第三章 TM4C123GH6PM 微控制器

微控制器是由处理器（CPU）和存储器、以及一些外设（Peripheral）集成到一片芯片里构成的。外设和 CPU 相对独立，且以寄存器的形式呈现给 CPU，CPU 通过总线对寄存器进行读写，实现对外设的配置和数据通信。常用的外设包括通用输入输出 GPIO、定时器 TIMER、模拟输入输出（ADC、DAC）、串行通信（UART、SSI、I2C、CAN）等等。

1 TM4C123GH6PM 微控制器的结构

Tiva 系列 TM4C123x 是 TI 公司推出的一系列基于 ARM Cortex-M4F 的微控制器；其工作速率达 80 MHz，集成了 256KB Flash 及 32KB SRAM，具备可编程 GPIO 和能使用备用电池的休眠模块。Tiva 系列微控制器面向各种高性能、低成本和低功耗的应用，如：便携设备、运动控制、测试仪表和工厂自动化。

本书使用 TM4C123GH6PM 微控制器作为学习对象，采用 EK-TM4C123GXL LaunchPad 开发板作为实验教学装置。TM4C123GH6PM 微控制器的工作性能和内部外设如表 3-1 所示。

表 3-1 TM4C123GH6PM

特性	描述
性能	
内核	ARMCortex-M4F 处理器核心
性能	80-MHz 运行速度；100DMIPS 性能
Flash	256 KB 单周期 Flash 存储器
系统 SRAM	32 KB 单周期访问的 SRAM
EEPROM	2KB EEPROM
内置 ROM	搭载 TivaWare™（适用于 C 系列）软件的内置 ROM
通信接口	
通用异步收发器 (UART)	8 个 UART
同步串行接口（SSI）	4 个 SSI 模块
内部集成电路 (I2C)	6 个 I2C 模块，具有四种传输速率（包括高速模式）
控制器局域网 (CAN)	2 个 CAN 2.0 A/B 控制器
USB	USB 2.0 OTG/Host/Device
系统集成	
微型直接存储器访问 (µDMA)	ARM®PrimeCell® 32 通道的可配置 uDMA 控制器
通用定时器 (GPTM)	6 个 16/32 位 GPTM 模块和 6 个 32/64 位宽 GPTM 模块
看门狗定时器 (WDT)	2 个看门狗定时器
休眠模块 (HIB)	带备用电池的低功耗休眠模块
通用输入/输出端口 (GPIO)	6 个物理 GPIO 模块
运动控制	
PWM	2 个 PWM 模块，共 16 个高级 PWM 信号
QEI	2 QEI 模块
模拟支持	
模-数转换器（ADC）	2 个 12 位 ADC 模块，每个的最大采样速率达 1M 次采样每秒

模拟比较器	2 个独立集成的模拟比较器
数字比较器	16 个数字比较器
JTAG 和串行线调试（SWD）	一个 JTAG 模块，带集成的 ARM SWD
封装信息	
封装	64 管脚 LQFP
工作温度范围（环境）	工业（-40°C 到 85°C）温度范围

TM4C123GH6PM 的结构框图如图 3-1 所示：

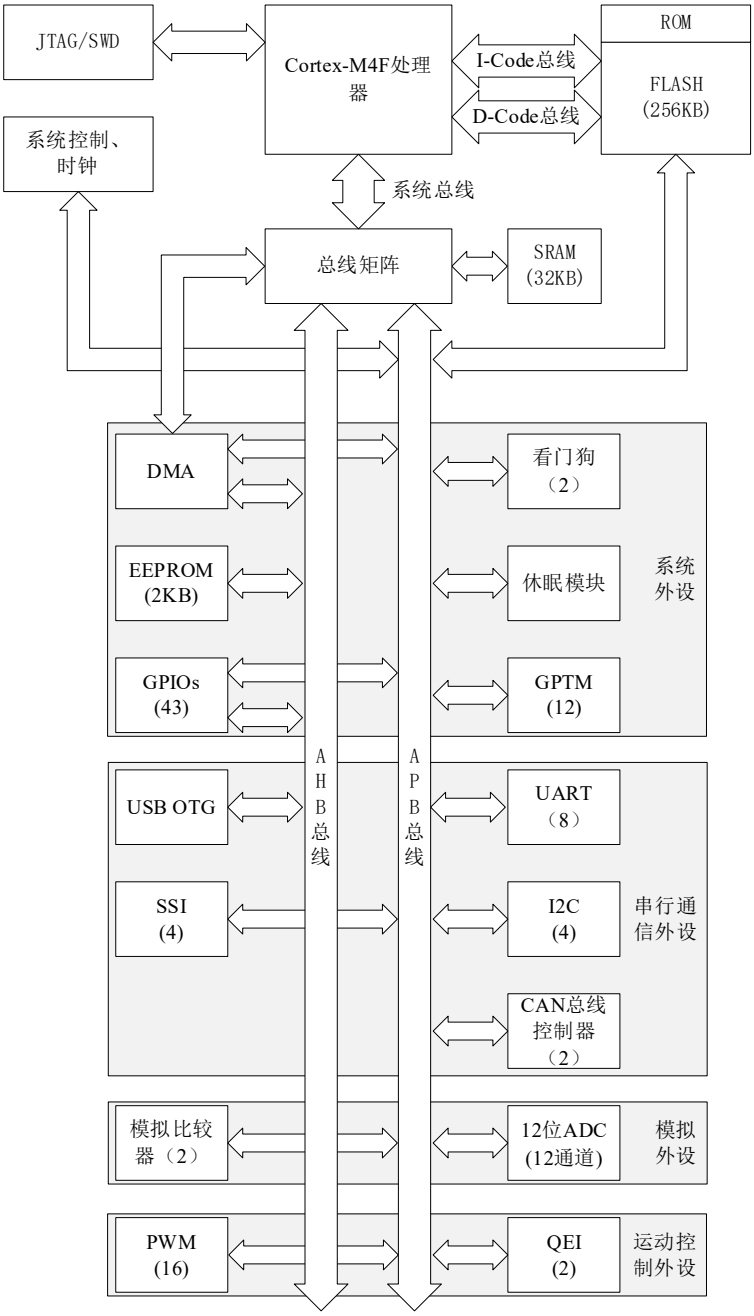

图 3-1 TM4C123GH6PM 结构框图

AHB 总线用于片上高性能模块的连接，APB 总线连接低带宽的外设。从图 3-1 可看到，TM4C123GH6PM 的 DMA 和 GPIO 可通过 AHB 和 APB 总线访问，EEPROM 和 USB OTG 只能通过 AHB 总线访问，其它外设则只能通过 APB 总线访问。

TM4C123GH6PM 的 LQFP 封装有 64 个引脚，其中 43 个为 GPIO 引脚，引脚图如图 3-2 所示。

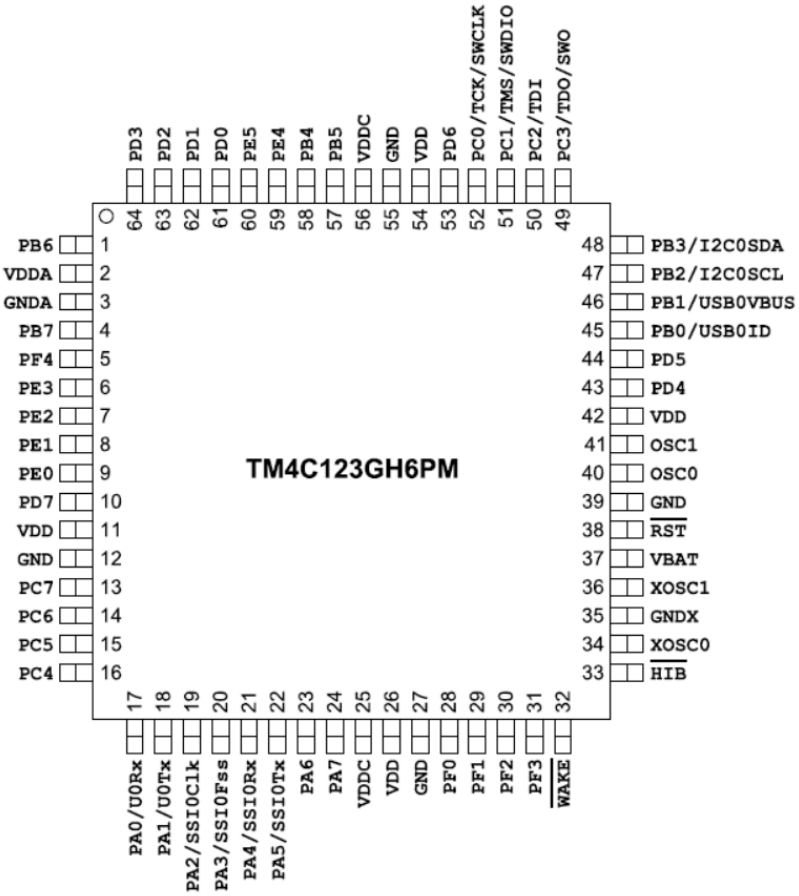

图 3-2 TM4C123GH6PM 的 LQFP 封装引脚图

2 电源

TM4C123GH6PM 微控制器提供一个内部集成的 LDO 稳压器，将外部电源的 3.3V 电压降压到 1.2V，用于给内部的处理器及大多数逻辑电路供电。图 3-4 描述了电源架构。

VDDA 可用于设备上的所有的模拟电路，如模数转换器 ADC，典型值 3.3V。

VDD：I/O 和某些逻辑的电源，典型值 3.3V。

VDDC：为主要的逻辑部分（包括处理器内核以及大部分片上外设）的电源。该管脚上的电压为 1.2V，由片上 LDO 提供。

VBAT：休眠模块的电源供应源，它通常连接到电池的正极端并用作备用电池，典型值 3.3V。

在 Tiva LauchPad 开发板上，直接将 VDD 和模拟电路电源 VDDA、休眠模块电源 VBAT 相连，将数字地 GND 和模拟地 GNDA 相连。

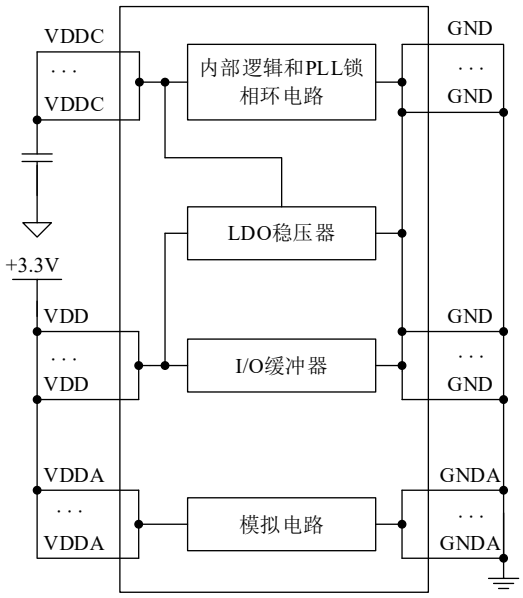

图 3-4 电源结构

3 复位

复位就是使处理器恢复到起始状态。TM4C123GH6PM 有 6 种复位方式：

1) 上电复位 PRO (Power-on Reset) : 当器件上电以及从休眠模式唤醒时产生复位。
2) 外部复位引脚/RST 引脚：/RST 低电平时会产生复位。
3) 内部掉电复位 BOR（Brown-on Detector）：当电源 VDD 低于门限电压 VBTH 时，系统将产生一个复位或中断。
4) 软件复位：软件即可以使整个器件复位，也可以单独使内核或某个外设复位。
5) 看门狗定时器复位：看门狗定时器第二次产生中断时复位。
6) MOSC 失效复位：当主振荡器运行太快或太慢时将产生错误，称为 MOSC 失效，此时将产生复位。

复位后，会根据 BOOTCFG 寄存器中的配置及相应的 GPIO 信号，决定执行 ROM 中的 Boot Loader 或是 FLASH 上的应用程序。如果从 FLASH 启动，则会从 0X000 0004 地址的复位向量开始执行。

外部复位引脚/RST 电路如下图所示：复位引脚/RST 通过一个上拉电阻接到电源正极，平时为高电平，按下按键为低电平；为了抗干扰，通常在/RST 引脚连接 RC 电路进行滤波。

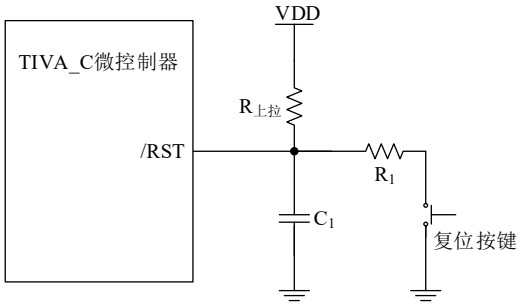

图 3-5 复位电路

4 时钟体系

系统时钟是处理器的运行时钟，一般外设的运行速度比处理器要慢，所以外设都有自己的时钟。外设时钟也都是通过基本时钟源或系统时钟分频和锁相环倍频得到。

4.1 基本时钟源

Tiva 系列微控制器中使用了多个时钟源：

内部高精度振荡器（PIOSC）：该振荡器是 Tiva 微控制器片内自带的时钟源，它无需使用任何外部元件。**PIOSC** 提供的时钟频率是 16MHz，在室温下精度为 1%，在全部工作温度范围内也有 3%的精度。

内部高精度振荡器 4 分频（4MHz）：对 PIOSC 做 4 分频，频率为 4MHz，精度为 1%。

外部主振荡器（MOSC）：可以使用外部接入的晶振，也可以使用外部输入的单端时钟源。在 Tiva LaunchPad 开发板上，外部主振荡器连接的是 16M 晶振。

内置低频振荡器（LFIOSC）：频率为 30kHz，精度为 50%。一般在系统进入深度睡眠模式时使用该时钟源。

休眠模式时钟源(HOSC)：频率为 32.768Hz，由外部接入。该时钟源用于实时时钟，或者为深度睡眠或休眠低功耗模式提供时钟。

表 3-2 时钟源

时钟源	频率	驱动 PLL	用于系统时钟
内部高精度振荡器 **PIOSC**	16MHz	是	是
内部高精度振荡器 4 分频	4MHz	否	是
外部主振荡器 **MOSC**	16MHz	是	是
内置低频振荡器 **LFIOSC**	30kHz	否	是
休眠模式时钟源 **HOSC**	32.768Hz	否	是

内部 PLL：锁相环(Phase Locked Loop)用作将微控制器的内部时钟和外部时钟保持同步，消除两者的相位差，还可以将时钟倍频，产生频率更高的时钟。Tiva 系列的内部 PLL 可以将时钟频率提升到 400MHz。PLL 的输入震荡源频率应控制在 5MHz——25MHz 之间。

4.2 时钟树

时钟树列出了系统时钟以及各外设时钟的走向。从时钟树可以看到，系统时钟有两种方式产生。

1) 使用锁相环的输出频率为 400MHz 的时钟，经过 2 分频(200MHz)和系统分频后得到。锁相环的输入只能是外部主振荡器（MOSC）或者内部高精度振荡器（PIOSC）。
2) 选择外部主振荡器（MOSC）、内部高精度振荡器（PIOSC）、内部高精度振荡器 4 分频（4MHz）、内置低频振荡器（LFIOSC）和休眠模式时钟源(HOSC)中的 1 个，经过系统分频得到。

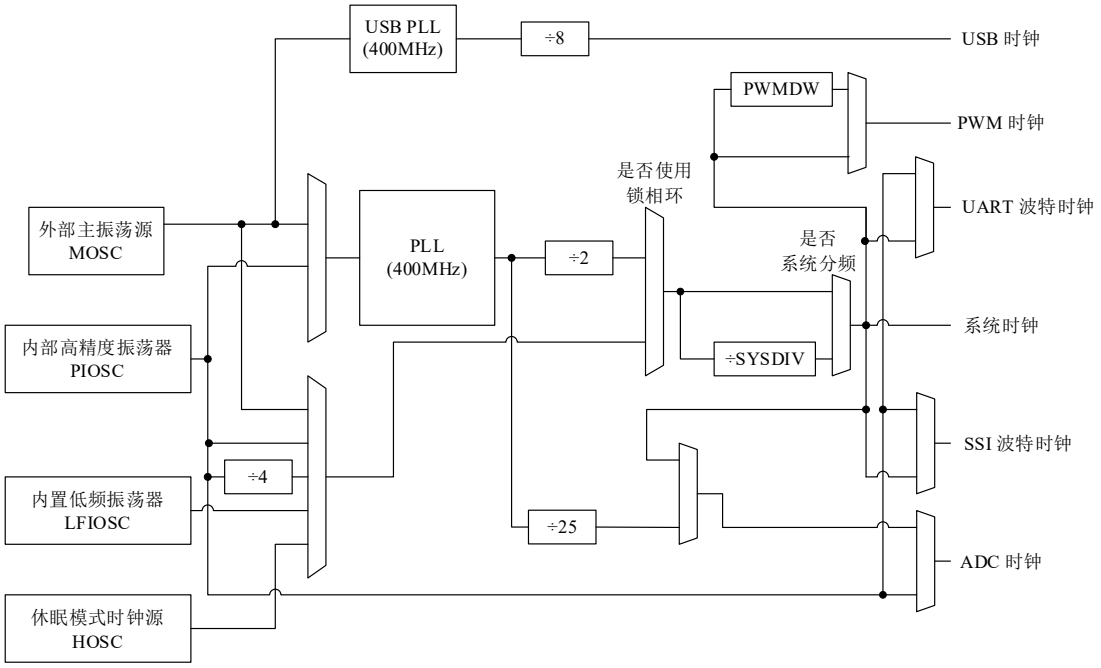

图 3-6 时钟树

4.3 初始化和配置

对 PLL 的配置可通过直接对 RCC/RCC2 寄存器执行写操作来实现。如果要使用 RCC2 寄存器，那么必须置位 USERCC2 位。成功改变基于 PLL 的系统时钟所需的步骤如下：

1) 通过置位 RCC 寄存器的 BYPASS 位并清零 USESYS 位来旁路 PLL 和系统时钟分频器，从而配置微控制器在“原始”时钟源运行，在切换系统时钟到 PLL 前允许新的 PLL 配置有效。
2) 选择晶振值(XTAL)和振荡器源 (OSCSRC)，清零 RCC/RCC2 的 PWRDN 位。设置 XTAL 域可以自动为相应晶振提供有效的 PLL 配置数据，清零 PWRDN 位可以给 PLL 及其输出供电并启用它们。
3) 在 RCC/RCC2 中选择需要的系统分频器(SYSDIV)并在 RCC 中置位 USESYS 位。SYSDIV 域决定了微控制器的系统频率。
4) 通过查询原始中断状态(RIS)寄存器的 PLLLRIS 位来等待 PLL 被锁定。
5) 通过清零 RCC/RCC2 的 BYPASS 位来启用 PLL。

4.4 时钟库函数

TivaWare 提供了系统时钟设定的库函数，使得系统时钟的配置变得很简单。

4.4.1 函数 SysCtlClockSet

功能：设置设备的时钟

原型：void SysCtlClockSet(uint32_t ui32Config);

参数：ui32Config 为设备时钟所需要的配置。

描述：该函数功能配置了设备的时钟，如：输入晶振频率、振荡器、PLL、系统时钟分频等。参数 ui32Config 是几个不同值的逻辑或，它们分别都被分成多个组，其中在每个组

中只能选择一个值。

是否使用 PLL：SYSCTL_USE_PLL（使用 PLL）或者 SYSCTL_USE_OSC（不使用 PLL）。当使用 PLL，且 PLL 的时钟源为外部主振荡器时，需告知 PLL 外部主振荡器的频率，如：SYSCTL_XTAL_4MHZ，.... SYSCTL_XTAL_25MHz。参数值低于 SYSCTL_XTAL_5MHZ 是无效的。

振荡器的选择为如下之一：SYSCTL_OSC_MAIN，SYSCTL_OSC_INT, SYSCTL_OSC_INT4，SYSCTL_OSC_INT30，SYSCTL_OSC_EXT32。其中 SYSCTL_OSC_EXT32 仅应用于拥有休眠模式的设备，而且只有当休眠模式使能时才有效。选择外部主振荡器作为系统时钟源时使用 SYSCTL_USE_OSC|SYSCTL_OSC_MAIN。

系统时钟分频器选择为以下值之一：SYSCTL_SYSDIV_1，.... SYSCTL_SYSDIV_64。

例 1：不使用 PLL，选择内部高精度振荡器（PIOSC）作为时钟源，系统分频数为 1，让系统时钟为 16M：

```
SysCtlClockSet(SYSCTL_SYSDIV_1|SYSCTL_USE_OSC| SYSCTL_OSC_INT);
```

例 2：使用 PLL，PLL 的时钟源为 16MHz 的外部主振荡器，则 PLL 输出固定 200M(2 分频)，再使用 4 分频的系统分频，让系统时钟为 50MHz：

```
SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
```

4.4.2 函数 SysCtlPeripheralEnable

功能：使能外设。

原型：void SysCtlPeripheralEnable(uint32_t ui32Peripheral)；

参数：ui32Peripheral 是要使能的外设

描述：函数的功能是使能一个外设。上电时，所有外设为禁止状态；它们必须在使能后才能操作或响应寄存器的读/写。所谓外设禁止实际上是禁止给外设提供的时钟信号，外设此时不能工作，其功耗也基本为 0；使能外设就是使能外设时钟。

外设参数必须为一下参数之一：SYSCTL_PERIPH_ADC0,SYSCTL_PERIPH_ADC1,, SYSCTL_PERIPH_GPIOA,SYSCTL_PERIPH_GPIOB,SYSCTL_PERIPH_GPIOC,SYSCTL_PERIPH_GPIOD, SYSCTL_PERIPH_GPIOE,SYSCTL_PERIPH_GPIOF, SYSCTL_PERIPH_HIBERNATE.....。其余参数请查看外设驱动库文档。

注：写使能外设后，要经过 5 个时钟周期外设才真正被使能。在这期间，试图访问外设将导致总线错误。要注意确保在这段时间内，外设没有被访问。

4.4.3 函数 SysCtlClockGet

功能：获取处理器时钟频率。

原型：uint32_t SysCtlClockGet(void)；

参数：无

描述：该函数获得处理器时钟频率和外设模块的时钟频率（除了 PWM，它有自己的时钟分配，其他外设有不同的时钟）。

5 存储器及映射

TM4C123GH6PM 微控制器带有 32KB 的 SRAM、内部 ROM、256KB 的 flash 和 2KB 的 EEPROM。Cortex-M4F 系列处理器对内存和外设进行了统一编址，各存储器和外设在同一个地址空间内。FLASH、ROM、SRAM 和外设对应的地址如下：

表 3-3 TM4C123GH6PM 存储器映射

存储器类型	起始地址	结束地址	大小
FLASH	0x00000000	0x0003FFFF	256K
ROM	0x10000000	0x1FFFFFFF	
SRAM	0x20000000	0x20007FFF	32K
SRAM 位带别名区	0x22000000	0x220FFFFF	1M
外设	0x40000000	0x400FFFFF	
外设位带别名区	0x42000000	0x43FFFFFF	

5.1 位带（Bit-band）

计算机中往往采用字节编址，1个地址对应1个字节(8bit)的数据，要对单一的位(bit)进行读写操作，得通过繁琐的位运算来进行。位带可以给单一的位赋予地址，这个地址被称为“位带别名地址”，对这个地址进行读写相当于读写相应的位，这样大大方便了位的操作。在Cortex M4中，有两个区域实现了位带。一个是SRAM区的最低1MB范围，第二个则是片内外设区的最低1MB范围。这两个区域除了可以像普通的RAM一样使用外，它们还都有自己的“位带别名区”。位带别名区的每个32位地址对应位带区的其中一位。当访问位带别名区的这些地址时，就可以达到访问原始位的目的。

SRAM 和外设的位带别名区基址分别为 0x22000000 和 0x42000000。

位带别名地址的计算公式为：

$$\text{位带别名地址} = \text{位带别名区基址} + (\text{偏移量} \times 32) + (\text{位编号} \times 4)$$

例如：如果要修改 SRAM 地址 0x20000000 的第 6 位，则位带别名地址计算如下：

$$\text{位带别名地址} = 0x22000000 + (0x0000 \times 32) + (6 \times 4) = 0x22000018$$

因此，当需要读/写位带地址 0x20000000 的第 6 位时，只需要对相应的位带别名地址 0x22000018 执行读/写操作指令即可。

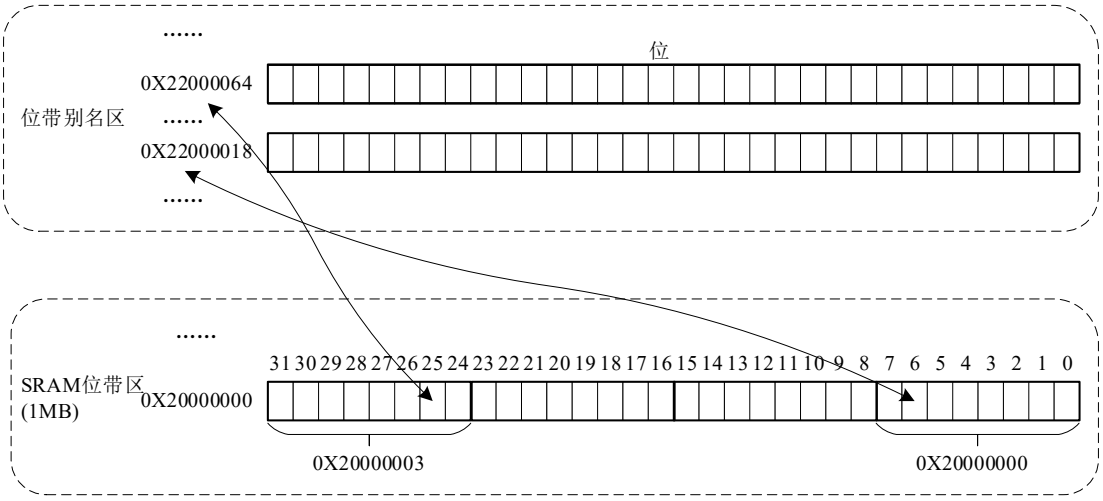

图 3-7 位带

Tivaware目录下的inc/hw_types.h宏定义了位带的变换，从而可以通过HWREGBITW(x,b)宏定义得到位带别名地址：

```
#define HWREGBITW(x,b) HWREG(((uint32_t)(x)&0xF0000000)|0x02000000|(((uint32_t)(x)&0x000FFFFF)
<<5)|((b)<<2))
```

其中x是地址，b是第几位，例如将0x20000100的第3位设置为1：

```
HWREGBITW(0x20000100,3) = 1;
```

5.2 SRAM

SRAM 是易失性存储器，掉电后数据会丢失，适合存储需要经常修改的数据。

随机存取存储器，可对存储器的任一个存储单元按需要随时进行信息读出或写入，且读/写速度快。

5.3 ROM

ROM 是非易失性存储器，只能读出，不能用一般的方法写入。TM4C123GH6PM 的 ROM 已预编程，包含以下组件：

TivaWare 引导加载程序 Boot Load 和向量表；

TivaWare 外围设备的驱动程序库(DriverLib)；

高级加密标准（AES）加密表；

循环冗余检查(CRC) 错误检测功能。

5.4 Flash

FLASH 是非易失性存储器，掉电后数据不会丢失，适合存储程序和不经常修改的数据。当系统时钟速度为 40 MHz 或更慢时，FLASH 可以在一个周期内读取。当处理器系统时钟大于 40MHz 时，FLASH 控制器将启用预取指缓存器。在这种模式下，Flash 存储器以一半的系统时钟工作，预取指缓存器每个时钟周期读取两个 32 位的字，使代码的连续执行不需要等待。

Flash 存储器由一系列 1KB 的块组织在一起，这些块可以被单独擦除。flash 器件的写入操作只能在空的或已擦除的单元内进行，所以很多时候，在进行写入操作之前必须先执行整

块擦除，擦除块会使块中所有内容块被重置为 1。

Tiva 器件具有 Flash 存储器保护机制，每两个 1KB 的块配成 2K 的保护块，保护块允许被设置为只读或只执行，为系统提供不同级别的代码保护。只读块不能被删除或编程，系统可保护这些块的内容不被修改。同时，只执行块不能被删除或编程，不能被控制器或调试器读取，只能通过取指机制来取出其中保存的指令。

5.5 EEPROM

EPPROM 是非易失性存储器，掉电后数据不会丢失，适合存储不经常修改的数据。Tiva 的 EEPROM 模块接口为用户提供了随机访问以及顺序访问的读写样式。同时，EEPROM 模块提供了对块的锁定机制，可防止在特殊情况下对块的读写操作保护。密码存取模式使用户程序可锁定 16 字对齐的一个或多个 EEPROM 块控制访问。

每个 EEPROM 由 32 块组成，每块由 16 个字组成。对 EEPROM 的读取可以是按字节或半字长，并且，这些读取操作也不需要依照字边界对齐。每次读取，会读取整个字长，任何不需要的数据将被简单地忽略。每次写入必须要按字对齐。要向 EEPROM 写入一个字节，应先读出值，修改相应的字节，再进行回写。

EEPROM 的每个块都是根据偏移量寻址的，可通过块选择寄存器对操作的块进行选择。当前操作的块由当前块寄存器 EEBLOCK 决定，当前偏移量由当前寄存器 EEOFFSET 决定。EEPROM 的块是受单独保护的。若尝试对未受权的块读写将返回 0xFFFF.FFFF 的错误值，并且 EEDONE 寄存器也将被置位。

6 低功耗模式

对于很多采用电池供电的便携式设备，需要尽量节省能量消耗，低功耗性能也成为微控制器的一个重要指标。TM4C123GH6PM 有 4 种工作模式，以应对各种情况下的低功耗需求：

正常模式：在该模式下，处理器正常运行指令，外设正常运行。

睡眠模式：睡眠模式下，停止处理器时钟和存储器时钟，不执行代码，但对于有些智能外设，其时钟频率不变，仍然可以运行。中断事件能使系统回归到运行模式。

深度睡眠模式：停止系统时钟并关闭 PLL 和 Flash 存储器，不执行代码。外设运行在内置低频振荡器的 30KHz 频率下。在该模式下，中断事件能使系统回归到运行模式。

休眠模式：处理器和外设的电源都关断，只维持休眠模块的供电。电源供电可以利用外部信号 WAKE 触发恢复，也可以利用内置实时时钟(RTC)，在经过一段特定的时间之后恢复。休眠模块可以由外部电池或者辅助电源单独供电。

低功耗模式可以让处理器在没有外部任务需要执行时保持休眠，当有外部请求后，唤醒处理器进行工作，任务完成后处理器再度进入休眠。

7 库函数编程和直接访问寄存器编程

7.1 TivaWare 函数库

TI 公司为 Tiva 系列微控制器的开发提供了软件开发工具包 (SDK)，即 TivaWare。

TivaWare 实际上是一个函数库，是为微控制器许多常用的基础操作编写的函数集合。有了这些规范化、模块化的库函数，就避免了开发者繁琐的、不规范的底层寄存器编程工作，减少了错误出现的可能，大大提高了开发效率。使用库函数开发和直接操作寄存器开发相比，其代码可能占用更多存储空间，执行效率也可能低一些，但代码的可读性更好，可移植性更强，开发更容易。在微控制器性能已不是瓶颈的今天，库函数开发已是主流的方式。

TivaWare 中提供了 Tiva 系列微控制器的外设驱动库(drivelib)，对每个外设都提供了一系列的驱动函数。此外还有图形驱动库(grlib)、高精度数学运算函数库(IQmath)、传感器库(sensorlib)、USB 库(usblib)、以及第三方软件包(thir_party)。TivaWare 的文件目录如图 3-8 所示，在 docs 目录下有各个库的说明文档；inc 目录中是 Tiva 系列微控制器的硬件寄存器头文件，包含了各寄存器、位域及操作的宏定义；utils 中包含一些函数使用的程序范例。

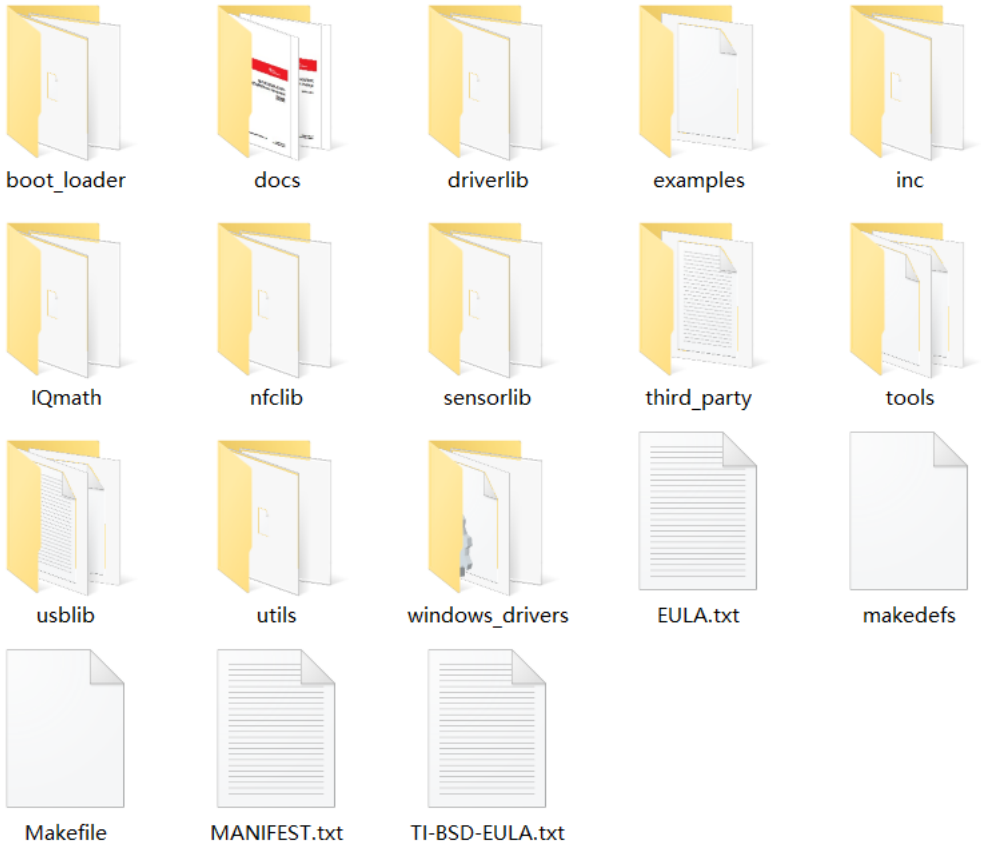

图 3-8 TivaWare 文件目录

外设驱动库（drivelib）是用来访问 Tiva 系列微控制器外设的一系列驱动函数。开发者只要掌握了外设操作的基本过程，即使对相关寄存器细节并不了解，也可以通过库函数来实现功能。下面是 drivelib 的相关目录和文件。

Docs 目录下的 SW-TM4C-DRL-UG-2.1.4.178.pdf 是外设驱动库的说明文档，其中对每一个外设驱动函数都有详细说明。

在 driverlib 目录下有各外设的库函数*.c 文件和头文件，包含了各外设驱动库函数的源代码。driverlib 支持 CCS 、Keil、IAR EW 等多种开发工具，因此在 driverlib 目录下也有对应的目录，里面存放着对应编译工具链接的 drivelib 库文件（drivelib.lib），可以直接使用这

些库文件来调用库函数。

在 inc 目录下有各外设的头文件(hw_*.h)，描述了每个外设的寄存器、寄存器中的位域和直接寄存器操作的宏定义。库函数需要使用这些头文件来访问外设，开发者也可以使用这些头文件来实现直接访问寄存器编程。

绝大多数的 Tiva 系列微控制器都将外设驱动程序库存储在了片内 ROM 中。这样做的好处是可以直接调用内部 ROM 中的库函数代码，减小了 Flash 中的代码空间。

7.2 直接访问寄存器编程

值得注意的是，库函数并没有实现所有的外设功能，许多比较复杂或者用得较少的功能，库函数并没有提供。因此，仅仅会使用库函数开发是不够的，开发者还需要掌握直接访问寄存器编程的方法。

直接使用寄存器编程，开发者需要了解外设的寄存器框架、每个寄存器及其各位域的含义及作用，各寄存器相互之间的联系和影响，要实现某个功能时操作的顺序及内容。这些往往需要仔细阅读微控制器厂商提供的数据手册和开发文档，详细了解相关内容。很多时候，直接学习相关库函数的源代码也是一个很好的方法，学习库函数的实现方式和风格，或者在其基础上进行修改，实现新的功能。下面以直接访问寄存器编写一个 Flash 的写操作函数来说明。

7.2.1 Flash 控制器的寄存器

在“tm4c123gh6pm 数据手册的 8.1 节内部存储器”中列出了 Flash 控制器的寄存器框架。

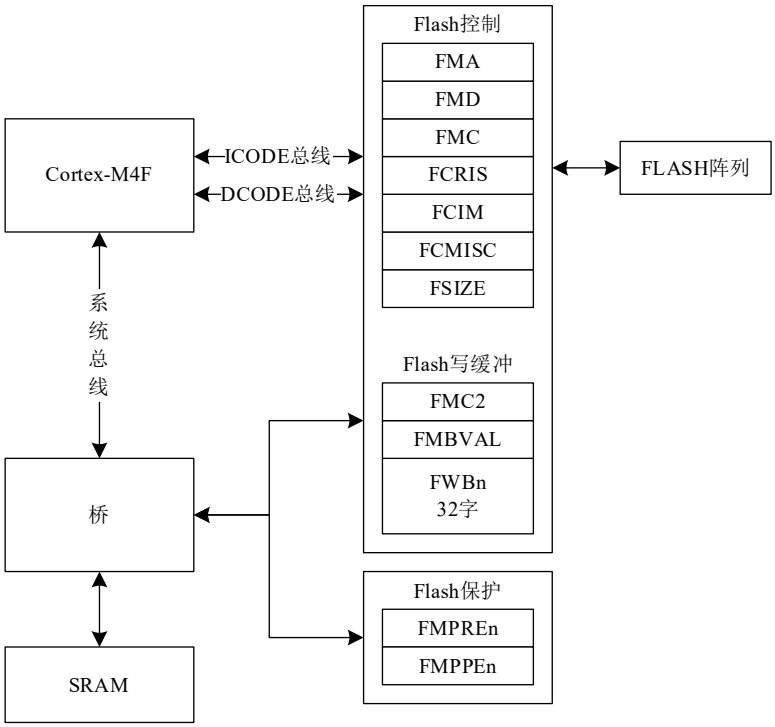

图 3-9 Flash 控制器寄存器框架图

在“数据手册 8.3 节寄存器映射”中列出了相关寄存器的信息。偏移量是指寄存器地址

对于模块基地址的偏移量，例如 Flash 控制器的基地址为 0x400F E000, FMD 寄存器的地址为基地址加偏移量，即 0x400F E004。复位是指复位时寄存器的初始值。

表 3-4 Flash 控制器的寄存器映射

偏移量	名称	类型	复位	描述
0x000	FMA	R/W	0x0000 0000	Flash 存储器地址寄存器
0x004	FMD	R/W	0x0000 0000	Flash 存储器数据寄存器
0x008	FMC	R/W	0x0000 0000	Flash 存储器控制寄存器
0x00C	FCRIS	RO	0x0000 0000	Flash 控制器原始中断状态
0x010	FCIM	R/W	0x0000 0000	Flash 控制器中断屏蔽寄存器
0x014	FCMISC	R/W1C	0x0000 0000	Flash 控制器可屏蔽中断的状态和清除
0x020	FMC2	R/W	0x0000 0000	Flash 存储器控制寄存器 2
0x030	FWBVAL	R/W	0x0000 0000	Flash 写缓冲器有效
0x100 -0x17C	FWBn	R/W	0x0000 0000	Flash 写缓冲器 n
0xFC0	FSIZE	RO	0x0000 007F	Flash 容量寄存器
0x130	FMPRE0	R/W	0xFFFF FFFF	Flash 存储器保护读取启用寄存器 0
0x200	FMPRE0	R/W	0xFFFF FFFF	Flash 存储器保护读取启用寄存器 0
0x134	FMPPE0	R/W	0xFFFF FFFF	Flash 存储器保护编程启用寄存器 0
0x400	FMPPE0	R/W	0xFFFF FFFF	Flash 存储器保护编程启用寄存器 0
0x204	FMPRE1	R/W	0xFFFF FFFF	Flash 存储器保护读取启用寄存器 1
0x208	FMPPE2	R/W	0xFFFF FFFF	Flash 存储器保护读取启用寄存器 2
0x20C	FMPRE3	R/W	0xFFFF FFFF	Flash 存储器保护读取启用寄存器 3
0x404	FMPPE1	R/W	0xFFFF FFFF	Flash 存储器保护编程启用寄存器 1
0x408	FMPPE2	R/W	0xFFFF FFFF	Flash 存储器保护编程启用寄存器 2
0x40C	FMPPE3	R/W	0xFFFF FFFF	Flash 存储器保护编程启用寄存器 3

在“数据手册 8.4 节 Flash 存储器寄存器描述”中可以查询到所有 Flash 控制和 Flash 缓冲寄存器的详细说明。这里将简单介绍与 Flash 写相关的寄存器。

1) FMA 寄存器存放对 Flash 存储器操作的起始地址，此地址必须字（4 字节）对齐，即必须能被 4 整除。在使用写缓冲区写入时，必须 32 字（128 字节）对齐，即能被 128 整除。
2) FMD 寄存器存放要向 Flash 写入的数据。
3) FMC 寄存器是 Flash 存储器控制寄存器，其位域如表 3-5：

表 3-5 FMC 寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WRKEY															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留												COMT	MERASE	ERASE	WRITE

WRKEY 为写密钥，为减少意外写入，要对 Flash 执行写操作，须将密钥值值 0xA442 或 0x71D5 写入 WRKEY 域中。

WRITE 置 1 将把 FMD 中的数据写入 FMA 中的指定的位置；读 WRITE 可以得知写入是否完成。

4) FMC2 寄存器的作用是启动写缓冲区写入。

表 3-6 FMC2 寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WRKEY															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留															WRBUF

WRKEY 和 FMC 寄存器中的 WRKEY 作用完全相同，为写密钥。

WRBUF 置 1 将会把写缓冲区 FWBn 中的数据写入 FMA 中的指定的位置；读 WRBUF 可以得知写入是否完成。

5) FCMISC 寄存器是中断状态和清除寄存器，可以通过这个寄存器查询中断和错误状态，以及清除中断和错误状态。

6) FWBVAL 寄存器记录了哪些 FWBn 寄存器（写缓冲区）被写入了数据，在写缓冲器中的数据写入 Flash 后，这些标志将被清除。可以通过此寄存器检查写缓冲区是否有数据。

7) FWBn 寄存器共有 32 个，n 的取值范围为 0~31，它们共同组成了 32 字的写缓冲区。

7.2.2 Flash 的写缓冲器写入库函数

在“数据手册 8.2 节功能说明”中可以查询到，Flash 的写入操作有单字写入和写缓冲区写入两种方式。写缓冲区写入是先将数据写入写缓冲区 FWBn 内，再将写缓冲区内的数据烧写到 Flash，最多可以一次写入 32 个字，其具体步骤为：

1) 将源数据写入 FWBn 寄存器。

2) 将目标地址写入 FMA 寄存器，该地址必须是一个 32 字对齐的地址。

3) 将密钥值写入 FMC2 寄存器的 WRKEY 位域，置位 WRBUF 启动 Flash 写入。

4) 查询 FMC2 寄存器，直到 Flash 写入完成（WRBUF 位被清零）。

TivaWare 提供了 Flash 的写缓冲器写入库函数 FlashProgram()，可以在 Docs 目录下的 SW-TM4C-DRL-UG-2.1.4.178.pdf 中查询到其详细使用方法(P243)。在函数的参数中，pui32Data 为写入数据的指针；ui32Address 为写入的 flash 起始地址；ui32Count 为写入的数据数量（按字节计算）。在 TivaWare 的 drivelib 目录中的 flash.c 文件中可以找到其源代码。

```
//---------------------------------------------------------------------------------------------
//flash.c
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_flash.h"
#include "inc/hw_ints.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/flash.h"
#include "driverlib/interrupt.h"
  ……
int32_t FlashProgram(uint32_t *pui32Data, uint32_t ui32Address, uint32_t ui32Count)
{
```

```
    // 检查输入参数的合法性，如果不合法则终止程序执行
    ASSERT(!(ui32Address & 3));  //检查地址是否为 4 的整数倍
    ASSERT(!(ui32Count & 3));    //检查传输数量为 4 的整数倍

   HWREG(FLASH_FCMISC) = (FLASH_FCMISC_AMISC | FLASH_FCMISC_VOLTMISC |FLASH_FCMISC_INVDMISC |
FLASH_FCMISC_PROGMISC);          // 清除中断和错误状态标志

    while(ui32Count)
    {
        HWREG(FLASH_FMA) = ui32Address & ~(0x7f); //将目标地址写入 FMA，该地址必须 32 字对齐
        // 将源数据写入 FWBn 写缓冲区
        while(((ui32Address & 0x7c) || (HWREG(FLASH_FWBVAL) == 0)) && (ui32Count != 0))
        {
            HWREG(FLASH_FWBN + (ui32Address & 0x7c)) = *pui32Data++;
            ui32Address += 4;
            ui32Count -= 4;
        }

        HWREG(FLASH_FMC2) = FLASH_FMC2_WRKEY | FLASH_FMC2_WRBUF; //将密钥和 WRBUF 位写入 FMC2 寄存器

        while(HWREG(FLASH_FMC2) & FLASH_FMC2_WRBUF)        //查询 FMC2 寄存器，直到 WRBUF 位被清零
        {
        }
    }

      if(HWREG(FLASH_FCRIS) & (FLASH_FCRIS_ARIS | FLASH_FCRIS_VOLTRIS | FLASH_FCRIS_INVDRIS |
FLASH_FCRIS_PROGRIS))              // 读取中断和错误状态标志，检查 flash 烧写过程中是否有中断或错误
    {
        return(-1);
    }

    return(0);
}
……
//---------------------------------------------------------------------------------------------
```

在头文件 inc\hw_flash.h 中定义了相关寄存器、位域和寄存器操作的宏定义。
例如寄存器宏定义，后面为寄存器地址：

```
#define FLASH_FMA               0x400FD000  // Flash Memory Address
#define FLASH_FMD               0x400FD004  // Flash Memory Data
#define FLASH_FMC               0x400FD008  // Flash Memory Control
#define FLASH_FCMISC            0x400FD014  // Flash Masked Interrupt Status and clear
#define FLASH_FMC2              0x400FD020  // Flash Memory Control 2
#define FLASH_FWBVAL            0x400FD030  // Flash Write Buffer Valid
```

例如寄存器 FMC2 中 WRBUF 位域的宏定义，其为寄存器 FMC2 的最低 1 位，相应位取 1，即为 0x0000001：

```
#define FLASH_FMC2_WRBUF        0x00000001  // Buffered Flash Memory Write
```

寄存器操作的宏定义，例如密钥值的宏定义：

```
#define FLASH_FMC2_WRKEY        0xA4420000  // FLASH write key
```

FlashProgram 库函数的程序流程图如图 3-10 所示：

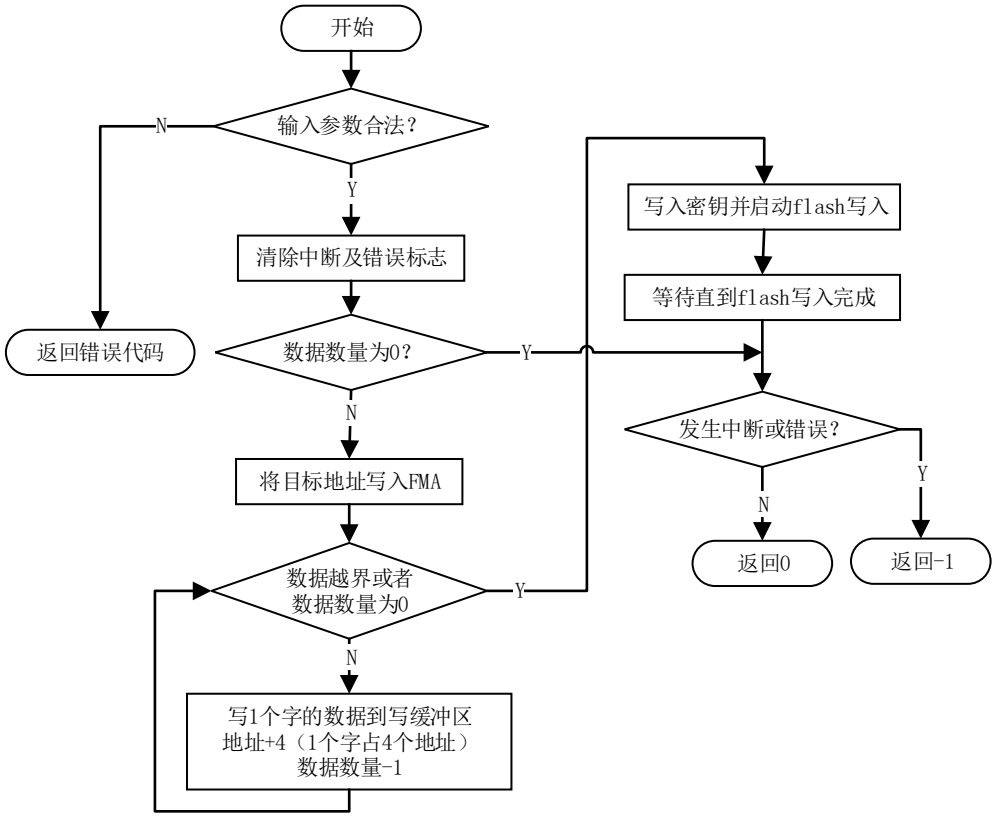

图 3-10 flash 写缓冲区写入程序流程图

7.2.3 Flash 写入的直接寄存器编程

TivaWare 没有提供 flash 单字写入的库函数，下面参考库函数，使用直接访问寄存器的方式编写 flash 单字写入函数。在“数据手册 8.2 节”中可以查询到单字写入的操作步骤：

1) 将源数据写入 FMD 寄存器。
2) 将目标地址写入 FMA 寄存器。
3) 将 Flash 存储器写入密钥和 WRITE 位写入 FMC 寄存器。要写入 Flash 存储器，必须将值 0xA442 或 0x71D5 写入 WRKEY 域中，具体取决于 BOOTCFG 寄存器 KEY 位的值。
4) 查询 FMC 寄存器，直至 WRITE 位被清零。

请大家查询 inc/hw_flash.h，得到相关寄存器、位域和操作的宏定义，修改 FlashProgram 库函数，尝试编写单字写入库函数 FlashProgramOne，并编写程序验证。

习题

1) 编写一个 systick 中断程序，重载值设为 2000000，每次中断控制 LED 改变一次状态（亮和灭）。其时钟源设为系统时钟，分别使用外部时钟和内部时钟源将系统时钟修改为 80MHz，50MHz，16MHz，8MHz，其中 8MHz 不使用 PLL,并观察 led 灯的闪烁快慢。