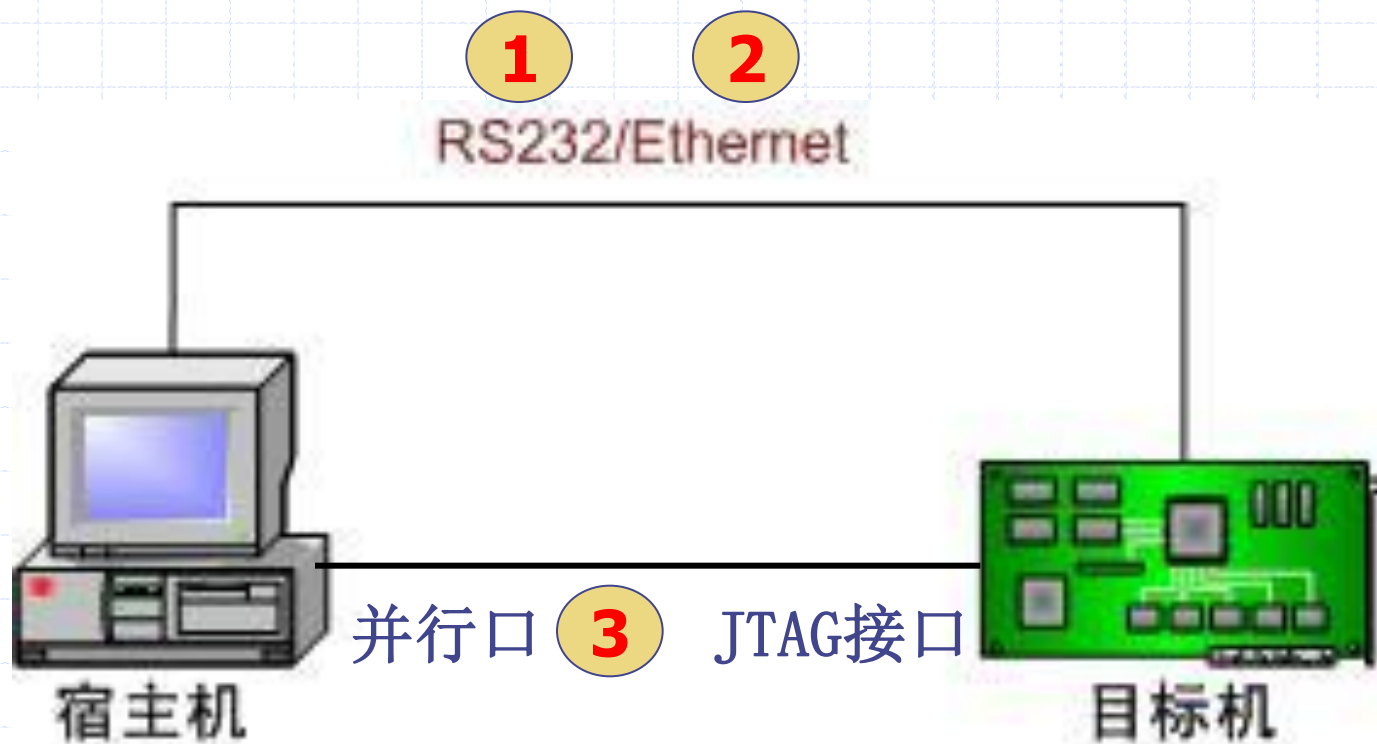
串口

网络

USB

并口

3种常见通信模式



通讯模式1—串口通讯

◆ 特点及应用场合

- 驱动实现最简单
- 传输速度慢，距离短，不适合大数据量、长距离数据传输
- 需要在宿主机、目标机两端均提供驱动
- 常用于宿主机—目标机的字符流通讯

Linux宿主机串口通讯简介—minicom

◆ Minicom对串口数据传输的配置

```
[root@XSBase home]# minicom -s
```

- 若目标机接在COM1上，则输入/dev/ttyS0;若接在COM2上则输入/dev/ttyS1。
- Speed为115200
- Parity bit为No
- Data bit为8
- Stop bits为1

Linux宿主机串口通讯简介—minicom

◆ 设置正确后，目标板启动显示信息如下：



The image shows a terminal window titled 'root@localhost:~'. The menu bar includes '文件(F)', '编辑(E)', '查看(V)', '终端(T)', '转到(G)', and '帮助(H)'. The terminal output displays the following text:

```
XSBASE-R1  
Copyright (C) 2002 Endoor Co.,. Ltd.  
Support: http://www.endoor.com  
  
Autoboot in progress, press any key to stop ..  
Autoboot aborted  
Type "help" to get a list of commands  
XSBASE>
```


通讯模式2—网络通讯

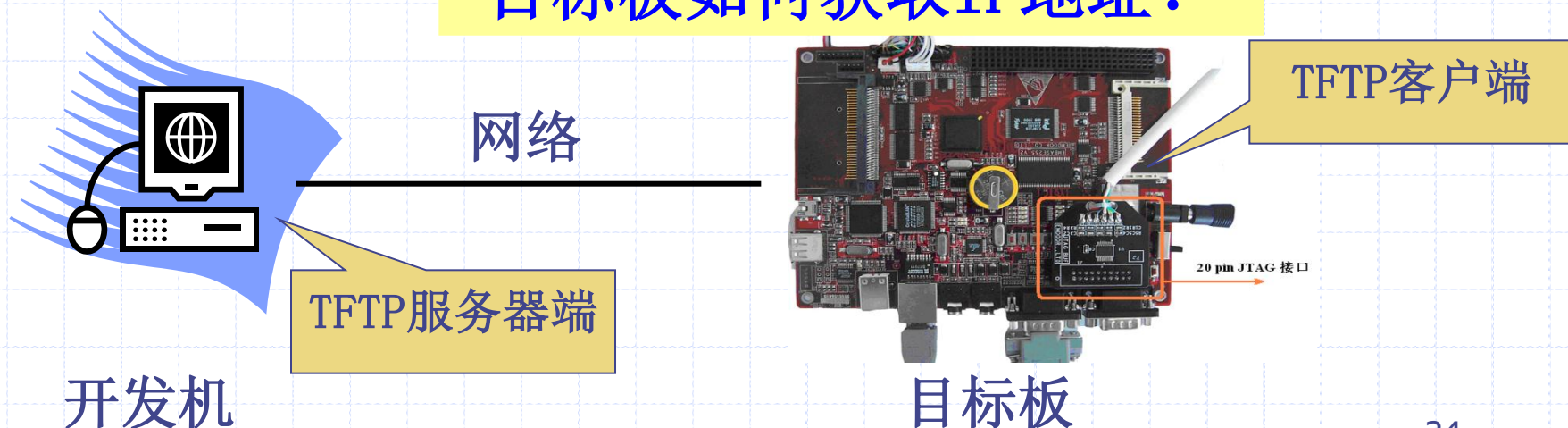
◆ 特点及应用场合

- 驱动实现相对复杂，一般采用精简的网络通讯协议，如TFTP进行通讯
- 常用于宿主机—目标机的大数据量数据传输，可以作为串口通讯的补充
- 需要在宿主机、目标机两端均提供驱动
- 宿主机端实现服务器，目标机端提供客户端

宿主机端网络通讯简介—TFTP协议

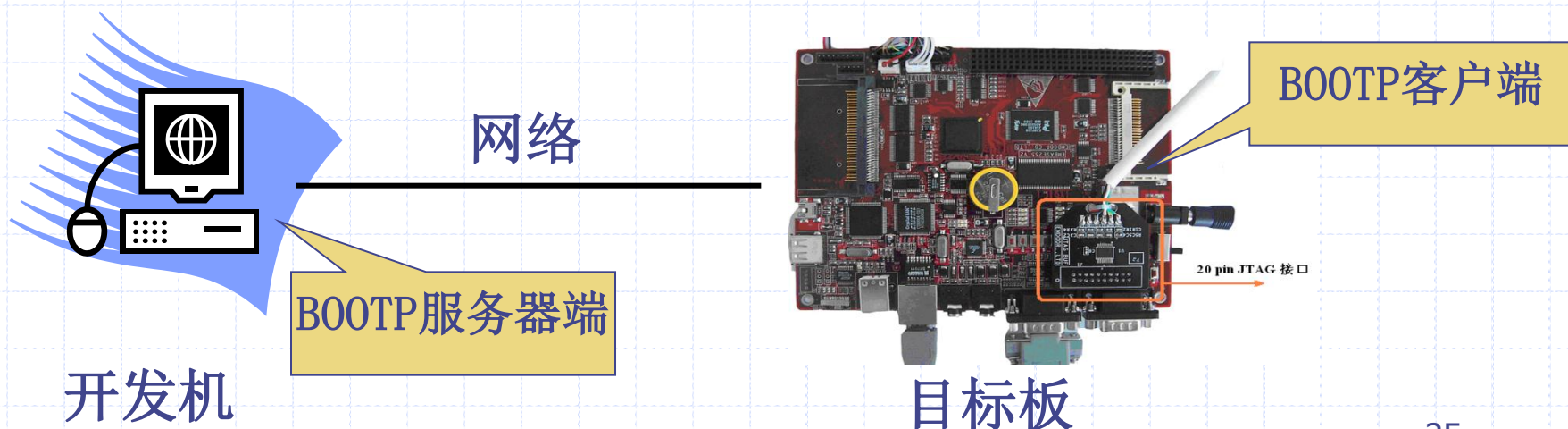
- ◆ TFTP服务的全称是简单文件传输协议（Trivial File Transfer Protocol）
- ◆ TFTP可以看成是一个简化了的FTP
- ◆ TFTP服务器端安装在宿主机，TFTP客户端由目标板实现，目标板需要获取IP地址

目标板如何获取IP地址？



IP地址获取协议简介—BOOTP协议(1/2)

- ◆ BOOTP服务的全称是BootStrap Protocol
- ◆ BOOTP服务是DHCP服务的前身，可以实现客户端IP地址的获取
- ◆ BOOTP服务端使用TCP/IP网络协议中的UDP 67/68两个通讯端口



IP地址获取协议简介—BOOTP协议(2/2)

采用BOOTP协议，目标板获取IP地址过程

- ◆ 在目标板启动BOOTP命令，用广播形式以IP地址0.0.0.0向网络中发出IP地址查询的请求，该请求帧包含客户机的网卡MAC地址信息。
- ◆ 主机平台运行BOOTP服务的服务器接收到请求帧，根据这帧中的MAC地址，在BOOTPTAB数据库中查找MAC的记录，如果没有此MAC的记录则不响应这个请求；如果有就分配IP地址。

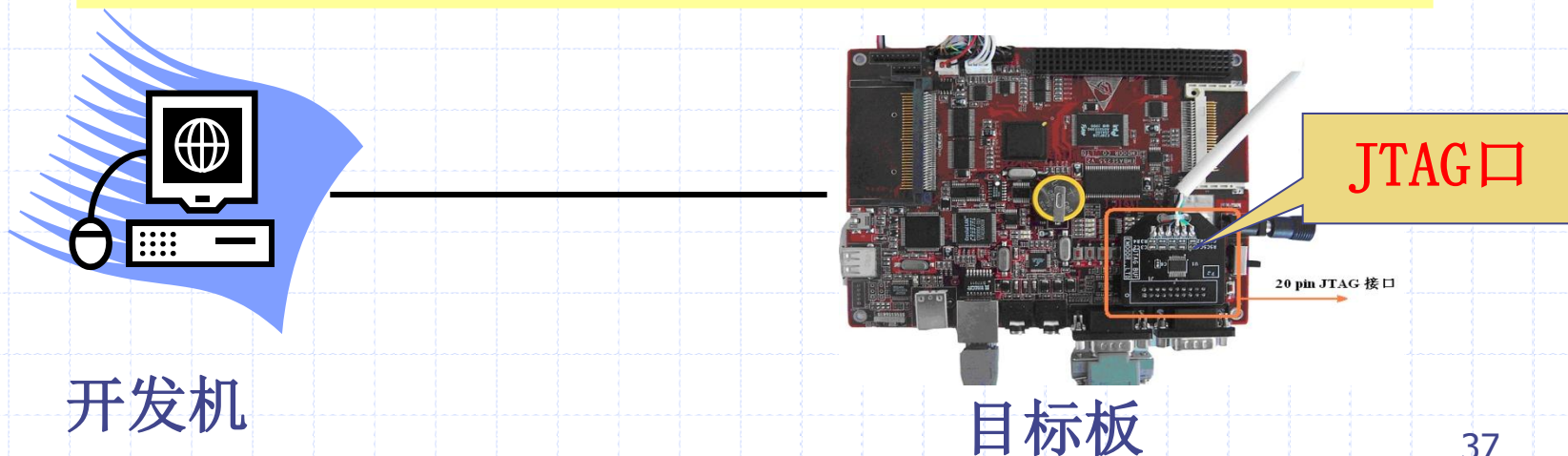
特别提示：目标板的IP地址由开发机分配，两个IP地址需在一个网段

两个问题

问题1：目标板的第一个运行软件如何从开发机传递过去？（串口？网络？）

答案：JTAG，芯片级接口

问题2：第一个运行的软件叫什么？有什么特点？
答案：Bootloader，硬件相关性



通讯模式3 — JTAG简介

- ◆ JTAG (Joint Test Action Group, 联合测试行动小组) 是1985年制定的检测PCB和IC芯片的一个标准。
- ◆ 1990年被修改后成为IEEE的一个标准, 即IEEE1149.1-1990。
- ◆ 通过这个标准, 可以对具有JTAG接口的芯片硬件电路进行边界扫描和故障检测。

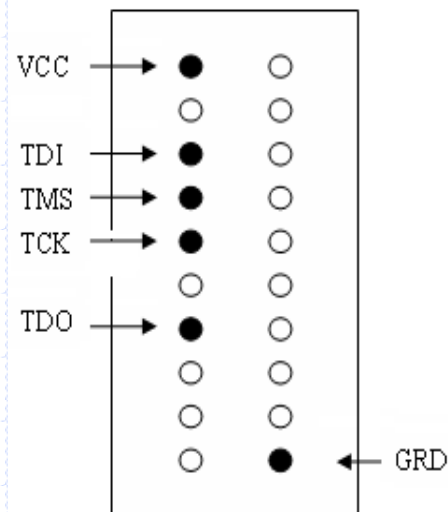
通讯模式3 — JTAG基本原理

◆边界扫描原理

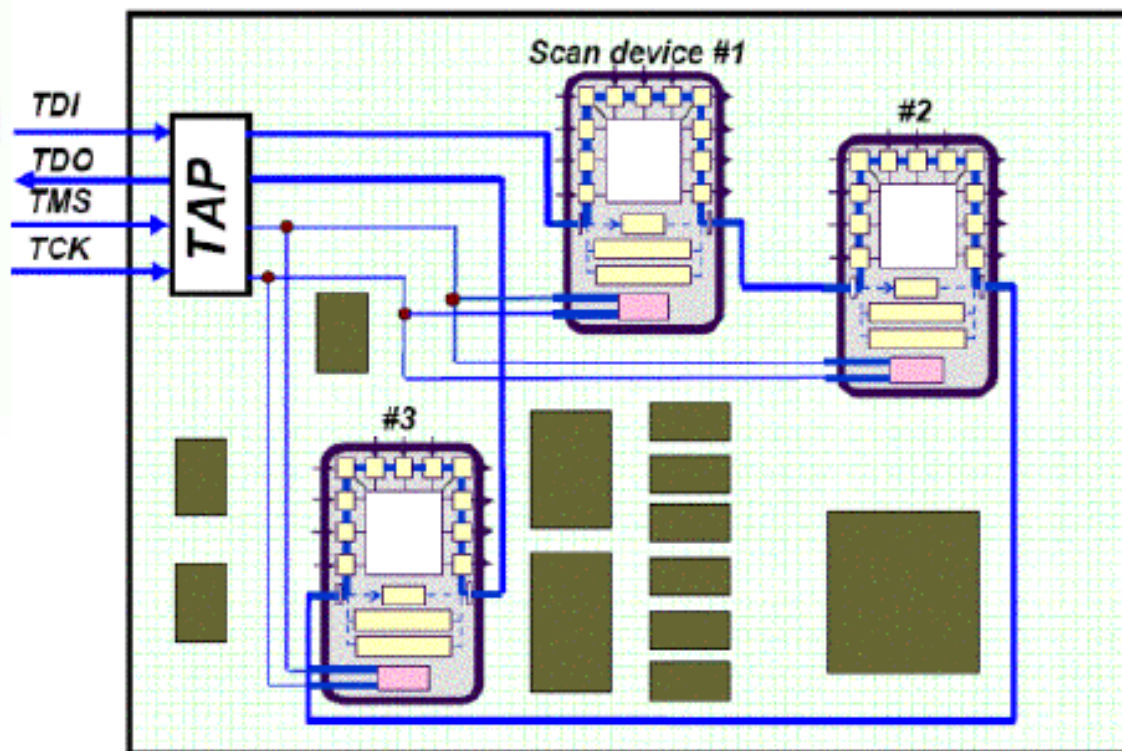
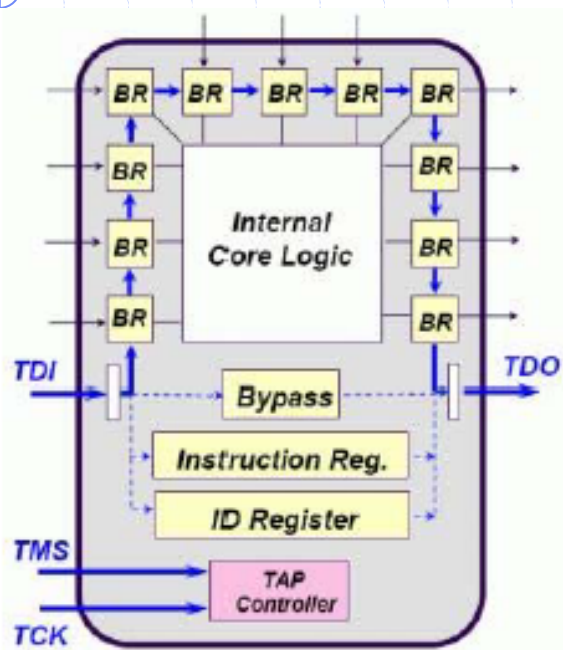
- 在芯片的每一个管脚均增加一个移位寄存器单元，称为**边界扫描寄存器**（BR），这些边界扫描寄存器可以将芯片和外围的输入输出隔离开来。
- 通过边界扫描寄存器单元，可以实现对芯片输入输出信号的观察和控制。
- 对于芯片的输入管脚，通过与之相连的边界扫描寄存器单元把信号（数据）加载到该管脚中去。
- 对于芯片的输出管脚，也可通过与之相连的边界扫描寄存器“捕获”该管脚上的输出信号。

通讯模式3 — JTAG接口介绍

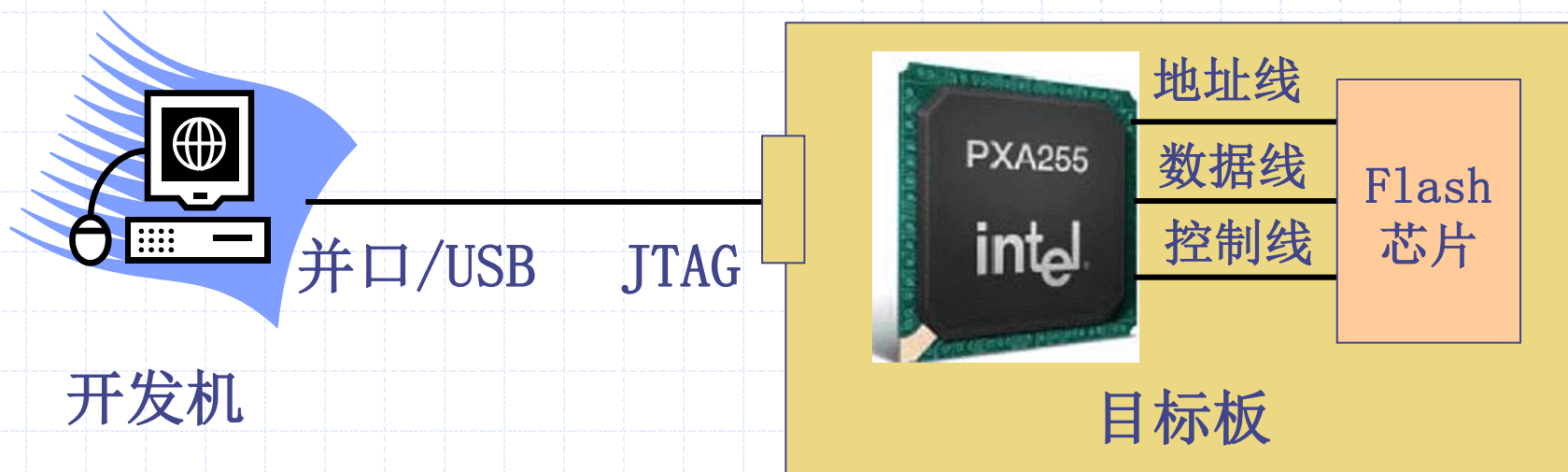
| 引脚名 | 英文全名 | 说明 |
|-----|-----------------|--------|
| TDI | Test Data In | 测试数据输入 |
| TMS | TestMode Select | 测试模式选择 |
| TCK | Test Clock Out | 测试时钟输入 |
| TDO | Test Data Out | 测试数据输出 |



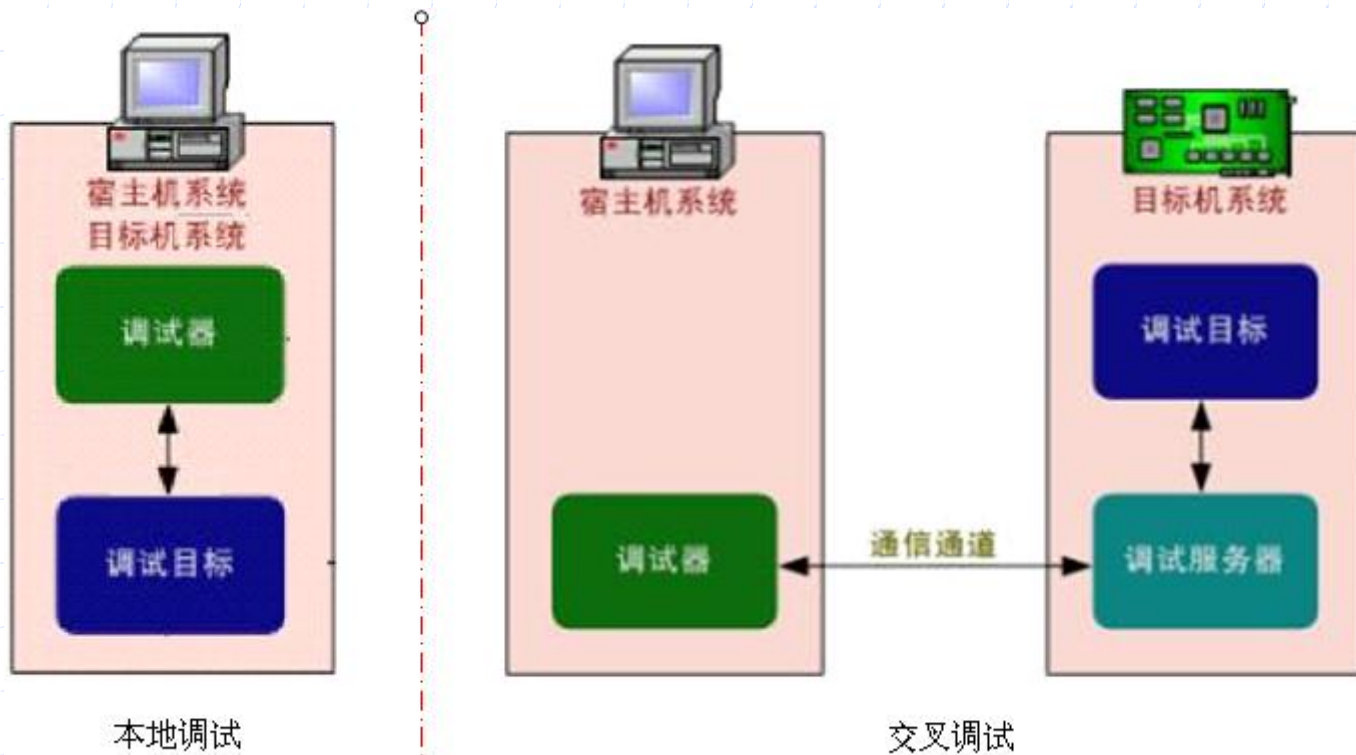
通讯模式3 — JTAG接口应用举例



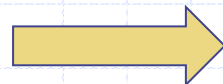
Bootloader烧写—JTAG烧写Flash原理



交叉调试环境构建



GDB本地调试



GDB Server交叉调试

GDB Server交叉调试环境构建 (1/5)

◆ 首先编译宿主机上的GDB调试器

- 解压gdb压缩包
- 运行./configure --target=arm-linux --prefix=/usr/local/arm-gdb -v创建 make 文件, 执行make, make install
- arm-linux-gdb将生成在 /usr/local/arm-gdb/bin

GDB Server交叉调试环境构建（2/5）

◆接着编译目标机上的GDB Server程序：

- 首先进入GDB目录编译，执行export PATH=\$PATH:/usr/local/arm-gdb/bin，然后执行./configure --target=arm-linux --host=arm-linux
- 在gdbserver目录，键入make CC = /usr/local/xsbase-arm-linux-R1/bin/arm-linux-gcc编译用于目标机的stub程序
- 如果没有编译错误gdbserver将生成在gdb/gdbserver目录下

GDB Server交叉调试环境构建（3/5）

- ◆ 建立宿主机和目标板之间的TCP/IP连接
- ◆ 宿主机需与目标板的IP在一个网段
- ◆ 用`arm-linux-gcc -g`编译具有调试信息的代码
- ◆ 下载代码和gdbserver到目标板

GDB Server交叉调试环境构建（4/5）

◆ 建立gdb和gdbserver的连接

- 在目标板上运行gdbserver

```
[root@HYPER255 /root]$ ./gdbserver 192.168.100.216:1234 test  
Process test created; pid = 85  
Listening on port 1234
```

- 拷贝程序到主机平台上的/usr/local/arm-gdb/bin目录下，然后执行arm-linux-gdb。

GDB Server交叉调试环境构建（5/5）

- 连接到开发板

```
(gdb) target remote 192.168.100.50:1234  
Remote debugging using 192.168.100.50:1234  
0x40002980
```

- 如果连接成功的话，将出现下面的信息

```
.....  
Remote debugging from host 192.168.100.216
```




谢谢!

