第五章　24-Bit Σ -Δ模数转换器

5.1　本章引言

SD24 模块由多达四个（注释：根据封装不同最多 4 个，也可以是 2 个或者 3 个）独立的Σ-Δ模数转换器构成。该转换器是基于二阶过采样Σ-Δ调制器和数字抽取滤波器。抽取滤波器是具有高达 256 可选择过采样率梳型滤波器。额外的滤波可以用软件来完成。

一个Σ-Δ模数转换器基本上有两部分组成：模拟部分的调制器和数字部分的抽取滤波器。在 SD24 的调制器提供的 0 和 1 的位流的数字抽取滤波器。

数字滤波器的平均比特流从一个给定的比特数的调制器（通过过采样率指定）并且提供一个低码率的抽样为进一步给 CPU 处理。

众所周知平均值可增加转换的信噪比性能。与传统的 ADC 相比 4 通道过采样每个因子可以提高约 6 分贝或 1 位的信噪比。

实现一个 16 位分辨率的一个简单的 1 位 ADC 需要过采样率为 4^{15} = 1.073.741.824，这是不切实际的。

为了克服这个限制，Σ-Δ制器实现的技术称为噪声整形，由于一个实施的反馈环技术和集成的量化噪声被推动到更高的频率，因此低得多的过采样率足以达到高的分辨率。

SD24 特性包括：

- 二阶Σ-Δ架构
- 高达 7 个(这个数量跟库函数手册有出入啊)可同时采样的 ADC。(具体数量请参考对应型号手册)。
- 固定的 1.024 MHz 调制器输入频率。
- 软件可选择的内置和外置参考电压
- 软件可选的全通道温度传感器

下图给出 SD24 系统相关结构图，图 5.1 为基准信号系统图，图 5.2 为 ADC 模块结构图。根据图 5.2 内部应该有 7 个通道的 ADC,其中通道 0~5 是测量外部信号用，通道 6 是测量温度传感器模块的。秉承了 G2 测量内部温度的优良传统，对于自我温度监控很有帮助，可以很好的保证系统正常运行，比如温度过高时候休眠或降低系统频率，好神奇。

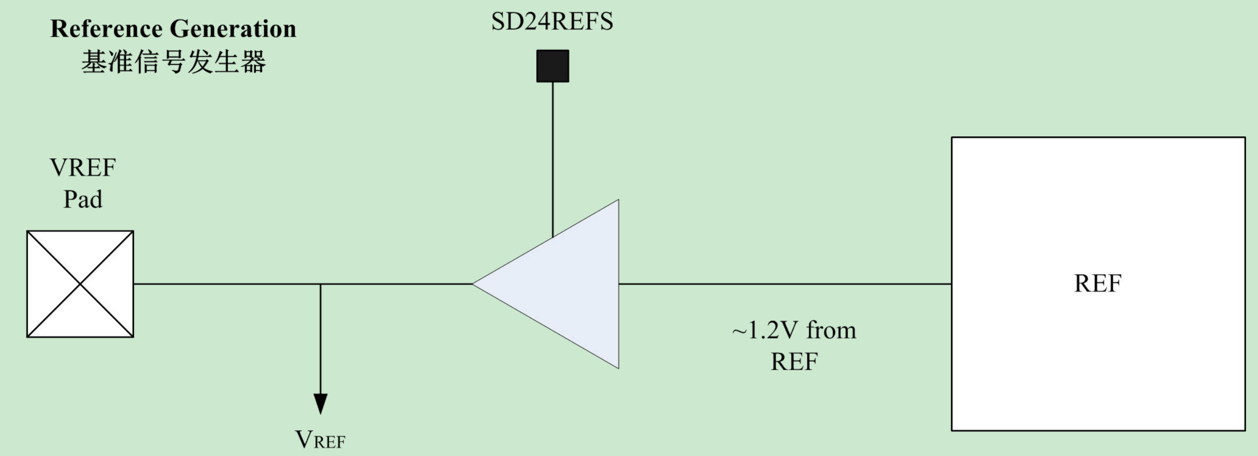

图 5.1

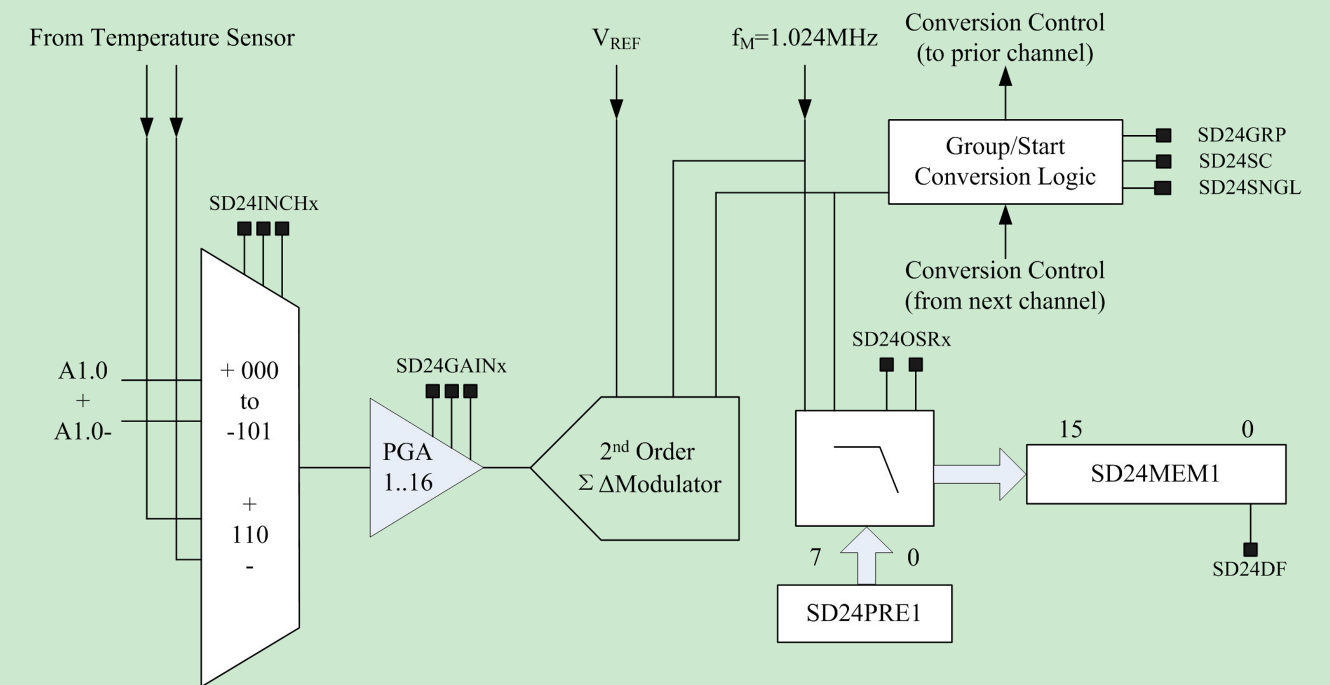

图 5.2

有兴趣的朋友自己阅读 MSP430i2xx 家族用户手册，我们这里对内部寄存器不作具体研究，下面开始学习相关库函数。

5.2 函数总览

1	void SD24_init (uint16_t baseAddress, uint8_t referenceSelect)
	初始化 SD24 模块
2	void SD24_initConverter (uint16_t baseAddress, uint16_t converter, uint16_t conversionMode)
	配置 SD24 转换器
3	void SD24_initConverterAdvanced (uint16_t baseAddress, SD24_initConverterAdvancedParam *param)
	配置 SD24 转换器—高级配置
4	void SD24_setConverterDataFormat (uint16_ baseAddress, uint16_t converter, uint16_t dataFormat)
	设置 SD24 转换器数据格式
5	void SD24_startConverterConversion (uint16_t baseAddress, uint8_t converter)
	转换器开始转换
6	void SD24_stopConverterConversion (uint16_t baseAddress, uint8_t converter)
	停止转换器转换
7	void SD24_setInputChannel (uint16_t baseAddress, uint8_t converter, uint8_t inputChannel)
	配置输入通道
8	void SD24_setInterruptDelay (uint16_t baseAddress, uint8_t converter, uint8_t interruptDelay)
	为中断触发配置延迟
9	void SD24_setOversampling (uint16_t baseAddress, uint8_t converter, uint16_t oversampleRatio)
	对转换器配置过采样率
10	void SD24_setGain (uint16_t baseAddress, uint8_t converter, uint8_t gain)
	为转换器配置增益
11	uint32_t SD24_getResults (uint16_t baseAddress, uint8_t converter)
	返回所选转换器的转换结果
12	uint16_t SD24_getHighWordResults (uint16_t baseAddress, uint8_t converter)
	返回转换结果的高字位
13	void SD24_enableInterrupt (uint16_t baseAddress, uint8_t converter, uint16_t mask)
	使能 SD24 模块中断。
14	void SD24_disableInterrupt (uint16_t baseAddress, uint8_t converter, uint16_t mask)
	关闭 SD24 模块中断
15	void SD24_clearInterrupt (uint16_t baseAddress, uint8_t converter, uint16_t mask)
	清除 SD24 模块中断（清除中断标志）。
16	uint16_t SD24_getInterruptStatus (uint16_t baseAddress, uint8_t converter, uint16_t mask)
	返回 SD24 模块的中断标志

函数分类

SD24 API 可以分成三组函数：那些处理初始化和转换的，那些处理中断的，还有那些处理 SD24 的辅助功能的。

SD24 初始化和转换功能：

- SD24_init()
- SD24_initConverter()
- SD24_initConverterAdvanced()
- SD24_startConverterConversion()
- SD24_stopConverterConversion()
- SD24_getResults()
- SD24_getHighWordResults()

SD24 处理中断的有：

- SD24 enableInterrupt()
- SD24 disableInterrupt()
- SD24 clearInterrupt()
- SD24 getInterruptStatus()

SD24 处理辅助功能的有：

- SD24 setInputChannel()
- SD24 setConverterDataFormat()
- SD24 setInterruptDelay()
- SD24 setOversampling()

SD24 setGain()

详细描述

void SD24_init (uint16_t baseAddress, uint8_t referenceSelect)

初始化 SD24 模块。此函数初始化 SD24 模块Σ-△模数转换。 具体的功能是为 SD24 核心设置用于转换的时钟源。SD24 中断初始化完成后将重置寄存器，并且将以给定的参数设置。另外，转换器配置设置是独立于此函数的。

参数

baseAddress	是 SD24 模块的基地址
referenceSelect	选择参考源 SD24 核心有效的值为： SD24_REF_EXTERNAL [Default] 默认的外部参考 SD24_REF_INTERNAL 选择内部参考

该函数修改寄存器 SD24BCTL0 的 SD24REFS 位。

返回：无。

void SD24_initConverter (uint16_t baseAddress, uint16_t converter, uint16_t conversionMode)

配置 SD24 转换器。

该函数初始化 SD24 模块的一个转换器。完成后转换器将准备好转换，并可开始用 **SD24_startConverterConversion()**。 额外的配置，如数据格式可以在 **SD24_setConverterDataFormat()** 中进行配置。

void SD24_initConverterAdvanced (uint16_t baseAddress, SD24_initConverterAdvancedParam *param)

配置 SD24 转换器——高级配置。

该函数初始化 SD24 模块的一个转换器。完成后，转换器将可以开始用 SD24_startConverterConversion()转换。

参数

baseAddress	是 SD24 模块的基地址
param	是指向转换器高级配置的结构体指针

该函数修改寄存器 SD24BCTL0 的 SD24REFS 位。

返回：无。

Param 具体成员如下：

1	SD24_initConverterAdvancedParam::converter	选择将要配置的转换器
2	SD24_initConverterAdvancedParam::conversionMode	转换模式，连续采样还是一次采样
3	SD24_initConverterAdvancedParam::groupEnable	分组使能，默认分组
4	SD24_initConverterAdvancedParam::inputChannel	输入通道选择，模拟通道还是温度通道
5	SD24_initConverterAdvancedParam::dataFormat	数据格式，二进制或 2 的补码形式
6	SD24_initConverterAdvancedParam::interruptDelay	中断延时设置，可以选择第一次采样中断或第四次采样中断
7	SD24_initConverterAdvancedParam::oversampleRatio	过采样率选择，可以选择四种：32,64,128,256
8	SD24_initConverterAdvancedParam::gain	增益选择，共四种，1,2,4,8,16，默认是 1

以上 8 个结构体成员变量具体选择范围请参考 sd24.h 文件，注意有个 inputChannel，TI 在介绍里写错了，漏了。

请认真查看头文件，把不懂的那个找出来我们一起讨论讨论。我对 3 和 5 还不是很了解，哪位高手看的明白讲出来。

void SD24_setConverterDataFormat (uint16_ baseAddress, uint16_t converter, uint16_t dataFormat)

设置 SD24 转换器的数据格式。

该函数设置转换器格式，以便由此产生的数据可以以 2 进制或 2 的补码形式查看。

参数

baseAddress	是 SD24 模块的基地址
converter	选择将要配置的转换器。可选值有： SD24_CONVERTER_0 SD24_CONVERTER_1 SD24_CONVERTER_2 SD24_CONVERTER_3
dataFormat	选择结果以怎样的数据格式，可选值有： SD24_DATA_FORMAT_BINARY [Default] SD24_DATA_FORMAT_2COMPLEMENT 修改寄存器 SD24CCTLx 的 SD24DFx 位。

返回值：无。

void SD24_startConverterConversion (uint16_t baseAddress, uint8_t converter)

转换器开始一个转换。

参数

baseAddress	是 SD24 模块的基地址
converter	选择转换器，可选的值有： SD24_CONVERTER_0 SD24_CONVERTER_1 SD24_CONVERTER_2 SD24_CONVERTER_3 修改寄存器 SD24CCTLx 寄存器的 SD24SC 位

该函数修改寄存器 SD24BCTLx 的 SD24SC 位。

返回：无。

void SD24_stopConverterConversion (uint16_t baseAddress, uint8_t converter)

停止转换器转换。

参数

baseAddress	是 SD24 模块的基地址
converter	选择将被停止的转换器，可选的值有： SD24_CONVERTER_0 SD24_CONVERTER_1 SD24_CONVERTER_2 SD24_CONVERTER_3 修改寄存器 SD24CCTLx 寄存器的 SD24SC 位

该函数修改寄存器 SD24BCTLx 的 SD24SC 位。

返回：无。

我们通过寄存器就可以看出来和上面的开始转换刚好是相反操作的一对。

void SD24_setInputChannel (uint16_t baseAddress, uint8_t converter, uint8_t inputChannel)

该函数用于配置输入通道。MSP430i2xx 系列单片机，用户可以选择模拟输入，也可以选择内部的温度传感器输入（跟 G2xx 类似的测量内部温度的方法）。

参数

baseAddress	是 SD24 模块的基地址
converter	选择将被停止的转换器，可选的值有： SD24_CONVERTER_0 SD24_CONVERTER_1 SD24_CONVERTER_2 SD24_CONVERTER_3 修改寄存器 SD24CCTLx 寄存器的 SD24SC 位
inputChannel	选择转换器的输入通道，可选值有： SD24_INPUT_CH_ANALOG SD24_INPUT_CH_TEMPERATURE 修改寄存器 SD24INCTLx 的 SD24INCHx 位

返回：无。

void SD24_setOversampling (uint16_t baseAddress, uint8_t converter, uint16_t oversampleRatio)

为给定的转换器配置过采样率。

参数

baseAddress	是 SD24 模块的基地址
converter	选择将被停止的转换器，可选的值有： SD24_CONVERTER_0 SD24_CONVERTER_1 SD24_CONVERTER_2 SD24_CONVERTER_3
oversampleRatio	选择转换器的过采样率，可选值有： SD24_OVERSAMPLE_32 SD24_OVERSAMPLE_64 SD24_OVERSAMPLE_128 SD24_OVERSAMPLE_256

	改寄存器 SD24OSRx 的 SD24OSRx 位。

返回：无。

void **SD24_setGain** (uint16_t baseAddress, uint8_t converter, uint8_t gain)

配置转换器增益。

参数

baseAddress	是 SD24 模块的基地址
converter	选择将被停止的转换器，可选的值有： SD24_CONVERTER_0 SD24_CONVERTER_1 SD24_CONVERTER_2 SD24_CONVERTER_3
gain	选择转换器的过采样率，可选值有： SD24_GAIN_1 默认值 SD24_GAIN_2 SD24_GAIN_4 SD24_GAIN_8 SD24_GAIN_16 修改寄存器 SD24INCTLx 的 SD24GAINx 位。

返回：无。

uint32_t **SD24_getResults** (uint16_t baseAddress, uint8_t converter)

返回转换器的结果。该函数从寄存器 SD24MEMx 高 16 位和低 16 位获取结果，并把他们连接起来，实际上该结果是一个最大为 24 位的数字（因此我们不用考虑是高位对齐还是低位对齐了，因为库函数已经做好了一切，该处问题是，到底是高位对齐还是低位对齐，O(∩_∩)O 哈哈~）。

参数

baseAddress	是 SD24 模块的基地址
converter	选择将要获取结果的转换器，可选的值有： SD24_CONVERTER_0 SD24_CONVERTER_1 SD24_CONVERTER_2 SD24_CONVERTER_3 修改寄存器 SD24CCTLx 寄存器的 SD24SC 位

该函数修改寄存器 SD24BCTLx 的 SD24SC 位。

返回：转换结果。

uint16_t **SD24_getHighWordResults** (uint16_t baseAddress, uint8_t converter)

该函数从寄存器 SD24MEMx 获取高 16 位结果，并返回该值。

参数

baseAddress	是 SD24 模块的基地址
converter	选择将要获取结果的转换器，可选的值有： SD24_CONVERTER_0 SD24_CONVERTER_1 SD24_CONVERTER_2 SD24_CONVERTER_3 修改寄存器 SD24CCTLx 寄存器的 SD24SC 位

该函数修改寄存器 SD24BCTLx 的 SD24SC 位。

返回：转换结果。

void **SD24_enableInterrupt** (uint16_t baseAddress, uint8_t converter, uint16_t mask)

该函数使能 SD24 模块对应中断。并不清除中断标志。

参数

baseAddress	是 SD24 模块的基地址
converter	选择转换器，可选的值有： SD24_CONVERTER_0 SD24_CONVERTER_1 SD24_CONVERTER_2 SD24_CONVERTER_3
mask	要被启用的变换器中断源的位掩码。掩码值是下面量值的逻辑或：

	SD24_CONVERTER_INTERRUPT SD24_CONVERTER_OVERFLOW_INTERRUPT 修改寄存器 SD24BIE 的 SD24OVIEx 位。

返回：无。

void **SD24_disableInterrupt** (uint16_t baseAddress, uint8_t converter, uint16_t mask)

关闭 SD24 模块中断功能。

参数

baseAddress	是 SD24 模块的基地址
converter	选择转换器，可选的值有： SD24_CONVERTER_0 SD24_CONVERTER_1 SD24_CONVERTER_2 SD24_CONVERTER_3
mask	要被关闭的变换器中断源的位掩码。掩码值是下面量值的逻辑或： SD24_CONVERTER_INTERRUPT SD24_CONVERTER_OVERFLOW_INTERRUPT 修改寄存器 SD24BIE 的 SD24OVIEx 位。

返回：无。

void **SD24_clearInterrupt** (uint16_t baseAddress, uint8_t converter, uint16_t mask)

该函数清除 SD24 模块中断标志。

参数

baseAddress	是 SD24 模块的基地址
converter	选择转换器，可选的值有： SD24_CONVERTER_0 SD24_CONVERTER_1 SD24_CONVERTER_2 SD24_CONVERTER_3
mask	要被清除掉的转换器对应的中断源的位掩码。掩码值是下面量值的逻辑或：

	SD24_CONVERTER_INTERRUPT SD24_CONVERTER_OVERFLOW_INTERRUPT 修改寄存器 SD24BIFG 的 SD24OVIFGx 位。

返回：无。

uint16_t **SD24_getInterruptStatus** (uint16_t baseAddress, uint8_t converter, uint16_t mask)

该函数可以获取 SD24 模块的中断标志位状态。

参数

baseAddress	是 SD24 模块的基地址
converter	选择转换器，可选的值有： SD24_CONVERTER_0 SD24_CONVERTER_1 SD24_CONVERTER_2 SD24_CONVERTER_3
mask	要返回的转换器中断源位掩码。掩码值是下面量值的逻辑或： SD24_CONVERTER_INTERRUPT SD24_CONVERTER_OVERFLOW_INTERRUPT

返回：下列量值的逻辑或：

SD24_CONVERTER_INTERRUPT

SD24_CONVERTER_OVERFLOW_INTERRUPT

表明对应掩码的中断状态。该函数可以知道是发生了什么中断。

5.3 例程

本例程非完整程序，旨在展示如何初始化和而是用 SD24 的 API，以开始一个单通道转换。

```
unsigned long results;
SD24_init(SD24_BASE, SD24_REF_INTERNAL); // Select internal REF
SD24_initConverterAdvancedParam param = {0};
```

```
param. converter = SD24_CONVERTER_2; // Select converter

param. conversionMode = SD24_SINGLE_MODE; // Select single mode

param. groupEnable = SD24_NOT_GROUPED; // No grouped

param. inputChannel = SD24_INPUT_CH_ANALOG; // Input from analog signal

param. dataFormat = SD24_DATA_FORMAT_2COMPLEMENT; // 2's complement data format

param. interruptDelay = SD24_FOURTH_SAMPLE_INTERRUPT; // 4th sample causes interrupt

param. oversampleRatio = SD24_OVERSAMPLE_256; // Oversampling ratio 256

param. gain = SD24_GAIN_1; // Preamplifier gain x1

SD24_initConverterAdvanced(SD24_BASE, &param);

delay cycles(0x3600); // Delay for 1.5V REF startup

while (1)

{

SD24_startConverterConversion(SD24_BASE,SD24_CONVERTER 2); // Set bit to start conversion

// Poll interrupt flag for channel 2

while( SD24_getInterruptStatus(SD24_BASE,_SD24_CONVERTER_2,SD24_CONVERTER_INTERRUPT) == 0 );

    results = SD24 getResults(SD24_BASE,SD24_CONVERTER_2); // Save CH2 results (clears IFG)

no operation(); // SET BREAKPOINT HERE

}
```

讨论：根据例程，我们可以了解如何使用 SD24 的基本流程。

1，选择参考 REF，是内置参考电压，还是选择外置参考电压。

2，高级配置转换器相关参数，共 8 个，这个很关键。

3，通过相关函数，将第 2 步配置的参数写入到系统。

4，延时一下，等待参考电压就绪，就可以启动转换器了。

5，根据所选转换器的中断标志位来判断转换完成了吗？

6，判断完成转换后就可以通过获取结果的相关函数，获取转换的结果了。

7，如果需要下次转换，记得清除中断标志。

不同情况下有不同的应用，具体详见库函数包里的例程。共有 6 种用法。

问题：param. groupEnable = SD24_NOT_GROUPED; // No grouped

分组是什么意思，大家可以认真阅读相关资料，积极探讨。

其他问题大家自己提，然后网友一起讨论。

第六章　FlashCtl-Flash 存储控制器

6.1　本章引言

闪存模块有一个联合控制器，可以控制编程和擦除操作。可以写入单个比特、 字节或字到闪存，但是闪存可擦除的最小单位是块（段)。闪存被划分为主要和信息存储块。在操作主存储块和信息存储块时候是没有差别的。代码和数据可以位于每个块。请参阅特定于设备的数据工作表，来确定每个块的起止地址，同时，还有完整的设备内存映射。本库函数提供用于闪存块的擦、写及操作状态检测的 API。

本驱动程序包含在 flashctl.c 文件里，头文件在 flashctl.h 里。

6.2　函数总览

1	void FlashCtl_eraseSegment (uint8_t *flash_ptr) 擦除闪存的一个块（段）
2	void FlashCtl_performMassErase (uint8_t *flash_ptr) 擦除全部闪存
3	bool FlashCtl_performEraseCheck (uint8_t *flash_ptr, uint16_t numberOfBytes) 擦除闪存记忆体检查。
4	void FlashCtl_write8 (uint8_t *data ptr, uint8_t *flash_ptr, uint16_t count) 以 8 位的字节单位写入闪存，通过引用传递（这里就是指针传递的意思)。
5	void FlashCtl_write16 (uint16_t *data ptr, uint16_t *flash_ptr, uint16_t count) 以 16 位的字单位写入闪存，通过引用传递
6	void FlashCtl_write32 (uint32_t *data ptr, uint32_t *flash_ptr, uint16_t count) 以 32 位的字单位写入闪存，通过引用传递
7	void FlashCtl_fillMemory32 (uint32_t value, uint32_t *flash_ptr, uint16_t count) 以 32 位的字单位写入闪存，通过值传递。
8	uint8_t FlashCtl_getStatus (uint8_t mask) 检查 FlashCtl_status，看看是否擦除或编程的当前正忙。
9	void FlashCtl_lockInfo (void) 锁数据存储区部分。
10	void FlashCtl_unlockInfo (void) 解锁数据存储区部分。
11	uint8_t FlashCtl_setupClock (uint32_t clockTargetFreq, uint32_t clockSourceFreq, uint16_t clockSource) 设置闪存模块时钟。

FlashCtl_segmentErase() 可以帮助清除单个块段的闪存。一个指向将要被擦除的闪存块段的指针传递给该函数。

FlashCtl_performEraseCheck() 帮助检查在闪存中的特定数量的字节目前是否被擦除。指向擦除检查和要检查的字节数的起始位置的指针传递到此函数。

根据即将被执行写入到闪存的类型，此库函数提供了 Api 用于闪存的写操作。

FlashCtl_write8() 函数以字节为单位写入闪存。FlashCtl_write16() 函数以字为单位写入闪存。FlashCtl_write32() 以长整型为单位写入闪存,通过引用（指针)。FlashCtl_memoryFill32() 函数以长整型，通过传递值方式写入闪存。FlashCtl_getStatus() 检测是否闪存当前忙于擦写状态。FlashCtl_lockInfo() 锁信数据储区。FlashCtl unlockInfo() 解锁数据存储区。

闪存 API 可以分为 4 组：那些处理闪存擦除的，那些写闪存的，那些获取闪存状态的，还有那些锁/解锁数据存储区的。

擦除操作的有：

FlashCtl_segmentErase()

FlashCtl_massErase()

写闪存操作的有：

FlashCtl_write8()

FlashCtl_write16()

FlashCtl_write32()

FlashCtl_memoryFill32()

获取状态的有：

FlashCtl getStatus()

FlashCtl performEraseCheck()

数据信息存储区（块、段）锁/解锁操作的有：

FlashCtl lockInfo()

FlashCtl unlockInfo()

闪存时钟设置的有： FlashCtl setupClock()

详解

void FlashCtl_eraseSegment (uint8_t *flash_ptr)

擦除单个闪存块段。

对于像 MSP430i204x 的 MCU，如果指定的部分是信息(数据)闪存段，调用此 API 之前必须调用 FLASH_unlockInfo。

参数

flash_ptr	指向即将被擦除的闪存块的指针

返回：无。

void FlashCtl_performMassErase (uint8_t *flash_ptr)

擦除全部闪存。对于像 MSP430i204x 的 MCU，如果 FLASH_unlockInfo API 是提前被执行过，此 API 将擦除主内存和信息内存。此外注意到擦除 MSP430i204x 的信息闪存，会影响位于信息内存的 TLV 标定常数。

参数

flash_ptr	指向即将被擦除的闪存块的指针

返回：无。

bool FlashCtl_performEraseCheck (uint8_t *flash_ptr, uint16_t numberOfBytes)

该函数检测闪存中的字节，来确认他们是否被擦除了（即设置为了 0xFF）。

参数

flash_ptr	指向即将被擦除的闪存块的开头区指针
numberOfBytes	将被检测的字节数量

返回：SUCCESS 或 FALL。

void FlashCtl_write8 (uint8_t *data ptr, uint8_t *flash_ptr, uint16_t count)

此函数将 count 数量的字节数组写入到闪存。假定已经擦掉，且解锁了闪存。FlashCtl_eraseSegment 可以用来擦除一段。

参数

data_ptr	指向即将被写的数据的指针
flash_ptr	指向被写入数据的闪存地址
count	要 写入的数量

返回：空

void FlashCtl_write16 (uint16_t *data ptr, uint16_t *flash_ptr, uint16_t count)

此函数将 count 数量的 16-bits 字数组写入到闪存。假定已经擦掉，且解锁了闪存。FlashCtl_eraseSegment 可以用来擦除一段。

参数

data_ptr	指向即将被写的数据的指针
flash_ptr	指向被写入数据的闪存地址
count	要 写入的数量

返回：空

void FlashCtl_write32 (uint32_t *data ptr, uint32_t *flash_ptr, uint16_t count)

此函数将 count 数量的 32-bits 长整型数组写入到闪存。假定已经擦掉，且解锁了闪存。FlashCtl_eraseSegment 可以用来擦除一段。

参数

data_ptr	指向即将被写的数据的指针
flash_ptr	指向被写入数据的闪存地址
count	要 写入的数量

返回：空

void FlashCtl_fillMemory32 (uint32_t value, uint32_t *flash_ptr, uint16_t count)

此函数将 count 次的 32-bits 长整型数据写入到闪存。假定已经擦掉，且解锁了闪存。FlashCtl_eraseSegment 可以用来擦除一段。

参数

value	用于装填内存的值（该值可以用来填充内存空间）
flash_ptr	指向被写入数据的闪存地址
count	要 写入的数量

返回：空

uint8_t FlashCtl_getStatus (uint8_t mask)

此函数将检查状态寄存器，以确定是否已准备好写闪存。

参数

mask	FLASHCTL 读取的状态掩码值是下面量值的逻辑或： LASHCTL READY FOR NEXT WRITE FLASHCTL ACCESS VIOLATION INTERRUPT FLAG FLASHCTL PASSWORD WRITTEN INCORRECTLY FLASHCTL BUSY

返回：下面量值的逻辑或

LASHCTL READY FOR NEXT WRITE

FLASHCTL ACCESS VIOLATION INTERRUPT FLAG

FLASHCTL PASSWORD WRITTEN INCORRECTLY

FLASHCTL BUSY

指示 FlashCtl 的状态。

void FlashCtl_lockInfo (void)

该函数通常在其他 API 函数为了执行重新锁信息闪存区擦写操作后被调用。

返回值：空

void FlashCtl_unlockInfo (void)

该函数在被其他函数执行擦写操作信息闪存区之前调用。

返回值：空

uint8_t FlashCtl_setupClock (uint32_t clockTargetFreq, uint32_t clockSourceFreq, uint16_t clockSource)

这个函数用于设置 flash 模块时钟。任何其他闪存的 API 函数在调用之前，通常会调用此函数。

参数

clockTargetFreq	目标时钟源频率 Hz
clockSourceFreq	时钟源频率 Hz
clockSource	闪存的时钟源。可选值有： FLASHCTL_MCLK （默认） FLASHCTL_SMCLK

返回：时钟设置结果，指示时钟设置成功或失败。

6.3 例程

```c
#include "driverlib.h"

// Address of the beginning of the Flash Information Segment
#define SEGSTART 0x1060
// Number of bytes within segment to write
#define SEG_LEN 16

// Value to write to segment
const uint32_t Value = 0xBEEFDEAD;

void write_InfoSeg(uint32_t value);

void main(void) {
    // Pointer to beginning of Flash segment
    uint32_t *flashPtr = (uint32_t *)SEGSTART;

    // Stop WDT
    WDT_hold(WDT_BASE);

    // Setting the DCO to use the internal resistor. DCO will be at 16.384MHz
    CS_setupDCO(CS_INTERNAL_RESISTOR);

    // Setting MCLK to DCO / 1. MCLK = 16.384MHz.
    CS_initClockSignal(CS_MCLK, CS_CLOCK_DIVIDER_1);

    // MCLK for Flash Timing Generator
    // Flash clock will run at ~390kHz. Datasheet recommends 257kHz - 476kHz
    FlashCtl_setupClock(390095, 16384000, FLASHCTL_MCLK);

    FlashCtl_unlockInfo();
    FlashCtl_fillMemory32(Value, flashPtr, SEG_LEN / 4);
    FlashCtl_lockInfo();

    while(1)
    {
        // Set breakpoint to view memory
        __no_operation();
    }
}
```

例程摘自库函数包里的例程。由例程可以方便我们理解库函数应用教程。首先通过指针标记要被操作的闪存块地址，之后设置系统的时钟系统，例程选择使用内部电阻模式的 DCO，且不分频。闪存时钟配置，可以看到三个参数的意义，第一个，闪存工作的频率，第二个，闪存引用的时钟源频率，第三个参数是闪存使用的时钟源。后面就要解锁数据存储区，之后写入数据，写完要记得再次锁数据存储区。

你学会了吗？欢迎大家一起讨论学习本科的感想，以及疑问，在帖子后面提出，本课程不再布置讨论主题与作业。试着根据例程修改，尝试例程中其他没有提到的 API 用法。