第 3 章　STM32F429 微控制器

3.1　STM32 系列微控制器

STM32 系列微控制器是意法半导体（STMicroelectronics）公司基于 ARM Cortex-M 内核开发的处理器，支持 32 位广泛的应用，支持包括高性能、实时功能、数字信号处理，以及低功耗、低电压操作，同时拥有一个完全集成和易用的开发环境。STM32 系列微控制器基于行业标准内核，提供了大量工具和软件选项，使该系列产品成为小型项目和完整平台的理想选择。STM32 系列微控制器的主要特点有性能高、电压低、功耗低、外设丰富、软件和硬件开发资源多、简单易用等。

3.1.1　STM32 系列微控制器概述

STM32 系列微控制器（MCU）主要包括主流级微控制器、高性能微控制器、超低功耗微控制器及无线微控制器，按照内核分类包括 Cortex-M0/M0+、Cortex-M3、Cortex-M4 及 Cortex-M7 系列微控制器。STM32 系列微控制器各系列简述表如表 3-1 所示。

表 3-1　STM32 系列微控制器各系列简述表

系列	微控制器
主流级微控制器	STM32 F0 系列：ARM Cortex-M0 入门级微控制器
	STM32 F1 系列：ARM Cortex-M3 基础型微控制器
	STM32 F3 系列：ARM Cortex-M4 混合信号微控制器
高性能微控制器	STM32 F2 系列：ARM Cortex-M3 高性能微控制器
	STM32 F4 系列：ARM Cortex-M4 高性能微控制器
	STM32 F7 系列：ARM Cortex-M7 高性能微控制器
	STM32 H7 系列：ARM Cortex-M7 超高性能微控制器
超低功耗微控制器	STM32 L0 系列：ARM Cortex-M0+低功耗微控制器
	STM32 L1 系列：ARM Cortex-M3 超低功耗微控制器
	STM32 L4 系列：ARM Cortex-M4 超低功耗微控制器
	STM32 L4+系列：ARM Cortex-M4 超低功耗高性能微控制器
无线微控制器	STM32 WB 系列：ARM Cortex-M4 和 Cortex-M0+双核无线微控制器

STM32 系列微控制器集成了大量常用的片上外设，这为嵌入式系统设计提供了极大的方便，促使设计者使用更小的面积开发出了更高功能密度的产品。STM32 系列微控制器通用的片上资源如下。

（1）大量 GPIO。

（2）多种通信外设：USART、SPI、I2C。

（3）14 个基本、通用和高级定时器。

（4）直接内存存取（DMA）。

（5）看门狗和实时时钟。

（6）锁相环（Phase Locked Loop，PLL）和实时时钟（Integrated Regulator PLL、Clock Circuit）。

（7）3 个 12 位数模转换器（DAC）。

（8）4 个 12 位模数转换器（ADC）。

（9）工作温度：−40～85℃（工业级），−40～125℃（汽车级）。

（10）低电压：2.0～3.6V。

（11）内部温度传感器（Temperature Sensor）。

3.1.2 芯片命名规则

STM32 系列微控制器命名方法如图 3-1 所示。

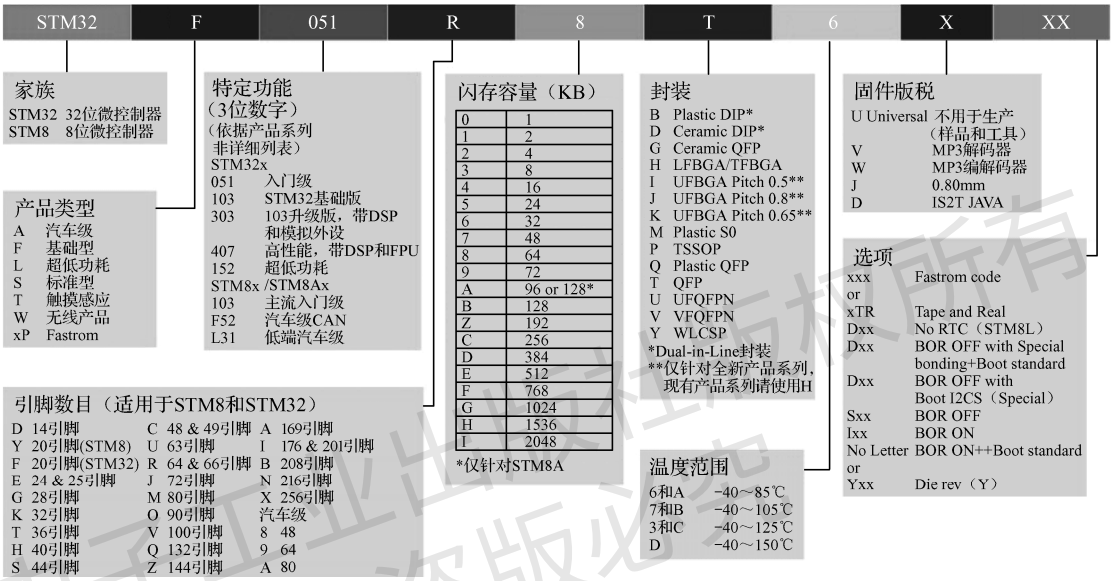

图 3-1 STM32 系列微控制器命名方法

STM32F429IGT6 命名中各位含义如表 3-2 所示。

表 3-2 STM32F429IGT6 命名中各位含义

组成	含义
家族	STM32 表示 32 位的微控制器
产品类型	F 表示基础型
特定功能	429 表示高性能，带 DSP、FPU、LTDC 控制器和 FMC
引脚数目	I 表示 176 引脚
闪存容量	G 表示 1024KB
封装	T 表示 QFP 封装，这是最常用的封装
温度	6 表示温度等级为 A（−40～85℃）

3.1.3 开发工具

进行 STM32F 系列微控制器的程序开发需要搭建一个交叉开发环境，其中包括计算机、开发软件、调试器、开发板或自己设计的电路板（包括 STM32F 系列微控制器）。程序交叉开发环境搭建示意图如图 3-2 所示。

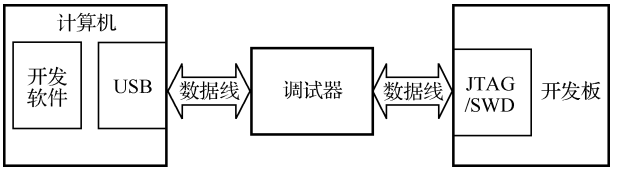

图 3-2　程序交叉开发环境搭建示意图

在计算机中使用开发软件编写应用程序，然后使用调试器将计算机和开发板连接起来，再将应用程序通过调试器下载到开发板中（一般是 STM32F 系列微控制器内部的存储器中），即可使用开发软件在线调试应用程序，在发现应用程序有错误时进行调试（语法错误一般能够在编译阶段排除，使用在线调试主要是为了发现程序中功能或逻辑错误），直至应用程序满足设计要求，大致有以下几个步骤。

（1）创建一个工程，选择一块目标芯片，并且做一些必要的工程配置。

（2）编写 C 或汇编源文件。

（3）编译应用程序。

（4）修改源程序中的错误。

（5）联机调试。

1．开发软件

选择合适的开发环境可以加快开发进度，节省开发成本。由于 STM32 系列微控制器基于 ARM 内核，所以很多基于 ARM 嵌入式开发环境都可用于 STM32 开发平台，开发者可以根据需求选择适合自己的开发环境。常用商业版软件有 Keil MDK 和 IAR EWARM。

1）Keil MDK

Keil MDK（MDK-ARM）是德国知名软件公司——Keil（现已并入 ARM 公司）开发的微控制器软件开发平台，是目前 ARM 内核单片机开发的主流工具。MDK-ARM 提供了包括 C 编译器、宏汇编、连接器、库管理和一个功能强大的仿真调试器在内的完整开发方案，通过一个集成开发环境将这些功能组合在一起集成了业内最领先的技术。MDK-ARM 支持 Cortex-M、Cortex-R 等 ARM 内核处理器，集成 Flash 烧写模块等，具备针对不同调试器的在线调试功能，已经成为 ARM 软件开发工具的标准。MDK-ARM 有 4 个可用版本，分别是 MDK-Lite、MDK-Basic、MDK- Standard、MDK-Professional。最新版本为 MDK-ARM Version 5.27（截止时间为 2019.03.25）。

2）IAR EWARM

IAR EWARM（Embedded Workbench for ARM）是瑞典IARSystems 公司为 ARM 微处理器开发的一个集成开发环境。它包含项目管理器、编辑器、C/C++编译器、汇编器、连接器和调试器，IAR EWARM 具有入门容易、使用方便、代码紧凑等特点。通过其内置的针对不同芯片的代码优化器，IAR EWARM 可以为 ARM 芯片生成非常高效、可靠的 Flash/PROMable 代码。IAR EWARM 中包含一个全软件的模拟程序，用户不需要任何硬件支持就可以模拟 ARM 内核、外围设备，甚至中断的软件运行环境。最新版本为 IAR EWARM 8.32（截止时间为 2019.03.25）。

2．调试工具

大部分 STM32F 系列微控制器内核包含用于高级调试功能的硬件，利用这些调试功能，可以在取指（指令断点）或取访问数据（数据断点）时使内核停止。当内核停止时，可以查询内核的内部状态和系统的外部状态。查询完成后，将恢复内核和系统并恢复程序执行。

当调试器与 STM32F 系列微控制器相连并进行调试时，将使用内核的硬件调试模块。调试接口（SWJ-DP）把 SWD-DP 和 JTAG-DP 功能合二为一，并且支持自动协议检测。

调试工具提供两个调试接口：串行（SWD）接口和 JTAG 接口。STM32F4 系列微控制器调试接口结构框图如图 3-3 所示。

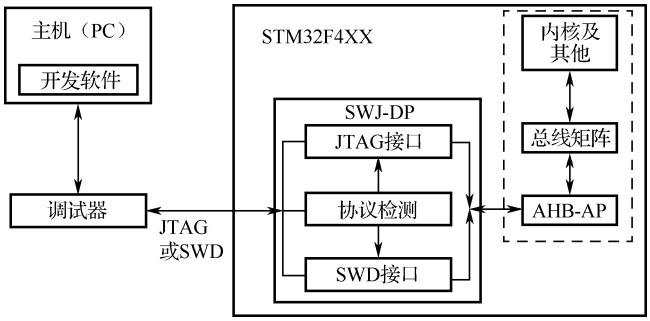

图 3-3　STM32F4 系列微控制器调试接口结构框图

STM32F4 系列微控制器调试接口引脚图如图 3-4 所示。

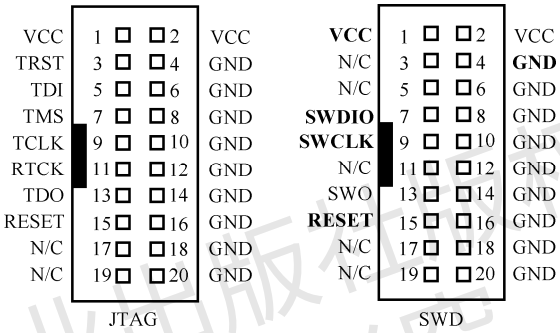

图 3-4　STM32F4 系列微控制器调试接口引脚图

STM32F4 系列微控制器调试接口各引脚功能如表 3-3 所示。

表 3-3　STM32F4 系列微控制器调试接口各引脚功能

仿真器接口	连接目标板	备注
1：VCC	微控制器电源 VCC	连接接口信号电平参考电压，一般直接连接电路板电源
2：VCC（可选）	微控制器电源 VCC	连接接口信号电平参考电压，一般直接连接电路板电源
3：TRST	TRST	测试复位，输入引脚，低电平有效
4：GND	GND	连接电路板 GND
5：TDI	TDI	TDI 接口是数据输入的接口，在 IEEE1149.1 标准里是强制要求的。所有要输入至特定寄存器的数据都是通过 TDI 接口一位一位串行输入的（由 TCK 驱动）
6：GND	GND	连接电路板 GND
7：TMS，SWDIO	TMS，SWDIO	TMS：测试模式选择（Test Mode State） SWDIO：串行数据输入、输出（Serial Wire Input and Output） 在 TCK 的上升沿有效。TMS 在 IEEE1149.1 标准里是强制要求的。TMS 信号用来控制 TAP 状态机的转换
8：GND	GND	连接电路板 GND
9：TCLK，SWCLK	TMS，SWCLK	TCLK：测试时钟（Test Clock），SWCLK：串行时钟（Serial Wire Clock） TCLK 与 SWCLK 在 IEEE1149.1 标准里是强制要求的。TCK 为 TAP 的操作提供了一个独立的、基本的时钟信号，TAP 的所有操作都是通过这个时钟信号来驱动的
10：GND	GND	连接电路板 GND

续表

仿真器接口	连接目标板	备注
11：RTCK	RTCK	目标端反馈给仿真器的时钟信号，用来同步 TCK 信号的产生，不使用时直接接地
12：GND	GND	连接电路板 GND
13：TDO	TDO	TDO 在 IEEE1149.1 标准里是强制要求的。TDO 接口是数据输出的接口。所有要从特定的寄存器中输出的数据都是通过 TDO 接口一位一位串行输出的（由 TCK 驱动）
14：GND	GND	连接电路板 GND
15：RESET	RESET	目标板上的系统复位信号相连，可以直接对目标系统复位。同时可以检测目标系统的复位情况，为了防止误触发，应在目标端加上适当的上拉电阻
16：GND	GND	连接电路板 GND
17：N/C	N/C	悬空
18：GND	GND	连接电路板 GND
19：N/C	N/C	悬空
20：GND	GND	连接电路板 GND

在实际的使用中，为了减少 JTAG 接口在电路板中占用的面积，通常使用双排的 10 针 JTAG 接口作为仿真接口使用。10 针 JTAG 接口电路图如图 3-5 所示。

串行调试（Serial Wire Debug，SWD），可以算是一种和 JTAG 不同的调试模式，二者使用的调试协议也应该不一样，所以最直接地体现在调试接口上，与 JTAG 接口的 20 个引脚相比，SWD 接口只需要 4 个引脚，结构简单。SWD 和传统的调试方式区别如下。

SWD 模式比 JTAG 在高速模式下更加可靠，在基本使用 JTAG 仿真模式的情况下是可以直接使用 SWD 模式的，只要仿真器支持，所以推荐大家使用这个模式。

在电路板空间紧张的时候，推荐使用 SWD 模式的接口。SWD 模式的接口只需要使用一个很小的 2.54mm 间距的 4 芯端子作仿真接口。由于它需要的引脚少，因此占用的 PCB 空间就小。SWD 接口电路图如图 3-6 所示。

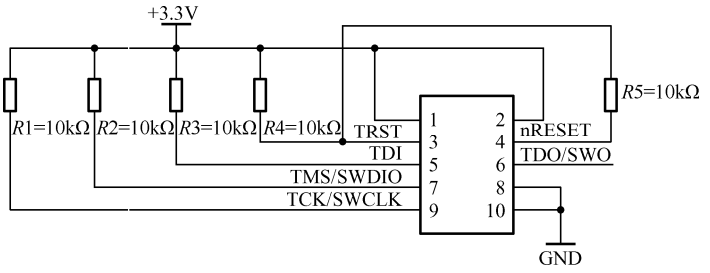

图 3-5　10 针 JTAG 接口电路图

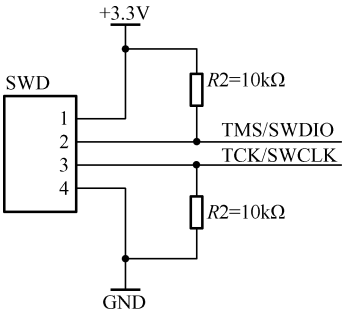

图 3-6　SWD 接口电路图

1）J-Link

J-Link 是 SEGGER 公司为支持仿真 ARM 内核芯片推出的 JTAG 仿真器。它是通用的开发工具，配合 MDK-ARM、IAR EWARM 等开发平台，可以实现对 ARM7、ARM9、ARM11、Cortex-M0/M1/M3/M4、Cortex-A5/A8/A9 等大多数 ARM 内核芯片的仿真。J-Link 需要安装驱动程序，才能配合开发平台使用。J-Link 仿真器有 J-Link Plus、J-Link Ultra、J-Link Ultra+、J-Link Pro、

J-Link EDU、J-Trace 等多个版本，可以根据不同的需求来选择不同的产品。J-Link 外观图如图 3-7 所示。

（1）JTAG 最高时钟频率可达 15MHz。

（2）目标板电压范围为 1.2～3.3V，5V 兼容。

（3）具有自动速度识别功能。

（4）支持编辑状态的断点设置，并在仿真状态下有效。可快速查看寄存器和方便配置外设。

（5）带 J-Link TCP/IP server，允许通过 TCP/ IP 网络使用 J-Link。

2）U-Link2

U-Link2 是 Keil 公司开发的仿真器，专用于 MDK-ARM 平台。在 MDK-ARM 平台下无须驱动，可直接使用。U-Link2 外观图如图 3-8 所示。

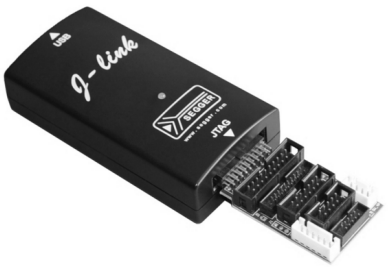

图 3-7　J-Link 外观图

图 3-8　U-Link2 外观图

（1）JTAG 最高时钟频率可达 10MHz。

（2）支持 Cortex-M 串行查看器（SWV）数据和时间跟踪，速度高达 1Mbit/s（UART 模式）。

（3）支持编辑状态的断点设置，并在仿真状态下有效。

（4）拥有独特的工具窗口，可快速查看寄存器和方便配置外设。

（5）存储区域/寄存器查看。

3）ST-Link

ST-Link 是 ST 公司为 STM8 系列和 STM32 系列微控制器设计的仿真器。ST-Link 外观图如图 3-9 所示。编程功能：可烧写 Flash ROM、EEPROM、AFR 等，需要安装驱动程序才能使用。

仿真功能：支持全速运行、单步调试、断点调试等调试方法，可查看 I/O 状态、变量数据等。

仿真性能：采用 USB2.0 接口进行仿真调试，单步调试，断点调试，反应速度快。

编程性能：采用 USB2.0 接口，进行 SWIM/JTAG/SWD 下载，下载速度快。

图 3-9　ST-Link 外观图

3．开发板

开发板是用来进行嵌入式系统开发的电路板，包括嵌入式微处理器、存储器、输入设备、输出设备、数据通路/总线和外部资源接口等一系列硬件组件。开发板是为初学者了解和学习嵌入式系统的硬件和软件而设计的，部分开发板还提供了基础集成开发环境、软件源代码和硬件原理图等。

4．软件开发形式

进行 STM32 系列微控制器的程序开发主要分为两种方式：直接寄存器操作开发方式和库函数开发方式。

使用直接寄存器操作开发方式需要完全熟悉微控制器的使用方法、流程及寄存器的配置方法。在编写程序时，直接面对寄存器，需要程序员自己编写所有操作代码。

库函数开发方式是在一个特定的库函数基础上进行程序开发的。库函数把对寄存器的操作抽象成了一系列操作微控制器的 API（Application Program Interface）函数，程序员只需要在功能层面熟悉 API 函数使用方法，然后按照一定的流程编写程序即可。

意法半导体公司提供了操作 STM32F 系列微控制器的标准函数库，该函数库包括对 STM32F 系列微控制器操作的基本 API 函数。开发者可调用这些函数接口来配置 STM32F 系列微控制器的寄存器，使开发人员得以脱离最底层的寄存器操作。使用函数库开发方式进行应用开发，有开发快速、易于阅读、维护成本低等优点。

库函数开发方式和直接寄存器操作开发方式之间的对比如图 3-10 所示。

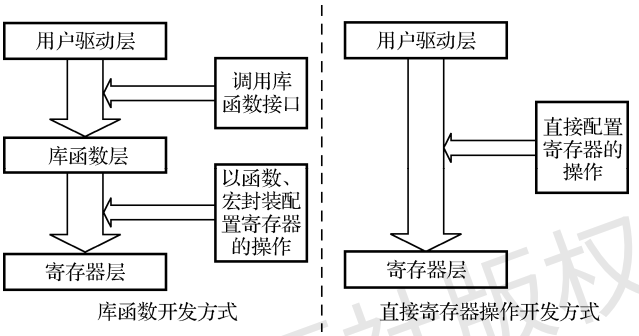

图 3-10　库函数开发方式和直接寄存器操作开发方式之间的对比

库函数是架设在寄存器层与用户驱动层之间的代码，向下处理与寄存器直接相关的配置，向上为用户提供配置寄存器的接口。库函数开发方式与直接寄存器操作开发方式的区别如表 3-4 所示。

表 3-4　库函数开发方式与直接寄存器操作开发方式的区别

软件开发形式	特点
库函数开发方式	更接程序员的思维：①用结构体封装寄存器参数；②用宏表示参数，意义明确；③用函数封装对寄存器的操作
	移植性好：代码的易读性好，使得驱动修改非常方便
直接寄存器操作开发方式	更接近机械思维：直接针对寄存器的某些为进行置 1 或清零操作，能清晰看到驱动代码控制的底层对象
	运行效率高：没有库函数层，省去代码为分层而消耗的资源

在实际使用过程中，大量使用库函数的地方主要集中在初始化阶段，在需要大量数据传输或响应操作等场合，库函数使用得并不是非常频繁，因此库函数开发方式和直接寄存器操作开发方式在运行效率上的区别不大。综合考虑两种方式的优缺点，在 STM32 系列微控制器实际开发过程中，推荐使用库函数开发方式。

为了理解 API 函数实现的原理，程序员需要掌握寄存器操作方式的编程方法，在后续章节将会就寄存器操作方法进行讲解。

3.1.4　STM32 标准函数库介绍

STM32 系列内核是 ARM 公司设计的处理器体系架构。ST 公司或其他芯片生产商，负责设计的是除内核之外的部件，被称为核外外设或片上外设、设备外设，如芯片内部的模数转换外

设 ADC、串口 UART、定时器 TIM 等。

为了解决不同的芯片生产商生产的 Cortex 微控制器软件的兼容性问题，ARM 公司与芯片生产商建立了 CMSIS 标准（Cortex Microcontroller Software Interface Standard），CMSIS 标准软件结构图如图 3-11 所示，它包括以下几部分。

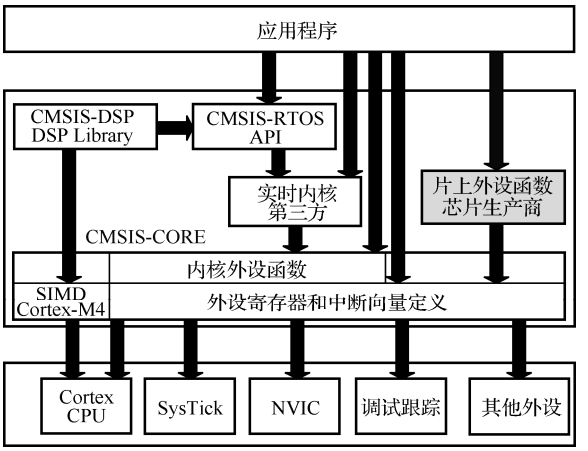

图 3-11　CMSIS 标准软件结构图

（1）CMSIS-CORE：提供 Cortex-M0、Cortex-M3、Cortex-M4、SC000 和 SC300 等处理器与外围寄存器之间的接口。

（2）CMSIS-DSP：包含以定点（分数 q7、q15、q31）和单精度浮点（32 位）实现的 60 多种函数的 DSP 函数库。

（3）CMSIS-RTOS API：用于线程控制、资源和时间管理的实时操作系统的标准化编程接口。

CMSIS 层位于硬件层与操作系统或用户层之间，它提供了与芯片生产商无关的硬件抽象层，可以为接口外设、实时操作系统提供简单的处理器软件接口，屏蔽了硬件差异，这对软件的移植是有极大好处的。

在此基础上，芯片生产商设计片上外设函数，从而形成一个完整的标准函数库，它包括以下几部分。

核内外设访问层（Core Peripheral Access Layer，CPAL）：该层由 ARM 公司负责实现，包括对核内寄存器名称、地址的定义，内核寄存器、NVIC、调试子系统的访问接口定义，以及对特殊用途寄存器的访问接口（如控制寄存器、程序状态寄存器）定义。由于对特殊用途寄存器的访问以内联方式定义，所以针对不同的编译器 ARM 公司统一用__INLINE 来屏蔽差异。该层定义的接口函数均是可重入的。

片上外设访问层（Device Peripheral Access Layer，DPAL）：该层由芯片生产商负责实现。该层的实现与 CPAL 类似，负责对硬件寄存器地址及外设访问接口进行定义。该层可调用 CPAL 层提供的接口函数，同时根据设备特性对异常向量表进行扩展，以处理相应外设的中断请求。

外设访问函数（Access Functions for Peripherals，AFP）：该层也由芯片生产商负责实现，主要提供访问片上外设的访问函数。STM32 的标准函数库按照 CMSIS 标准建立，是一个固件函数库，由程序、数据结构和宏组成，包括微控制器所有外设的性能特征。该固件函数库还包括每一个外设的驱动描述和应用实例，为开发者访问底层硬件提供了一个中间 API，通过使用固件函数库，无须深入掌握底层硬件细节，开发者就可以轻松应用每一个外设。因此，使用固态函数库可以大大减少开发者开发使用片内外设的时间，进而降低开发成本。每个外设驱动都由一组函数组成，这组函数覆盖了该外设所有功能。每个器件的开发都由一个通用的标准化的 API 函数驱动。

库文件主要包括 Libraries、Project、Utilities 三个文件夹和三个说明文件。库文件结构图如图 3-12 所示。

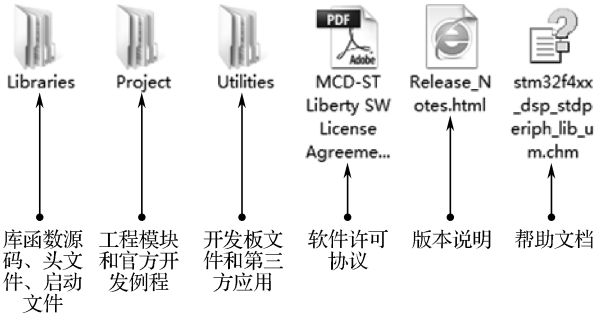

图 3-12　库文件结构图

Libraries 文件夹包含用到的函数库的所有文件，用户通过对这个文件夹中文件的使用来完成应用程序的设计。Project 文件夹包含 MDK-ARM、IAR EWARM 等集成开发环境工程模板和官方片上外设的一些例程。Utilities 文件夹包含开发板文件和第三方应用。

STM32 的标准函数库的组成机构如图 3-13 所示。

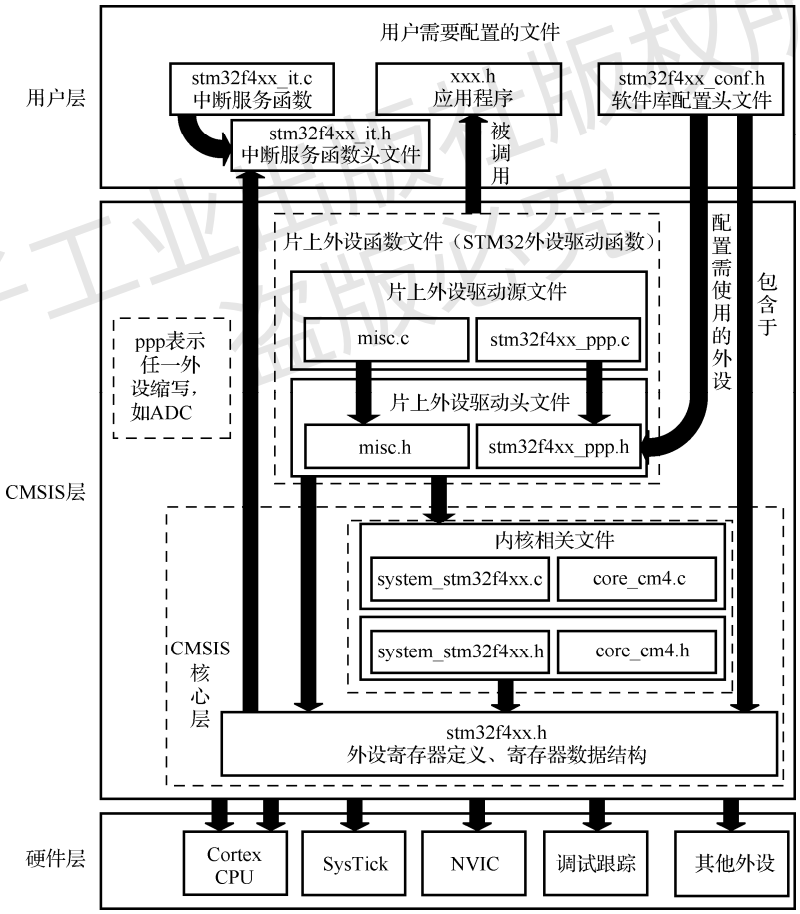

图 3-13　STM32 的标准函数库的组成机构

在实际应用中，需要按照图 3-13 中的结构把各环节用到的文件一并加载到集成开发环境的

工程之中，以在 MDK-ARM 下建立 STM32F429 应用工程为例，工程中用到的各程序功能和目录如表 3-5 所示。

表 3-5　工程中用到的各程序功能和目录

功能分类	文件名	功能说明	目录地址
启动文件	startup_stm32f429_439xx.s	启动文件	..\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\arm
外设相关	stm32f4xx.h	外设寄存器定义	..\Libraries\CMSIS\Device\ST\STM32F4xx\Include
	system_stm32f4xx.h	—	..\Libraries\CMSIS\Device\ST\STM32F4xx\Include
	system_stm32f4xx.c	用于配置系统时钟等	..\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates
	stm32f4xx_conf.h	可以选择应用中的外设	..\Project\STM32F4xx_StdPeriph_Templates
	stm32f4xx_ppp.h	外设标准函数库头文件	..\Libraries\STM32F4xx_StdPeriph_Driver\inc
	stm32f4xx_ppp.c	外设标准函数库源件	..\Libraries\STM32F4xx_StdPeriph_Driver\src
	misc.h	—	..\Libraries\STM32F4xx_StdPeriph_Driver\inc
	misc.c	NVIC、SysTick 相关函数	..\Libraries\STM32F4xx_StdPeriph_Driver\src
内核相关	core_cm4.h	内核寄存器定义	..\Libraries\CMSIS\Include
	core_cmFunc.h	操作内核相关，不常用	..\Libraries\CMSIS\Include
	core_cmInstr.h	内核指令定义	..\Libraries\CMSIS\Include
	core_cmSimd.h	SIMD 指令定义	..\Libraries\CMSIS\Include
通用	stdint.h	数据类型定义	..\Keil_v5\ARM\ARMCC\include\
用户相关	stm32f4xx_it.h	中断服务函数头文件	..\Project\STM32F4xx_StdPeriph_Templates
	stm32f4xx_it.c	用户编写的中断服务函数	..\Project\STM32F4xx_StdPeriph_Templates
	main.c	用户应用程序主程序入口	可自定义
	其他应用子程序	用户自定义应用功能	可自定义

启动文件 startup_stm32f429_439xx.s 实现了如下功能。

（1）初始化堆栈指针，SP=__initial_sp。

（2）初始化程序计数寄存器指针，PC=Reset_Handler。

（3）初始化中断向量表。

（4）配置系统时钟。

（5）调用 C 库函数__main 初始化用户堆栈，从而最终调用 main 函数进入 C 程序的世界。

在\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates 这个目录下，除 arm 文件夹外，还有 gcc_ride7、iar、SW4STM32、TrueSTUDIO 等文件夹，这些文件夹包含了对应编译平台的汇编启动文件，在实际使用时要根据编译平台来选择。如果使用其他型号的芯片，要在此处选择对应的启动文件，如 STM32F407 型号的芯片使用 startup_stm32f4xx.s 启动文件。

stm32f4xx.h 文件是一个 STM32 系列芯片底层相关的文件，比较重要。它包含了 STM32 标准函数库中所有外设寄存器地址和结构体类型定义，在使用到 STM32 标准函数库的地方都要包含这个头文件。

system_stm32f4xx.c 文件包含 STM32 系列芯片上电后初始化系统时钟、扩展外部存储器用

的函数。例如，用于上电后初始化时钟的 SystemInit 函数，对应的头文件是 system_stm32f4xx.h。STM32F429 系列的芯片，执行 SystemInit 函数后，系统时钟频率被初始化为 180MHz，如有需要可以修改这个文件的内容，将其设置成自己所需的时钟频率。

stm32f4xx_conf.h 文件被包含进 stm32f4xx.h 文件中。STM32 标准函数库支持所有 STM32F4 型号的芯片，但有的型号芯片外设功能比较多，所以在使用这个配置文件时需要根据芯片型号增减 STM32 标准函数库的外设文件。通过宏定义指定不同芯片的型号所包含的不同外设。例如，STM32F429 和 STM32F427 型号芯片片上外设有差异，分别使用了不同的宏定义不一样的头文件。

在 STM32 标准函数库的函数中，一般会包含输入参数检查，使用 assert_param 宏完成这一功能，当参数不符合要求时，会调用 assert_failed 函数，这个函数默认是空的。

stm32f4xx_ppp.c 文件定义了处理器中各个片上外设的驱动接口函数，这些文件是我们使用函数库的主体。操作片上外设所需要的功能函数都可以在相应的 stm32f4xx_ppp.c 文件中找到。ppp 表示任一外设缩写。例如，adc 对应的文件是 stm32f4xx_adc.c，对应的头文件是 stm32f4xx_adc.h。

图 3-14　STM32F429 型号芯片部分片上外设驱动文件和头文件

misc.c 文件提供了外设对内核中的 NVIC（中断向量控制器）的访问函数，如果需要使用中断，就必须把这个文件添加到工程。

图 3-14 显示了 STM32F429 型号芯片部分片上外设驱动文件和头文件。

与内核相关的头文件包括 core_cm4.h、core_cmFunc.h、core_cmInstr.h、core_cmSimd.h。这些文件是与 Cortex-M 内核设备相关的头文件，它们为采用 Cortex-M 内核设计 SoC 的芯片生产商设计的芯片外设提供了一个进入内核的接口，定义了一些与内核相关的寄存器。对于相同 Cortex-M 内核的不同芯片生产商的微控制器芯片，这些文件是一样的。在同目录下还有几个文件是 DSP 函数库使用的头文件。

stm32f4xx_it.c 文件是专门用来编写中断服务函数的。这个文件默认已经定义了一些系统异常（特殊中断）的接口函数，其他普通中断服务函数需要用户自己添加。中断服务函数放置的位置不是必须放在 stm32f4xx_it.c 文件中，可以在用户自定义的程序文件中定义，但是，中断服务函数名不能随便定义，需要和启动文件中的对应中断服务函数名保持一致。stm32f4xx_it.c 文件对应的头文件是 stm32f4xx_it.h。

stdint.h 文件是 C99 中的一个标准头文件，是独立于处理器之外的，位于 MDK-ARM 软件的安装目录下，主要作用是提供一些数据类型定义，在编程过程中经常用到这些定义的数据类型。这些新类型定义屏蔽了在不同芯片平台时，数据类型的差异（如不同平台下 int 类型数据可能是 16 位的也可能是 32 位的）。在不同平台上移植程序时，只需要改变 stdint.h 文件中的数据类型定义即可，这样简化了程序的移植过程。

core_cm4.h 和 stm32f4xx.h 文件包含了 stdint.h 这个头文件。

```
/*exact-width signed integer types*/
typedef   signed          char int8_t;
typedef   signed short    int int16_t;
typedef   signed          int int32_t;
typedef   signed          __INT64 int64_t;

/*exact-width unsigned integer types*/
typedef   unsigned        char uint8_t;
```

```
typedef   unsigned short    int uint16_t;
typedef   unsigned          int uint32_t;
typedef   unsigned          __INT64 uint64_t;
```

3.2 STM32F429 微控制器结构

STM32F429微控制器属于STM32F4系列微控制器，采用了最新的 180MHz 的Cortex-M4处理器内核，可取代当前基于微控制器和中低端独立数字信号处理器的双片解决方案，或者将两者整合成一个基于标准内核的数字信号控制器。微控制器与数字信号处理器整合还可提高能效，让用户使用支持 STM32 的强大研发生态系统。STM32 全系列产品在引脚、软件和外设上相互兼容，并配有巨大的开发支持生态系统，包括例程、设计 IP、低成本的探索工具和第三方开发工具，可提升设计系统扩展和软、硬件再用的灵活性，使 STM32 平台的投资回报率最大化。因此，与STM32F429微控制器的相关结构、原理及使用方法适用于其他STM32F4系列微控制器，对于使用相同封装形式和相同的功能的片上外设应用来讲，代码和电路可以公用。后面章节均以STM32F429微控制器为对象讲解芯片的内部构造、片上外设工作原理及应用方法。

图 3-15　STM32F429IGT6 芯片的外观图

STM32F429 微控制器的封装形式有 LQFP100（14mm×14mm）、LQFP144（20mm×20mm）、UFBGA169（7mm×7mm）、LQFP176（24mm× 24mm）、LQFP208（28mm×28mm）、UFBGA176（10mm×10mm）、TFBGA216（13mm×13mm）、WLCSP143。一般来讲，引脚数量越大可用的片内资源越多，在实际使用中根据应用需求选择合适的芯片。本书以 STM32F429IGT6 芯片为例进行讲解，对应的封装形式是 LQFP176（24mm×24mm）。STM32F429IGT6 芯片的外观图如图 3-15 所示。

3.2.1 芯片资源

STM32F429IGT6 芯片使用的是 Cortex-M4 处理器内核，支持 FPU 指令和 DSP 指令，集成了丰富的片上外设，能够满足大部分嵌入式系统设计应用。该芯片内部的资源如下。

1．内核

（1）32 位高性能 Cortex-M4 处理器。

（2）时钟：频率高达 180MHz。

（3）支持 FPU（浮点运算）和 DSP 指令。

2．I/O 端口

（1）共 176 个引脚，有 140 个 I/O 引脚。

（2）大部分 I/O 引脚都耐 5V（模拟通道除外）。

3．存储器容量

1024KB Flash，256KB SRAM。

4．LCD 控制器

分辨率为800像素×600像素，具备专用于图像处理的专业DMA——Chrom-Art Accelerator™（DMA2D）。

5．时钟、复位和电源管理

（1）1.8～3.6V 电源和 I/O 端口工作电压。

（2）上电复位、掉电复位和可编程的电压监控。

（3）强大的时钟系统：外部高速晶振、内部高速 RC 振荡器、内部低速 RC 振荡器、看门狗时钟、内部锁相环（倍频，其输出作为系统时钟）、外部低速晶振（主要作 RTC 时钟源）。

6．低功耗

睡眠、停止和待机三种低功耗模式。

7．模数转换器（ADC）

3 个 12 位 ADC（多达 24 个外部测量通道）。

8．数模转换器（DAC）

2 个 12 位 DAC。

9．直接内存存取（DMA）

16 个 DMA 通道，带 FIFO 和突发支持。

10．定时器

多达 17 个定时器。

11．通信接口

多达 21 个通信接口。

3 个 I2C 接口、8 个 USART、6 个 SPI 接口、1 个 SAI 接口、2 个 CAN2.0、1 个 SDIO。

12．USB OTG

2 个 USB OTG。

13．以太网媒体接入控制器（MAC）

1 个以太网媒体接入控制器。

3.2.2 芯片内部结构

STM32F429IGT6 芯片主系统由 32 位多层 AHB 总线矩阵构成，STM32F429 IGT6 芯片内部通过 8 条主控总线（S0～S7）和 7 条被控总线（M0～M6）组成的总线矩阵将 Cortex-M4 内核、存储器及片上外设连在一起。

1．8 条主控总线

（1）Cortex-M4 内核 I 总线、D 总线和 S 总线（S0～S2）。

S0：I 总线。此总线用于将 Cortex-M4 内核的指令总线连接到总线矩阵。内核通过此总线获取指令。此总线访问的对象是包含代码的存储器（内部 Flash/SRAM 或通过 FSMC 的外部存储器）。

S1：D 总线。此总线用于将 Cortex-M4 内核的数据总线和 64KB CCM 数据 RAM 连接到总线矩阵。内核通过此总线进行立即数加载和调试访问。此总线访问的对象是包含代码或数据的存储器（内部 Flash 或通过 FSMC 的外部存储器）。

S2：S 总线。此总线用于将 Cortex-M4 内核的系统总线连接到总线矩阵。此总线用于访问位于外设或 SRAM 中的数据。也可通过此总线获取指令（效率低于 I 总线）。此总线访问的对象是内部 SRAM（112KB、64KB 和 16KB）、包括 APB 外设在内的 AHB1 外设和 AHB2 外设，以及通过 FSMC 的外部存储器。

（2）DMA1 存储器总线、DMA2 存储器总线（S3、S4）。

S3、S4：DMA 存储器总线。此总线用于将 DMA 存储器总线主接口连接到总线矩阵。DMA 通过此总线来执行存储器数据的传入和传出。此总线访问的对象是如下数据存储器：内部 SRAM（112KB、64KB、16KB）及通过 FSMC 的外部存储器。

（3）DMA2 外设总线（S5）。

S5：DMA2 外设总线。此总线用于将 DMA2 外设总线主接口连接到总线矩阵。DMA 通过此总线访问 AHB 外设或执行存储器间的数据传输。此总线访问的对象是 AHB 和 APB 外设及数

据存储器（内部 SRAM 及通过 FSMC 的外部存储器）。

（4）以太网 DMA 总线（S6）。

S6：以太网 DMA 总线。此总线用于将以太网 DMA 主接口连接到总线矩阵。以太网 DMA 通过此总线向存储器存取数据。此总线访问的对象是如下数据存储器：内部 SRAM（112KB、64KB 和 16KB）及通过 FSMC 的外部存储器。

（5）USB OTG HS DMA 总线（S7）。

S7：USB OTG HS DMA 总线。此总线用于将 USB OTG HS DMA 主接口连接到总线矩阵。USB OTG DMA 通过此总线向存储器加载/存储数据。此总线访问的对象是如下数据存储器：内部 SRAM（112KB、64KB 和 16KB）及通过 FSMC 的外部存储器。

2．7 条被控总线

（1）内部 Flash I 总线（M0）。

（2）内部 Flash D 总线（M1）。

（3）主要内部 SRAM1（112KB）总线（M2）。

（4）辅助内部 SRAM2（16KB）总线（M3）。

（5）辅助内部 SRAM3（64KB）总线（仅适用于 STM32F42 系列和 STM32F43 系列器件）（M7）。

（6）AHB1 外设（包括 AHB-APB 总线桥和 APB 外设）总线（M5）。

（7）AHB2 外设总线（M4）。

（8）FSMC 总线（M6）。FSMC 借助总线矩阵，可以实现主控总线到被控总线的访问，这样即使在多个高速外设同时运行期间，系统也可以实现并发访问和高效运行。

主控总线所连接的设备是数据通信的发起端，通过矩阵总线可以和与其相交被控总线上连接的设备进行通信。例如，Cortex-M4 内核可以通过 S0 总线与 M0 总线、M2 总线和 M6 总线连接 Flash、SRAM1 及 FSMC 进行数据通信。STM32F429IGT6 芯片总线矩阵结构图如图 3-16 所示。

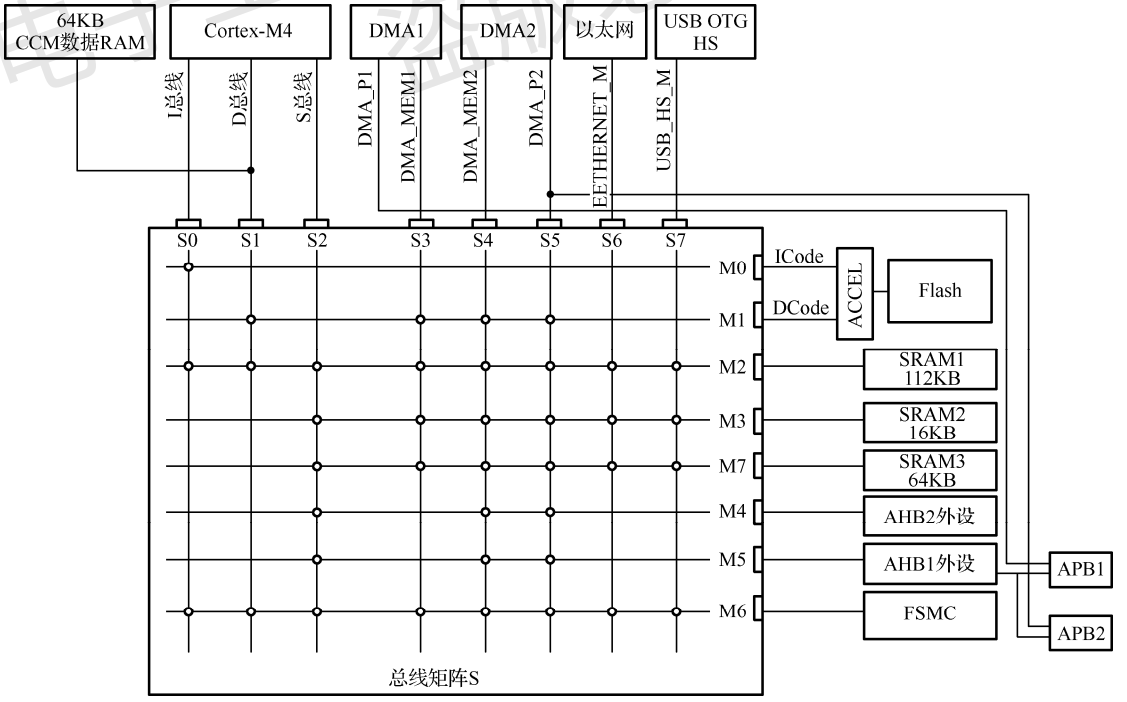

图 3-16　STM32F429IGT6 芯片总线矩阵结构图

在总线矩阵的互联下，将芯片内部的所有设备连接在一起，形成如图 3-17 所示的芯片内部构造结构图。

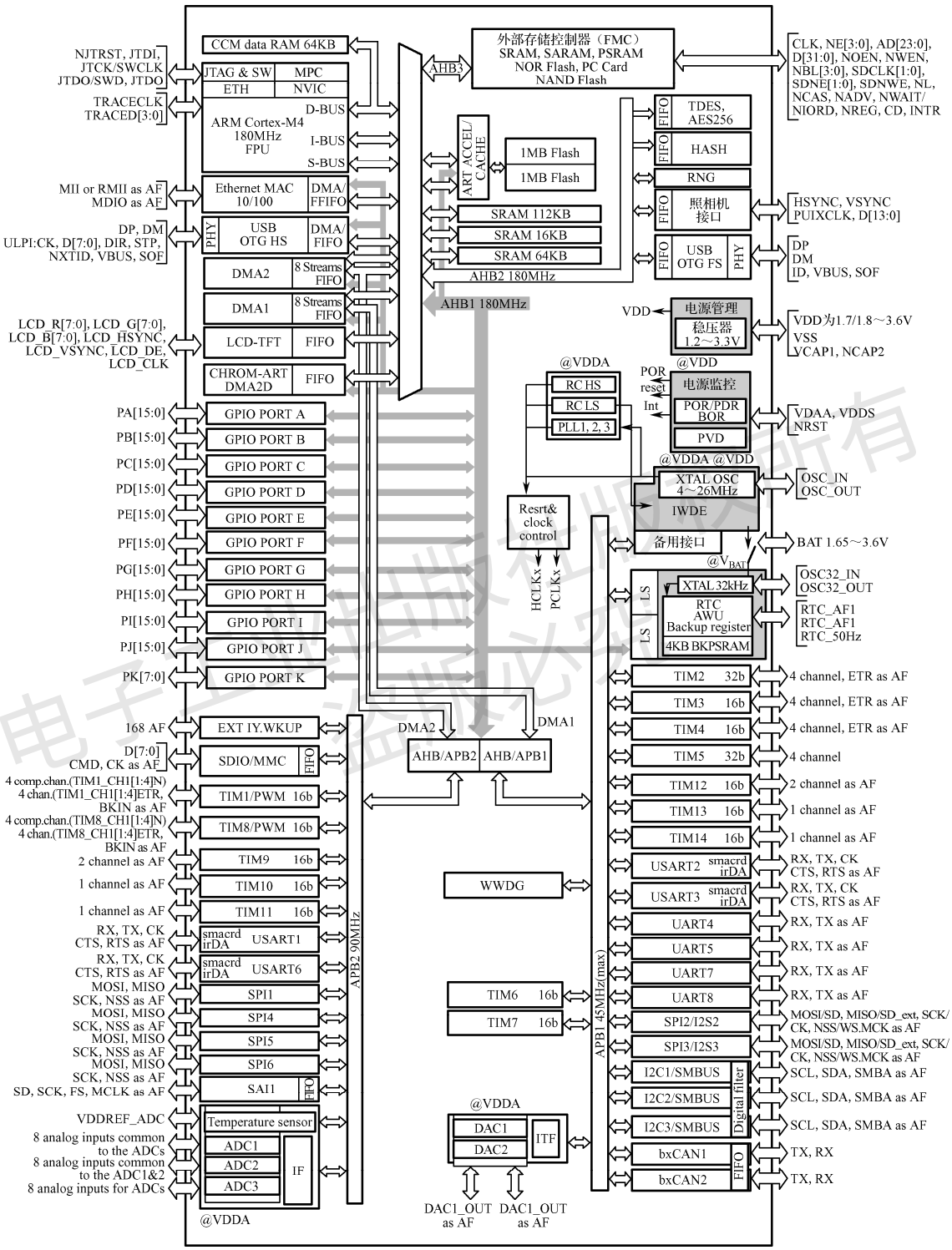

图 3-17　STM32F429IGT6 芯片内部构造结构图

3.2.3 芯片引脚和功能

STM32F429IGT6 芯片引脚示意图如图 3-18 所示。图 3-18 只列出了每个引脚的基本功能。但是，由于芯片内部集成功能较多，实际引脚有限，因此多数引脚为复用引脚（一个引脚可复用为多个功能）。例如，56 号引脚可以作为 PB0、TIM1_CH2N、TIM3_CH3、TIM8_CH2N、LCD_R3、OTG_HS_ULPI_D1、ETH_MII_RXD2、EVENTOUT、ADC12_IN8。对于每个引脚的功能定义请查看《STM32F427XX、STM32F429XX 数据手册》。

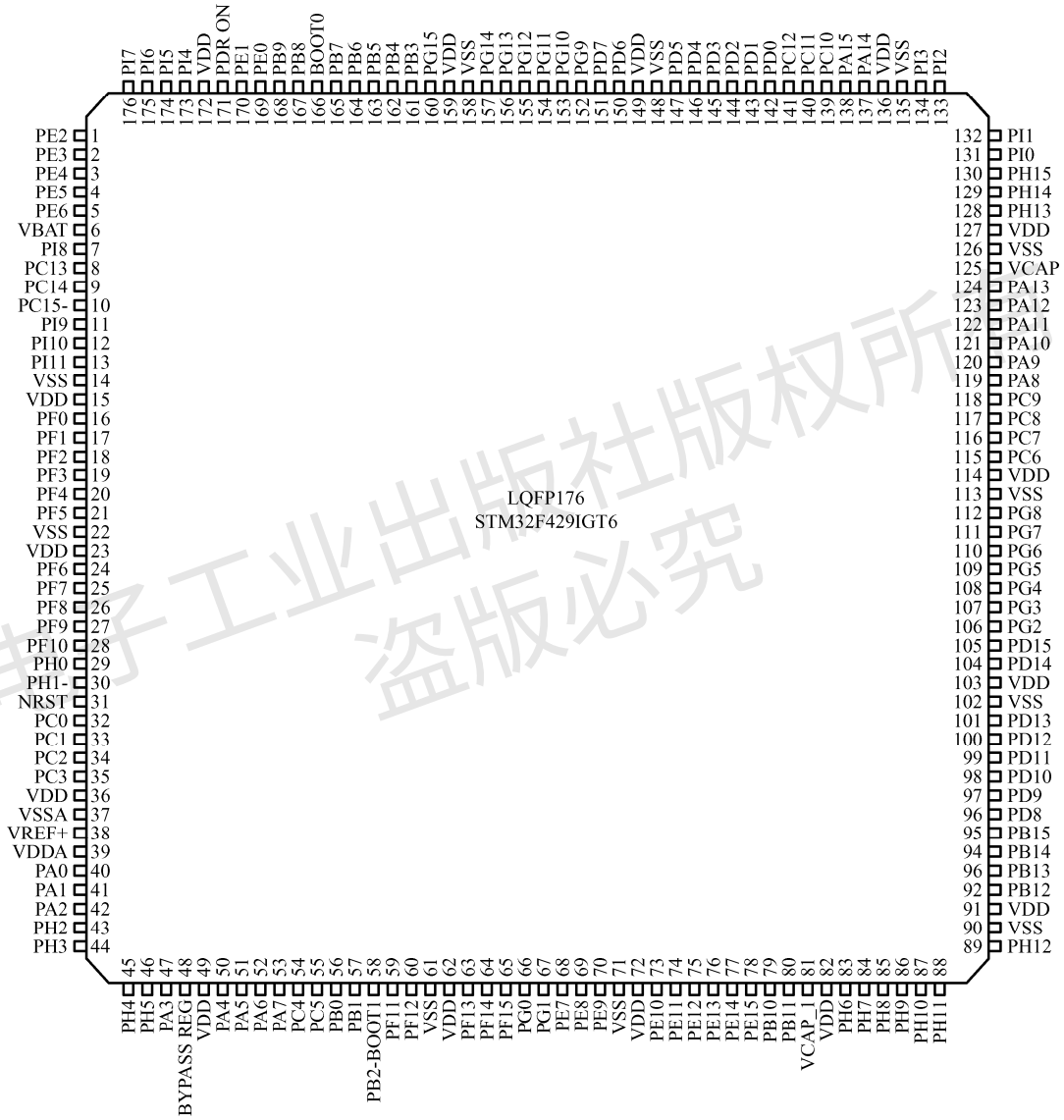

图 3-18　STM32F429IGT6 芯片引脚示意图

对微控制器引脚的说明主要从引脚序号、引脚名称、引脚类型、I/O 结构、注意事项、复用功能及额外功能等方面进行。微控制器引脚说明如表 3-6 所示。

表 3-6　微控制器引脚说明

名称	说明
引脚序号	阿拉伯数字表示 LQFP 封装，英文字母+阿拉伯数字表示 BGA 封装
引脚名称	指复位状态下的引脚名称
引脚类型	S 表示电源引脚
	I 表示输入引脚
	I/O 表示输入/输出引脚
I/O 结构	FT 表示兼容 5V
	TTa 表示只支持 3V3，且直接到 ADC
	B 表示 BOOT 引脚
	RST 表示复位引脚，内部带弱上拉引脚
注意事项	对某些 I/O 引脚要注意的事项的特别说明
复用功能	I/O 引脚的复用功能，通过 GPIO*x*_AFR 寄存器来配置选择。一个 I/O 引脚可以复用成多个功能
额外功能	I/O 引脚的额外功能，通过直连的外设寄存器配置来选择

引脚类型主要有电源引脚、晶振 I/O 引脚、下载 I/O 引脚、BOOT I/O 引脚、复位 I/O 引脚及 GPIO 引脚等，具体如表 3-7 所示。

表 3-7　微控制器引脚类型

引脚分类	引脚说明
电源引脚	V_{BAT}、V_{DD}、V_{SS}、V_{DDA}、V_{SSA}、V_{REF+}、V_{REF-}等
晶振 I/O 引脚	主晶振 I/O 引脚、RTC 晶振 I/O 引脚
下载 I/O 引脚	用于 JTAG 下载的 I/O 引脚：JTMS、JTCK、JTDI、JTDO、NJTRST
BOOT I/O 引脚	BOOT0、BOOT1，用于设置系统的启动方式
复位 I/O 引脚	NRST，用于外部复位
GPIO 引脚	专用器件接到专用的总线，比如 I2C、SPI、SDIO、FSMC、DCMI，这些总线的器件需要接到专用的 I/O 引脚，普通的元器件接到 GPIO 引脚，如蜂鸣器、LED 等元器件用普通的 GPIO 引脚，如果还有剩下的 I/O 引脚，可根据项目需要引出或者不引出

在表 3-7 中由前 5 部分 I/O 引脚组成的系统也叫作最小系统。

STM32F4 系列微控制器的所有标准输入引脚都是 CMOS 的，但与 TTL 兼容。

STM32F4 系列微控制器的所有容忍 5V 电压的输入引脚都是 TTL 的，但与 CMOS 兼容。

在输出模式下，在供电电压 2.7～3.6V 的范围内，STM32F4 系列微控制器所有的输出引脚都是与 TTL 兼容的。

表 3-8 列出了部分引脚的功能，其中 1 号引脚复位功能是 PE2（GPIOE 的 2 号引脚），引脚类型是 I/O 类型，具有容忍 5V 电压功能，并能复用成 TRACECLK、SPI4_SCK、SAI1_MCLK_A、ETH_MII_TXD3、FMC_A23、EVENTOUT 功能，大部分 GPIO 引脚都具备类似 PE2 的功能。6 号、14 号、15 号引脚是电源引脚，分别是 V_{BAT}、V_{SS}（电源负极）、V_{DD}（电源正极），除这几个引脚外，还有其他电源引脚。31 号引脚是复位引脚。166 号引脚是 STM32F429IGT6 微控制器的自举控制引脚。更多引脚功能请查阅《STM32F427XX、STM32F429XX 数据手册》。

表 3-8　部分引脚的功能说明

序号	名称	类型	I/O 结构	复用功能
1	PE2	I/O	FT	TRACECLK，SPI4_SCK，SAI1_MCLK_A，ETH_MII_TXD3，FMC_A23，EVENTOUT
6	V_{BAT}	S	—	无
14	V_{SS}	S	—	无
15	V_{DD}	S	—	无
31	NRST	I/O	RST	无
166	BOOT0	I	B	无

3.2.4　电源系统

STM32F429 微控制器的工作电压（V_{DD}）范围为 1.8～3.6V。嵌入式线性调压器用于提供内部 1.2V 数字电源。当主电源 V_{DD} 断电时，可通过 V_{BAT} 引脚为实时时钟（RTC）、RTC 备份寄存器和备份 SRAM（BKP SRAM）供电。电源系统主要分为备份电路、ADC 电路及调压器主供电电路三部分。STM32F429 微控制器内部电源系统结构图如图 3-19 所示。

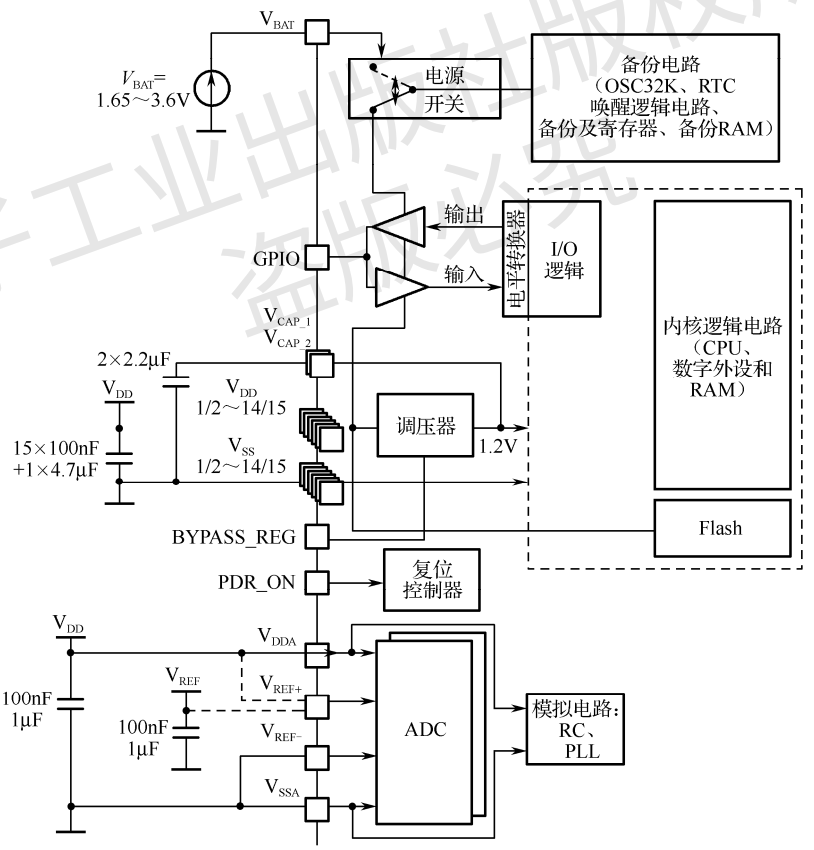

图 3-19　STM32F429 微控制器内部电源系统结构图

1．独立 ADC 电源和参考电压

为了提高转换精度，ADC 配有独立电源，可以单独滤波并屏蔽 PCB 上的噪声。

ADC 电源电压从单独的 V_{DDA} 引脚输入。

V_{SSA} 引脚提供了独立的电源接地连接。

为了确保在测量低电压时具有更高的精度，用户可以在 V_{REF} 引脚上连接单独的 ADC 外部参考电压输入。V_{REF} 介于 1.8V 到 V_{DDA} 之间。

2．备份电路

要想系统的电源 V_{DD} 关闭后保留 RTC 备份寄存器和备份 SRAM 的内容并为 RTC 供电，需要将 V_{BAT} 引脚连接到通过电池或其他电源供电的可选备用电压。在实际电路设计中，一般用两个二极管来设计 V_{BAT} 电源供电：将两个二极管阴极连在一起，并与 V_{BAT} 引脚连接，其中一个二极管连接电池电源输出脚，另外一个二极管的阳极连接 V_{DD} 引脚。

3．调压器主供电电路

嵌入式线性调压器为除备份域和待机电路外的所有数字电路供电，包括内核和所有片上外设、总线等，调压器输出电压约为 1.2V。此调压器需要将两个外部电容连接到专用引脚 V_{CAP_1} 和 V_{CAP_2}，用于对 1.2V 输出电压进行纹波处理。根据应用模式的不同，可采用以下三种不同的模式工作。

① 运行模式，调压器为 1.2V 域（内核、存储器和数字外设）提供全功率。

② 停止模式，调压器为 1.2V 域提供低功率，保留寄存器和内部 SRAM 的内容。

③ 待机模式，调压器掉电。除待机电路和备份域外，寄存器和 SRAM 的内容都将丢失。

当 BYPASS_REG 引脚连接 V_{DD} 引脚时，调压器被禁止，这时就需要通过 V_{CAP_1} 和 V_{CAP_2} 两个引脚提供 1.2V 工作电源。

STM32F429 微控制器内部具有电源监控器，用于检测 V_{DD} 引脚的电压，以实现复位功能及掉电紧急处理功能，保证系统可靠地运行，检测功能包括上电复位（POR）、掉电复位（PDR）、欠压复位（BOR）和可编程电压检测器（PVD）。要想使能复位控制器，需要将 PDR_ON 引脚连接到 V_{DD} 引脚。

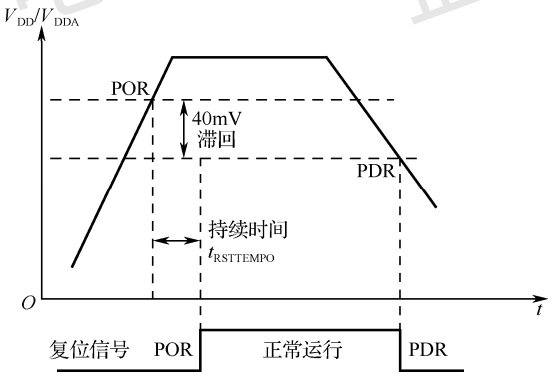

图 3-20　POR 和 PDR 状态图

1）上电复位与掉电复位

上电复位（Power On Reset，POR）功能是在 V_{DD} 由低向高上升越过规定的阈值（典型值为 1.72V）之前，保持芯片复位，在越过这个阈值后的一小段时间（$t_{RSTTEMPO}$，最大值为 3ms，最小值为 0.5ms）后，结束复位并取复位向量，开始执行指令。掉电复位（Power Down Reset，PDR）功能是在 V_{DD} 由高向低下降越过规定的阈值（典型值为 1.68V）后，将在芯片内部产生复位。

POR 和 PDR 状态图如图 3-20 所示。

2）欠压复位

通过设定欠压复位（Brownout Reset，BOR）的电压阈值，可以实现灵活的电压监控方案。在 V_{DD} 由低向高上升越过规定的阈值上限之前，保持芯片复位。在 V_{DD} 由高向低下降越过规定的阈值下限后，将在芯片内部产生复位。BOR 状态图如图 3-21 所示。通过对器件选项字节进行配置可以设定 BOR 的级别。可设置的 BOR 阈值如表 3-9 所示。

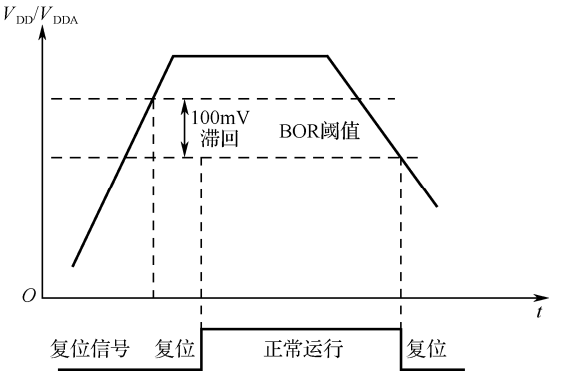

图 3-21　BOR 状态图

表 3-9　可设置的 BOR 阈值

级别	条件	典型值/V
1 级 BOR 阈值	下降沿	2.19
	上升沿	2.29
2 级 BOR 阈值	下降沿	2.5
	上升沿	2.59
3 级 BOR 阈值	下降沿	2.83
	上升沿	2.92

3）可编程电压检测器

可编程电压检测器（Programmable Votage Detector，PVD）的作用是监视供电电压，在 V_{DD} 下降到给定的阈值以下时，PVD 产生一个中断，通知软件做紧急处理。当 V_{DD} 又恢复到给定的阈值以上时，PVD 也会产生一个中断，通知软件供电恢复。供电下降的阈值与供电上升的 PVD 阈值有一个固定的差值，引入这个差值的目的是防止因电压在阈值上下小幅抖动而频繁地产生中断。PVD 状态图如图 3-22 所示。

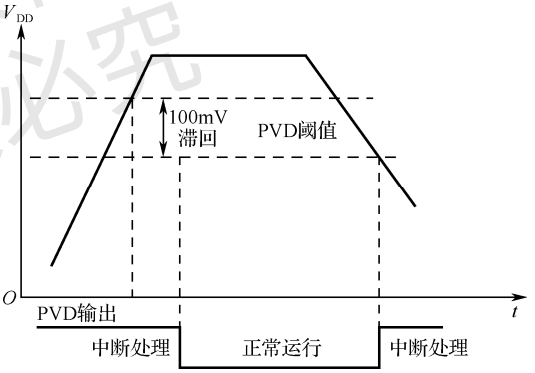

图 3-22　PVD 状态图

可设置的 PVD 阈值如表 3-10 所示。

表 3-10　可设置的 PVD 阈值

级别	条件	典型值/V	级别	条件	典型值/V
0 级 PVD 阈值	下降沿	2.14	4 级 PVD 阈值	下降沿	2.76
	上升沿	2.04		上升沿	2.66
1 级 PVD 阈值	下降沿	2.3	5 级 PVD 阈值	下降沿	2.93
	上升沿	2.19		上升沿	2.84
2 级 PVD 阈值	下降沿	2.45	6 级 PVD 阈值	下降沿	3.03
	上升沿	2.35		上升沿	2.93
3 级 PVD 阈值	下降沿	2.6	7 级 PVD 阈值	下降沿	3.14
	上升沿	2.51		上升沿	3.03

3.2.5 复位系统

STM32F429 微控制器的复位共有三种类型，分别为系统复位、电源复位和备份域复位。除 RCC 时钟控制和状态寄存器（RCC_CSR）中的复位标志和备份域中的寄存器外，系统复位会将其他寄存器全部都复位为复位值。RESET 复位入口向量在存储器映射中固定在地址 0x00000004。

1．系统复位

当发生以下事件之一时，就会产生系统复位。

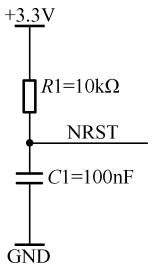

图 3-23　RC 复位电路

（1）NRST 引脚低电平（外部复位）。

（2）窗口看门狗计数结束（WWDG 复位）。

（3）独立看门狗计数结束（IWDG 复位）。

（4）软件复位（SW 复位）。

（5）低功耗管理复位。

使用 NRST 引脚低电平进行复位，需要设计一个复位电路。这个复位电路可以使用 RC 复位电路或专用复位芯片电路来实现。RC 复位电路如图 3-23 所示。

要对器件进行软件复位，必须将 Cortex-M4 应用中断和复位控制寄存器中的 SYSRESETREQ 位置 1，并可通过查看 RCC_CSR 中的复位标志确定。

2．电源复位

当发生以下事件之一时，就会产生电源复位。

（1）POR、PDR 或 BOR。

（2）在退出待机模式时。

除备份域中的寄存器外，电源复位会将其他寄存器设置为复位值。芯片内部的复位信号会在 NRST 引脚上输出。脉冲发生器用于保证复位脉冲持续时间，可确保每个内部复位源的复位脉冲都至少持续 20μs。

3．备份域复位

备份域复位会将所有 RTC 寄存器和 RCC 备份域控制寄存器（RCC_BDCR）复位为各自的复位值。BKPSRAM 不受此复位影响。备份域复位内部结构图如图 3-24 所示。

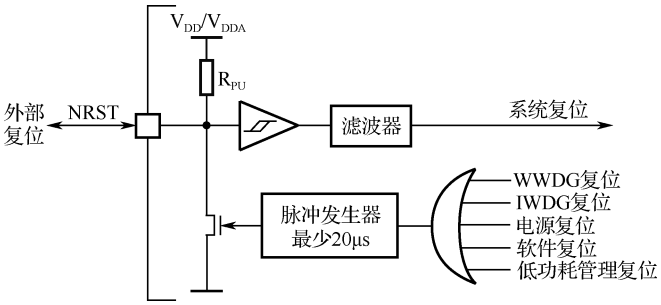

图 3-24　备份域复位内部结构图

当发生以下事件之一时，就会产生备份域复位。

（1）软件复位，通过将 RCC_BDCR 中的 BDRST 位置 1 触发。

（2）在电源 V_{DD} 和 V_{BAT} 都已掉电后，其中任何一个又再上电。

3.3 STM32F4 系列微控制器存储器映射和寄存器

3.3.1 存储器映射

存储器映射是指把程序存储器、数据存储器和寄存器等按照统一编址，分配在 4GB 地址空间内，用地址来表示对象。地址绝大多数是由厂家规定好的，用户只能用不能改。用户只能在外部扩展的 RAM 或 Flash 的情况下，对存储空间进行自定义。

STM32F429 微控制器的 4GB 可寻址的存储空间分为 8 个块，每个块 0.5GB。STM32F429 微控制器存储空间映射图如图 3-25 所示。

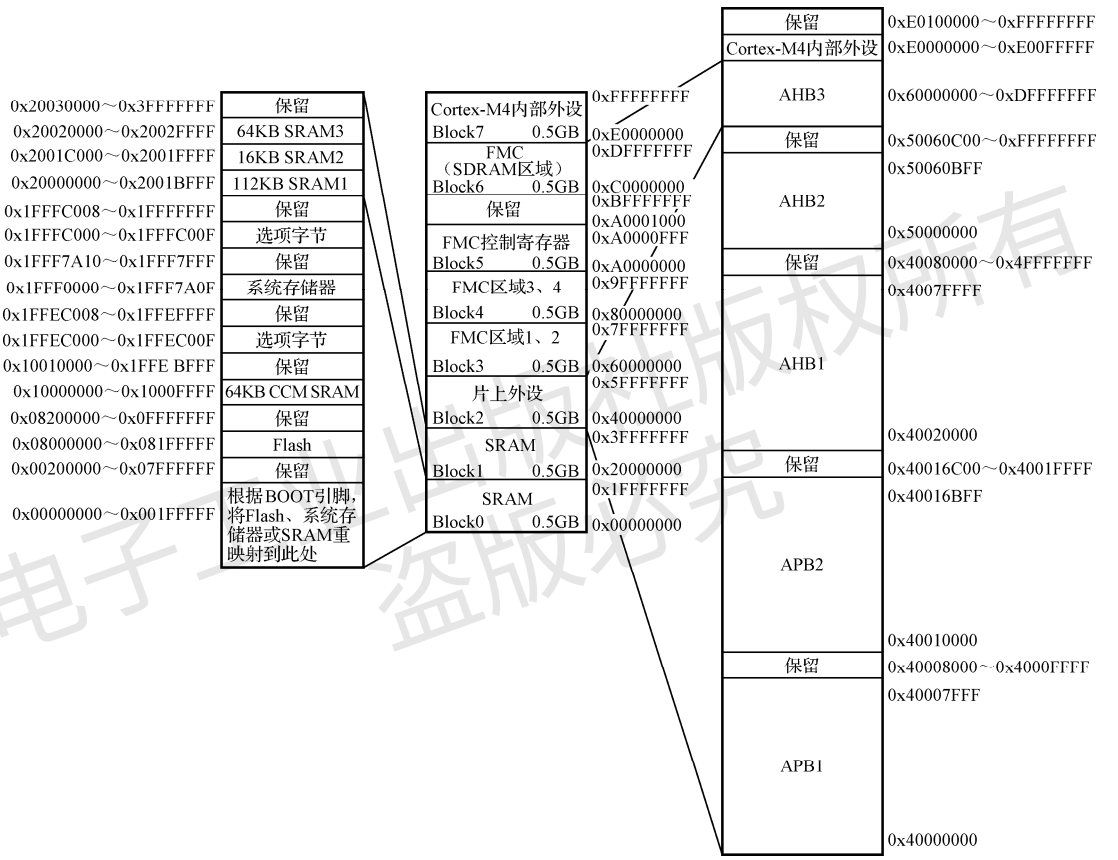

图 3-25　STM32F429 微控制器存储空间映射图

1．Block0 存储块功能

这一区域是代码区，我们编写的代码主要在这一区域运行，并集成了 Flash、CCM 及系统自举 Bootloader 等。

（1）0x00000000～0x001FFFFF：2MB。Cortex-M4 的复位地址是 0x00000000，而实际存储代码的位置可能在不同的存储介质和不同的存储位置，因此需要一种机制，使得存储其他位置的代码能够映射到 0x00000000 运行。STM32F429 微控制器可以根据 BOOT 引脚的不同设置，将 Flash、系统存储器或 SRAM 重映射到这一区域，以实现不同代码运行方式。

（2）0x08000000～0x081FFFFF：2MB。内部集成的 Flash，用户代码可以被烧写在此处。代码在内部 Flash 中运行是最常用的一种 BOOT 方式。

（3）0x10000000～0x1000FFFF：64KB。紧耦合（CCM）SRAM 区，没有被挂载在总线矩阵上，只能 CPU 访问，但访问速度很快。一般用于分配全局或静态变量、堆、栈空间。

（4）0x1FFEC000～0x1FFEC00F：16 个字节。0x1FFEC008～0x1FFEC00F 用作 Flash 扇区 12～23 的写保护。0x1FFEC000～0x1FFEC007 保留。

（5）0x1FFF0000～0x1FFF7A0F：0x1FFF0000～0x1FFF77FF 是系统存储器区，存储了意法半导体烧写的自举代码，通过这一部分代码可以实现程序的远程下载。0x1FFF7800～0x1FFF7A0F 是 528 个字节的一次编程区（One Time Programable，OTP）区域，前 512 个字节的 OTP（一次性可编程），用于存储用户数据 OTP 区域。最后 16 个额外字节，用于锁定对应的 OTP 数据块。

（6）0x1FFFC000～0x1FFFC00F：16 个字节。用于配置读写保护、BOR 级别、软件和硬件看门狗，以及器件处于待机或停止模式下的复位。如果芯片被锁住，那么可以在 RAM 中运行代码，修改相应的寄存器配置。0x1FFFC000～0x1FFFC007 是 ROP 和用户选项字节，0x1FFFC008～0x1FFFC00F 用作 Flash 扇区 0～11 的写保护。

2．Block1 存储块功能

这一存储块用于分配给内部集成的 SRAM，分为三部分：112KB 的 SRAM1、16KB 的 SRAM2 和 64KB 的 SRAM3，共 192KB。这一部分的 SRAM 被挂载在总线矩阵上，可以被 CPU 访问，也可以通过 DMA 控制器实现存储器与存储器之间，以及存储器与外设之间的数据通信。

SRAM1 通过 M2 总线与 I 总线、D 总线和 S 总线相连，因此 SRAM1 可以运行代码，也可以用作数据存储区。SRAM2 和 SRAM3 分别通过 M3 总线和 M7 总线与 S 总线连接，CPU 只能通过 S 总线访问这两个区域。

3．Block2 存储块功能

这一区域被分配给了片内外设的寄存器组。所有片上外设都挂载在高级高性能总线（Advanced High-performance Bus，AHB）和高级外设总线（Advanced Peripheral Bus，APB）上，CPU 通过 AHB 和 APB 访问片上外设，而 CPU 控制片上外设是通过访问片上外设对应的寄存器组实现的。AHB 分为 AHB1 和 AHB2，又通过 AHB1 的两个 AHB/APB 总线桥将 AHB1 连接到 APB1（低速）和 APB2（高速），从而实现 AHB 与两个 APB 之间完全同步连接。Block2 存储块功能分配如表 3-11 所示。

表 3-11　Block2 存储块功能分配

总线名称	总线地址范围	空间
APB1	0x40000000～0x40007FFF	32KB
APB2	0x40010000～0x40016BFF	27KB
AHB1	0x40020000～0x4007FFFF	384KB
AHB2	0x50000000～0x50060BFF	387KB

4．Block3～Block6 存储块功能

这几个存储块被分配给 AHB3，主要用于外部存储器的扩展，包括 SRAM、SDRAM、NOR Flash 和 NAND Flash。

1）Block3～Block4

（1）0x60000000～0x6FFFFFFF 是灵活的存储控制器（Flexible Memory Controller，FMC）的区域 1（Bank1），用于扩展 NOR Flash、PSRAM 和 SRAM。

（2）0x70000000～0x7FFFFFFF 和 0x80000000～0x8FFFFFFF 分别是 FMC 的区域 2（Bank2）

和区域 3（Bank3），用于扩展 NAND Flash。

（3）0x90000000～0x9FFFFFFF 是 FMC 的区域 4（Bank4），用于扩展 PC 卡。

2）Block6

0xC0000000～0xCFFFFFFF 和 0xD0000000～0xDFFFFFFF 是 SDRAM 的区域（Bank1 和 Bank2），用于扩展 SDRAM。

3）Block5

0xA0000000 开始分布了 FMC 的一些控制寄存器。

5．Block7 存储块功能

这个存储块的 0xE0000000～0xE00FFFFF 存储区域被分配给了 Cortex-M4 的内核寄存器，0xE00FFFFF～0xFFFFFFFF 的存储区域保留。

3.3.2 自举配置

存储器采用固定的存储器映射，代码区域起始地址为 0x00000000（通过 I 总线/D 总线访问），而数据区域起始地址为 0x20000000（通过系统总线访问）。Cortex-M4 CPU 始终通过 I 总线获取复位向量，这意味着只有代码区域（通常为 Flash）可以提供自举空间。STM32F4 系列微控制器实施一种特殊机制，可以从其他存储器（如内部 SRAM）进行自举。在 STM32F4 系列微控制器中，可通过 BOOT[1:0]引脚选择三种不同的自举模式。

复位后，用户可以通过设置 BOOT1 引脚和 BOOT0 引脚来选择需要的自举模式，如表 3-12 所示。BOOT0 引脚为专用引脚，而 BOOT1 引脚则与 GPIO 引脚共用。一旦完成对 BOOT1 引脚的采样，相应 GPIO 引脚即进入空闲状态，可用于其他用途。

表 3-12 自举模式配置

自举模式选择引脚		自举模式	自举空间
BOOT1	BOOT0		
x	0	主 Flash	选择主 Flash 作为自举空间
0	1	系统存储器	选择系统存储器作为自举空间
1	1	嵌入式 SRAM	选择嵌入式 SRAM 作为自举空间

例如，通常选择主 Flash 作为复位后指令执行的位置，那么就需要将 BOOT0 引脚接到低电平上。

如果器件从 SRAM 自举，那么在应用程序初始化代码中，需要使用 NVIC 异常及中断向量表和偏移寄存器来重新分配 SRAM 中的向量表。

当微控制器退出待机模式时，会对 BOOT 引脚重新采样。因此，当微控制器处于待机模式时，这些引脚必须保持所需的自举模式配置。在启动延迟结束后，CPU 将从地址 0x00000000 获取栈顶值，然后从始于 0x00000004 的自举存储器开始执行代码。

3.3.3 寄存器映射

把片上外设对应的寄存器在存储空间上分配地址的过程称为寄存器映射。与存储单元一样，每个寄存器（一般都是 32 位的）都有一个寻址地址，访问和操作寄存器和存储单元基本一致。寄存器是一个很重要的概念，应用程序对片上外设的初始化和控制都是通过对片上外设对应的一系列寄存器的修改、读写来实现的。可以说，寄存器是应用程序控制和操作硬件设备的接口。Cortex-M4 内核通过 S 总线经 M4 总线和 M5 总线，访问 AHB1、AHB2，以及 APB1 和 APB2 总线上挂载的片上外设的寄存器组，进而完成对相应片上外设的操作。所有片上外设寄存器组

都被分配在 Block2 中。

1．寄存器操作

以 GPIOA 的输出为例讲解寄存器的操作方法。控制 GPIOA 的输出功能的是数据输出寄存器（ODR），其地址是 0x40020014。每个 GPIO 端口有 16 个引脚，分别对应于 ODR 的低 16 位 ODR0～ODR15，高 16 位保留。GPIOA 的 ODR 如图 3-26 所示。

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

位31:16 保留，必须保持复位值。
位15:0 ODR*y*[15:0]：端口输出数据（*y*为0～15）这些位可通过软件读取和写入。 ④

图 3-26　GPIOA 的 ODR

图 3-26 中，①是寄存器中位段的操作权限，r 表示只读，w 表示只写，rw 表示可读可写；②是位段名；③是位段编号，从 0 开始；④是对寄存器各位段的使用说明，通过这一部分可以得到这一个寄存器所能实现的功能。

如果让 GPIOA 的 16 个引脚输出高电平，需要将 GPIOA 的 ODR 的低 16 位都设置为 1，通过以下的 C 语言程序实现：

```
*(unsigned int*)(0x40020014)=0xFFFF;//GPIOA 全部输出高电平
```

GPIOA 的 ODR 的地址是 0x40020014，但是这个地址在 C 语言编译器看来只是一个普通的变量，如果要使用地址 0x40020014 进行寄存器访问，则需要将 0x40020014 强制转换为指针，即(unsigned int *)(0x40020014)，其中 unsigned int 表示指针的类型是无符号整型。然后，使用指针运算符*进行指针操作，实现对 GPIOA 的 ODR 的操作。

为了方便记忆，通常使用宏定义对寄存器操作进行别名定义。例如：

```
#define   GPIOA_ODR     *(unsignedint*)(0x40021C14)
GPIOH_ODR=0xFFFF;
```

使用具有特定含义的别名，方便了对操作对象的记忆，增加了程序的可读性。STM32F4XX 的标准函数库使用大量的地址宏定义和结构体，实现了对微控制器片上外设寄存器的别名定义。

2．片上外设地址映射

在 STM32F4XX 的标准函数库中，片上外设寄存器的地址是通过如下形式得到的。

片上外设寄存器地址=片上外设存储块基地址（Block2 基地址，见图 3-25）+总线相对于片上外设存储块基地址的地址偏移+寄存器组相对于总线基地址的地址偏移+寄存器在寄存器组中的地址偏移。其中，片上外设存储块基地址= 0x40000000；APB1 总线相对于片上外设存储块基地址的地址偏移=0；APB2 总线相对于片上外设存储块基地址的地址偏移=0x00010000；AHB1 总线相对于片上外设存储块基地址的地址偏移=0x00020000；AHB2 总线相对于片上外设存储块基地址的地址偏移=0x10000000。

寄存器组相对于总线基地址的地址偏移和寄存器在寄存器组中的地址偏移根据芯片不同外设具体定义设置。

寄存器组相对于总线基地址的地址偏移在 stm32f4xx.h 文件中可以查到。

寄存器在寄存器组中的地址偏移可以在《STM32F4 参考手册》中各片上外设说明章节的最后一节中查到。

在此，以 GPIOA 为例，讲解 GPIOA 的寄存器组中各寄存器的地址。

GPIO 挂载在 AHB1 总线中，GPIO 的寄存器组相对于 AHB1 总线基地址的地址偏移如表 3-13 所示。

表 3-13　GPIO 的寄存器组相对于 AHB1 总线基地址的地址偏移

外设名称	外设寄存器组基地址	相对于 AHB1 总线基地址的地址偏移
GPIOA	0x40020000	0x0
GPIOB	0x40020400	0x00000400
GPIOC	0x40020800	0x00000800
GPIOD	0x40020C00	0x00000C00
GPIOE	0x40021000	0x00001000
GPIOF	0x40021400	0x00001400
GPIOG	0x40021800	0x00001800
GPIOH	0x40021C00	0x00001C00

GPIOA 对应的各个寄存器的地址偏移如表 3-14 所示。

表 3-14　GPIOA 对应的各个寄存器的地址偏移

寄存器名称	寄存器地址	相对于 GPIOA 寄存器组基地址的地址偏移
GPIOA_MODER	0x40020000	0x00
GPIOA_OTYPER	0x40020004	0x04
GPIOA_OSPEEDR	0x40020008	0x08
GPIOA_PUPDR	0x4002000C	0x0C
GPIOA_IDR	0x40020010	0x10
GPIOA_ODR	0x40020014	0x14
GPIOA_BSRR	0x40020018	0x18
GPIOA_LCKR	0x40020000	0x1C
GPIOA_AFRL	0x40020020	0x20
GPIOA_AFRH	0x40020024	0x24

综上所述，GPIOA 的 ODR 的地址是 GPIOA_ODR=0x40000000+0x00020000+0x0+ 0x14。

其中，0x40000000 是片上外设存储块基地址；0x00020000 是 AHB1 总线相对于片上外设存储块基地址的地址偏移；0x0 是 GPIOA 寄存器组相对于 AHB1 总线基地址的地址偏移；0x14 是 ODR 在 GPIOA 寄存器组中的地址偏移。

3．函数库对片上外设寄存器的封装

函数库使用各个片上外设寄存器组的基地址和寄存器组结构体实现对寄存器的封装和定义，这部分内容在函数库的 stm32f4xx.h 文件中能找到。在此以 GPIO 为例，说明封装的方法。

1）寄存器组基地址宏定义

stm32f4xx.h 文件定义了片上外设存储块基地址、总线基地址及各个片上外设寄存器组基地址，在此以 GPIO 为例给出相应的定义，更多内容请查阅《STM32F4 参考手册》。

```
/*片上外设存储块基地址*/
#define PERIPH_BASE              ((unsigned int)0x40000000)
/*总线基地址*/
#define APB1PERIPH_BASE          PERIPH_BASE
#define APB2PERIPH_BASE          (PERIPH_BASE+0x00010000)
#define AHB1PERIPH_BASE          (PERIPH_BASE+0x00020000)
```

```
#define AHB2PERIPH_BASE         (PERIPH_BASE+0x10000000)
/*GPIO 寄存器组基地址*/
#define GPIOA_BASE              (AHB1PERIPH_BASE+0x0000)
#define GPIOB_BASE              (AHB1PERIPH_BASE+0x0400)
#define GPIOC_BASE              (AHB1PERIPH_BASE+0x0800)
#define GPIOD_BASE              (AHB1PERIPH_BASE+0x0C00)
#define GPIOE_BASE              (AHB1PERIPH_BASE+0x1000)
#define GPIOF_BASE              (AHB1PERIPH_BASE+0x1400)
#define GPIOG_BASE              (AHB1PERIPH_BASE+0x1800)
#define GPIOH_BASE              (AHB1PERIPH_BASE+0x1C00)
```

2）片上外设寄存器组结构体封装

stm32f4xx.h 文件给各个片上外设寄存器组定义了结构体类型，在此以 GPIO 寄存器组的结构体定义进行说明，对应的自定义 GPIO 结构体类型是 GPIO_TypeDef，定义如下：

```
typedef struct {
    uint32_t MODER;     //GPIO 模式寄存器                    地址偏移：0x00
    uint32_t OTYPER;    //GPIO 输出类型寄存器                地址偏移：0x04
    uint32_t OSPEEDR;   //GPIO 输出速度寄存器                地址偏移：0x08
    uint32_t PUPDR;     //GPIO 上拉/下拉寄存器               地址偏移：0x0C
    uint32_t IDR;       //GPIO 输入数据寄存器                地址偏移：0x10
    uint32_t ODR;       //GPIO 输出数据寄存器                地址偏移：0x14
    uint16_t BSRRL;     //GPIO 置位/复位寄存器低 16 位部分   地址偏移：0x18
    uint16_t BSRRH;     //GPIO 置位/复位寄存器高 16 位部分   地址偏移：0x1A
    uint32_t LCKR;      //GPIO 配置锁定寄存器                地址偏移：0x1C
    uint32_t AFR[2];    //GPIO 复用功能配置寄存器            地址偏移：0x20～0x24
} GPIO_TypeDef;
```

GPIO_TypeDef 结构体中成员的定义顺序与 GPIO 寄存器组中每个成员的偏移顺序保持一致，所有 GPIO 的寄存器组封装相同。其中，成员 BSRRL 和 BSRRH 是置位/复位寄存器的低 16 位和高 16 位，成员 AFR[2]包含了 AFRL 和 AFRH 两个寄存器。

3）库函数中片上外设操作对象定义

结合各个 GPIO 的寄存器组基地址和 GPIO_TypeDef 结构体类型，定义出每个 GPIO 在库函数中的操作对象，定义如下：

```
/*把地址强制转换成 GPIO_TypeDef 类型地址*/
#define GPIOA                   ((GPIO_TypeDef *) GPIOA_BASE)
#define GPIOB                   ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOC                   ((GPIO_TypeDef *) GPIOC_BASE)
#define GPIOD                   ((GPIO_TypeDef *) GPIOD_BASE)
#define GPIOE                   ((GPIO_TypeDef *) GPIOE_BASE)
#define GPIOF                   ((GPIO_TypeDef *) GPIOF_BASE)
#define GPIOG                   ((GPIO_TypeDef *) GPIOG_BASE)
#define GPIOH                   ((GPIO_TypeDef *) GPIOH_BASE)
```

需要注意的是，每一个操作对象都是一个结构体指针。其他片上外设的操作对象定义方法相同，更多内容请查阅《STM32F4 参考手册》。

在此基础上，就可以使用定义的结构体指针访问对应的寄存器。例如，将 GPIOA 的引脚都设置为高电平，可以使用如下方法实现：

```
GPIOA->ODR=0xFFFF;
```

4）读—修改—写操作

在对寄存器进行操作的时候，经常需要修改寄存器的部分字段，但不能影响其他字段，这就需要通过读—修改—写的方式实现。这种操作可以只修改寄存器中部分目标字段，而不影响其他字段。

常用操作如下。

（1）位与：&，可实现目标字段的清零，而不影响其他字段。

一般格式：操作对象&=屏蔽字。

屏蔽字的目标字段设置为 0，其他位设置为 1。

（2）位或：|，可实现目标字段的置位，而不影响其他字段。

一般格式：操作对象|=屏蔽字。

屏蔽字的目标字段设置为 1，其他位设置为 0。

（3）异或：^，可实现目标字段的取反，而不影响其他字段。

一般格式：操作对象^=屏蔽字。

屏蔽字的目标字段设置为 1，其他位设置为 0。

例如，将 GPIOA 端口的 3 号、10 号引脚输出低电平，使用位与操作将 GPIOA 的 ODR 的 3 位和 10 位清零实现。

```
GPIOA->ODR&=~(1<<3 | 1<<10);
```

其中，~(1<<3 | 1<<10)是操作需要的屏蔽字，将 1 左移 3 位和 1 左移 10 位合并在一起，然后取反得到。

例如，将 GPIOA 端口的 2 号、8 号引脚输出高电平，使用位或操作将 GPIOA 的 ODR 的 2 位和 8 位置位实现。

```
GPIOA->ODR |= (1<<2) | (1<<8);
```

其中，(1<<2) | (1<<8)是操作需要的屏蔽字，将 1 左移 2 位和 1 左移 8 位合并在一起得到。

例如，将 GPIOA 端口的 1 号、11 号和 13 号引脚的电平反转，使用异或操作将 GPIOA 的 ODR 的 1 位、11 位和 13 位取反实现。

```
GPIOA->ODR ^= (1<<1) | (1<<11) |(1<<13);
```

其中，(1<<1) | (1<<11) |(1<<13)是操作需要的屏蔽字，将 1 左移 1 位、1 左移 11 位和 1 左移 13 位合并在一起得到。

习题

1．STM32F1 系列和 STM32F4 系列微控制器分别使用什么内核？

2．请列举 STM32F1 系列微控制器中 3 个以上具体型号微控制器（全称）。

3．请列举 STM32F4 系列微控制器中 3 个以上具体型号微控制器（全称）。

4．写出 MDK-ARM 的安装过程（最好自己安装一遍），并安装 STM32F4 系列微控制器的器件支持包。

5．请说明处理器中寄存器的功能。

6．请说明 STM32F103RCT6 微控制器和 STM32F407ZET6 微控制器命名中各部分含义。

7．请查询《STM32F429IGT6 数据手册》，写出 68 号引脚都有哪些功能，是否是耐 5V 引脚？

8．STM32F429IGT6 微控制器是否支持浮点运算功能？

9．STM32F429IGT6 微控制器内部集成的程序存储器空间大小为_____，SRAM 的空间大小为_____。

10．STM32F429IGT6 微控制器的工作电压范围是_____。

11．STM32F407ZET6 微控制器命名中的 32、F、429、I、G、T、6 的含义分别是什么？

12．STM32F29IGT6 微控制器内部是通过将_____内核和片上外设连在一起的。

13．STM32F29IGT6 微控制器的引脚有哪些类型？

14．一个 STM32F29IGT6 微控制器最小系统板一般包含哪些器件？

15．列举出一个你知道的开发板型号，并指出上面使用的微控制器型号和内核。

16．写出 STM32F429IGT6 微控制器内部 Flash 的地址范围。

17．写出 TIM1 和 USART1 对应的寄存器占用的地址范围。

18．假设一个寄存器的地址为 0x40000400，怎么将数据 0x55aa 写入这一寄存器？

19．将一个数据中个别位清零，其他位保持不变，使用逻辑_____操作；将一个数据中个别位置位，其他位保持不变，使用逻辑_____操作；将一个数据中个别位取反，其他位保持不变，使用逻辑_____操作。

20．如何将 GPIOH 端口（包括 16 引脚，编号为 0～15）的 0 号、2 号、3 号、6 号、7 号引脚设置为低电平，将 1 号、8 号、9 号引脚设置为高电平？请使用两条 C 指令完成。

21．已知 GPIOH 的 ODR 的地址是 0x40021C14，怎么通过 C 程序实现 GPIOH 端口的全部引脚输出低电平？

22．已知 GPIOH 的 ODR 的地址是 0x40021C14，怎么通过 C 程序实现 GPIOH 端口的 2 号、3 号、6 号、8 号引脚输出高电平，而其他引脚电平不变？

23．已知 GPIOH 的 ODR 的地址是 0x40021C14，怎么通过 C 程序实现 GPIOH 端口的 1 号、4 号、9 号、11 号引脚输出低电平，而其他引脚电平不变？

24．已知 GPIOH 的 ODR 的地址是 0x40021C14，怎么通过 C 程序实现 GPIOH 端口的 2 号、4 号、6 号、8 号引脚输出电平状态反转，而其他引脚电平不变？

25．已知 GPIOH 的 IDR 的地址是 0x40021C10，怎么通过 C 程序实现 GPIOH 端口的 2 号引脚电平读取到微控制器内部？假设已经定义了一个变量 PIN_Level 用于存储读取的结果。

26．使用库文件中定义的 GPIO 宏，实现第 21～25 题的功能。

27．STM32 的标准函数库的 CMSIS 层包含_____和_____两部分。

28．STM32F429 的异常/中断向量表在_____文件中，NVIC 相关操作在_____文件中，系统时钟初始化在_____文件中。

29．在 stm32f4xx.h 文件中，GPIOA 的定义形式是_____，GPIO_TypeDef 结构体类型所维护的成员是_____。

30．在 stm32f4xx_GPIO.h 文件中，GPIO_Pin_0 的定义形式是_____。