STM32 与飞行控制

作者

郑 扬 珞

wunschunreif@sjtu.edu.cn

2019–02–15

免责声明

本文件及其所含信息均按“原样”提供，作者不作任何明示或暗示的保证，包括但不限于使用本文件所含信息不会侵犯任何权利或任何适销性或特定用途适用性的暗示保证。

目录

图片索引

表格索引

代码索引

缩略语

SoC System on a chip

RM Reference Manual

1. 绪论

1.1. 为什么要学习 STM32

自动化控制技术在飞行器制造的过程中日趋重要，姿态稳定、舵面控制等都要用到电子控制，而单片机以其重量轻、耗电少、控制逻辑灵活、易于学习应用等诸多优势，成为航模电子系统的首选控制器类型。在各类单片机中，STM32 具有很高的性价比：丰富的片上外设、完备的库函数、灵活的控制方式，以及远高出同类产品的时钟频率，使其成为在各个领域都有很强通用性的单片机产品，而它同样适合作为飞行控制器的核心，因此，我们有必要学习这款单片机，了解它的编程方式、具体功能，以便开发更为强大的控制系统。
（以上都是胡编的，我也不知道为什么要学习 STM32。。。）

1.2. 如何学习 STM32

作为一种应用技术，STM32 的最佳学习方式自然是实践，因此希望读者在阅读本文档的同时，能够多动手进行编程实践。虽然本文档的定位是一份涵盖作者认为编程中所有可能遇到的问题的事无巨细的指南，但不可否认仍有许多的问题只有亲自动手才能发现并解决，从而作为日后的经验积累下来。在这个过程中，我们也欢迎读者对本文档的内容做出斧正与完善。
此外，官方文档始终是学习 STM32 的权威帮助，作者在此推荐 STM32 的 Datasheet 和 Reference Manual，以及其标准外设库的文档，其中包含大量的示例代码以供学习参考。当然，遇到难以解决的问题时，求助网络可能是最快的解决方式，毕竟 STM32 在国内拥有相当大的群众基础，大多数问题都有网友进行总结了。

1.3. 预备知识

本文档认为读者已经基本掌握了下列预备知识：

- 计算机基本操作及其大致原理的了解

- C 语言程序设计（本文将采用 C11 标准），包括熟练掌握 2 阶以下的指针、函数指针、结构体、枚举等知识
- 电子技术，只需掌握简单的电路分析、一些数字电路的基本知识，但要求有一定的实践基础，例如能够正确地连接电路

1.4. 局限性

本文档编写的目的是阐述 STM32 单片机在一般的飞行控制中需要用到的外设的编程控制方法，对于其他的外设讨论很少。此外，本文档将着重介绍 STM32F1 系列的单片机，而并不会涉及性能更强的 STM32F4、STM32F7 等系列的单片机。

本章的最后，祝愿各位读者在今年上半年迅速掌握 STM32 这款单片机，软硬两开花！

Part I.

STM32 基础知识

这一部分主要讲解 STM32 的基本程序设计方法。内容包括:

- STM32 简介
- 开发环境搭建——基于 macOS 与 GCC
- GPIO 操作及其中断
- SysTick 的应用
- 未完待续。。。

完成这一部分的学习，读者就具备了利用 STM32 编写不太复杂的控制程序的能力，可以实现许多有趣的小项目。

2. STM32 及其编程简介

2.1. STM32 是什么

STM32 是指意法半导体（STMicroelectronics）制造的一系列 32 位 ARM 架构单片机，这些单片机依据它们使用的核心不同，分为许多系列，例如本文档主要介绍的 F1 系列采用的是 Cortex-M3 内核[1]。作为单片机，STM32 不仅包含了 CPU 核，还集成了 Flash 程序存储器、sRAM 内存，以及诸如 GPIO 控制器的丰富外设，构成了一个片上系统（System on a chip (SoC) ）。此外，STM32F1 系列单片机的时钟频率可以高达 72MHz，远超 Arduino UNO 的时钟频率，因此可以实现一些复杂的、繁重的控制任务，以及一些常用的算法。

本文档主要介绍的是 STM32F103C8 单片机，其中，这个命名包含的信息有：

- STM32 - 表示 ST 生产的 32 位单片机
- F103 - 该单片机的系列
- C - 引脚数为 48 个
- 8 - Flash 大小为 64KiB[2]

此外，它的内存容量为 20KiB，是一款“中密度”单片机产品。

关于 STM32F103C8 的更详细的数据可以从它的 Datasheet[3]中查到，这里不再继续对其进行介绍。

2.2. STM32 程序怎么写

虽然我们认为这篇文档的读者以及掌握了基本的 C 语言程序设计，但是，在一个高度抽象的计算机上编程，与这里为单片机编程有着相当大的区别，所以，我们有必要从本质出发认识 STM32 程序的设计方法。

[1] https://en.wikipedia.org/wiki/STM32
[2] 1 KiB = 1024 Bytes
[3] https://www.st.com/resource/en/datasheet/stm32f103c8.pdf

2.2.1. 程序的本质是什么

程序的本质，当然是计算，毕竟程序是为“计算机”设计的。计算，可以认为是程序唯一的本质，CPU 的所有行为不过是从存储介质中取得运算数，按程序对其计算，再将结果写入规定的存储介质中。可是这样，单片机是如何完成各种强大的控制功能的呢？这些功能，实际上是通过逻辑电路来实现的，CPU 将控制这些逻辑电路的参数写入规定好的寄存器中，就可以告诉相应的逻辑电路该产生什么行为，除非遭遇了宇宙射线什么的，否则这些逻辑器件总是会乖乖听话。这就是单片机控制功能的来源。不如举个例子，查阅 STM32 的参考手册[4]（Reference Manual (RM)），可以知道 GPIO 这个外设有一个寄存器叫做“ODR (Port output data register)”，可以控制 GPIO 的输出电平，如下图：

9.2.4 Port output data register (GPIOx_ODR) (x=A..G)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy:** Port output data (y= 0 .. 15)

These bits can be read and written by software and can be accessed in Word mode only.

Note: For atomic bit set/reset, the ODR bits can be individually set and cleared by writing to the GPIOx_BSRR register (x = A .. G).

图 2.1.: ODR 寄存器

假设 GPIOA 已经正确初始化，那么只要我们向 GPIOA 的 ODR 寄存器写入 0x01，就可以控制 PA1 引脚输出高电平。为此，我们的程序可能长得像下面这样：

```
1 GPIOA->ODR = 0x01;
```

代码清单 2.1: 控制 PA1 输出高电平

实际上，完成这个操作的确这么简单，这一行程序的功能也没有超出“计算”的范围，因为它就是向一个指定的位置写入了一个预先算好的数据。

[4]https://www.st.com/resource/en/reference_manual/cd00171190.pdf

所以我们说，程序的本质是计算。