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。

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)，然后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由一名工程师来承担，软件在全部开发工作中的比例很小。随着时间的推移，硬件设计变得越来越复杂，软件的体量也急剧增长，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，开发者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复用与移植的基础与保障。

5.1.1 嵌入式硬件构件概念与嵌入式硬件构件分类

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，进行下一个应用的开发时，硬件电路原理图就往往需要从零开始，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，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 之外。

与硬件模块的概念不同，嵌入式硬件构件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与消费关系，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三种类型。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。这三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。这样，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框在其中，对外接口引到虚线框之外，填上接口网标。

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

(1) 元器件命名格式：对于核心构件，其元器件直接以编号命名，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，电容名称为 C1、C2，电感名称为 L1、L2，指示灯名称为 E1、E2，二极管名称为 D1、D2，三极管名称为 Q1、Q2，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。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，还要兼顾各构件的接口特性、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：凡使用该 MCU 进行硬件系统设计时，核心构件均可以直接“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，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，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，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。接口注释名称前的构件名称则可有可无，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(以斜体标注)，Power-OUT

和 LTC6903-OUT 均为接口网标。

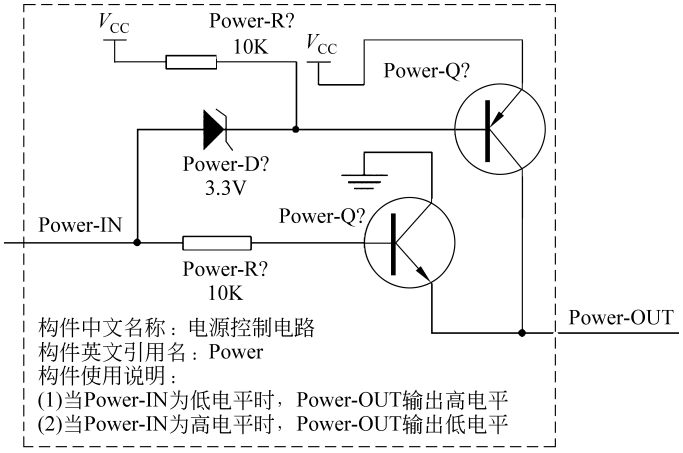

图 5-1　电源控制构件

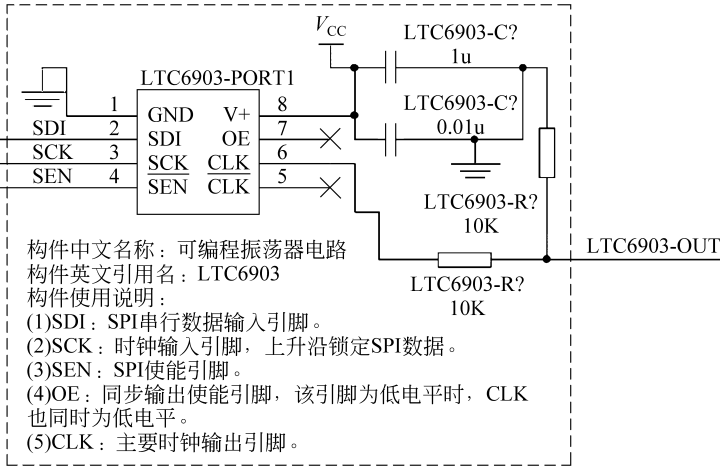

图 5-2　可变频率产生构件

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，仅有向上一级构件交付所用的需求接口，因而接口标识均为斜体标注的接口注释。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。

5. 使用硬件构件组装系统的方法

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。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。

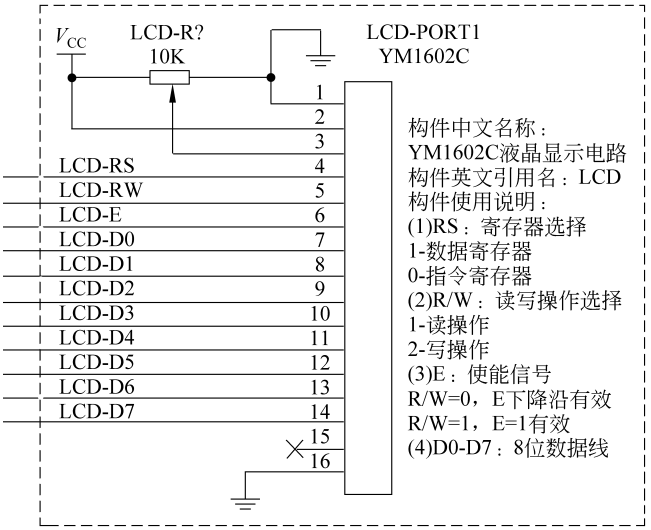

图 5-3　LCD 构件

图 5-4　键盘构件

（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)指在软件系统中具有相对独立功能、可以明确辨识的构件实体。

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。下面为嵌入式底层驱动构件进行简明定义。

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 IO)口和一些内置功能模块,可将通用 I/O 口的驱动程序封装为 GPIO 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“外设”。由这些“外设”的驱动程序封装而成的软件构件称为底层外设构件。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。

由图 5-5 可以看出,底层外设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。高层构件则可以调用底层外设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,以便其他构件头文件中包含该头文件。

5.2.3 嵌入式软件构件分类

为了便于理解与应用,可以把嵌入式构件分为基础构件、应用构件与软件构件三种类型。

1. 基础构件

基础构件是根据 MCU 内部功能模块的基本知识要素,针对 MCU 引脚功能或 MCU 内

---

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,相应组织形式会有变化,但实质不变。

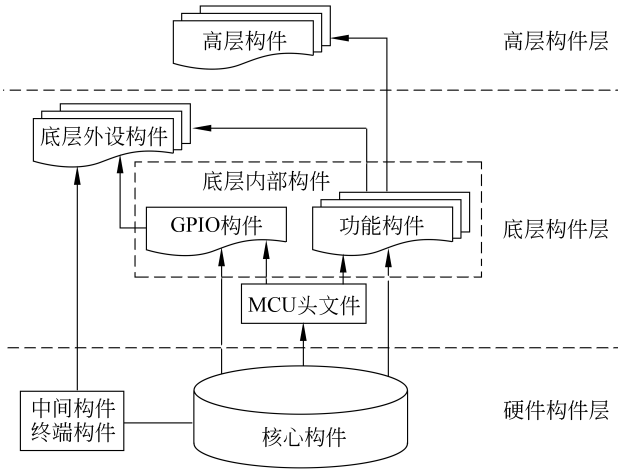

图 5-5　嵌入式硬件构件与软件构件结合的层次模型

部功能，利用 MCU 内部寄存器所制作的直接干预硬件的构件。基础构件是面向芯片级的、符合软件工程封装规范的硬件驱动构件，也称为底层硬件驱动构件，常简称作底层构件、驱动构件。常用的基础构件主要有 GPIO 构件、UART 构件、Flash 构件、ADC 构件、PWM 构件、SPI 构件、I2C 构件等。

基础构件的特点是面向芯片，不考虑具体应用，以功能模块独立性为准则进行封装。面向芯片，表明在设计基础构件时，不仅应该考虑具体应用项目，还要屏蔽芯片之间的差异，尽可能把基础构件的接口函数与参数设计成与芯片无关，既便于理解与移植，也便于保证调用基础构件的上层软件的可复用性；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

应用构件是通过调用芯片的基础构件制作的，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，以硬件模块独立性为准则进行封装。例如，若一个 LCD 硬件模块是 SPI 接口的，则 LCD 构件调用基础构件 SPI，完成对 LCD 显示屏控制的封装。也可以把 printf 函数纳入应用构件，因为它调用串口构件。printf 函数调用的一般形式为 printf("格式控制字符串", 输出表列)，本书使用的 printf 函数可通过串口向外传输数据。

3. 软件构件

软件构件是一个面向对象的、具有规范接口和确定的上下文依赖的组装单元，它能够被独立使用或被其他构件调用。本书使用的软件构件概念狭义地限制在与硬件无关层面。其特点是面向实际算法，以功能独立性为准则进行封装，具有底层硬件无关性，例如排序算法、队列操作、链表操作及人工智能相关算法等。

5.2.4　基础构件的基本特征与表现形式

基础构件即底层硬件驱动构件，是嵌入式软件与硬件打交道的必经之路。开发应用软件时，需要通过底层硬件驱动构件提供的应用程序接口与硬件打交道。封装好的底层硬件驱动构件能减少重复劳动，使广大 MCU 应用开发者专注于应用软件稳定性与功能设计，提

高开发的效率和稳定性。

为了把基础构件设计好、封装好，读者应首先了解构件的基本特征与形式。

1. 构件的基本特征

封装性、可移植性与可复用性是软件构件的基本特性。在嵌入式软件领域中，由于软件与硬件紧密联系的特性，与硬件紧密相连的基础构件的生产成为嵌入式软件开发的重要内容之一。良好的基础构件具备如下特性：

(1) 封装性。在内部封装实现细节，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。基础构件的可移植性指同样功能的构件，在不改动或少改动的前提下，方便地移植到同系列及不同系列的芯片内，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项目时，基础构件应该做到复用。可复用性使得上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。不同芯片的基础构件复用需在可移植性基础上进行。

2. 构件的表现形式

底层驱动构件即基础构件，是与硬件直接打交道的程序。它被设计成具有一定独立性的功能模块，由头文件和源程序文件两部分组成[①]。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以及声明对外接口函数的语句。良好的构件头文件应该相当于构件使用说明书，使用者不需要查看源程序就能使用构件。

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等。

将构件分为头文件与源程序文件两个独立的部分，意义在于，头文件中包含对构件的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。

构件中的函数名使用“构件名_函数功能名”形式命名，以便明确标识该函数属于哪个构件，实现什么功能。

构件中的内部调用函数不在头文件中声明，其声明直接放在源程序头部，不做注释，只做声明。函数头注释及函数实体在对外接口函数后给出。

从RTOS角度来说，构件应该是与RTOS无关的，这样才能保证构件的可移植性与可复用性。

---

① 特别强调一下，根据软件工程的基本原则，一个基础构件只能由一个头文件和一个源程序文件组成，头文件是构件的使用说明。

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着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。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　基础构件设计的基本原则

为了能够把基础构件设计好、封装好，开发者还要了解构件设计的基本原则。

在设计基础构件时，最关键的工作是要对构件的共性和个性进行分析，从而设计出合理的、必要的对外接口函数，使得一个基础构件可以直接应用到使用同一芯片的不同工程中，不需任何修改。

根据构件的封装性、描述性、可移植性、可复用性的基本特征，基础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。基础构件与底层硬件打交道，在应用系统中位于最底层。遵循层次化原则设计基础构件需要做到：

针对应用场景和服务对象，分层组织构件。设计基础构件的过程中，有一些与处理器相关的、描述了芯片寄存器映射的内容，是所有基础构件都需要使用的。将这些内容组织成基础构件的公共内容，作为基础构件的基础。在基础构件的基础上，还可使用高级的扩展构件调用基础构件功能，从而实现更加复杂的服务。

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，但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。即使要通过对 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，应该在上一层次的程序中调用。平级构件是相互不可见的，只有深入理解这一点，并遵守之，才能更好地设计出规范的基础构件。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

易用性在于让调用者能够快速理解构件提供的功能并能快速正确使用。遵循易用性原则设计基础构件需要做到：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，要避免编写代码量过多。函数的代码量过多会致使其难以理解与维护，并且容易出错。若一个函数的功能比较复杂，可将其“化整为零”，先编写多个规模较小，功能单一的子函数，再进行组合，实现整体的功能。

3. 鲁棒性原则

鲁棒性在于为调用者提供安全的服务，以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基础构件，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；不忽视编译警告错误；使用分支判断时，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”安排合理的处理程序，

可以增强鲁棒性。

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因素。遵循内存可靠使用原则设计基础构件,需要做到:

(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,更为可靠。例如,在设计基础构件时,尽量不要使用 malloc、new 动态申请内存。

(2) 谨慎地使用变量。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所产生的中间结果,原因是使用变量暂存数据会影响数据的时效性。

(3) 防止“野指针”。为避免指向非法地址,定义指针变量时必须初始化。

(4) 防止缓冲区溢出。使用缓冲区时,建议预留不小于 20%的冗余。在对缓冲区填充前,先检测数据长度,防止缓冲区溢出。

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。本节给出了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) 使用完整英文单词或约定俗成的缩写,有明确含义。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) 为了便于代码复用,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) 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“_”。

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进行以下说明。

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。