1.嵌入式系统的定义，特点，分类，典型应用

定义

实际上**“嵌入式计算机系统”**

- IEEE：是用于控制、监视或辅助操作机器和设备的装置，从应用上考虑
- 国内：以应用为中心，以计算机技术为基础，软硬件可裁减，适用于应用系统对功能、可靠性、成本体积等有严格要求的专用计算机系统
- 其他：包含某些较大设备产品中的计算机系统，目的为该设备提供监视和控制服务

包括可编程计算机在内的本身不用做通用计算机的设备

特点

- 形式多样，面向特定应用
- 由多样的处理器和处理器体系结构支持
- 关注成本
- 有实时性和可靠性
- 使用的操作系统一般实行多种处理器、可剪裁、轻量级、可固化
- 开发需要专门工具和方法

分类

按处理器位数：4位、8位、16位、32位(主流)、64位 按应用：信息家电、汽车电子、通信、移动终端、工业控制 按速度：强实时系统(ms、us)，一般实时系统(s)，弱实时系统(s+) 按确定性：硬实时系统(严格要求响应事件、否则或崩溃或错误)、软实时系统 按复杂程度：循环轮询系统、有限状态机系统、前后台系统、单处理器多任务系统、多处理器多任务系统

典型应用

可穿戴设备、通信、工控、航天航空、智能家居、智能驾驶

2.嵌入式系统、IOT、CPS的基本组成。

嵌入式系统

嵌入式硬件和软件

1. 硬件:微处理器为核心集成存储器和系统专用的输入输出设备
2. 软件:初始化代码及驱动、嵌入式操作系统和应用程序有机结合,形成系统特定的一体化软件

从上到下：应用软件、中间件、运行/实时内核、驱动、系统硬件

CPS

CPS，信息物理系统，计算进程和物理进程的统一体，集计算、通信、控制于一体的下一代智能系统，由嵌入式系统、互联网和控制器组成

典型应用是汽车电子的GPS定位系统

 e11c5c1dcf6c208a86977b4db6fad487

IOT

互联网与嵌入式系统发展到高级阶段的融合。是物理设备、联网设备、智能设备和其他嵌入式电子设备之间的互联网络，使这些对象能够收集和交换数据。

组成：传感器、控制器、终端、无线模组、通信网络、应用开发平台

3.嵌入式系统设计

嵌入式系统面临挑战

- 可靠性：如何保证系统可靠工作
- 实时性：如何满足时限要求，如何处理多项功能在时间上协调一致
- 成本：硬件
- 功耗：如何降低
- 易升级：如何设计以保证系统可以升级

传统开发过程

1. 系统在一开始就被划分为软件和硬件两大部分
2. 软件和硬件独立进行行开发设计
3. “Hardware first” approach often adopted

问题

软硬件之间的交互受到很大限制、系统集成相对滞后，从上到下越早开始集成越好

结果

设计质量差、设计修改代价高、研制周期不能有保障

软硬件划分

1. 嵌入式系统的设计涉及硬件与软件部件
2. 硬件和软件具有双重性
3. 软硬件变动对系统的决策造成影响
4. 划分和选择需要考虑多种因素
5. 硬件和软件的双重性是划分决策的前提

由软件实现的部分：

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

双重性部分：

- 算法：加密/解密、编码/解码、压缩/解压。
- 数学运算:浮点运算、FFT。

硬件实现部分：

4.嵌入式硬件系统基础

4.1嵌入式微处理器基础

冯诺伊曼结构与哈佛结构

冯：

数据和程序放在同一个存储单元，统一编址，指令和数据通过同一个总线访问

哈：

数据和程序存储在不同的存储空间中，即程序存储器和数据存储器是两个相互独立的存储器。

系统中设置的两条总线(程序总线和数据总线)

不能使用自修改代码

CISC与RISC

CISC：

复杂指令集：

硬件复杂度高，芯片成本高；指令周期较长；寻址模式复杂，寄存器少；高级语言支持硬件完成

RISC：

精简指令集：

软件复杂度高，芯片成本低；高级语言支持软件完成；寻址模式简单，寄存器较多；指令周期短

流水线技术

- 影响因素：
  - 结构阻滞：指令太多硬件无法满足
  - 数据阻滞：后继指令需要前面的执行结果
  - 控制阻滞：遇到改变PC的指令
- 分支指令影响：
  - 处理器执行跳转工作使流水线顺序指令无效，需要重新形成流水线
- 解决方法：分支预测技术
- 补充：
  - RISC机器用来减少指令周期的一种技术，提高处理器和总线的使用率
  - 提高CPU利用率，并行计算

分类：

- MPU（嵌入式微处理单元）
  - 对应通用计算机的CPU
  - 体积小功耗小成本低
- MCU（嵌入式微控制器）
  - 将主要硬件集成为一个芯片
  - 单片化

- 嵌入式DSP：
  - 专用于信号处理，编译效率和指令执行速度高
- 嵌入式SoC：
  - 绝大多数系统构件在一个系统芯片内

选型

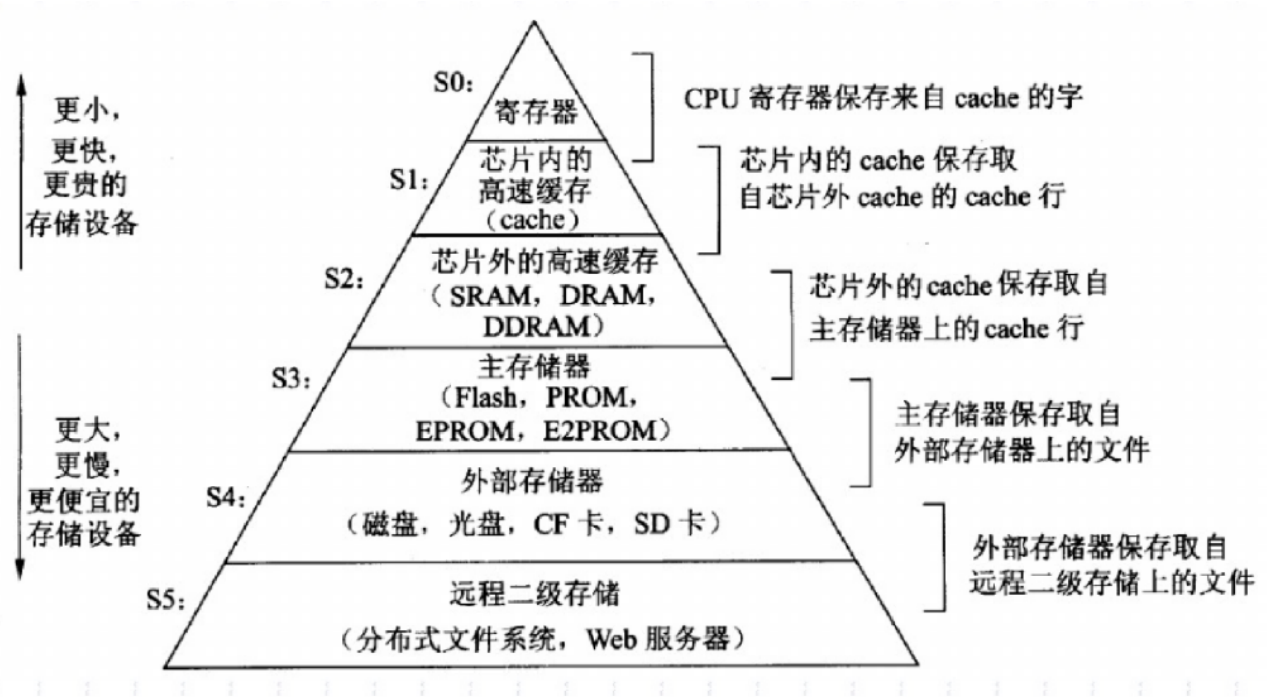
- 有效且经济地满足任务计算需求：
  - 速度、容易升级、单位成本等
- 软件开发工具可用性
- 微控制器广泛可用性和可靠来源

watchdog

- 定义：
  - 定时器，要求程序每隔一段时间向其输出信号，超过规定时间就给MCU发送复位信号
- 作用
  - 检查单片机运行状态，防止受到干扰后跑飞导致崩溃，及时复位
- 原理
  - 看门狗一个接口与单片机一个IO控线引脚相连，开启定时器。。。

4.2存储体系

存储器系统层次结构



ROM种类和选型

- ROM：无法修改
- PROM：一次修改
- EPROM：多次
- FLASH：快闪

FLASH种类和选型

- NOR
  - 成本高，存储代码，坏块处理少
- NAND
  - 成本低，用于存储数据，无法修正

RAM种类与选型

- SRAM：快，贵，小，不刷新
- DRAM：慢点，刷新
- SDRAM

4.3嵌入式总线

总线结构

一组电线加通信协议

常见总线及特点

- 单总线结构
  - 单一系统总线来连接CPU、主存、IO，总线只能分时，吞吐率受限制
- 双总线结构
  - 在主存和CPU之间加一条存储总线，减轻系统总线负担
- 多总线结构
  - 又增加IO总线（多个外部设备与通道传输数据的通路）

输入输出编程

- 忙等IO
  - 用指令检查设备是否就绪，CPU不能同时执行其他，难处理同时发生的IO
- 中断IO
  - 基于子程序调用，下一条指令为子程序调用预定位置，保存返回位置
- 忙等效率低，中断允许设备更改CPU控制流

可编程IO

选择控制寄存器或数据缓存区的三种方法

- 独立的IO端口：需要专门指令完成
- 内存映射IO
  - 优点：控制寄存器看作变量，完全可用C编写IO设备驱动，阻滞用户进程执行IO无需保护机制
  - 缺点：需有选择性禁用功能，增加复杂度
- 混合解决方案
  - 内存映射的IO数据缓存区和独立IO端口的控制寄存器

5.嵌入式软件

5.1软件基础知识

嵌入式软件特点

- 内存有限
- CPU恰好满足要求
- 实时性要求
- 开发流程中执行步骤复杂
- 往往使用交叉编译器

嵌入式软件和桌面软件对比

- 内存受限，影响开发工具和语言的选择
- 运行的操作系统不同
- CPU考虑成本往往恰好满足需求
- 对于实时性有要求，要求可预测性
- 相比桌面软件开发流程不同，前几部较相似单执行步骤复杂·1
- 一般设备开机就开始执行知道关机
- 开发工具不同，往往使用交叉编译器

软件分类

- 系统软件
- 支撑软件
- 应用软件

嵌入式软件体系结构（重要！！！！！）

- 轮询：
    - 无优先级，一切按顺序执行，所有任务时间总和，变更影响大
    - 需要确定所有任务时间
    - 简单，没有共享数据问题

```
void main(void) { while(TRUE) {

if (device_A requires service) service device_A

if (device_B requires service) service device_B

if (device_A requires service) service device_A

if (device_C requires service) service device_C

if (device_A requires service) service device_A

... and so on, making sure high-priority device_A is always serviced on time

} }
```

- 有限状态机

- 非顺序执行，一个状态决定下一个状态
- 优先级有现在的状态决定
- 响应时间：所有任务综合
- 简单，变更影响重大

```
while(1) {
   switch(state) {
      case IDLE:
         check_buttons();
         LEDisplay_hex(NUM1);
         if (BUTTON1 | BUTTON2 | BUTTON3)
            state=SHOW;
         break;
      case SHOW:
         NUM1=0;
         if (BUTTON1) NUM1 += 0x0001;
         if (BUTTON2) NUM1 += 0x0010;
         if (BUTTON3) NUM1 += 0x0100;
         state=IDLE;
         break;
   }
}
```

- 中断轮询
  - 中断优于主循环
  - 响应时间：所有任务加中断执行时间
  - 更改对中断影响小，主循环同轮询
  - 要处理与中断服务程序的共享数据
  - ```
    - BOOL flag_A = FALSE; /* Flag for device_A follow-up processing */
      /* Interrupt Service Routine for high priority device_A */
      ISR_A(void) {
         ... handle urgent requirements for device_A in the ISR,
         then set flag for follow-up processing in the main loop ...
         flag_A = TRUE;
      }
      void main(void) {
         while(TRUE) {
            if (flag_A)
               //中断任务优先
               flag_A = FALSE
               //... do follow-up processing with data from device_A
            if (device_B requires service)
               service device_B
    ```
    ```

```
        if (device_C requires service)
           service device_C
        //... and so on until all high and low priority devices have
been serviced
     }
  }
-
```

- 仅有中断
  - 共享数据处理同上
  - 变化影响同上
  - 响应时间：中断执行时间
  - 中断优先
  - ISR过多会有问题，高优先级中断执行时间过长可能导致错过某些中断
- 函数队列调度
  - 优先级：中断有优先，其他按序执行
  - 变更影响低
  - 共享数据同上
  - 执行时间：最长任务执行时间
  - 优化：添加时间队列、添加任务优先级
  - ```
    #define MAX_TASKS 20
    typedef int(*FuncPtr)();
    FuncPtr tasks[MAX_TASKS]
    int current_task = 0;
    void add_task(FuncPtr func) {
       int n;
       for(n=current_task+1;n<MAX_TASKS-1;n++) {
          if(tasks[n]==NULL) {
             tasks[n]=func;
             return;
          }
       }
       for(n=0;n<current_task;n++) {
          if(tasks[n]==NULL) {
             tasks[n]=func;
             return;
       }
    }

    id display_task() {
       LEDisplay_hex(NUM1);
       add_task(button_task);
    }
    void button_task() {
    ```
    ```

```
    check_buttons();
    NUM1=0;
    if (BUTTON1) NUM1 += 0x0001;
    if (BUTTON2) NUM1 += 0x0010;
    if (BUTTON3) NUM1 += 0x0100;
    add_task(display_task);
}

main() {
    LEDisplay_init();
    LEDisplay_clear();
    init_buttons();

    add_task(button_task);

    while(1) {
        if(tasks[current_task]==NULL) {
        ;
        }
        else {
            (*tasks[current_task])();
            tasks[current_task]=NULL;
        }
        current_task++;
        if(current_task>=MAX_TASKS) current_task=0;
    }
}
```

5.2操作系统基础知识

RTOS概念、特点、选型

- 概念：对外来事件能在限定时间内做出预定质量处理的计算机系统
- 特点
  - 实时性、可靠性、可裁剪、可预测性、可扩展性、抢占式内核、紧凑
- 选型
  - 首先确定是否使用RTOS（系统对延迟时间有要求、对事件处理复杂度有要求、对RAM、ROM有限制）
  - 再根据成本和可靠性、实时性、工具链、模块丰富程度、RAM、ROM占用量等

任务管理

进程、线程、任务的概念

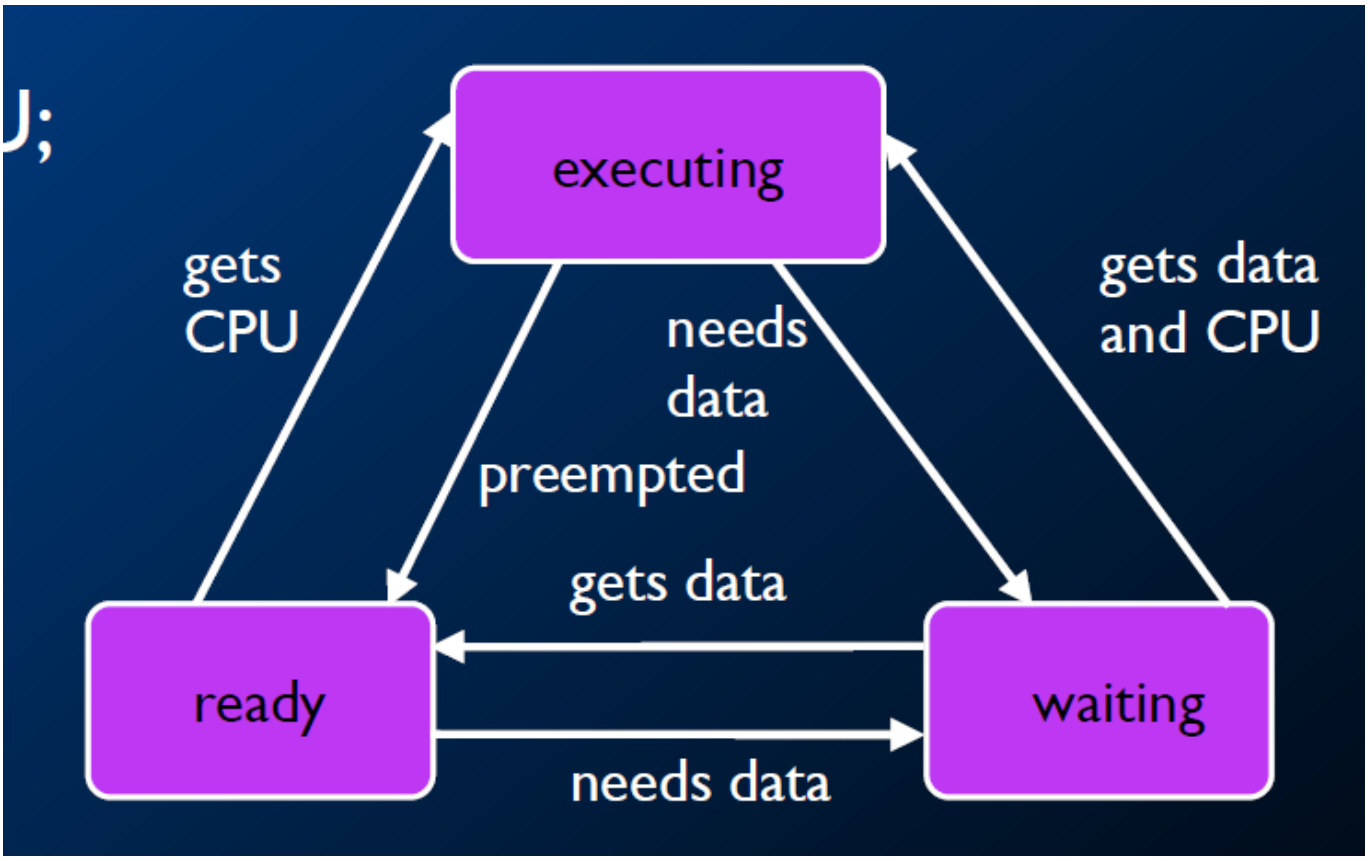
- 进程：操作系统资源分配的基本单位，程序关于某数据集合的一次运行活动
- 线程：处理器调度和执行的基本单位
- 任务；最抽象，软件完成一个活动

任务控制块

描述一个任务核心数据机构

包括堆栈指针、任务状态、优先级等，任务创建时创建TCB表

任务状态及转换



任务调度

- 可抢占调度
  - 系统将处理机分配给优先级最高的进程，更好满足紧迫作业的要求。
- 不可抢占调度
  - 系统将处理机分配给优先级最高的进程后，该进程一直执行下去
- 先来先服务
  - 处理机分配给最先来的进程
- 时间片轮转算法：
  - 先将进程按照先进先出原则排列成一个队列，让首进程在CPU运行一个时间片的时间(通常10-100ms)后，将该进程放置在队列末尾
- 优先级算法
  - 系统将处理器分配给就绪队列中优先数最高的进程(分为抢占式优先级算法和非抢占式优先级算法)

调度算法题（重要！！！！）

- RMS
  - 固定优先级、所有任务独立
- EDF算法题

任务间通信

- 共享内存、小心、管道、信号

- 使用邮箱和消息队列
- 管理核心：事件控制块

同步与互斥

- 同步
    - 通过开关中断实现
- 互斥
    - 使用信号量
    - 线程通过信号量机制访问共享资源时，它们需要使用互斥对象以确保数据完整性。
    - 互斥危害：优先级反转
        - 当高优先级访问共享资源时该资源已被低优先级任务占有，因而造成其被许多低优先级任务阻塞
        - 解决方式1:优先级继承协议
            - A任务申请共享资源时发现优先级较低的C使用，将C的优先级提升到自身，当C使用结束后恢复其原有优先级
        - 解决方式2:优先级天花板
            - 当任务申请某资源时，把**该任务的优先级提升到可访问这个资源的所有任务中的最高优先级**

临界区

- 定义
    - 访问共有资源的程序片段，而这些共有资源无法被多个进程锁访问
- 实现
    - 信号量PV操作（实现互斥访问、行为同步）、中断
    - 处理临界段代码时关中断，处理完毕开中断

嵌入式系统静态和动态内存管理

固定分区的存储管理方法OSMemCreate，OSMemGet， OSMemPut

1. put时必须放入get时去除的内存块
    1. Heap1：不支持free，简单的数组分配，适用于确定性任务
    2. Heap2：简单的free，会导致碎片，适用于每次申请和释放内存大小相同的任务
    3. Heap3：调用了malloc和free,通过挂起调度器达到了线程安全性

6.嵌入式实时内核

ucOS-II如何从64位系统位扩展到256位

- 修改`os_cfg.h`中`#define OS_LOWEST_PRIO 63 -> 254`
- ucos-II可以自动根据`OS_LOWEST_PRIO`的大小判断TBL大小和算法，不需要修改，对于旧版，则`ucos_ii.h`中修改TCB表的大小(8->16)，在`os_core.c`中个修改算法
    - 将16位修改成低8位和高8位：若低8位为0，则在高8位，反之则为低8位，找到高低之和，再进行进一步判断

    - 总之将8组 * 8个/组->16组 * 16个每组，高八位计算按位运算模式使用Map，并在最后计算优先级时+8再算。

- 代码如下

```
static void OS_SchedNew (void)
{
#if OS_LOWEST_PRIO <= 63//μC/OS-II v2.7之前方式
  INT8U y;
  y = OSUnMapTbl[OSRdyGrp];
  OSPrioHighRdy = (INT8U)((y << 3) + OSUnMapTbl[OSRdyTbl[y]]);
#else
  INT8U y;
  INT16U *ptbl;
  //OSRdyGrp为16位
  if ((OSRdyGrp & 0xFF) != 0) {
    y = OSUnMapTbl[OSRdyGrp & 0xFF];
  } else {
    y = OSUnMapTbl[(OSRdyGrp >> 8) & 0xFF] + 8;//矩形组号y>=8
  }
  ptbl = &OSRdyTbl[y];//取出x方向的16bit数据
  if ((*ptbl & 0xFF) != 0) {
    OSPrioHighRdy = (INT8U)((y << 4) + OSUnMapTbl[(*ptbl & 0xFF)]);//*16
  }
  else {
    OSPrioHighRdy = (INT8U)((y << 4) + OSUnMapTbl[(*ptbl >> 8) & 0xFF] + 8);
  }
#endif
}
```

一些原理

- 开始是所有TCB位于空闲TCB链表（单向），有任务加进来把指针指向的TCB赋予它，加入使用链表（双向）

- **任务就绪表**

    - 有OSRdyGrp和OSRdyTbl【】，前者的某一位代表该优先级（8个）是否有任务就绪

    - OSRdyTbl【】中(0, 0)是优先级最高的任务，(7, 7)是优先级最低的

    - 根据优先级确定就绪表（非常简单）

    - 可以根据就绪表确定最高优先级（高三为）

    - OSUnMapTbl[256]优先级判定表，枚举产生

任务调度

- μC/OS-II可抢占式，总是运行最高优先级

- 任务调度所花时间与建立的任务数目无关
- 不支持时间片轮转
- 确定那个任务优先级最高、选那个任务执行有调度器完成

任务的栈空间

静态分配：在编译的时候分配

动态分配：在任务运行的时候使用malloc()函数来动态申请内存空间

可能出现：内存碎片问题

中断和时钟

由于某种事件的发生而导致程序流程的改变

条件

- 至少一个中断源发出中断信号
- 系统允许中断，未屏蔽中断信号

时钟节拍

特殊中断

徐用户提供周期性信号源，实现时间延时和确认超时

频率取决于用户程序精度

用户必须在多任务系统启动后开启时钟节拍

存储管理

- 是实模式存储管理
  - 不划分用户空间和内核空间
  - 任务只有线程，只有运行上下文和栈是独享的，其他都是共享
- 采用固定分区的存储管理
- 在ANSI C中可以用malloc()和free()两个函数动态地分配内存和释放内存。在嵌入式实时操作系统中，容易产生碎片。
  - 改进了算法，使得它们可以分配和释放固定大小的内存块。这样一来，malloc()和free()函数的执行时间也是固定的了

7.建模

有限状态机

- 定义
  - 用于设计计算机程序的行为模型，由与转换相关的有限个状态组成。转换就是一组从一个状态开始并在另一个状态结束的动作，有触发器启动，触发器可以是事件或条件
- 应用：
  - 自动售货机、交通信号灯、电子游戏、文本解析、正则表达式匹配、CPU控制器、、、

补充问题

为什么嵌入式软件要交叉开发？比较宿主机和目标机的差异

为了满足代码编写编译调试与运行环境有不同要求的情况。嵌入式系统是面向特定应用的一体化软件，不具备软件编写编译等功能，这么做可以降低嵌入式系统复杂度，降低成本 差异: 宿主机是通用计算机系统，而目标机则多为嵌入式系统 宿主机具有完整的开发软件如IDE、Complier、Debugger等，用于软件的编写，生成的二进制可执行代码则通过串口传输，在专用的面向应用的目标机上运行，并采取交叉方式进行调试 过程 宿主机上开发，模拟调试 通过串口或网络传到目标机上 目标机上基于监视器和操作系统的调试 目标脱离宿主机运行 特征 通用计算机 嵌入式系统 形式与类型 看得见的计算机、按照其体系结构、运算速度和结构规模等因素分为大中小型机和微机 看不见的计算机，形式多样，应用领域广泛，按应用来分 组成 通用处理器、标准总线和外设，软件和硬件相对独立 面向应用的嵌入式微处理器，总线和外部接口多集成在处理器内部。软件与硬件是紧密集成在一起的 开发方式 开发平台和运行平台都是通用计算机 采用交叉开发方式，开发平台一般是通用计算机，运行平台是嵌入式系统 二次开发性 应用程序可重新编制 一般不能再编程