目录

1

嵌入式软件设计

2

嵌入式软件设计

- 嵌入式软件架构与层次
- 代码优化
- 代码可靠性
- 代码实现的安全性（后续章节再讲）

3

嵌入式软件架构与层次

■ 嵌入式系统分为4层，硬件层、驱动层、系统层和应用层。

1、 **硬件层** 处理器，存储器、通用设备接口（例如通信模块）和I/O接口、扩展设备接口以及电源等；

2、 **驱动层**包含硬件抽象层HAL、板级支持包BSP和设备驱动程序；

3、 **系统层** RTOS、文件系统、GUI、网络系统及通用组件模块组成；

4、 **应用层** 基于实时系统开发的应用程序组成。

4

嵌入式C语言编程规范

- 对于嵌入式系统，C语言在开发速度，软件可靠性及软件质量等方面都有着明显的优势。
- 嵌入式系统受其硬件成本、功耗、体积以及运行环境的限制，需非常注重代码的时间和空间效率。目前，在嵌入式系统开发中C语言应用得最广泛；
- C 语言具有很强的功能性和可移植性。但在嵌入式系统开发中，出于对低价产品的需求，系统的计算能力和存储容量都非常有限，因此如何利用好这些资源就显得十分重要。
- 开发人员应注意嵌入式 C语言和标准 C 语言的区别，减少生成代码长度，提高程序执行效率，在程序设计中对代码进行优化。
- **优化永远是追求一种平衡，而不是走极端。**

5

充分利用硬件

- 最大可能地利用各种硬件设备自身的特点来减小其运转开销；
- 减少中断次数（例如通信BUF）
- 利用DMA传输方式。

6

变量类型

- 不同的数据类型所生成的机器代码长度相差很多，变量类型选取的范围越小运行速度越快，占用的内存越少。
- **我们应按照实际需要合理的选用数据类型：**
  - 能够使用字符型(char)定义的变量，就不要使用整型(int)变量来定义；
  - 能够使用整型变量定义的变量就不要用长整型(long int)；
  - 能不使用浮点型(float)变量就不要使用浮点型变量；
- 相同类型的数据类型，有无符号对机器代码长度也有影响；
- 在定义变量后不要超过变量的作用范围，如果超过变量的范围赋值，编译器并不报错，但程序运行结果却错了，而且这样的错误很难发现。

算法优化

- 算法优化指对程序时空复杂度的优化：在 PC 机上进行程序设计时一般不必过多关注程序代码的长短，只需考虑功能的实现，但嵌入式系统就必须考虑系统的硬件资源，在程序设计时，应尽量采用生成代码短的算法，在不影响程序功能实现的情况下优化算法。
- 减少运算的强度
  - （1）查表
  - （2）求余运算
  - （3）平方运算
  - （4）用移位实现乘除法运算，避免不必要的整数除法
  - （5）提取公共的子表达式

适当的使用宏提高程序的时间效率

- 在C程序中使用宏代码可以提高程序的时间效率。
- 宏代码本身不是函数，但使用起来像函数。
- 函数调用要使用系统的栈来保存数据，同时CPU在函数调用时需要保存和恢复当前的现场，进行入栈和出栈操作，所以函数调用需要CPU时间。
- 宏占用的是代码空间，省去了压栈、返回出栈等过程，从而提高了程序的执行速度.
- 但宏破坏了程序的可读性，使排错更加麻烦，但对于嵌入式系统，为了达到要求的性能，嵌入宏代码常常是必须的做法。

内嵌汇编

- 程序中对时间要求苛刻的部分可以用内嵌汇编来重写，以带来速度上的显着提高和确定的时间要求。
- 开发和测试汇编代码是一件辛苦、细致的工作，要慎重选择要用汇编的部分；
- 内嵌汇编与CPU紧密相关，不利于代码重用和移植；
- 在程序中，存在一个80-20原则，即20%的程序消耗了80%的运行时间，因而我们要改进效率，最主要是考虑改进那20%的代码。

提高循环语言的效率

- 在多重循环中，应将最长的循环放在最内层，最短的循环放在最外层.这样可以减少 CPU跨切循环的次数；
- 充分分解小的循环；
- 公共表达式放在循环外；
- 使用指针，少使用数组；
- 判断条件用0；（例如自减延时函数）；
- while与do while：do…while编译后的代码的长度短于while；
- WHILE(1)与FOR(;;)：FOR(;;)编译后的代码的长度短于WHILE(1)；
- 循环展开；
- 相关循环放到一个循环里。

提高 switch 语句的效率

- Switch 可能转化成多种不同算法的代码。其中最常见的是跳转表和比较链/树。
- 当switch用比较链的方式转化时，编译器会产生if-else-if的嵌套代码，并按照顺序进行比较，匹配时就跳转到满足条件的语句执行。所以可以对case的值依照发生的可能性进行排序，把最有可能的放在第一位，这样可以提高性能。
- 在case中推荐使用小的连续的整数（枚举），因为在这种情况下，编译器会把switch 转化成跳转表。
- 将大的switch语句转为嵌套switch语句，当switch语句中的case标号很多时，为了减少比较的次数，明智的做法是把大switch语句转为嵌套switch语句。把发生频率高的case标号放在一个switch语句中，并且是嵌套switch语句的最外层，发生相对频率相对低的case标号放在另一个switch语句中。

其他语句

- if >= 与< 0 ，减少条件与条件嵌套，充分利用PSW标志位
- 使用自增、自减和复合赋值运算符；
- 避免浮点运算；
- 优化赋值语句（比如定义赋值-编译器使用常数表转移）；
- 使用Thumb 指令集。

13

函数优化

- Inline函数：这个关键字请求编译器用函数内部的代码替换所有对于指出的函数的调用。这样做在两个方面快于函数调用：
  1、省去了调用指令需要的执行时间；
  2、省去了传递变元和传递过程需要的时间。
  但是使用这种方法程序长度变大了，因此需要更多的ROM。使用这种优化在Inline函数频繁调用并且只包含几行代码的时候是最有效的。
- 不定义不使用的返回值；
- 减少函数调用参数；
- 所有函数都应该有原型定义；
- 参数及返回值参数尽量同MCU字宽；
- 临时变量要少，可使用寄存器；
- 尽可能使用常量；
- 把本地函数声明为静态的(static)；

14

变量

- **register变量**
- 在最内层循环避免使用全局变量和静态变量；
- 同时声明多个变量优于单独声明变量；
- 短变量名优于长变量名，应尽量使变量名短一点；
- 在代码开始前声明变量；
- 使用尽量小的数据类型。

15

尽量避免使用标准库

- 使用标准库可以加快开发进度,但由于标准库需要处理用户所有可能遇到的情况,所以很多标准库代码很大；
- 很多标准库使用资源情况不清晰；（比如栈、执行时间和效率）

16

采用数学方法优化程序

- 例如:求 1~100 的和m = 100*(100+1)/2; 数学公式. (a1 + an)*n/2;
- 减少除法和取模的运算；
- 位操作：灵活的位操作可以有效地提高程序运行的效率。比如用位操作代替除法:比如（128 / 8 ）转换成（128 >> 3）；
- 优化算法和数据结构对提高代码的效率有很大的帮助。有时候时间效率和空间效率是对立的,此时应分析哪个更重要，做出适当折中。
- 在进行优化的时候不要片面的追求紧凑的代码,因为紧凑的代码并不能产生高效率的机器码。
- 避免递归计算。
- 代码优化可能影响程序可读性。

17

存储器分配

- 嵌入式系统存储器容量有限，程序中所有的变量,包含的库函数以及堆栈等都使用有限的内存:全局变量在整个程序范围内都有效,程序执行完后才会释放；静态变量的作用范围也是整个程序,只有局部变量中的动态变量在函数执行完后会释放。
- 全局变量和静态变量可直接寻址，速度快；动态变量一般间接寻址,速度慢。
- 提高内存使用效率在程序中应尽量使用局部变量；
- 提高时间效率在程序中应尽量使用全局变量和静态变量；
- 使用 malloc 函数申请内存之后一定要用 free 函数进行释放。
- 充分利用结构体和联合类型。

18

嵌入式代码优化是平衡艺术

- 代码长度/内存使用/执行效率需要平衡；
- 现在的 C 编译器会自动对代码进行优化，但这些优化是对执行速度和代码长度的平衡，不同编译器代码优化策略不同，效果差异大；
- 若要获得更好代码优化，需要手工对代码进行优化。
- 一般来说编程技巧对代码的优化贡献是20%；算法和数据结构对代码的优化贡献是80%；
- 代码优化可能影响代码可读性和可维护性；
- 能量优化嵌入式代码优化的新方向。

嵌入式软件的可靠性

- 程序存储器
- RAM
- 不挥发存储器EEPROM/FLASH
- 输入
- 输出

程序存储器

- 复位区分（上电、WDT复位、软件复位、欠压）、透明恢复
- 自检测试（开机、周期、键控、接线检查）
- 参数入口/出口检查
- 代码入口/出口执行队列检查
- 代码陷阱与错误捕获（关键位置NOP、未用空间NOP）

程序存储器

- WDT（硬件、软件、窗口）
- SLEEP躲避干扰
- 时钟变频抗干扰、内部时钟校准与同步
- 周期性校正与标定（特别是传感器）
- 外设复位与控制（复位延时）
- 时间限制（超时处理）与延时（绝对时间与相对时间）

RAM

- 数据单元（数据+校验码）
- 多编码备份（注意物理空间距离）
- 冷热启动区分（标志位与内存特征字）
- 内存清理(申请与释放、碎片管理)
- 堆栈与通信区防溢出

EEPROM/FLASH

- 安全存储

  数据单元、多编码备份、密文存储
- 防拔处理（参考COS的原子进程）
- 寿命与页面管理（参考COS）

输入

- 周期性重复配置
- 周期性输入与滤波（例如程序判断、算术平均、加权平均、去极值算术平均、低通/高通/带通滤波）
- 输入部件寿命管理（例如按键次数）
- 时间限制（超时处理）与延时（绝对时间与相对时间）

输出

- 内存可靠性高于端口，利用端口数据备份周期性刷新内容与配置（注意：内容在前）
- 输出部件寿命管理（例如LED、马达、喇叭）
- 输出与反馈
- 时间限制（超时处理）与延时（绝对时间与相对时间）

通信

- 通信处理参见输入、输出、RAM部分和接口章节；

嵌入式软件可维护性

- 一. 可理解性
- 二. 可靠性
- 三. 可测试性
- 四. 可修改性
- 五. 可移植性
- 六. 效率
- 七. 可使用性