

CMT2219A 配置流程

本章将指引用户进行如何使用 MCU 对 CMT2219A 进行配置参数。

一. 需要准备的工具和软件

- Arduino 1.0.5 版本 IDE（可选，也可以使用用户熟悉的 MCU 平台）
- HopeDuino 板（可选，也可以使用用户熟悉的实验板）
- USB 连接线（A 口转 B 口）
- CMOSTEK USB Programmer（可选）
- CMOSTEK RFPDK V1.38（注意使用最新版本，本文编辑时最新为 V1.38）
- RFM219S 模块（基于 CMT2219A 芯片）及配套的转换板



RFM219S



RFM119

二. CMT2219A 两种工作模式

➤ Active Mode

Operation Settings				
Operation Mode <input checked="" type="radio"/> Active <input type="radio"/> Passive	Duty-Cycle Mode On	Sleep Timer On	Sleep Time (3-134152192) 3 ms	Rx Timer On
	Rx Time (0.04-2683043) 2000.00 ms	Rx Time Ext (0.04-2683043) NA ms	Rx Early-Exit Off	State After Rx Exit NA
	Wake-On Radio Off	Wake-On Condition NA		

Active 模式下的 CMT2219A 是完全独立工作，完成解调好的数据，通过 DOUT（GPO4、Pin5）输出。在此模式下，用户无需通过 SPI 接口进行操控（不允许用户通过 SPI 操控）；SPI 接口仅适用于 USB Programmer 烧录接口。另外，在 Duty-Cycle 模式配合下，可以实现低功耗处理。但用户需要务必清楚，即使 Duty-Cycle 开启，仍是在 Active 模式下，则 CMT2219A 依然是完全独立工作，芯片内部进行状态调度，而无需 MCU 的介入，也不允许 MCU 介入干预。

注意: Active 模式适合用于 MCU 软件解码的应用场合, 需要生产时通过 CMOSTEK 提供烧录设备(如: USB Programmer 或 Off-line Writer) 进行预先烧录用户需要的参数。

➤ Passive Mode

Operation Settings				
Operation Mode <input type="radio"/> Active <input checked="" type="radio"/> Passive	Duty-Cycle Mode Off	Sleep Timer On	Sleep Time (3-134152192) 400 ms	Rx Timer On
	Rx Time (0.04-2683043) 2000.00 ms	Rx Time Ext (0.04-2683043) NA ms	Rx Early-Exit Off	State After Rx Exit NA
	Wake-On Radio Off	Wake-On Condition NA		

Passive 模式下的 CMT2219A 是工作在非完全独立工作模式，需要 MCU 介入操作才能正常工作起来。另外，在 SleepTimer、RxTimer、以及 DataMode（Direct、Buffer、Packet）模式搭配下，组合状态较多，灵活方便：

SleepTimer	RxTimer	Data Mode		说明
		Direct	Buffer/Packet	
×	×	√	×	同样用于 MCU 的软件解码的应用，与 Active 模式类似，主要区别在于，MCU 可以通过 SPI 进行参数配置（如修改频率、速率等）；并需要 MCU 介入工作状态切换（如切换到接收、睡眠、待机 etc.）。
×	×	×	√	基本与上面模式相同，区别在于 CMT2219A 内部进行数据解码，并填充到 FIFO 供 MCU 读取使用；这种也是同类芯片最为常见的接收模式。
×	√	--	--	RxTimer 有效，此时 CMT2219A 处于半自动模式，需要 MCU 介入控制，通过 SPI 配置其进入 Rx 模式；但进入 Rx 模式后，MCU 等待 RxTimeOut 中断（RX_TMO_FLAG），不允许 MCU 在 Rx 状态下无故切换到其它状态。
√	×	--	--	同样是半自动模式，需要 MCU 介入控制，通过 SPI 配置其进入睡眠模式；但进入睡眠模式后，MCU 等待 SleepTimeOut 中断（SL_TMO_FLAG），不允许 MCU 在 Sleep 状态下无故切换到其它状态。
√	√	--	--	上述两个半自动模式结合，MCU 需要根据 RxTimeOut 和 SleepTimeOut 两个中断信号，通过 SPI 交替让芯片处于 Rx 和 Sleep 两个状态切换，不允许在没有产生这两个中断情况下无故切换。可以简化 MCU 的设计流程。

注意：

1. SleepTimer 和 RxTimer 两个组合，开启任意一个或两个情况下，CMT2219A 状态是半自动模式。此时 MCU 进行状态切换是有限制的，需要按照上表的建议执行。如果需要完全受控于的 MCU，则需要保证这两个 Timer 都关闭；
2. DataMode（Direct、Buffer、Packet）不影响 SleepTimer 和 RxTimer 的组合功能；仅取决于 CMT2219A 是否内部执行解码填充 FIFO；还是由外部 MCU 进行解码。

三、参数配置流程

习惯常规用法的用户，多半都希望通过 MCU 对 CMT2219A 进行参数配置，以避免在生产时额外通过工具进行烧录芯片参数。面对 CMT2219A 多种灵活的工作模式，在不确定手上芯片（或模块）当前烧录参数如何情况下，配置参数流程势必是需要十分注意。为确保用户配置可靠，请严格按下流程操作：

- 芯片（系统）上电等待 20ms 以上；
- 关闭半自动机制；
- 切换到 TUNE 模式，并等待大于 10ms 以上；
- 切换到 SLEEP 模式，并等待大于 1ms 以上；
- 关闭半自动机制时钟校准；

- 切换到 STANDBY 模式;
- 按照 RFPDK 生成的 “*_reg.exp” 内的 62 个参数进行配置;
- 读取 0x3D 寄存器值, 并将其值的 Bit5 置 1 后, 回写到 0x3D 寄存器中;
- 至此, CMT2219A 配置已经完成, 并停留在 Standby 模式。

注意: 完成配置流程后, 操控权交由 MCU, MCU 可以让 CMT2219A 状态进行切换; 但是, 需要结合 SleepTimer 和 RxTimer 配置, 若两者其中一个或两个开启, 则需要考虑 CMT2219A 芯片内部的半自动机制介入, MCU 需要等待其对应的 TimeOut 中断 (TMO 中断), 期间不能无故切换状态。

上述流程示例代码如下:

```
byte tmp;
```

```
//芯片 (系统) 上电等待 20ms 以上  
_delay_ms(50);
```

```
//关闭半自动机制
```

```
tmp = Spi3.bSpi3Read(0x1F);    //close duty-cycle  
tmp &= 0x7F;  
Spi3.vSpi3Write(0x1F00+tmp);
```

```
tmp = Spi3.bSpi3Read(0x23);    //close sleep-timer  
tmp &= 0x7F;  
Spi3.vSpi3Write(0x2300+tmp);
```

```
tmp = Spi3.bSpi3Read(0x25);    //close receive-timer  
tmp &= 0x7F;  
Spi3.vSpi3Write(0x2500+tmp);
```

```
//切换到 TUNE 模式
```

```
vGoFS();  
_delay_ms(10);
```

```
//切换到 SLEEP 模式
```

```
vGoSleep();  
_delay_ms(2);
```

```
//关闭半自动机制时钟校准
```

```
tmp = Spi3.bSpi3Read(0x1F);    //close LFOSC A  
tmp &= 0x9F;  
Spi3.vSpi3Write(0x1F00+tmp);
```

```
tmp = Spi3.bSpi3Read(0x2D);    //close LFOSC B  
tmp &= 0xDF;
```

```
Spi3.vSpi3Write(0x2D00+tmp);
```

```
//切换到 STANDBY 状态
```

```
vGoStandby();
```

```
//按照 RFPDK 生成的 “*_reg.exp” 内的 62 个参数进行配置;
```

```
for(i=0; i<62; i++) //cfg table form exp file
```

```
    Spi3.vSpi3Write(((word)i<<8)|cfg[i]);
```

```
//读取 0x3D 寄存器值，并将其值的 Bit5 置 1 后，回写到 0x3D 寄存器中;
```

```
tmp = Spi3.bSpi3Read(0x3D);
```

```
tmp |= 0x20;
```

```
Spi3.vSpi3Write(0x3D00+tmp);
```

HOPERF