第5章

嵌入式硬件构件与底层驱动构件基本规范

本章导读：本章主要分析嵌入式系统构件化设计的重要性和必要性，给出嵌入式硬件构件的概念、嵌入式硬件构件的分类、基于嵌入式硬件构件的电路原理图设计简明规则；给出嵌入式底层驱动构件的概念与层次模型；给出底层驱动构件的封装规范，包括构件设计的基本思想与基本原则、编码风格基本规范、头文件及源程序设计规范；给出硬件构件及底层软件构件的重用与移植方法。本章的目的是期望通过一定的规范，提高嵌入式软硬件设计的可重用性和可移植性。


视频讲解

5.1 嵌入式硬件构件

机械、建筑等传统产业的运作模式是先生产符合标准的构件(零部件)，再将标准构件按照规则组装成实际产品。其中，构件(Component)是核心和基础，复用是必需的手段。传统产业的成功充分证明了这种模式的可行性和正确性。软件产业的发展借鉴了这种模式，为标准软件构件的生产和复用确立了举足轻重的地位。

随着微控制器及应用处理器内部 Flash 存储器可靠性的提高及擦写方式的变化，内部 RAM 及 Flash 存储器容量的增大，以及外部模块内置化程度的提高，嵌入式系统的设计复杂性、设计规模及开发手段已经发生了根本变化。在嵌入式系统发展的最初阶段，嵌入式系统硬件和软件设计通常是由一个工程师来承担，软件在整个工作中的比例很小。随着时间的推移，硬件设计变得越来越复杂，软件的数量也急剧增长，嵌入式开发人员也由一人发展为由若干人组成的开发团队。为此，希望提高软硬件设计的可复用性与可移植性，构件的设计与应用是可复用性与可移植性的基础与保障。

5.1.1 嵌入式硬件构件概念及其分类

要提高硬件设计的可重用性与可移植性，就必须有工程师共同遵守的硬件设计规范。设计人员若凭借个人工作经验和习惯的积累进行系统硬件电路的设计，在开发完一个嵌入式应用系统后进行下一个应用开发时，硬件电路原理图往往需要从零开始，并重新绘制；或

者在一个类似的原理图上修改,但容易出错。因此,需把构件的思想引入硬件原理图设计中。

1. 嵌入式硬件构件概念

什么是嵌入式硬件构件?它与人们常说的硬件模块有什么不同?

众所周知,嵌入式硬件是任何嵌入式产品不可分割的重要组成部分,是整个嵌入式系统的构建基础,嵌入式应用程序和操作系统都运行在特定的硬件体系上。一个以 MCU 为核心的嵌入式系统通常包括电源、写入器接口电路、硬件支撑电路、UART、USB、Flash、AD、DA、LCD、键盘、传感器输入电路、通信电路、信号放大电路、驱动电路等硬件模块。其中,有些模块集成在 MCU 内部,有些模块位于 MCU 外部。

与硬件模块的概念不同,**嵌入式硬件构件是指将一个或多个硬件功能模块、支撑电路及其功能描述封装成一个可重用的硬件实体,并提供一系列规范的输入输出接口**。由定义可知,传统概念中的硬件模块是硬件构件的组成部分,一个硬件构件可能包含一个或多个硬件功能模块。

2. 嵌入式硬件构件分类

根据接口之间的生产消费关系,**接口可分为供给接口和需求接口两类**。根据所拥有接口类型的不同,硬件构件分为**核心构件**、**中间构件**和**终端构件** 3 种类型。**核心构件**只有供给接口,没有需求接口。也就是说,它只为其他硬件构件提供服务,而不接受服务。在以单 MCU 为核心的嵌入式系统中,MCU 的最小系统就是典型的核心构件。**中间构件**既有需求接口又有供给接口,即它不仅能够接受其他构件提供的服务,而且也能够为其他构件提供服务。而**终端构件**只有需求接口,它只接受其他构件提供的服务。这 3 种类型构件的区别如表 5-1 所示。

表 5-1 核心构件、中间构件和终端构件的区别

类 型	供给接口	需求接口	举 例
核心构件	有	无	芯片的硬件最小系统
中间构件	有	有	电源控制构件、232 电平转换构件
终端构件	无	有	LCD 构件、LED 构件、键盘构件

利用硬件构件进行嵌入式系统硬件设计之前,应该进行硬件构件的合理划分。按照一定规则,设计与系统目标功能无关的构件个体,然后进行“组装”,完成具体系统的硬件设计。所以,这些构件个体也可以被组装到其他嵌入式系统中。在硬件构件被应用到具体系统时,在绘制电路原理图阶段,设计人员需要做的仅是为需求接口添加接口**网标**[①]。

5.1.2 基于嵌入式硬件构件的电路原理图设计简明规则

在绘制原理图时,一个硬件构件使用一个虚线框,把硬件构件的电路及文字描述括在其中,将外接口引到虚线框之外,填上接口网标。

① 电路原理图中网标是指一种连线标识名称,凡是网标相同的地方,表示是连接在一起的。与此对应的还有文字标识,它仅是一种注释说明,不具备电路连接功能。

1. 硬件构件设计的通用规则

在设计硬件构件的电路原理图时，需遵循以下基本原则。

(1) 元器件命名格式。对于核心构件，其元器件可直接按编号命名，同种类型的元器件命名时冠以相同的字母前缀。例如，电阻名称为 R1、R2 等，电容名称为 C1、C2 等，电感名称为 L1、L2 等，指示灯名称为 E1、E2 等，二极管名称为 D1、D2 等，三极管名称为 Q1、Q2 等，开关名称为 K1、K2 等。对于中间构件和终端构件，其元器件命名格式采用“构件名-标志字符?”。例如，LCD 构件中所有的电阻名称统一为“LCD-R?”，电容名称统一为“LCD-C?”。当构件原理图应用到具体系统中时，可借助原理图编辑软件为其自动编号。

(2) 为硬件构件添加详细的文字描述，包括中文名称、英文名称、功能描述、接口描述、注意事项等，以增强原理图的可读性。其中，中英文名称应简洁明了。

(3) 将前两步产生的内容封装在一个虚线框内，组成硬件构件的内部实体。

(4) 为该硬件构件添加与其他构件交互的输入输出接口标识。接口标识有两种：接口注释和接口网标。它们的区别是：接口注释标于虚线框以内是为构件接口所做的解释性文字，目的是帮助设计人员在使用该构件时，理解该接口的含义和功能；而接口网标位于虚线框之外，且具有电路连接特性。为使原理图阅读者便于区分，接口注释采用斜体字。

在进行核心构件、中间构件和终端构件的设计时，除了要遵循上述的通用规则外，还要兼顾各自的接口特性、地位和作用。

2. 核心构件设计规则

设计核心构件时，需考虑的问题是：核心构件能为其他构件提供哪些信号？核心构件其实就是某型号 MCU 的硬件最小系统。**核心构件设计的目标是：凡是使用该 MCU 进行硬件系统设计时，核心构件可以直接“组装”到系统中，无须任何改动**。为了实现这一目标，在设计核心构件的实体时必须考虑细致、周全，包括稳定性、扩展性等，且封装要完整。核心构件的接口都是为其他构件提供服务的，因此接口标识均为接口网标。在进行接口设计时，需将所有可能使用到的引脚都标注上接口网标(无须考虑核心构件将会用到怎样的系统中去)。若同一引脚具有不同功能，则接口网标依据第一功能选项命名。遵循上述规则设计核心构件的好处是：当使用核心构件和其他构件一起组装系统时，只要考虑其他构件将要连接到核心构件的哪个接口(无须考虑核心构件将要连接到其他构件的哪个接口)，这也符合设计人员的思维习惯。

3. 中间构件设计规则

设计中间构件时，需考虑的问题是：中间构件需要接收哪些信号，以及提供哪些信号？中间构件是核心构件与终端构件之间通信的桥梁。在进行中间构件的实体封装时，实体的涉及范围应从构件功能和编程接口两方面考虑。一个中间构件应具有明确的且相对独立的功能，它既要有接收其他构件提供服务的接口，即需求接口，又要有为其他构件提供服务的接口，即供给接口。描述需求接口采用接口注释，处于虚线框内，描述供给接口采用接口网标，处于虚线框外。

中间构件的接口数目没有核心构件那样丰富。为了直观起见，设计中间构件时，将构件的需求接口放置在构件实体的左侧，供给接口放置在构件实体的右侧。接口网标的命名规则是：构件名称-引脚信号/功能名称。而接口注释名称前的构件名称可有可无，它的命名隐含了相应的引脚功能。

如图 5-1 和图 5-2 所示,电源控制构件和可变频率产生构件是常用的中间构件。图 5-1 中的 Power-IN 和图 5-2 中的 SDI、SCK 和 SEN 均为接口注释,Power-OUT 和 LTC6903-OUT 均为接口网标。

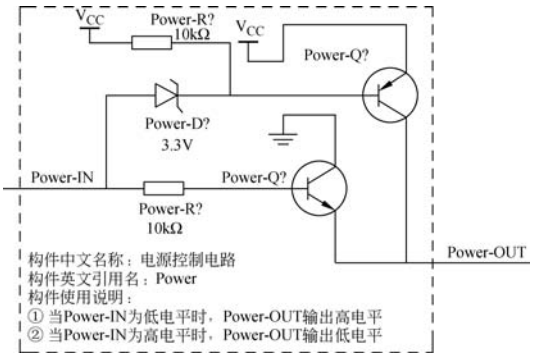


图 5-1 电源控制构件

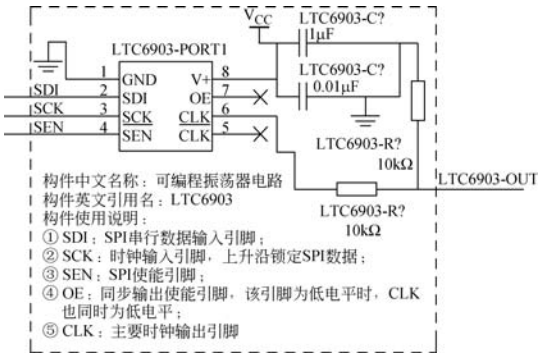


图 5-2 可变频率产生构件

4. 终端构件设计规则

设计终端构件时,需考虑的问题是:**终端构件需要什么信号才能工作**?终端构件是嵌入式系统中最常见的构件,它没有供给接口,仅有与上一级构件交付的需求接口。LCD(YM1602C)构件、LED 构件、指示灯构件及键盘构件等都是典型的终端构件,如图 5-3 和图 5-4 所示。

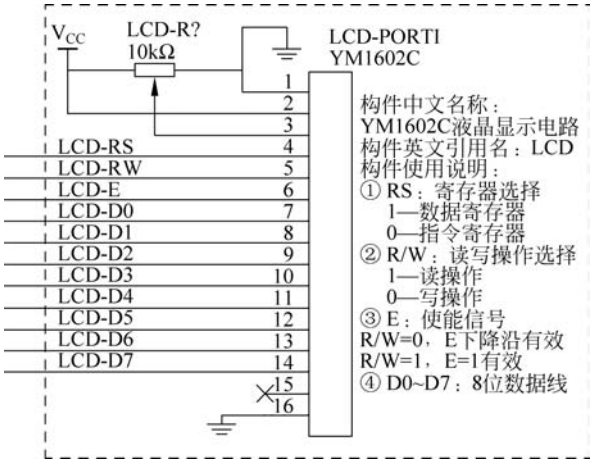


图 5-3 LCD 构件

图 5-4 键盘构件

5. 使用硬件构件组装系统的方法

对于核心构件,在应用到具体的系统中时,不必做任何改动。具有相同 MCU 的应用系统,其核心构件也完全相同。对于中间构件和终端构件,在应用到具体的系统中时,仅需为需求接口添加接口网标;在不同的系统中,虽然接口网标名称不同,但构件实体内部却完全相同。

使用硬件构件化思想设计嵌入式硬件系统的过程与步骤如下。

(1) 根据系统的功能划分出若干个硬件构件。
(2) 将所有硬件构件原理图“组装”在一起。
(3) 为中间构件和终端构件添加接口网标。

5.2 嵌入式底层驱动构件的概念与层次模型

嵌入式系统是软件与硬件的综合体,硬件设计和软件设计是相辅相成的。嵌入式系统中的驱动程序是直接工作在各种硬件设备上的软件,是硬件和高层软件之间的桥梁。正是通过驱动程序,各种硬件设备才能正常运行,达到既定的工作效果。

5.2.1 嵌入式底层驱动构件的概念

要提高软件设计可复用性与可移植性,就必须充分理解和应用软件构件技术。“提高代码质量和生产力的唯一最佳方法就是**复用**好的代码”,软件构件技术是软件复用实现的重要方法,也是软件复用技术研究的重点。

构件(Component)是可重用的实体,它包含了合乎规范的接口和功能实现,能够被独立部署和被第三方组装[①]。

软件构件(Software Component)是指在软件系统中具有相对独立的功能、可以明确辨识构件实体。

嵌入式软件构件(Embedded Software Component)是实现一定嵌入式系统功能的一组封装的、规范的、可重用的、具有嵌入特性的软件构件单元,是组织嵌入式系统功能的基本单位。嵌入式软件分为高层软件构件和底层软件构件(底层驱动构件)。高层软件构件与硬件无关,如实现嵌入式软件算法的算法构件、队列构件等;而底层驱动构件与硬件密不可分,是硬件驱动程序的构件化封装。下面给出嵌入式底层驱动构件的简明定义。

嵌入式底层驱动构件简称底层驱动构件或硬件驱动构件,是直接面向硬件操作的程序代码及函数接口的使用说明。规范的底层驱动构件由头文件(.h)及源程序文件(.c)构成[②],头文件(.h)应该是底层驱动构件简明且完备的使用说明。也就是说,在不需查看源程序文件的情况下,就能够完全使用该构件进行上一层程序的开发。因此,设计底层驱动构件必须有基本规范,5.3 节将阐述底层驱动构件的封装规范。

5.2.2 嵌入式硬件构件与软件构件结合的层次模型

前面内容提到,在硬件构件中,核心构件为 MCU 的最小系统。通常,MCU 内部包含 GPIO(即通用 I/O)接口和一些内置功能模块,可将通用 I/O 接口的驱动程序封装为 GPIO

① NATO Communications and Information Systems Agency. NATO Standard for Development of Reusable Software Components[S], 1991.

② 底层驱动构件若不使用 C 语言编程,相应组织形式有变化,但实质不变。

驱动构件,将各内置功能模块的驱动程序封装为功能构件。芯片内含模块的功能构件有串行通信构件、Flash 构件、定时器构件等。

在硬件构件层中,相对于核心构件而言,中间构件和终端构件是核心构件的外部设备。由这些外部设备的驱动程序封装而成的软件构件称为底层外部设备构件。注意:并不是所有的中间构件和终端构件都可以作为编程对象。例如,键盘、LED、LCD 等硬件构件与编程有关,而电平转换硬件构件就与编程无关,因而不存在相应的底层驱动程序,也就没有相应的软件构件。嵌入式硬件构件与软件构件的层次模型如图 5-5 所示。

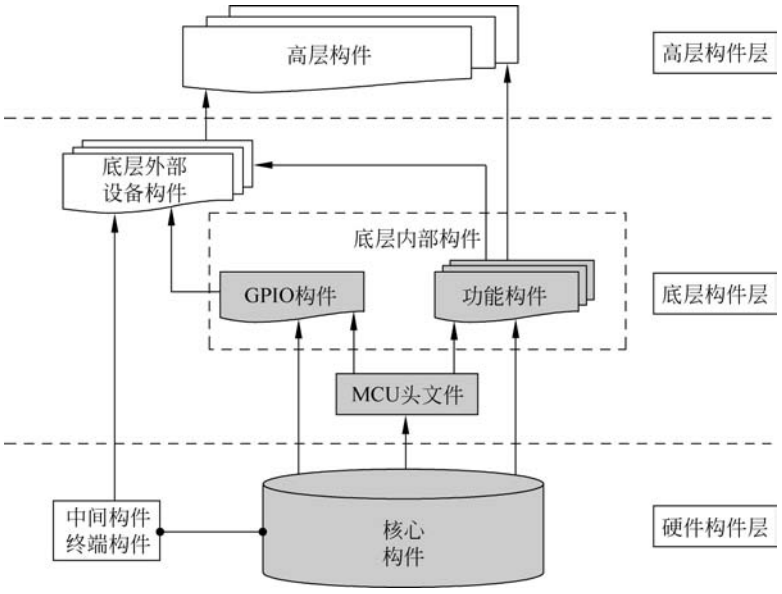


图 5-5 嵌入式硬件构件与软件构件结合的层次模型

由图 5-5 中可以看出,底层外部设备构件可以调用底层内部构件,如 LCD 构件可以调用 GPIO 驱动构件、PCF8563 构件(时钟构件)可以调用 I2C 构件等。而高层构件可以调用底层外部设备构件和底层内部构件中的功能构件,而不能直接调用 GPIO 驱动构件。另外,考虑到几乎所有的底层内部构件都涉及 MCU 各种寄存器的使用,因此将 MCU 的所有寄存器定义组织在一起,形成 MCU 头文件,以便其他构件的头文件中包含该头文件。

5.2.3 嵌入式软件构件分类

为了更加清晰地理解构件层次,可以按与硬件的密切程度及调用关系,把嵌入式构件分为基础构件、应用构件和软件构件 3 类。

1. 基础构件

基础构件是面向芯片级的硬件驱动构件,是符合软件工程封装规范的芯片硬件驱动程序。**其特点是面向芯片,以知识要素为核心,以模块独立性为准则进行封装。**

其中,面向芯片表明在设计基础构件时,不考虑具体应用项目。以知识要素为核心,尽可能把基础构件的接口函数与参数设计成芯片无关性,既便于理解与移植,也便于保证调用基础构件上层软件的可复用性。这里以 GPIO 构件为例简要说明封装 GPIO 底层驱动构件

的知识要素：①GPIO 引脚可以被定义为输入和输出两种情况；②若是输入，则程序需要获得引脚的状态(逻辑 1 或 0)；若是输出，则程序可以设置引脚状态(逻辑 1 或 0)；③若被定义为输入引脚，还有引脚上拉、下拉问题，以及中断使能/除能问题；④若是中断使能，还有边沿触发方式、电平触发方式、上升沿/下降沿触发方式等问题。基于这些知识要素设计 GPIO 底层驱动构件的函数及参数。参数的数据类型要使用基本类型，而不使用构造类型，便于接口函数芯片间的可移植性。模块独立性是指设计芯片的某一模块底层驱动构件时，不要涉及其他平行模块。

2. 应用构件

应用构件是调用芯片基础构件而制作完成的，符合软件工程封装规范的、面向实际应用硬件模块的驱动构件。**其特点是面向实际应用硬件模块，以知识要素为核心，以模块独立性为准则进行封装**。

3. 软件构件

嵌入式系统中的软件构件是不直接与硬件相关的，但符合软件工程封装规范，是实现一个完整功能的函数。**其特点是面向实际算法，以知识要素为核心，以功能独立性为准则进行封装**，如链表操作、队列操作、排序算法、加密算法等。

5.3 底层驱动构件的封装规范

驱动程序的开发在嵌入式系统的开发中具有举足轻重的地位。驱动程序的好坏直接关系整个嵌入式系统的稳定性和可靠性。然而，开发出完备、稳定的底层驱动构件并非易事，所以为了提高底层驱动构件的可移植性和可复用性，特制定底层驱动构件的封装规范。

5.3.1 构件设计的基本思想与基本原则

1. 构件设计的基本思想

底层构件是与硬件直接交互的软件，它被组织成具有一定独立性的功能模块，由头文件(.h)和源程序文件(.c)两部分组成。构件的头文件名和源程序文件名一致，且为构件名。

构件的头文件中，主要包含必要的引用文件、描述构件功能特性的宏定义语句及声明对外接口函数。良好的构件头文件应该成为构件使用说明，不需要使用者查看源程序。

构件的源程序文件中包含构件的头文件、内部函数的声明、对外接口函数的实现。

将构件分为头文件与源程序文件两个独立的部分，其意义在于：头文件中包含对构件使用信息的完整描述，为用户使用构件提供充分必要的说明，构件提供服务的实现细节被封装在源程序文件中；调用者通过构件对外接口获取服务，而不必关心服务函数的具体实现细节。这就是构件设计的基本内容。

在设计底层构件时，最关键的工作是对构件的共性和个性进行分析，设计出合理的、必要的对外接口函数及其形参。**尽量做到：当一个底层构件应用到不同系统中时，仅需修改构件的头文件，对于构件的源程序文件则不必修改或改动很小**。

2. 构件设计的基本原则

在嵌入式软件领域中,由于软件与硬件紧密联系的特性,使得与硬件紧密相连的底层驱动构件的生产成为嵌入式软件开发的重要内容之一。良好的底层驱动构件具备如下特性。

(1) **封装性**。在内部封装的实现细节中,采用独立的内部结构以减少对外部环境的依赖。调用者只通过构件接口获得相应功能,内部实现的调整将不会影响构件调用者的使用。

(2) **描述性**。构件必须提供规范的函数名称、清晰的接口信息、参数含义与范围、必要的注意事项等描述,为调用者提供统一、规范的使用信息。

(3) **可移植性**。底层构件的可移植性是指同样功能的构件,如何做到不改动或少改动,而方便地移植到同系列及不同系列芯片内,以减少重复劳动。

(4) **可复用性**。在满足一定使用要求时,构件不经过任何修改就可以直接使用,特别是使用同一芯片开发不同的项目,底层驱动构件应该做到复用。可复用性使得高层调用者对构件的使用不因底层实现的变化而有所改变,提高了嵌入式软件的开发效率、可靠性与可维护性。不同芯片的底层驱动构件的可复用性需在可移植性基础上进行。

为了使构件设计满足**封装性、描述性、可移植性、可复用性**的基本要求,嵌入式底层驱动构件的开发,应遵循**层次化、易用性、鲁棒性及对内存的可靠使用原则**。

1) 层次化原则

层次化设计要求清晰地组织构件之间的关联关系。底层驱动构件与底层硬件交互,在应用系统中位于最底层。遵循层次化原则设计底层驱动构件需要做到以下两点。

(1) 针对应用场景和服务对象,分层组织构件。在设计底层驱动构件的过程中,有一些是与处理器相关的、描述了芯片寄存器映射的内容,这些是所有底层驱动构件都需要使用的,将这些内容组织成底层驱动构件的公共内容,作为底层驱动构件的基础。在底层驱动构件的基础上,还可以使用高级的扩展构件调用底层驱动构件功能,从而实现更加复杂的服务。

(2) 在构件的层次模型中,**上层构件可以调用下层构件提供的服务,同一层次的构件不存在相互依赖关系,不能相互调用**。例如,Flash 模块与 UART 模块是平级模块,不能在编写 Flash 构件时,调用 UART 驱动构件。即使通过 UART 驱动构件函数的调用在 PC 屏幕上显示 Flash 构件测试信息,也不能在 Flash 构件内含有调用 UART 驱动构件函数的语句,应该编写上一层次的程序调用。平级构件是相互不可见的,只有深入理解并遵守,才能更好地设计出规范的底层驱动构件。在操作系统下,平级构件不可见特性尤为重要。

2) 易用性原则

易用性能够让调用者快速理解构件提供服务的功能并进行使用。遵循易用性原则设计底层驱动构件需要做到:**函数名简洁且达意;接口参数清晰,范围明确;使用说明语言精练规范,避免二义性**。此外,在函数的实现方面,避免编写代码量过多。函数的代码量过多不仅难以理解与维护,而且容易出错。若一个函数的功能比较复杂,可将其"化整为零",通过编写多个规模较小且功能单一的子函数,再进行组合,以实现最终的功能。

3) 鲁棒性原则

鲁棒性为调用者提供安全的服务,避免在程序运行过程中出现异常状况。遵循鲁棒性原则设计底层驱动构件需要做到:**在明确函数输入输出的取值范围、提供清晰接口描述的同时,在函数实现的内部要有对输入参数的检测,对超出合法范围的输入参数进行必要的处**

理；使用分支判断时，要确保对分支条件判断的完整性，对默认分支进行处理。例如，对 if 结构中的 else 分支和 switch 结构中的 default 分支安排合理的处理程序。同时，不能忽视编译警告错误。

4）内存可靠使用原则

对内存的可靠使用是保证系统安全、稳定运行的一个重要的考虑因素。遵循内存可靠使用原则设计底层驱动构件需要做到以下 5 点。

（1）优先使用静态分配内存。相比于人工参与的动态分配内存，静态分配内存由编译器维护，因此更为可靠。

（2）谨慎使用变量。当可以直接读写硬件寄存器时，不使用变量替代。避免使用变量暂存简单计算产生的中间结果。使用变量暂存数据将会影响数据的时效性。

（3）检测空指针。定义指针变量时必须初始化，防止产生空指针。

（4）检测缓冲区溢出，并为内存中的缓冲区预留不小于 20％的冗余。使用缓冲区时，对填充数据长度进行检测，不允许向缓冲区中填充超出容量的数据。

（5）对内存的使用情况进行评估。

5.3.2 编码风格基本规范

良好的编码风格能够提高程序代码的可读性和可维护性，而使用统一的编码风格在团队合作编写一系列程序代码时，无疑能够提高集体的工作效率。本节给出编码风格的基本规范，主要涉及文件、函数、变量、宏及结构体类型的命名规范，空格与空行、缩进、断行等的排版规范，以及文件头、函数头、行及边等的注释规范。

1. 文件、函数、变量、宏及结构体类型的命名规范

命名的基本原则如下。

（1）命名清晰明了，有明确含义，使用完整单词或约定俗成的缩写。通常，较短的单词可通过去掉元音字母形成缩写；较长的单词可取单词的头几个字母形成缩写，即“见名知意”。命名中若使用特殊约定或缩写，要有注释说明。

（2）命名风格要自始至终保持一致。

（3）为了代码复用，命名中应避免使用与具体项目相关的前缀。

（4）为了便于管理，对程序实体的命名要体现出所属构件的名称。

（5）使用英语命名。

（6）除宏命名外，名称字符串全部小写，以下画线（_）作为单词的分隔符。首尾字母不用下画线。

针对嵌入式底层驱动构件的设计需要，对文件、函数、变量、宏及数据结构类型的命令特别做以下说明。

1）文件的命名

底层驱动构件在具体设计时分为两个文件，其中头文件命名为<构件名>.h，源文件命名为<构件名>.c，且<构件名>表示具体的硬件模块的名称。例如，GPIO 驱动构件对应的两个文件为 gpio.h 和 gpio.c。

2) 函数的命名

底层驱动构件的函数从属于驱动构件,驱动函数的命名除要体现函数的功能外,还需要使用命名前缀和后缀标识其所属的构件及不同的实现方式。

函数名前缀:底层驱动构件中定义的所有函数均使用<构件名>_前缀表示其所属的驱动构件模块。例如,GPIO 驱动构件提供的服务接口函数命名为 gpio_init(初始化)、gpio_set(设定引脚状态)、gpio_get(获取引脚状态)等。

函数名后缀:对同一服务不同方式的实现,需使用后缀加以区分。这样做的好处是,当使用底层构件组装软件系统时,避免构件之间出现同名现象。同时,名称要有"顾名思义"的效果。

3) 函数形参变量与函数内局部变量的命名

对嵌入式底层驱动构件进行编码的过程中,需要考虑对底层驱动函数形参变量及驱动函数内部局部变量的命名。

函数形参变量:函数形参变量名是使用函数时理解形参的最直观印象,表示传递参数的功能说明。特别是,若传入底层驱动函数接口的参数是指针类型,则在命名时应使用_ptr后缀加以标识。

局部变量:局部变量的命名与函数形参变量类似。但函数形参变量名一般不取单个字符(如 i、j、k)进行命名,而 i、j、k 作为局部循环变量是允许的。这是因为变量,尤其是局部变量,如果用单个字符表示,很容易写错(如 i 写成 j),在编译时很难检查出来,就有可能因为这个错误花费大量的查错时间。

4) 宏常量及宏函数的命名

宏常量及宏函数的命名全部使用大写字符,使用下画线(_)作为分隔符。例如,在构件公共要素中定义开关中断的宏为:

```
#define ENABLE_INTERRUPTS  asm(" CPSIE  i")    //"开"总中断
#define DISABLE_INTERRUPTS asm(" CPSID  i")    //"关"总中断
```

5) 结构体类型的命名、类型定义与变量声明

(1) 结构体类型名称使用小写字母命名(< defined_struct_name >),定义结构体类型变量时,全部使用大写字母命名(< DEFINED_STRUCT_NAME >)。

(2) 对结构体内部字段全部使用大写字母命名(< ELEM_NAME >)。

(3) 定义类型时,同时声明一个结构体变量和结构体指针变量。

模板为:

```
typedef  struct  <defined_struct_name>
{
    <elem_type_1>   <ELEM_NAME_1>;       //对字段 1 含义的说明
    <elem_type_2>   <ELEM_NAME_2>;       //对字段 2 含义的说明
    …
} <DEFINED_STRUCT_NAME>, * <DEFINED_STRUCT_NAME_PTR>;
```

例如,当要定义一个描述 UART 设备初始化参数结构体类型时,可有如下定义:

```
typedef  struct  uart_init
{
    uint_8      DEV_ID:              // 串口设备号
    uint_32     BAUD_RATE:           //串口通信波特率
} UART_INIT_STRUCT,   * UART_INIT_PTR;
```

这样，uart_init 就是一种结构体类型；而 UART_INIT_STRUCT 是一个 uart_init 类型变量；UART_INIT_PTR 是 uart_init 类型指针变量。

2. 排版

对程序进行排版是指，通过插入空格与空行，使用缩进、断行等手段，调整代码的书面版式，**使代码整体美观、清晰，从而提高代码的可读性**。

1）空行与空格

关于空行：相对独立的程序块之间需加空行。关于空格：在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格，必要时可加两个空格；进行非对等操作时，如果是关系密切的立即操作符（如->），其后不应加空格。采用这种松散方式编写代码的目的是使代码更加清晰。例如，只在逗号、分号后面加空格；在比较操作符、赋值操作符“=”“+=”、算术操作符“+”“%”、逻辑操作符“&&”、位域操作符“<<”“^”等双目操作符的前后加空格；在“!”“~”“++”“--”“&”（地址运算符）等单目操作符前后不加空格；在“->”“.”前后不加空格；在 if、for、while、switch 等与后面括号间加空格，使关键字更为突出、明显。

2）缩进

使用空格缩进，建议不使用 Tab 键，这样代码复制打印不会造成错乱。代码的每一级均往右缩进 4 个空格的位置。函数或过程的开始、结构的定义、循环、判断等语句中的代码都要采用缩进风格，case 语句下的情况处理语句也要遵从语句缩进要求。

3）断行

建议**较长的语句（>78 字符）要分成多行书写**，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读；对于循环、判断等语句中，若有较长的表达式或语句，则要进行适应的划分，长表达式要在低优先级操作符处划分新行，操作符放在新行之首；若函数或过程中的参数较长，则要进行适当的划分；建议不要把多个短语句写在一行中，即一行只写一条语句。特殊情况如，“if （x>3）x=3;”可以写在一行；对于 if、for、do、while、case、switch、default 等语句后的程序块分界符（如 C/C++语言的花括号“{”和“}”）应各自独占一行，并且位于同一列，且与以上保留字左对齐。

3. 注释

在程序代码中使用注释，有助于对程序的阅读理解，说明程序在“做什么”，解释代码的目的、功能和采用的方法。编写注释时要注意：一般情况源程序有效注释量在 30%左右，注释语言必须准确、易懂、简洁，编写和修改代码的同时，处理好相应的注释，**C 语言中建议采用“//”注释，不建议使用段注释“/*…*/”。保留段注释用于调试，便于注释不用的代码**。

为规范嵌入式底层驱动构件的注释，下面对文件头注释、函数头注释、行注释与边注释做必要的说明。

1) 文件头注释

底层驱动构件的接口头文件和实现源文件的开始位置,使用文件头注释。例如:

```
//=====================================================================
//文件名称: gpio.h
//功能概要: GPIO底层驱动构件头文件
//版权所有: SD-EAI&IoT Lab.
//版本更新: 2020-11-01  V1.0
//=====================================================================
```

2) 函数头注释

在驱动函数的接口声明和函数实现之前,使用函数头注释详细说明驱动函数提供的服务。在构件的头文件中必须添加完整的函数头注释,为构件使用者提供充分的使用信息。构件的源文件对用户是透明的,因此,在必要时可适当简化函数头注释的内容。例如:

```
//=====================================================================
//函数名称: gpio_init
//函数返回: 无
//参数说明: port_pin: (端口号)|(引脚号)(例如,PT2|(2) 表示为2口5脚)
//          dir: 引脚方向(0=输入,1=输出,可用引脚方向宏定义)
//          state: 端口引脚初始状态(0=低电平,1=高电平)
//功能概要: 初始化指定端口引脚作为GPIO引脚功能,并定义为输入或输出。若是输出,
//          还需指定初始状态是低电平或高电平
//=====================================================================
```

3) 整行注释与边注释

整行注释文字,主要是对至下一个整行注释之前的代码进行功能概括与说明。边注释位于一行程序的尾端,对本语句或至下一边注释之间的语句进行功能概括与说明。此外,分支语句(条件分支、循环语句等)需在结束的"}"右方做边注释,表明该程序块结束的标记为"end_…",尤其在多重嵌套时需要有该结束标记。对于有特别含义的变量、常量,如果其命名不是充分自注释的,在声明时都必须加以注释,说明其含义。变量、常量、宏的注释应放在其上方相邻位置(行注释)或右方(边注释)。

5.3.3 头文件的设计规范

头文件描述了构件的接口,用户通过头文件获取构件服务。在本节中,对底层驱动构件头文件内容的编写加以规范,从程序编码结构、包含文件的处理、宏定义及设计服务接口等方面进行说明。

1. 编码框架

编写每个构件的头文件时,应使用"#ifndef…#define…#endif"的编码结构,防止对头文件的重复包含。例如,若定义GPIO驱动构件,在其头文件gpio.h中,应包含如下内容。

```
#ifndef  _GPIO_H
#define  _GPIO_H
…… // 文件内容
#endif
```

2. 包含文件

包含文件命令为#include,包含文件的语句统一安排在构件的头文件中,而在相应构件的源文件中仅包含本构件的头文件。将包含文件的语句统一置于构件的头文件中,使文件间的引用关系能够更加清晰地呈现。

3. 使用宏定义

宏定义命令为#define,使用宏定义可以替换代码内容,替换内容可以是常数、字符串,甚至还可以是带参数的函数。利用宏定义的替换特性,当需要变更程序的宏常量或宏函数时,只需一次性修改宏定义的内容,程序中每个出现宏常量或宏函数的地方均会自动更新。

对于宏常数,通常可使用宏定义表示构件中的常量,为常量值提供有意义的别名。例如,在灯的亮暗状态与对应 GPIO 引脚高低电平的对应关系需根据外接电路而定。此时,将表示灯状态的电平信号值用宏常量的方式定义,编程时使用其宏定义。当使用的外部电路发生变化时,仅需将宏常量定义做适当变更,而不必改动程序代码。

```
#define LIGHT_ON    0    //灯亮
#define LIGHT_OFF   1    //灯暗
```

对于宏函数,可以使用宏函数实现构件对外部请求服务的接口映射。在设计构件时,有时会需要应用环境为构件的基本活动提供服务。此时,采用宏函数表示构件对外部请求服务的接口,在构件中不关心请求服务的实现方式,这就为构件在不同应用环境下的移植提供了较强的灵活性。

4. 声明对外接口函数,包含对外接口函数的使用说明

底层驱动构件通过外接口函数为调用者提供简明而完备的服务,对外接口函数的声明及使用说明(即函数的头注释)包含于头文件中。

5. 特别说明

为某款芯片编写硬件驱动构件时,不同的构件存在公共使用的内容,可将这些内容放入头文件 cpu.h 中,供制作构件时使用,举例如下。

(1) 开关总中断的宏定义语句。高级语言没有对应语句,可以使用内嵌汇编的方式定义开关中断的语句:

```
#define ENABLE_INTERRUPTS  asm(" CPSIE  i")    //"开"总中断
#define DISABLE_INTERRUPTS asm(" CPSID  i")    //"关"总中断
```

(2) 一位操作的宏函数。将编程时经常用到对寄存器的某一位进行操作,即对寄存器的置位、清位及获得寄存器某一位状态的操作,定义为宏函数。设置寄存器某一位为 1,称为置位;设置寄存器某一位为 0,称为清位。这些操作在底层驱动编程时经常用到。置位与清位的基本原则是:当对寄存器的某一位进行置位或清位操作时,不能干扰该寄存器的其

他位；否则可能出现意想不到的错误。

综合利用“<<”“>>”“|”“&”“～”等位运算符，可以实现置位与清位，且不影响其他位的功能。下面以8位寄存器为例进行说明，其方法适用于各种位数的寄存器。设R为8位寄存器，下面描述将R的某一位置位与清位，而不干预其他位的编程方法。

置位。要将R的第3位置1，其他位不变，方法为：R |= (1<<3)。其中，1<<3的结果是“0b00001000”，R |= (1<<3)也就是R=R|0b00001000。任何数和0做“或”操作，结果为任何数；任何数和1做“或”操作，结果为1。这样操作可达到对R的第3位置1，但不影响其他位的目的。

清位。要将R的第2位清0，其他位不变，方法为：R &=～(1<<2)。其中，～(1<<2)的结果是0b11111011，R&=～(1<<2)也就是R=R&0b11111011。任何数和1做“与”操作，结果为任何数；任何数和0做“与”操作，结果为0。这样操作可达到对R的第2位清0，但不影响其他位的目的。

获得某一位的状态。(R>>4) & 1是获得R第4位的状态；R>>4是将R右移4位，将R的第4位移至第0位，即最后1位，再和1做“与”操作，也就是和0b00000001做“与”操作，保留R最后1位的值，以此得到第4位的状态值。

为了方便使用，把这种方法改为带参数的“宏函数”，并且简明定义，放在头文件cpu.h中。使用该“宏”的文件，可以包含头文件cpu.h。

```
#define BSET(bit,Register)  ((Register)| = (1 <<(bit)))     //置寄存器的一位
#define BCLR(bit,Register)  ((Register) &= ～(1≪(bit)))  //清寄存器的一位
#define BGET(bit,Register)  (((Register) >> (bit)) & 1)     //获得寄存器一位的状态
```

这样，可以使用BSET、BCLR、BGET这些容易理解与记忆的标识，进行寄存器的置位、清位及获得寄存器某一位状态的操作。

(3) 重定义基本数据类型。嵌入式程序设计与一般的程序设计有所不同，在嵌入式程序中使用的大多数是底层硬件的存储单元或寄存器，所以在编写程序代码时，使用的基本数据类型多以8位、16位、32位数据长度为单位。不同的编译器为基本整型数据类型分配的位数存在不同，但在编写嵌入式程序时要明确使用变量的字长，因此，需根据具体编译器重新定义嵌入式基本数据类型。重新定义后，不仅书写方便，也有利于软件的移植。例如：

```
typedef volatile uint8_t      vuint8_t;  //不优化无符号8位数,字节
typedef volatile uint16_t     vuint16_t; //不优化无符号16位数,字
typedef volatile uint32_t     vuint32_t; //不优化无符号32位数,长字
typedef volatile int8_t       vint_8;    //不优化有符号8位数
typedef volatile int16_t      vint_16;   //不优化有符号16位数
typedef volatile int16_t      vint_32;   //不优化有符号32位数
```

通常，有一些数据类型不能进行优化处理。在此，对不优化数据类型的定义做特别说明。不优化数据类型的修饰关键字是volatile，它用于通知编译器，对其后定义的变量不能随意进行优化，因此，编译器会安排该变量使用系统存储区的具体地址单元，编译后的程序每次需要存储或读取该变量时，都会直接访问该变量的地址。若没有volatile关键字，则编译器可能会暂时使用CPU寄存器来存储，以优化存储和读取。这样，CPU寄存器和变量地

址的内容很可能会出现不一致现象。对 MCU 映像寄存器的操作就不能优化，否则，对 I/O 接口的写入可能被“优化”写入 CPU 内部寄存器中，这样就会出现混乱。常用的 volatile 变量使用场合有设备的硬件寄存器、中断服务例程中访问到的非自动变量、操作系统环境下多线程应用中被几个任务共享的变量。

5.3.4 源程序文件的设计规范

编写底层驱动构件实现源文件的基本要求，是实现构件通过服务接口对外提供全部服务的功能。为确保构件工作的独立性，实现构件高内聚、低耦合的设计要求，将构件的实现内容封装在源文件内部。对于底层驱动构件的调用者而言，通过服务接口获取服务，不需要了解驱动构件提供服务的具体运行细节。因此，功能实现和封装是编写底层驱动构件实现源文件的主要考虑内容。

1. 源程序文件中的#include

底层驱动构件的源文件(.c)中，只允许一处使用#include 包含自身头文件。需要包含的内容需在自身构件的头文件中包含，以便有统一、清晰的程序结构。

2. 合理设计与实现对外接口函数与内部函数

驱动构件源程序文件中的函数包含对外接口函数与内部函数。对外接口函数供上层应用程序调用，其头注释需完整表述函数名、函数功能、入口参数、函数返回值、使用说明、函数适用范围等信息，以增强程序的可读性。在构件中封装比较复杂功能的函数时，代码量不宜过长。此时，应当将其中功能相对独立的部分封装成子函数。这些子函数仅在构件内部使用，不提供对外服务，因此被称为内部函数。为将内部函数的访问范围限制在构件的源文件内部，在创建内部函数时，应使用 static 关键字作为修饰符。内部函数的声明放在所有对外接口函数程序的上部，代码实现放在对外接口函数程序的后部。

一般地，实现底层驱动构件的功能，需要同芯片片内模块的特殊功能寄存器交互，通过对相应寄存器的配置实现对设备的驱动。某些配置过程对配置的先后顺序和时序有特殊要求，在编写驱动程序时要特别注意。

对外接口函数实现完成后，复制其头注释于头文件中，作为构件的使用说明。参考样例见电子资源中的 GPIO 构件及 Light 构件(各样例工程下均有)。

3. 不使用全局变量

全局变量的作用范围可以扩大到整个应用程序，其中存放的内容在应用程序的任何一处都可以随意被修改，一般可用于在不同程序单元间传递数据。但是，若在底层驱动构件中使用全局变量，其他程序即使不通过构件提供的接口也可以访问到构件内部，这无疑对构件的正常工作带来隐患。从软件工程理论中对封装特性的要求上看，也不利于构件设计高内聚、低耦合的要求。因此，在编写驱动构件程序时，严格禁止使用全局变量。用户与构件交互只能通过服务接口进行，即所有的数据传递都要通过函数的形参来接收，而不是使用全局变量。

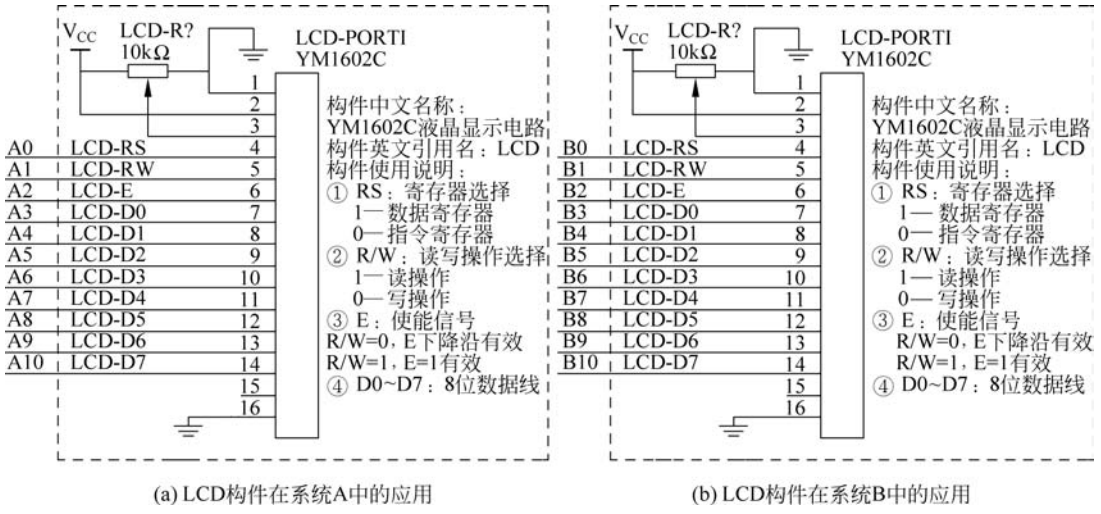
5.4 硬件构件及其驱动构件的复用与移植方法

复用是指在一个系统中，同一构件可被重复使用多次；移植是指将一个系统中使用到的构件应用到另外一个系统中。

5.4.1 硬件构件的复用与移植

对于以单 MCU 为核心的嵌入式应用系统而言,当用硬件构件组装硬件系统时,核心构件(即最小系统)有且只有一个,而中间构件和终端构件可有多个,并且相同类型的构件可出现多次。下面以终端构件 LCD 为例,介绍硬件构件的移植方法。其中,A0～A10 和 B0～B10 是芯片相关引脚,但不涉及具体芯片。

在应用系统 A 中,若 LCD 的数据线(LCD-D0～LCD-D7)与芯片的通用 I/O 接口的 A3～A10 相连,A0～A2 作为 LCD 的控制信号传送口。其中,LCD 寄存器选择信号 LCD-RS 与 A0 引脚连接,读写信号 LCD-RW 与 A1 引脚连接,使能信号 LCD-E 与 A2 引脚连接,则 LCD 硬件构件实例如图 5-6(a)所示。虚线框左边的文字(如 A0、A1 等)为接口网标,虚线框右边的文字(如 LCD-RS、LCD-RW 等)为接口注释。



(a) LCD构件在系统A中的应用　　(b) LCD构件在系统B中的应用

图 5-6　LCD 构件在实际系统中的应用

在应用系统 B 中,若 LCD 的数据线(LCD-D0～LCD-D7)与芯片的通用 I/O 接口的 B3～B10 相连,B0～B2 引脚分别作为寄存器选择信号 LCD-RS、读写信号 LCD-RW、使能信号 LCD-E,则 LCD 硬件构件实例如图 5-6(b)所示。

5.4.2 驱动构件的移植

当一个已设计好的底层构件移植到另外一个嵌入式系统中时,其头文件和程序文件是否需要改动,要视具体情况而定。例如,系统的核心构件发生改变(即 MCU 型号改变)时,底层内部构件头文件和某些对外接口函数也要随之改变,如模块初始化函数。

对于外接硬件构件,如果不改动程序文件,而只改动头文件,则头文件就必须被充分设计。以 LCD 构件为例,与图 5-6(a)相对应的底层构件头文件 lcd.h 可使用如下编写。

```
//=====================================================================
//文件名称: lcd.h
//功能概要: lcd构件头文件
// 版权所有: SD-EAI&IoT Lab
//=====================================================================

#ifndef LCD_H
#define LCD_H

#include "gpio.h"

#define LCDRS     A0          //LCD寄存器选择信号
#define LCDRW     A1          //LCD读写信号
#define LCDE      A2          //LCD读写信号
//LCD数据引脚
#define LCD_D7    A3
#define LCD_D6    A4
#define LCD_D5    A5
#define LCD_D4    A6
#define LCD_D3    A7
#define LCD_D2    A8
#define LCD_D1    A9
#define LCD_D0    A10
//=====================================================================
//函数名称: LCDInit
//函数返回: 无
//参数说明: 无
//功能概要: LCD初始化
//=====================================================================
void LCDInit();

//=====================================================================
//函数名称: LCDShow
//函数返回: 无
//参数说明: data[32]: 需要显示的数组
//功能概要: LCD显示数组的内容
//=====================================================================
void LCDShow(uint_8 data[32]);

#endif
```

当LCD硬件构件按照图5-6(b)中进行移植时,显示数据传送接口和控制信号传送接口发生了改变,只需修改头文件,而不需修改lcd.c文件。

本书给出构件化设计方法的目的是,在进行软硬件移植时,设计人员所做的改动应尽量小,而不是不做任何改动。希望改动尽可能在头文件中进行,而不希望改动程序文件。

本章小结

本章属于方法论内容,与具体芯片无关。主要阐述嵌入式硬件构件及底层驱动构件的基本规范。

1. 关于嵌入式硬件构件概念

嵌入式硬件构件是指将一个或多个硬件功能模块、支撑电路及其功能描述封装成一个可复用的硬件实体,并提供一系列规范的输入输出接口。嵌入式硬件构件根据接口之间的生产消费关系,接口可分为供给接口和需求接口两类。根据所拥有接口类型的不同,硬件构件分为核心构件、中间构件和终端构件3种类型。核心构件只有供给接口,没有需求接口,它只为其他硬件构件提供服务,而不接受服务。中间构件既有需求接口又有供给接口,它不仅能够接受其他构件提供的服务,还能够为其他构件提供服务。终端构件只有需求接口,它只接受其他构件提供的服务。设计核心构件时,需考虑的问题是:“核心构件能为其他构件提供哪些信号?”设计中间构件时,需考虑的问题是:“中间构件需要接受哪些信号,以及提供哪些信号?”设计终端构件时,需考虑的问题是:“终端构件需要什么信号才能工作?”

2. 关于嵌入式底层驱动构件设计原则与规范

嵌入式底层驱动构件是直接面向硬件操作的程序代码及使用说明。规范的底层驱动构件由头文件(.h)及源程序文件(.c)文件构成。头文件(.h)是底层驱动构件简明且完备的使用说明,即在不查看源程序文件情况下,就能够完全使用该构件进行上一层程序的开发,这也是设计底层驱动构件最值得遵循的原则。

在设计实现驱动构件的源程序文件时,需要合理设计外接口函数与内部函数。外接口函数供上层应用程序调用,其头注释需完整表述函数名、函数功能、入口参数、函数返回值、使用说明、函数适用范围等信息,以增强程序的可读性。在具体代码实现时,严格禁止使用全局变量。

3. 关于构件的移植与复用

在嵌入式硬件原理图设计中,要充分利用嵌入式硬件进行复用设计;在嵌入式软件编程时,涉及与硬件直接联系时,应尽可能复用底层驱动构件。若无可复用的底层驱动构件,应该按照基本规范设计驱动构件,然后再进行应用程序开发。

习题

1. 简述嵌入式硬件构件概念及嵌入式硬件构件分类。
2. 简述核心构件、中间构件和终端构件的含义及设计规则。
3. 简述嵌入式底层驱动构件的基本内涵。
4. 在设计嵌入式底层驱动构件时,其对外接口函数设计的基本原则有哪些?
5. 举例说明在什么情况下使用宏定义。
6. 举例说明底层构件的移植方法。
7. 利用C语言,自行设计一个底层驱动构件,并进行调试。
8. 利用一种汇编语言,设计一个底层驱动构件,并进行调试,同时与C语言设计的底层驱动构件进行简明比较。