第七章　电源管理模块（PMM）

7.1　本章引言

PMM 提供管理电源及监视其相关设备的所有函数。它的主要功能是：生成一个电源电压的核心逻辑；提供监督和监测设备的电压(VCC)和核心(VCORE)电压发生器的机制。

PMM 使用一个集成的低压差稳压器(LDO)，用于从 VCC 提供的初级电压产生一个二级核心电压(VCORE)。VCORE 提供给 CPU、内存(flash / RAM),和数字模块,而 VCC 提供给 I / O 模块和模拟模块。VCORE 输出使用一个通过 PMM 内置的参考模块产生的电压基准维持。输入或调节器的初级侧被称为其高侧。输出或次级侧被称为其低侧。

PMM 特性包括：

电源电压（VCC）范围：2.2 V 至 3.6 V

高端掉电复位（BOTH）

针对 VCC 的阈值等级可编程电源电压监控和外部管脚（VMONIN）对内部参考监控。

1.8 V 固定电压设备核心（VCORE）发生器

电源电压监控器（SVS）的 VCORE

为整个器件和集成温度传感器提供精确的 1.16-V 参考。

7.2　函数总览

1	**void PMM_setupVoltageMonitor(uint8_t voltageMonitorLevel)**
	设置电压监视器
2	**void PMM_calibrateReference (void)**
	校准
3	**void PMM_setRegulatorStatus (uint8_t status)**
	配置 PMM 校准器状态
4	**void PMM_unlockIOConfiguration (void)**
	解锁 IO
5	**void PMM_enableInterrupt (uint8_t mask)**
	使能中断
6	**void PMM_disableInterrupt (uint8_t mask)**
	关闭中断
7	**void PMM_getInterrupt (uint8_t mask)**
	返回中断状态
8	**void PMM_clearInterrupt (uint8_t mask)**
	清除中断掩码

函数分类

PMM API 可以分成三组函数：那些设置 PMM 的，那些使用 LPM4.5 模式的，还有那些用于 PMM 中断的。

设置 PMM 的有：

void PMM_setupVoltageMonitor(uint8_t voltageMonitorLevel)SD24_initConverter()

void PMM_calibrateReference (void)SD24_startConverterConversion()

用于使用 LPM4.5 模式的有：

void PMM_setRegulatorStatus (uint8_t status)

void PMM_unlockIOConfiguration (void)

用于 PMM 中断的有：

void PMM_enableInterrupt (uint8_t mask)

void PMM_disableInterrupt (uint8_t mask)

void PMM_getInterrupt (uint8_t mask)

void PMM_clearInterrupt (uint8_t mask)

详细描述

void PMM_setupVoltageMonitor(uint8_t voltageMonitorLevel)

校准。

修改寄存器 REFCAL0 和 REFCAL1.

返回值：空。

void PMM_clearInterrupt (uint8_t mask)

清中断掩码。

参数

mask	掩码值是下面数值的逻辑或

	PMM_LPM45_INTERRUPT	LPM4.5 中断

返回：无。

void PMM_disableInterrupt (uint8_t mask)

关闭中断。

参数

mask	掩码值是下面数值的逻辑或 **PMM_VMON_INTERRUPT** 电压监视器中断

返回：无。

void PMM_ensableInterrupt (uint8_t mask)

开启中断。

参数

mask	掩码值是下面数值的逻辑或 **PMM_VMON_INTERRUPT** 电压监视器中断

返回：无。

unit8_t PMM_getInterruptStatus (uint8_t mask)

返回中断状态。

参数

mask	掩码值是下面数值的逻辑或 **PMM_VMON_INTERRUPT** 电压监视器中断 **PMM_LPM45_INTERRUPT** LPM4.5 中断

返回：下面量的逻辑或。

PMM_VMON_INTERRUPT 电压监视器中断

PMM_LPM45_INTERRUPT LPM4.5 中断

指示中断掩码状态

void PMM_setRegulatorStatus (uint8_t status)

设置 PMM 校准器的状态

参数

mask	可选值有： **PMM_REGULATOR_ON** 开启 PMM 校准器
	PMM_REGULATOR_OFF 关闭 PMM 校准器 修改 LPM45CTL 寄存器 REGOFF 注释：手册解释的刚好颠倒，是不是错了？？

修改 LPM45CTL 寄存器。

返回：无。

void PMM_setupVoltageMonitor(uint8_t voltageMonitorLevel)

配置电压监视器

参数

voltageMonitorLevel	可选值有： **PMM_DISABLE_VMON** 关闭电压监视器 **PMM_DVCC_2350MV** **DVCC 与 2350mV 比较** **PMM_DVCC_2650MV** **DVCC 与 2650mV 比较** **PMM_DVCC_2850MV** **DVCC 与 2850mV 比较** **PMM_VMONIN_1160MV** **VMONIN 与 1160mV 比较** 修改寄存器 VMONCTL 的 REGOFF 位

修改 VMONCTL 寄存器。

void PMM_unlockIOConfiguration (void)

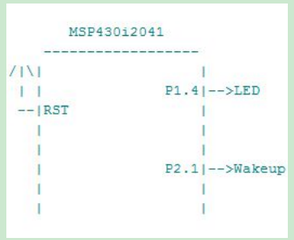
解锁 IO.

修改寄存器 LMP45CTL 的 LOCKLPM45 位。

返回：无。

7.3 例程

描述：在这个例子中，设备设置进入 LPM4.5.在进入 LPM4.5 之前 LED 是开启的来只是进入 LPM4.5。退出 LPM4.5 由 P2.1 的上升沿触发，以导致设备的重置。一旦重置，LPM4.5 中断设置和设备开始仅仅翻转 LED 状态。该例程示范如何配置设备的 LPM4.5 和从它成功的中断



```
#include "driverlib.h"
#define GPIO_PIN_ALL (GPIO_PIN0 | GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3 |
GPIO_PIN4 | GPIO_PIN5 | GPIO_PIN6 | GPIO_PIN7)
int main(void) {
WDT_hold(WDT_BASE);
// Configure GPIO for low current numbers while in LPM4.5
//下面四条函数把 P1 和 P2 的八个 IO 配置成输出，并拉低电平。
GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN_ALL);
GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN_ALL);
GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN_ALL);
GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN_ALL);
// Determine if coming out of LPM 4.5
// 判断是否从 LPM4.5 出来
if(PMM_getInterruptStatus(PMM_LPM45_INTERRUPT))
{
PMM_clearInterrupt(PMM_LPM45_INTERRUPT);
// Need to unlock IO after exiting LPM4.5 so LED will blink
// 需要解锁后退出 LPM4.5 IO,这样 LED 将会闪
PMM_unlockIOConfiguration();
}
else
{
// Configure exit of LPM4.5 on P2.1 interrupt
// 配置 P2.1 的中断以退出 LPM4.5,P2.1 设置为输入，选择上升沿中断，清除中断源，使能中断
GPIO_setAsInputPin(GPIO_PORT_P2, GPIO_PIN1);
GPIO_selectInterruptEdge(GPIO_PORT_P2, GPIO_PIN1,
GPIO_LOW_TO_HIGH_TRANSITION);
GPIO_clearInterrupt(GPIO_PORT_P2, GPIO_PIN1);
GPIO_enableInterrupt(GPIO_PORT_P2, GPIO_PIN1);
// Turn on LED to indicate we are about to enter LPM4.5
// 开 LED，指示我们进入了 LPM4.5,LED 在 P1.4 管脚，输出高电平。
GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN4);
// Turn off regulator so we can enter LPM4.5
// 关校准器，这样我们才能进入 LPM4.5,之后通过特殊寄存器指令进入 LPM4.5
PMM_setRegulatorStatus(PMM_REGULATOR_OFF);
__bis_SR_register(LPM4_bits);
}
// Slow down clock so we can see LED blink
//减缓时钟,所以我们可以看到 LED 闪烁
// Configure MCLK = ~1MHz
// DCO 默认采用 16.384Mhz,获取 1MHz 频率，需要进行 16 分频，在循环体翻转，延时 1s
CS_initClockSignal(CS_MCLK, CS_CLOCK_DIVIDER_16);
while(1)
{
GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN4);
__delay_cycles(100000);
}
}
```

7.4 问题

1.查找本系列相关资料，了解 LPM4.5，帖子后面回复发表个人理解与见解。

2.查找该家族手册了解 calibrateReference 是怎么回事。

3.查看 driverlib_2_00_00_16\examples\MSP430i2xx\pmm_pmm_ex2_vmon1160mV.c 例程，学习相关 APP 用法，并在帖子后参与内容学习与讨论，可以是对该例程的理解，也可以是相关疑问或问题。

第八章　Tag Length Value(TLV)

6.1　本章引言

TLV（标签长度值）是一个存储在闪存的表,其中包含特定于设备的信息。包含重要的信息为使用和校准设备。一个 TLV 内容的列表也可在特定于设备的数据表 （在 TLV 部分），其具体功能上的解释在 MSP430i2xx Family User's Guide 中提供。

本驱动程序包含在 tlv.c 文件里，头文件在 tlv.h 里。

6.2　函数总览

1	void TLV_getInfo (uint8_t tag,uint8_t *length , uint16_t **data_address)
	获取 TLV 信息
2	bool TLV_performChecksumCheck (void)
	执行对 TLV 校验检查

这组 API 用于查询 TLV 结构中的信息。

TLV_getInfo() 该函数获取一个标签的值和这个标签的长度。

TLV_performaChecksumCheck() 该函数对 TLV 执行 CRC 检查。

详解

void TLV_getInfo (uint8_t tag,uint8_t *length , uint16_t **data_address)

获取 TLV 信息。

TLV 结构使用一个标签或标识表段的基地址信息存储的地方。这可以用于检索设备的校准常数或找出有关该设备的详细信息。此函数可检索请求标签的地址和标签的长度。请参考你设备的手册，确认标签是否可用。

参数

tag	代表的标签需要检索的信息。可用值有： TLV_CHECKSUM TLV_TAG_DIE_RECORD TLV_LENGTH_DIE_RECORD TLV_VAFER_LOT_ID TLV_DIE_X_POSITION TLV_DIE_Y_POSITION TLV_TEST_RESULTS TLV_REF_CALIBRATION_TAG TLV_REF_CALIBRATION_LENGTH TLV_REF_CALIBRATION_REFCAL1 TLV_REF_CALIBRATION_REFCAL0 TLV_DCO_CALIBRATION_TAG TLV_DCO_CALIBRATION_LENGTH TLV_DCO_CALIBRATION_CSIRFCAL TLV_DCO_CALIBRATION_CSIRTCAL TLV_DCO_CALIBRATION_CSERFCAL TLV_DCO_CALIBRATION_CSIETCAL TLV_SD24_CALIBRATION_TAG TLV_SD24_CALIBRATION_LENGTH TLV_SD24_CALIBRATION_SD24TRIM
lentgh	通过间接引用作为返回值执行。此函数检索 TLV 标签长度值。一旦调用该函数，该值通过*length 指定，被应用层使用。
data_address	通过间接引用作为返回值执行。一旦该函数被调用，data_address 指向的指针值从指定检索 TAG 标签。

返回：无。

bool TLV_performChecksumCheck (void)

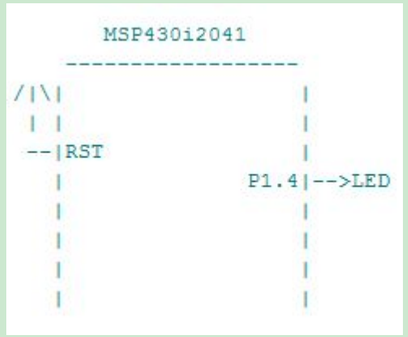
执行 TLV 的校验和检测。

2 秒补充校验和计算的 TLV 中存储的数据。如果计算出的校验和是等于在 TLV 存储的校验和，用户可知道 TLV 未损坏。这个 API 可以在 BOR 之后使用写配置常量到相应的寄存器之前。

返回值：如果 TLV 校验和匹配存储在 TLV 的值，即为真，否则为假。

6.3 例程

描述：在该例程中，TLV的校验和是被检测，以确认TLV没有毁坏。如果TLV没有错误，LED点亮。造成错误，首先备份TLV，然后擦除它的一部分。这将导致发生校验和错误，LED不会被点亮。




```
#include "driverlib.h"
int main(void) {
bool result;
// Stop the WDT
WDT_hold(WDT_BASE);
// Check the TLV checksum
result = TLV_performChecksumCheck();
// Turn on LED if test passed
if(result)
{
GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN4);
}
else
{
GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN4);
}
// LED for indicating checksum result
GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN4);
__bis_SR_register(LPM0_bits);
}
```

该例程看似很简单，那么我们的问题就是 TLV 具体是用来干什么的？查找资料参与讨论。

回顾：

我们一共学习了 8 种外设使用方法，还有很多功能我们没有学习，相信你也能够结合这几节课的内容自学其他的部分，如果你要用得其他部分到的话。根据这四节课程相信你也对 MSP430i2xx 家族，以及相关库函数的结构和编写的特点有了一定的认识，对未来深入学习打下了基础。MSP430i2xx 是为工业级应用设计的，我们可以用于电表和水表的应用设计，特别精简的 CS 系统，让你的作品不会因为外部晶振和复杂的时钟设计而产生累赘感。该家族集成的 DCO 拥有超高的精度，可以完胜你关于时钟相关的大部分应用。ADC 模块使用了 SD24，高精度的模数模块可以胜任你所有的 ADC 检测应用，该 ADC 在仪表应用有出色的表现。本课程会提供若干相关应用文档下载，帮助你更好的了解 430 所擅长的领域。该系列 I/O 与以往 430 不同的地方在于不存在上下拉电阻，这个可以参考该家族技术手册了解。最后祝大家学习进步，本版主能力有限，部分翻译可能很不到位，也希望该教材起到抛砖引玉的效果，重要的是让你知道 TI 推出了一款新的 430 单片机，它更加适合工业级的设计应用。同时开心的告诉大家，TI 终于把 430 抄底功耗特性引入了 32 位机，新推出的 MSP432 单片机拥有 430 的超低功耗特性，同时具备 32 位 ARM 核心。另外 TI 也推出的有 MSP430FRxx 系列，该系列采用铁电存储，并提供库函数集，在 430 家族 16 位机里具备最为丰富的外设和功能，有兴趣的可以了解一下。