ManThink

OMx02

LoRaWAN 模块使用指南

Specification Version 1.7

目录

1. 总体介绍

1.1 概述

OMx02 模组集成了一款低功耗的单片机（KL17x CortexM0+内核）和支持扩频调制的射频前端（SX1276/SX1278）。OMx02 将 MCU 的硬件资源（GPIO，SPI，IIC，UART）开放给开发者，方便用户基于 OMx02 进行二次开发，从而节省了产品开发周期，提高了产品一次性开发的稳定性，降低了产品的成本。

基于 OMx02 及其他系列产品，门思科技提供自主研发的物联网实时操作系统 Man-Pregnante Operating System(MPOS)及配套的 LoRa/LoRaWAN 协议栈（LWS），基于 MPOS+LWS，可以快速实现基于 LoRa/LoRaWAN 更多更丰富的应用。

MPOS+LWS 通过 lib 文件的形式提供，用户可以通过 OMx02 的硬件资源（SPI，IIC,UART，GPIO）和 MPOS+LWS 库文件快速设计出自己的 LoRaWAN/LoRa 传感器。MPOS+LWS 支持以下特性

- 精准的周期性任务设置和调度，
- 支持信号量，
- 支持一次性事件，
- 支持中断
- 支持精准的 UTC 时间
- 支持 MCU 底层驱动
- 支持硬件驱动的重置
- 支持多 bin

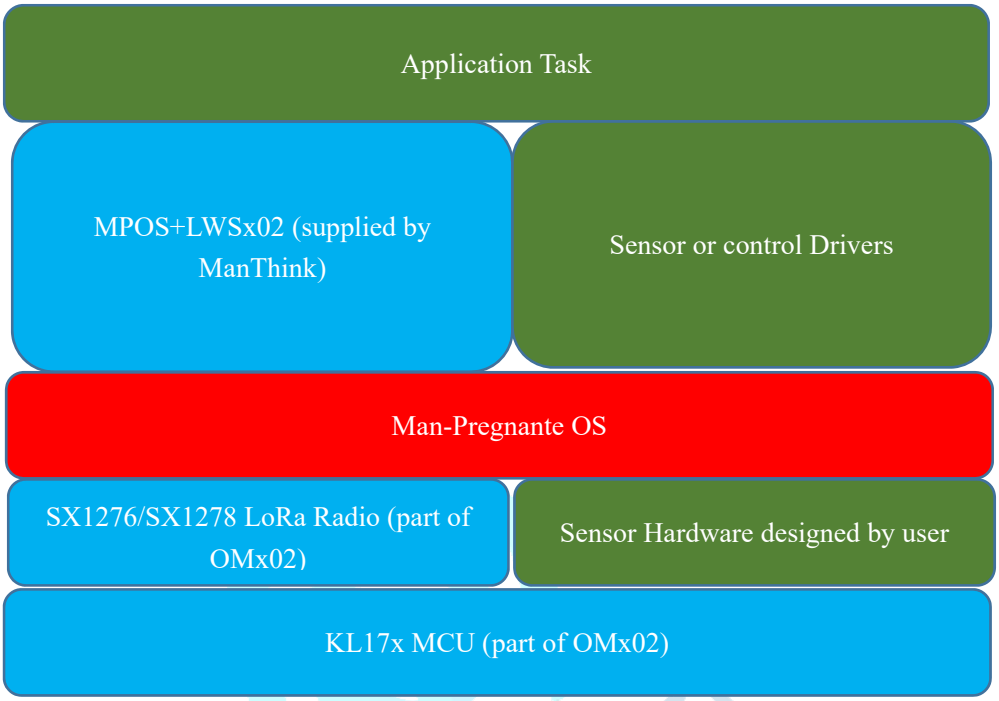
- LWS 支持通过 API 的调用实现 LoRaWAN 的 ClassA，ClassB 和 ClassC，
- 支持 SW 模式
- 支持 FUOTA 等功能。

除了门思提供的 LoRaWAN 协议栈，开发一个 LoRaWAN 的产品还需要传感器的底层驱动和其他的一些硬件控制，这些应用驱动的开发不在本文档描述之内，门思不会提供这部分代码的技术支持。

1.2 特性

- 兼容 LoRaWAN Class A, Class B,Class C 协议（LoRaWAN1.0.2）
- 全球唯一的 64 位标识码(EUI-64™)
- 集成了实时的操作系统(Man-Pregnante Operating System)
- 简单易用的周期性任务
- 丰富的外设接口（SPI，IIC，UART，GPIO)

1.3 架构


Application Task
MPOS+LWSx02 (supplied by ManThink)
Sensor or control Drivers
Man-Pregnante OS
SX1276/SX1278 LoRa Radio (part of OMx02)
Sensor Hardware designed by user
KL17x MCU (part of OMx02)

2. SDK 概述

OMx02 的 SDK 是由库文件（MPOS_LWSx02V1.0.a）、头文件和基于 EWARM 的工程文件构成，SDK 提供的 API 函数实现了对驱动函数，MPOS 系统函数和 LoRaWAN 功能函数的操作。

MPOS 使用基于消息机制的模式通知用户相关事件，并且由轻量的实时操作系统来管理系统的事件和用户的任务。

2.1 禁用资源

OMx02 的一些资源已经被协议栈使用了，禁止用户使用，使用下面列出的禁用资源会导致未知的问题：

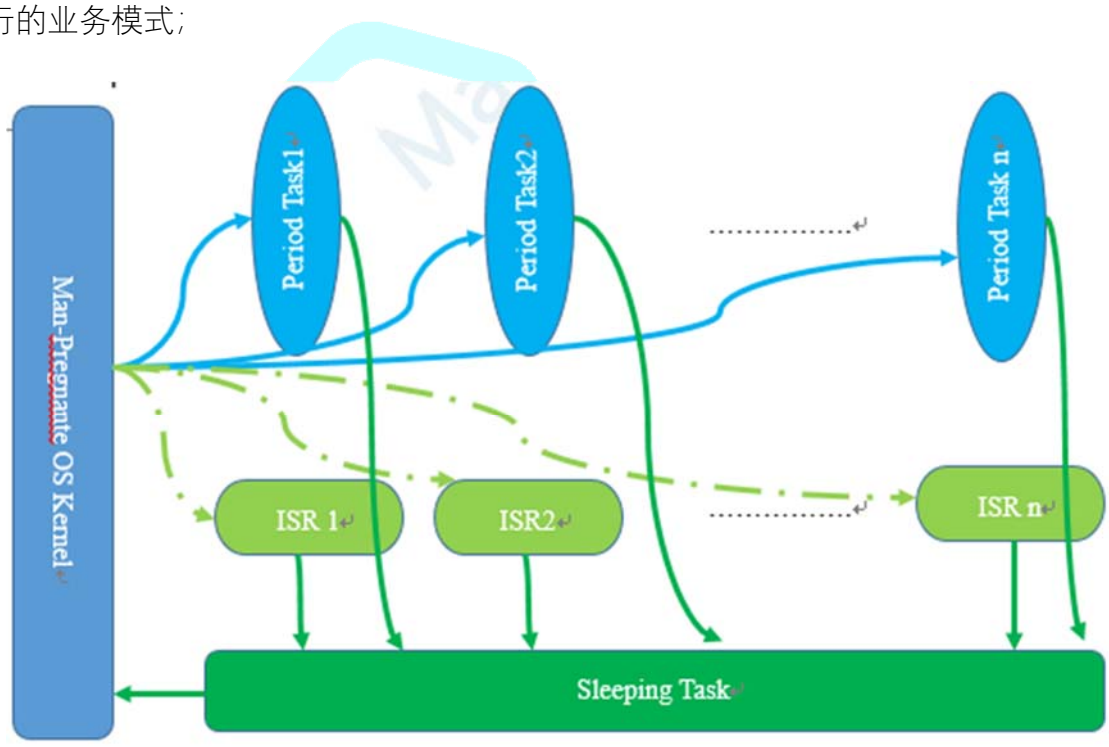
- PA1 , PA2 , PA4
- PB0
- PC5,PC6
- PE0,PE1,PE16,PE17,PE18,PE19
- SPI0 , PIT , LPTIM

2.2 版本信息

序号	版本	支持硬件	功能	备注
1	MPOS_LWS402lite	OM402,OM402S	MPOS,ClassA,ClassC, SW 模式 410-510MHz	
2	MPOS_LWS802lite	OM802,OM802S	MPOS,ClassA,ClassC, SW 模式 860-1020MHz	
3	MPOS_LWS402	OM402S	MPOS,ClassA,ClassB，ClassC SW 模式，多播，FUOTA，多 bin ,410-510MHz	
4	MPOS_LWS802	OM802S	MPOS,ClassA,ClassB,ClassC, SW 模式，多播，FUOTA，多 bin 860-1020MHz	
5	MPOS_LWS411	OM411	MPOS,ClassA,ClassB，ClassC SW 模式，多播，FUOTA，多 bin ,410-510MHz	
6	MPOS_LWS811	OM811	MPOS,ClassA,ClassB,ClassC, SW 模式，多播，FUOTA，多 bin 860-1020MHz	

3. Man-Pregnant Operating System

Man-Pregnant Operating System（MPOS）操作系统专为低功耗和硬件资源偏少的应用而设计，内核代码小于 5kB，可以在 cortex-M0 内核上正常运行。MPOS 是一个时间精度较高的实时调度系统，它提供了三种类型的任务：周期性任务，中断任务和单次事件任务，这三种任务可以帮助客户快速而轻松的实现 IOT 应用，单次事件任务可以将优先级低的任务通过 hook 的方式在系统空闲时运行以避免影响任务时序，中断任务可以实现硬件触发的事件，周期的任务可以实现周期性执行的业务模式；



图片 2 :MPOS 的流程架构

如果没有任务运行，MPOS 的功耗策略是会自动休眠。MPOS 管理所有任务的状态，并且可以在精准的时间唤醒并执行的任务，待任务执行完立即进入休眠状态。在这种策略下，休眠电流不足 3 微安,进入休眠的时间小于 1 毫秒。

3.1 单次事件

当单次事件使能时，它将被执行一次，用户可以把事件链接到一个函数，MPOS 有三个单次事件向用户开放，当没有周期任务执行时，单次事件将在 5 毫秒内被执行。

3.2 周期任务

周期任务是周期性运行的任务，用户需要设置以毫秒为单位的周期，并链接到具体函数，周期任务的时间精度是 5mS±50ppm/秒。

周期性任务可以设置运行次数，并可以在任务执行时对执行次数进行动态修改以满足更多的应用场景。

周期性任务的周期可以在任务执行时进行修改，修改后，下次任务的启动时间将以新的周期执行。

3.3 中断任务

MPOS 可以提供硬件中断函数接口，用户可以在其中添加代码。MPOS 中提供三个单次事件，用户可以在中断内启用单次事件，并将单次事件挂接到自己的函数，也可以在中断函数中直接编写代码以实现最高的优先级的响应。

MPOS 提供 Hook 函数，用户需要将自己定义的中断函数赋值给相应的硬件中断 hook 函数以实现中断服务的功能。

3.4 信号量

MPOS 通过全局变量实现任务之间的通信，对于硬件资源占用和全局变量的

占用可以通过信号量的方式进行保护。

3.5 优先级

MP 操作系统有三个等级的优先级，最高等级的优先级的任务在中断中执行，用户可以在中断函数中编写简单的代码以实现最高优先级的任务。不推荐在中断函数中实现复杂的代码，这样会降低操作系统的效率，建议用单次事件来实现中断函数。

第二个等级的优先级任务是周期任务，所有这个优先级的任务都是基于时间的。MP 操作系统首先会执行时间最早的任务，当某个任务正在运行时，其他任务不会抢占 MCU。

第三个等级的优先级任务是单次事件任务，MP 操作系统定义了几个 Hook 函数，只要启用就执行一次。如果没有中断任务和周期任务运行，Hook 函数将被执行一次。当它被执行了，Hook 函数将被禁止，直到下一个被启用并链接到一个新的函数。

4. MPOS 的 API

4.1 MPOS 的功能位

MPOS 定义了一些功能位以实现不同的功能，功能位在全局变量<StackFunction>中定义。

4.1.1 StackFunction.watchdog

如果该位设置 1，则 MCU 的看门狗将被启用，应用程序应在 8 秒之内喂狗。当调试的时候，可以置 0，禁止看门狗。

4.1.2 StackFunction.nosleep

如果该位设置 1，则 MPOS 将执行与正常模式功能相同的虚拟休眠模式；该位清零则系统会在无任务执行时自动执行休眠指令，实现低功耗功能。任务从执行完毕到进入休眠的时间<1mS.

建议在调试时启用这个虚拟休眠模式，并在产品发布时禁用该模式。

4.1.3 StackFunction.uart1

如果该位设置 1，MPOS 将在内部初始化 UART1，用户可以使用 UART1 的 API 来实现其 UART1 应用。

4.1.4 StackFunction.lorawan

如果该位设置 1，MPOS 将使能模块的 LoRaWAN 功能。

4.1.5 StackFunction.MTprotocol

如果该位设置为 1，则 UART1 通信的协议必须使用 ManThink 制定的通信协议格式。当 MCU 处于休眠状态时，UART1 可以接收到数据，MPOS 会自动解析数据帧，然后提供给用户的解析函数做进一步解析。如果该位设置为 0，则 UAR

T1 在 MCU 休眠时不能接收任何数据，当 MCU 未休眠时，MPOS 将会转发数据给用户。

4.2 系统 API

用于系统硬件驱动的 API 函数。

4.2.1 mpos_driver.Clock_Init

初始化 MCU 的参数，主时钟的频率将被设置为 48MHz 的内部 RC 振荡器。

void **mpos_driver.Clock_Init ()**。

4.2.2 mpos_driver.mcureset

通过调用此函数，可以使 MCU 复位。

void mpos_driver.mcureset ()。

4.2.4 mpos_driver.kickdog

通过这个函数进行喂狗操作，用户可以选择启用或禁用内部看门狗。如果看门狗启用，则应用程序必须通过此函数周期性地喂狗。看门狗的设置时间是 8 秒，用户创建的任务中需要有喂狗的操作，在系统的循环中，系统会自动执行喂狗操作。

void **mpos_driver.kickdog ()**。

4.2.5 mpos_osfun.SysParaInit

系统参数的初始化。

void **mpos_osfun.SysParaInit ()**。

4.2.6 mpos_osfun. DelayUs

系统的延时函数，延时单位:微秒

void **mpos_osfun. DelayUs (unsigned long** nCount**)**。

nCount: 需要延时的微秒。

4.2.7 mpos_osfun. enableInterrupts

使能全局中断。

void **mpos_osfun. enableInterrupts** ().

4.2.8 mpos_osfun. disableInterrupts

禁能全局中断。

void **mpos_osfun. disableInterrupts** ().

4.3 功能性函数

4.3.1 mpos_osfun. memcpy1

从源 src 所指的内存地址的起始位置开始复制 size 个字节到目标 dst 所指的内存地址的起始地址中。

void mpos_osfun. memcpy1 (uint8_t *dst, uint8_t *src, uint16_t size)

dst:需要拷贝数据的目标地址

src: 需要拷贝数据的源地址

size: 需要拷贝数据的字节数

4.3.2 mpos_osfun. os_rlsbf2

把两个 8 位数据转换成一个 16 位小端模式的数据

u2_t mpos_osfun. os_rlsbf2 (u1_t *buf)

buf:两个 8 位数据的首字节地址

return value : 16 位小端模式的数据

4.3.3 mpos_osfun. os_rlsbf4

把 4 个 8 位数据转换成一个 32 位小端模式的数据

u4_t mpos_osfun. os_rlsbf4 (u1_t *buf)

buf:4 个 8 位数据的首字节地址

return value : 32 位小端模式的数据

4.3.4 mpos_osfun. os_rmsbf4

把 4 个 8 位数据转换成一个 32 位大端模式的数据

u4_t mpos_osfun. os_rlsbf4 (u1_t *buf)

buf:4 个 8 位数据的首字节地址

return value：32 位大端模式的数据

4.3.5 mpos_osfun. os_wlsbf2

把 1 个 16 位小端模式的数据转换成 2 个 8 位的数据

void mpos_osfun. os_wlsbf2 (u1_t *buf , u2_t v)

buf:2 个 8 位数据的首字节地址

v：16 位小端模式的数据

4.3.6 mpos_osfun. os_wlsbf3

把 1 个 24 位小端模式的数据转换成 3 个 8 位的数据

void mpos_osfun. os_wlsbf2 (u1_t *buf , u4_t v)

buf:3 个 8 位数据的首字节地址

v：32 位小端模式的数据

4.3.7 mpos_osfun. os_wlsbf4

把 1 个 32 位小端模式的数据转换成 4 个 8 位的数据

void mpos_osfun. os_wlsbf2 (u1_t *buf , u4_t v)

buf:4 个 8 位数据的首字节地址

v：32 位小端模式的数据

4.3.8 mpos_osfun. os_wlsbf8

把 1 个 64 位小端模式的数据转换成 8 个 8 位的数据

void mpos_osfun. os_wlsbf2 (u1_t *buf , u8_t v)

buf:8 个 8 位数据的首字节地址

v：64 位小端模式的数据

4.3.9 mpos_osfun. os_wmsbf3

把 1 个 24 位大端模式的数据转换成 3 个 8 位的数据

void mpos_osfun. os_wmsbf3 (u1_t *buf , u4_t v)

buf:3 个 8 位数据的首字节地址

v : 32 位大端模式的数据

4.3.10 mpos_osfun. os_wmsbf4

把 1 个 32 位大端模式的数据转换成 4 个 8 位的数据

void mpos_osfun. os_wmsbf4 (u1_t *buf , u4_t v)

buf:4 个 8 位数据的首字节地址

v : 32 位大端模式的数据

4.3.11 mpos_osfun. os_wmsbf4

把 1 个 32 位大端模式的数据转换成 4 个 8 位的数据

void mpos_osfun. os_wmsbf4 (u1_t *buf , u4_t v)

buf:4 个 8 位数据的首字节地址

v : 32 位的数据

4.3.12 mpos_osfun. mp_modeACRC

计算多个 8 位数据的 CRC;

u2_t mpos_osfun. mp_modeACRC (unsigned char *buf,unsigned int lenth,u2_t poly)

buf:数据缓存区的首字节地址

v : 数据的长度，单位：字节

poly:计算 CRC 需要的多项式

return value :计算 CRC 的结果

4.3.13 mpos_osfun. crc8_ccit

计算多个 8 位数据的 CRC8;

u1_t mpos_osfun. crc8_ccit (const uint8_t * data, unsigned size)

data:数据缓存区的首字节地址

size: 数据的长度，单位：字节

return value :计算 CRC 的结果

4.4 周期任务的 API

MPOS 为用户提供一种基于周期性任务的工作方式，任务被创建后将按照设定周期运行，用户可以设定任务被执行的次数。任务的执行周期和执行次数可以在任务实体中进行实时修改。

周期性任务可以用来读取传感器的信息，发送数据，控制 GPIO 用于实现物联网设备的一些基本功能。

用户使用 S_periodTask 结构体定义一个周期运行任务,在结构体里面有 5 个变量来管理一个任务。

- Task

  这是一个回调函数，任务周期到时会执行一次。

- Interval

  任务周期的单位是毫秒，精度是 5ms±50ppm/s.

- Cycles

  这个周期任务执行的次数，每执行一次自动减一，递减到 0 以后函数不再执行。

  注意：如果设置为 0xFFFFFFFF 时，此任务会永久执行。

4.4.1 mpos_osfun.Task_Setup

在任务初始化后，用户必须调用此函数才能将任务安装到系统中。

```
void mpos_osfun.Task_Setup (S_periodTask * task)
```

task：需要调用的任务函数指针

4.4.2 mpos_osfun.Task_SetPeriod

用于设置任务执行的周期。

void mpos_osfun. Task_SetPeriod (S_periodTask * task,clock_time_t period)

task:需要设置周期的任务函数的指针

Period:需要设置的周期，单位 :毫秒

4.4.3 mpos_osfun.Task_Restart

重启启动任务函数。

void mpos_osfun.Task_Restart (S_periodTask * task)

task:需要重新启动的任务函数的指针

4.4.4 mpos_osfun.Task_Sleep

设置任务函数休眠的周期，需要在任务函数内部调用此函数，用于改变任务的执行周期;

void mpos_osfun. Task_Sleep (clock_time_t period)

period:设置任务需要休眠的周期，单位毫秒。

4.4.5 mpos_osfun.Task_ExcuteNow

立即启动任务函数的执行。

void mpos_osfun. Task_ExcuteNow (S_periodTask * task)

task:需要立即执行的任务函数的指针

4.4.6 mpos_osfun.Task_Stop

立即停止此任务函数的执行。

void mpos_osfun.Task_Stop (S_periodTask * task)

task:需要立即停止的任务函数的指针

4.4.7 mpos_osfun. Task_Remove

删除此任务函数。

```
void  mpos_osfun. Task_Remove (S_periodTask * task)
```

task:需要删除的任务函数的指针

4.5 信号量

信号量用于对调用资源的保护，比如 RS-485 通信，存在多个任务调用 UART 的场景，调用任务需要通过信号量对 UART 进行保护。

信号量用 SEMPHORE 进行定义。

4.5.1 Task_sem_init

初始化信号量，将信号量初始化为具有一定资源量，信号量在使用前必须经过此初始化过程。

void mpos_osfun.Task_sem_init(SEMPHORE * mysem, int counts)

mysem : 信号量指针。

counts: 信号量资源数

4.5.2 Task_sem_release

信号量释放，当资源用完之后，需要调用此函数释放信号量。

void mpos_osfun.Task_sem_release(SEMPHORE * mysem)

mysem : 信号量指针。

4.5.3 Task_sem_waitOne

信号量等待，如果函数返回 true，说明资源可用。函数返回 false，则在指定时间延时后会重新检查该资源，直到获取该资源。

对于高优先级别的任务，可以将 waitTime 设置短一些，则可以以更高优先级获取资源。

bool mpos_osfun.Task_sem_waitOne(S_periodTask* task,SEMPHORE* mysem,s8_t waitTime)

task: 调用该信号量的任务。

mysem：信号量指针。

waitTime：下次检查信号量的时间，单位：毫秒。

4.6 单次事件的 API

当单次事件被启用，单次事件将被执行一次，在 MPOS 中有三个单次事件。

4.6.1 单次事件的使能位

变量 LPInfo.wakingBit.Bits 中有 3 个位用于启用/禁用下面列出的单次事件：

- LPInfo.wakingBit.Bits.OnceEvent1
- LPInfo.wakingBit.Bits.OnceEvent2
- LPInfo.wakingBit.Bits.OnceEvent3

有 3 个 Hook 函数用于把用户的函数挂钩到单次事件：

- void (* HookExcuteOnce1)()
- void (* HookExcuteOnce2)()
- void (* HookExcuteOnce3)()

4.6.2 执行单次事件的流程

启用单次事件需要两个步骤， 首先置位 OnceEvent 位，然后将用户的函数链接到具体的钩子函数。例如：

```
LPInfo.wakingBit.Bits.OnceEvent1=1;

HookExcuteOnce1=OnceEventTest1;
```

4.7 中断

基于 KL17 的 MPOS 开启了 5 个中断事件，用户可以通过 5 个 Hook 函数实现其中断功能。

4.7.1 HOOK_USART1_IRQHandler

定义 UART1 的 Hook 函数，该函数已经被设置为 MPOS 的内部函数用以实现 UART1 接口。 如果用户想重新编写 UART1 函数，这个 Hook 函数可以重新设置为用户的函数。

4.7.2 HOOK_USART2_IRQHandler

定义 UART2 的 Hook 函数， 开发者需要将 Hook 函数可以赋值为用户的 UART2 中断函数。

4.7.3 HOOK_RTC_IRQHandler

定义 RTC 中断的 Hook 函数， 用户可以通过设置这个 Hook 函数来实现它们的 RTC 中断函数。

4.7.4 HOOK_EXTIPORTBCDE_WKIRQHandler

定义外部 IO 中断的 Hook 函数，可以唤醒休眠的 MCU。

4.7.5 HOOK_EXTIPORTBCDE_GNIRQHandler

定义外部普通 IO 中断的 Hook 函数，不可以唤醒休眠的 MCU，在变量 SysStatus.Bits.WkIntBits 中定义的几个中断源：

- Bit0 表示 PB0 触发的中断
- Bit1 表示 PC1 触发的中断
- Bit4 表示 PC5 触发的中断
- Bit5 表示 PC6 触发的中断

- Bit6 表示 PD4 触发的中断
- Bit7 表示 PD6 触发的中断

4.8 唤醒位

如果没有任何任务运行，MCU 将进入深度休眠状态，MPOS 为用户保留一些唤醒位来定义应用程序唤醒事件。 有两个唤醒位，一个是正常的唤醒位，另一个是在轻度休眠时的唤醒位。 如果正常唤醒位置位，MCU 进入深度休眠状态，如果轻唤醒位置位，MCU 进入轻休眠状态。

4.8.1 LPInfo.wakingBit.UserEvent & LPInfo.wakingBit.RFU

- LPInfo.wakingBit 的大小：

  UserEvent 是 1bit，LPInfo.wakingBit.RFU 的大小是 5 位，如果这些位不都是 0，则 MCU 将不会进入休眠状态，直到这些位被清零。

RFU 保留给用户使用，当用户的某种状态或者应用需要让 MCU 保持运行，将某 bit 的 RFU 置高，当用户应用需要让 MCU 休眠，则需要将所有的 RFU 清 0.

4.8.2 LPInfo. lightWakingBit.UserEvent & LPInfo.lightWakingBit.RFU

- LPInfo.lightWakingBit 的大小：

  UserEvent 是 1bit 和 LPInfo.lightWakingBit.RFU 是 9 位，如果这些位设置为非 0，MCU 将进入轻休眠状态以降低电流消耗，直到功能位清零。

RFU 保留给用户使用，当用户的某种状态或者应用需要让 MCU 保持轻休眠运行，将某 bit 的 RFU 置高，当用户应用需要让 MCU 休眠，则需要将所有的 RFU 清 0.

5. MCU 外设驱动的 API

ManThink 为 KL17 提供驱动程序，可用于操作连接到 OMx02 上的传感器，驱动程序包括 Flash，GPIO，UART，后续 ManThink 提供更多的驱动程序。 如果用户需要的驱动程序不包含在这个库里面，用户可以自己编写或者联系 ManThink 来添加驱动程序。

5.1 Flash

MPOS 为开发者预留了两块 Flash 区域存储用户数据：APP 参数和 DataBase 区。

APP 参数预留了 200 字节的参数空间，可以直接调用 APP 参数操作的 API 进行读取和写入（详见 6.1.12 和 6.1.13 ）。

DataBase 区域预留了 1022 字节的参数空间，操作地址方式为偏置地址：2-1023. 使用 mpos_driver.Database_Read 和 mpos_driver.Database_Write 函数实现对该数据区的读写操作。

5.1.1mpos_driver.Database_Read

用于读取 DataBase 区域存储的数数据。

unsigned char mpos_driver.Database_Read(unsigned int Addr,unsigned char *RxBuffer,unsigned char Length)

Addr: Addr 是在固定的起始地址上的一个偏移地址，地址范围 2-1023.

RxBuffer: RxBuffer 是要存储读出的数值的缓冲器

Length : 需要读取的字节数

5.1.2 mpos_driver.Database_Write

用于写数据到 DataBase 区域。

unsigned char mpos_driver.Database_Write (unsigned int Addr,unsigned char *RxBuffer,unsigned char Lenth)

Addr: Addr 是在固定的起始地址上的一个偏移地址，地址范围 2-1023。

RxBuffer: RxBuffer 是存放即将要写入 Flash 的数值的缓冲器

Length：需要写入的字节数

5.2 GPIO

5.2.1 mpos_driver.GPIO_Config

用于配置 GPIO 的工作模式。

void mpos_driver.GPIO_Config (GPIO_TypeDef* gpioport, unsigned char pin, unsigned short gpiocfg)

gpioport：要操作的目标端口，其值是 PORTA 或者 PORTB

pin　　：要操作的目标引脚，其值是 0-7

gpiocfg：GPIO 的模式，具体可参考文件“MT_KL17_GPIO.h”

5.2.2 mpos_driver.GPIO_PinSet

用于输出到 GPIO 引脚信号

void mpos_driver.GPIO_PinSet (GPIO_TypeDef* gpioport, unsigned char pin, unsigned char state)

gpioport：要操作的目标端口，其值是 PORTA 或者 PORTB

pin ：要操作的目标引脚，其值是 0-7

state :引脚输出的电平，0 表示低电平，1 表示高电平

5.2.3 mpos_driver.GPIO_PinGet

用于获取 GPIO 引脚的状态。

unsigned char mpos_driver.GPIO_PinGet (GPIO_TypeDef* gpioport, unsigned char pin)

gpioport：要操作的目标端口，其值是 PORTA 或者 PORTB

pin　　：要操作的目标引脚，其值是 0-7

return value: 读取到 GPIO 的状态，0 表示低电平，1 表示高电平

5.2.4 mpos_driver.GPIO_IT_Config

用于 GPIO 为中断引脚的相关配置。

void mpos_driver.GPIO_PinGet (GPIO_TypeDef* gpioport , unsigned char pin, GPIO_ITConfig_Type gpiocfg)

gpioport：要操作的目标端口，其值是 PORTA 或者 PORTB

pin　　：要操作的目标引脚，其值是 0-7

gpiocfg: 配置 GPIO 口的中断方式，具体可参考文件“MT_KL17_GPIO.h”

5.2.5 mpos_driver.GPIO_IT_Config

禁止使能 GPIO 的相关中断。

void mpos_driver.GPIO_IT_Diable (GPIO_TypeDef* gpioport , unsigned char pin)

gpioport：要操作的目标端口，其值是 PORTA 或者 PORTB

pin　　：要操作的目标引脚，其值是 0-7

5.3 UART1

5.3.1 mpos_driver.UART1_Init

用于初始化 UART1。

void mpos_driver.UART1_Init(uint32_t baudrate)

baudrate : 定义了 UART1 的波特率

举例: mpos_driver.UART1_Init(9600);

5.3.2 mpos_driver.UART1_SendBuffer

用于 UART1 发送一个数组。

unsigned char mpos_driver.UART1_SendBuffer(unsigned char* data,unsigned short len)

data: 定义了需要发送的列表的头指针

len: 定义了发送数组的长度

5.4 UART2

5.4.1 mpos_driver.UART2_Init

用于初始化 UART2。

void mpos_driver.UART2_Init(uint32_t baudrate)

baudrate : 定义了 UART2 的波特率。

举例: mpos_driver.UART2_Init(9600);

5.4.2 mpos_driver.UART2_SendBuffer

用于 UART2 发送一个数组。

unsigned char mpos_driver.UART2_SendBuffer(unsigned char* data,unsigned short len)

data: 定义了需要发送数组的头指针 。

len: 定义了发送数组的长度

5.5 SPI

5.5.1 mpos_driver.Spi_Init

用于初始化 SPI 。

void mpos_driver.Spi_Init (SPI_Type* SPIx)

SPIx：定义具体的 SPI

举例 ： mpos_driver.Spi_Init (SPI1);

5.5.2 mpos_driver. SPI_Cmd

用于使能或者禁能 SPI。

void mpos_driver. SPI_Cmd(SPI_Type* SPIx,FunctionalState NewState)

SPIx： 定义具体的 SPI。

NewState: ENABLE or DISABLE 用于使能和禁能 SPI。

举例: SPI_Cmd(ENABLE);

5.5.3 mpos_driver. SpiInOut

通过 SPI 发送数据或者读取数据

unsigned char mpos_driver.SpiInOut(SPI_Type* SPIx,unsigned char outData)

SPIx：定义具体的 SPI

outData: 需要 SPI 发送的数据

Return value: 在从设备读取到的 SPI 值

举例: unsigned char getdata= mpos_driver.SpiInOut(SPI1,0xFF);

5.6 ADC

5.6.1 mpos_driver.ADC_GetTempValue

通过 AD 对芯片内部的 PN 结的采样，获取芯片内部的温度。

uint8_t mpos_driver.ADC_GetTempValue (void)

Return value: 返回值为实际的温度值+100，单位：℃。

5.7 RTC

5.7.1 mpos_driver.RTC_ConvertSecondsToDatetime

把 UTC 时间的秒数转换为年月日时分秒的格式。

void mpos_driver.RTC_ConvertSecondsToDatetime (uint32_t seconds, T_RTC_DATETIME *datetime)

seconds: UTC 时间，单位:秒

datetime: 年月日的时间格式，具体请参考文件“MyInline.h”

5.7.2 mpos_driver.RTC_ConvertDatetimeToSeconds

把年月日时分秒的时间格式转换为 UTC 时间。

uint32_t mpos_driver.RTC_ConvertDatetimeToSeconds (T_RTC_DATETIME *datetime)

datetime: 年月日的时间格式，具体请参考文件“MyInline.h”

Return value: 转换完成的 UTC 时间

5.7.3 mpos_driver.RTC_SetDatetime

设置 RTC 的运行时间。

void mpos_driver.RTC_SetDatetime (T_RTC_DATETIME *datetime)

datetime: 年月日的时间格式，具体请参考文件“MyInline.h”

5.7.4 mpos_driver.RTC_GetDatetime

获取 RTC 的运行时间。

void mpos_driver.RTC_GetDatetime (T_RTC_DATETIME *datetime)

datetime: 年月日的时间格式，具体请参考文件“MyInline.h”

5.7.5 mpos_driver.RTC_SetSencond

设置 RTC 的运行时间。

void mpos_driver.RTC_Set Sencond (uint32_t sencond)

sencond: UTC 时间，单位：秒

5.7.6 mpos_driver.RTC_GetSencond

获取 RTC 的运行时间。

uint32_t mpos_driver.RTC_Get Sencond (void)

Return value: UTC 时间，单位：秒

6. LoRa/LoRaWAN 的 API

ManThink 通过库向用户提供 LoRa 和 LoRaWAN 协议栈，该协议栈被命名为 LWS，可以支持 LoRaWAN 的 ClassA、ClassB 和 ClassC,LoRaWAN 的应用层可以支持多播，FUOTA，SW 模式（参考不同版本的 SDK）。用户可以通过 API 函数自己定义发送和接收 LoRa 数据包， 该协议栈可以支持掌机模式和中继模式，以适应丰富的场景。用户可以通过修改 LWS 的功能位来启用或禁用某些功能。

LWS 是基于 MPOS 开发 LoRa+LoRaWAN 协议栈，射频的操作通过调用相应的 API 函数实现。射频的数据接收及 LoRaWAN 的相关事件则通过事件的方式通知到开发者。

本章涉及的大量 LoRaWAN 协议中的术语，建议阅读本章内容时请参考 LoRaWAN spec。

6.1 LWS 的系统 API

6.1.1 mpos_lws.LWS_init

用于初始化协议栈，用户应该在应用程序的开始运行此函数。

void mpos_lws.LWS_init (void)

6.1.2 mpos_lws. LoRaWANInit

用于初始化 LoRaWAN 参数，如果需要实现 LoRaWAN 的功能，必须调用该函数。

void mpos_lws.LoRaWANInit (void)

6.1.3 mpos_lws.LWS_SetDR

用于设置发送一个数据包的数据速率

void mpos_lws.LWS_SetDR (uint8_t DR)

DR: DR 的定义详见 LoRaWAN spec，在 LoRaWAN 中的 CN470 标准时，DR=0 对应于 SF12.

举例: mpos_lws.LWS_SetDR(0)

6.1.4 mpos_lws.LWS_SetFDD

用于设置设备是上行和下行是同频的还是异频。不同的 LoRaWAN 标准支持的方式不同，具体请参照 LoRaWAN spec。

void mpos_lws. LWS_SetFDD (unsigned char status)

status: 0: 设置同频，1: 设置异频。

举例: mpos_lws. LWS_SetFDD (0)

6.1.5 mpos_lws.LWS_SetADR

用于设置模块端是否启用速率自适应功能。

void mpos_lws.LWS_SetADR (unsigned char status);

status: status= 0: 不启用 ADR 功能，1: 启用 ADR 功能。

举例: mpos_lws.LWS_SetADR (1)，启用 ADR 功能。

6.1.6 mpos_lws.LWS_SetServerADR

用于设置 Server 端是否启用速率自适应功能。如果启用 Server ADR 的功能，终端设备的速率将由 Server 侧进行调整。

void mpos_lws.LWS_SetServerADR (unsigned char status);

status: status= 0: 启用终端侧 ADR 功能，1: 启用 Server 侧 ADR 功能。

举例: mpos_lws.LWS_SetServerADR (1)，启用 Server ADR 功能。

6.1.7 mpos_lws. LWS_SetLinkCheck

用于设置模块是否启用 LinkCheck 功能。

void mpos_lws. LWS_SetLinkCheck (unsigned char status);

status: status= 0: 不启用 LinkCheck 功能，1: 启用 LinkCheck 功能。

举例：mpos_lws. LWS_SetLinkCheck (1)，启用 LinkCheck 功能。

6.1.8 mpos_lws. paraFWSetAPPEUI

用于把 APPEui 的参数设置到模块的参数中

void mpos_lws. paraFWSetAPPEUI (u1_t * para,u8_t appeui);

para：模块的内部参数变量

Appeui：需要设置的 APPEui 参数

举例：mpos_lws.paraFWSetAPPEUI(paraFwReg.SFwRegister.AppEui,0x400101000B000301);

6.1.9 mpos_lws. paraRDSetFreq

用于把射频的频率等数据设置到模块的参数中

void mpos_lws. paraRDSetFreq (u1_t *para, u4_t freq,u1_t band);

para：模块的内部参数变量

freq：需要设置的频点等参数

band：需要设置的 band

举例：mpos_lws.paraRDSetFreq(paraRdReg.SRdRegister.Freq[0].SFreq,4703000000,0);

6.1.10 mpos_lws. paraRDSetChannelMap

用于设置模块的 channelmap 参数到模块的内部参数

void mpos_lws. paraRDSetChannelMap (u1_t *para, u2_t map);

para：模块的内部参数变量

map：需要设置的参数

举例：mpos_lws. paraRDSetChannelMap (paraRdReg.SRdRegister.channelMap,0x00FF);

6.1.11 mpos_lws. paraFWSetDevKey

用于设置模块 APPKey 到模块的内部参数

void mpos_lws. paraFWSetDevKey (u1_t *para, u1_t *appkey);

para: 模块的内部参数变量

appkey: 需要设置的参数

举例: mpos_lws. paraFWSetDevKey (paraFwReg.SFwRegister.DevKey,DevKey)

6.1.12 mpos_lws.paraAPPGet

把模块的内部 APP 参数读取到变量中

void mpos_lws. paraAPPGet (u1_t *para ,u1_t len);

para: 读取到的参数

len: 需要读取的 APP 参数的长度

举例: mpos_lws. paraAPPGet (u1_t * para,u1_t len)

6.1.13 mpos_lws.paraAPPSave

把固定长度的 APP 参数存储到模块的 Flash 中

void mpos_lws. paraAPPSave (u1_t *para ,u1_t len);

para: 待写入的参数

len: 需要读取的 APP 参数的长度

举例: mpos_lws. paraAPPSave (u1_t * para,u1_t len)

6.1.14 mpos_lws.LWS_SetClassMode

用于设置设备的工作模式

void mpos_lws.LWS_SetClassMode (uint8_t calssMode)

classMode: 工作模式已经写入到了枚举变量，它的值是 ClassA、ClassB 和 ClassC

举例: mpos_lws.LWS_SetClassMode (Class_A);

6.1.15 mpos_lws.JoinReset

用于使设备重新加入到网络

LWERRO_t mpos_lws.JoinReset (LWOP_tmode)

mode: =LWOP_REJOIN

Return value: 返回执行重新入网的操作结果

举例: mpos_lws.JoinReset (LWOP_REJOIN);

6.1.16 mpos_lws.LWS_GetUPseq

获取 LoRaWAN 的上行帧号。

u4_t LWS_GetUPseq();

6.1.17 mpos_lws.LWS_GetUTCTime

获取 UTC 时间

u4_t LWS_GetUTCTime();

6.1.18 mpos_lws.LWS_SetDeviceTimeReq

调用该函数后，协议栈将从 server 侧同步 UTC 时间。

void LWS_SetDeviceTimeReq();

6.2 LWS 的 FW 寄存器

FW 寄存器定义了 LWS 协议栈的固件参数，用户可以通过 API [mpos_lws.ParaFWGet]和[mpos_lws.ParaFWSave]读取和修改其值，我们建议重新运行协议栈以确保正确的参数设置。

6.2.1 FW registers table

Register	Length(octet)	type	Description
Module Type	1	无符号字符型	只读
Hardware version	1	无符号字符型	只读
Firmware version	1	无符号字符型	只读
DevEUI	8	无符号字符型数组	只读 (全球唯一设备标识符)
APPEUI	8	无符号字符型数组	设备应用的标识符
DevAddr	4	无符号字符型数组	节点地址
NetID	4	无符号字符型数组	网络标识符
NwkSKey	16	无符号字符型数组	网络会话秘钥
AppSKey	16	无符号字符型数组	应用层会话秘钥
APPKey	16	无符号字符型数组	应用层秘钥
RxWinDelay1	1	无符号字符型	接收数据包的第一个窗口的延时时间
RxWinDelay2	1	无符号字符型	接收数据包的第二个窗口的延时时间
JoinDelay1	1	无符号字符型	接收入网回复包的第一个窗口延时时间
JoinDelay2	1	无符号字符型	接收入网回复包的第二个窗口延时时间
ReTx	1	无符号字符型	Confirm 上行数据包的重传次数
globalDutyRate	1	无符号字符型	全局占空比
DutyBand[4]	8	4 个 2 字节无符号字符型数组	频段占空比
DevTimeReqDuty	1	无符号字符型	是指发送多少包数据后和 server 自动对一次时间
ShadowBits	1	无符号字符型	该寄存器仅供内部是使用
RFU	19	无符号字符型	保留

6.2.2 Structure of FW registers

```
typedef struct t_CRO_FW
{
   unsigned char MType;
   unsigned char HwVersion;
   unsigned char FmVersion;
   unsigned char DevEui[8];
   unsigned char AppEui[8];
   unsigned char DevAddr[4];
   unsigned char NetID[4];
   unsigned char NwkSKey[16];
   unsigned char APPSKey[16];
   unsigned char DevKey[16];
   unsigned char RxWinDelay1;
   unsigned char RxWinDelay2;
   unsigned char JoinDelay1;
   unsigned char JoinDelay2;
   unsigned char ReTx;
   unsigned char globalDutyRate;
   unsigned char DutyBand[4][2];
   unsigned char DevTimeReqDuty;
   U_FW_SHDOW    shadowBits;
   unsigned char RFU[19];
} PACKED T_CRO_FW;
```

6.2.3 mpos_lws. paraFWGet

把模块的内部 FW 参数读取到变量中

void mpos_lws. paraFWGet (U_CROFW *para);

para: 模块的内部参数

举例: mpos_lws. paraFWGet (&paraFwReg)

6.2.4 mpos_lws. paraFWSave

存储 FW 参数到模块 flash 中

void mpos_lws. paraFWSave (U_CROFW *para);

para: FW 参数

举例: mpos_lws. paraFWSave (&paraFwReg)

6.3 LWS 的 Radio 寄存器

Radio 寄存器定义了 LWS 协议栈的有关射频的参数，用户可以通过应用接口函数[mpos_lws.ParaRDGet] 和 [mpos_lws.ParaRDSave]修改其值，我们建议重启协议栈以保证其参数的正确性。

6.3.1 Table of Radio registers

Register	Length(octet)	type	Description
dn2Feq	4	无符号字符型数组	定义了第二个接收窗口的频率，小端模式，单位：赫兹，例如：433500000Hz 数值： 0x60 0xAF 0xD6 0x19
dlsetting	1	无符号字符型	定义了第二个接收窗口的速率和第一个窗口的 offset.参考 LoRaWAN spec
ChannelMap	2	无符号字符型数组	使用的信道的状态(每一位定义了具体的信道是否使用)
Power	1	无符号字符型	射频的发送功率,单位：dBm 范围：-5~22 实际的发射射频功率=(Power-2)dBm
Freq[n]	4	无符号字符型数组	定义了各个信道的频率，小端模式， 单位：赫兹，例如：433500000Hz 数值： 0x60 0xAF 0xD6 0x19 n=0~16
DrRange[n]	1	无符号字符型数组	在频点 Freq[n]时，模块允许的频率范围 Bit0~bit3 定义了频点为 Freq[n]时最小的 R。 Bit4~Bit7 义了频点为 Freq[n]时最大的 DR n=0~16 具体内容请参考 LoRaWAN 规格书
channelMapcntl	2	无符号字符型数组	channelMap 控制字（仅供内部使用）

6.3.2 Structrue of Radio registers

U_CRORD 的结构类型。

```
typedef struct t_CRO_RD
{
   unsigned char     dn2Freq[4];
   U_DLSETIING dlsetting;
   unsigned char     channelMap[2];
   unsigned char     Power;
   S_FREQDR    Freq[16];
   u2_t aliCHcntl;
}PACKED T_CRO_RD;
typedef union u_CRO_RD
{
   T_CRO_RD SRdRegister;
   unsigned char Bytes[88];
}PACKED U_CRORD;
```

6.3.3 mpos_lws.paraRDGet

把模块的内部 RD 参数读取到变量中

void mpos_lws.paraRDGet (U_CRORD *para);

para: 模块的内部参数

举例: mpos_lws.paraRDGet (&paraRdReg)

6.3.4 mpos_lws.paraRDSave

存储 RD 参数到模块 flash 中

void mpos_lws.paraRDSave (U_CRORD *para);

para: RD 参数

举例: mpos_lws.paraRDSave (&paraRdReg)

6.4 CF 的寄存器

CF 的寄存器定义 LoRaWAN 功能的参数， 用户可以通过修改参数来改变系统的工作模式，可以通过函数[mpos_lws.paraCFGet]获取参数和函数[mpos_lws.paraCFSave]设置参数。

6.4.1 Table of CFregisters

Register	Size	Type	Description
exCNF	2	无符号字符型数组	模式设置字节
			Bit0=1： 窗口 1 使能 Bit1=1：窗口 2 使能 Bit2=1： 多播 Ping 使能 Bit3=1：休眠唤醒使能 OM402: Bit4~Bit7=0：CN470/EU433 Bit4~Bit7=1：LinkWAN 标准使能 OM802: Bit4~Bit7=0：EU868 Bit4~Bit7=1：US902 Bit4~Bit7=2：KR923 Bit4~Bit7=3：AS920 Bit4~Bit7=4：AU915 Bit4~Bit7=5：CN920 Bit8=1：保留 Bit9=1：模块收到上一包数据 RSSI 输出使能 Bit10=1：模块收到上一包数据 SNR 输出使能 Bit11=1：中继功能使能 Bit12=1：DwellTime 功能使能 Bit13=1：LinkWAN 标准低速率使能 Bit14=1：SW 模式时始终处于接收状态 Bit15=1：SW 模式时发送正常前导码 Bitx=0：禁止此项功能 具体请参考 LoRaWAN spec
MulDevAddr	4	无符号字符型数组	多播的 DevAddr
MulNSessionKey	16	无符号字符型数组	多播的 NetworkSessionKey
MulASessionKey	16	无符号字符型数组	多播的 AppSessionKey
BeaconFreq	4	无符号字符型数组	Beacon 的频点

BeaconDR	1	无符号字符型	Beacon 的 datarate
BeaconPeriod	1	无符号字符型	Beacon 的周期=（BeaconPeriod*128） 单位：秒
BeaconAirTime	4	无符号字符型数组	Beacon 在空中的传输时间 例：SF=9 时，144580 ms SF=10 时，305150 ms
pingFreq	4	无符号字符型数组	Pingslot 的频点
pingDR	1	无符号字符型	Pingslot 的 datarate
pingIntvExp	1	无符号字符型	每个 Beacon 周期中 Pingslot 的数量=2^ pingIntvExp
MulPingFreq	4	无符号字符型数组	多播 ping 的频点
MulPingDR	1	无符号字符型	多播 Pingslot 的 datarate
MulPingIntvExp	1	无符号字符型	每个 Beacon 周期中多播 Pingslot 的数量=2^ pingIntvExp
SWSF	1	无符号字符型	SW 模式所用的扩频因子
SWBW	1	无符号字符型	SW 模式所用的带宽 例，7：BW=125kHz 8：BW=250kHz 9：BW=500kHz
SWFreq	4	无符号字符型数组	SW 模式所用的频点
SWPeriod	1	无符号字符型	SW 模式周期=50ms*SWperiod
uprepeat	1	无符号字符型	Unconfirm 数据包的重传次数
JoinCHMap	2	无符号字符型数组	在 LinkWAN 标准下使用哪个频段入网 具体可参考 LinkWAN 标准
DRRange	1	无符号字符型	Bit0~Bit3：定义了 datarate 的下限值 Bit4~Bit7：定义了 datarate 的上限值
RelayWin2	1	无符号字符型	中继模式的接收窗口延时开启时间 单位：秒
RelayFreqMap	2	无符号字符型数组	中继模式可以使用的信道（每位定义了每个信道状态：开启或关闭）
RelayDR	1	无符号字符型	中继模式时所使用的 datarate
RelayPeriod	1	无符号字符型	中继模式下发送前导码的时长 = RelayPeriod*50 单位：ms
RelayDelayTx	1	无符号字符型	RFU
MaxDwellTime	1	无符号字符型	Bit0~Bit3：最大发射功率限制 单位:dBm Bit4：上行的 DwellTime 使能 Bit5：下行的 DwellTime 使能

			Bit6~Bit7: RFU

6.4.2 Structure of CF registers

```
typedef union
{
   u2_t ByteS;
   struct
   {
      u2_t window1Enable :1;
      u2_t window2Enable :1;
      u2_t MulPingEnable :1;
      u2_t SleepWakeEnable:1;
      u2_t Standard:4;
      u2_t Tmode:1;
      u2_t RssiEnable:1;
      u2_t SNREnable:1;
      u2_t RelayEnable:1;
      u2_t DwellTime:1;
      u2_t aliLowDR:1;
      u2_t SWRXON   :1;
      u2_t SWTXF    :1;
   } PACKED Bits;
}  U_EXSTACKCNF;

typedef union
{
   u1_t ByteS;
   struct
   {
      u1_t MaxEIRP :4;
      u1_t UplinkDwellTime :1;
      u1_t DownlinkDwellTime :1;
      u1_t RFU:2;
   } PACKED Bits;
}  U_MAXDWELLTIME;

typedef struct t_CRO_CF
{
      U_EXSTACKCNF      exStackFunction;
      unsigned char     MulDevAddr[4];
      unsigned char     MulNSessionKey[16];
      unsigned char     MulASessionKey[16];
      unsigned char     BeaconFreq[4];
      unsigned char     BeaconDR;
      unsigned char      BeaconPeriod;
```

```
    unsigned char    BeaconAirTime[4];
    unsigned char    pingFreq[4];
    unsigned char    pingDR;
    unsigned char    pingIntvExp;
    unsigned char    MulPingFreq[4];
    unsigned char    MulPingDR;
    unsigned char    MulPingIntvExp;
    unsigned char    SWSF;
    unsigned char    SWBW;
    unsigned char    SWFreq[4];
    unsigned char    SWPeriod;
    unsigned char    uprepeat;
    unsigned char    JoinCHMap[2];
    unsigned char    DRRange;
    unsigned char    RelayWin2;
    unsigned char    RelayFreqMap[2];
    unsigned char     RelayDR;
    unsigned char    RelayPeriod;
    unsigned char    RelayDelayTx;
    U_MAXDWELLTIME    MaxDwellTime;
}PACKED  T_CRO_CF;

typedef union U_CRO_CF
{
    T_CRO_CF  CFRegister;
    unsigned char  Bytes[82];
}PACKED  U_CROCF;
```

6.4.3 mpos_lws.paraCFGet

把模块的内部 CF 参数读取到变量中

void mpos_lws.paraCFGet (U_CROCF *para);

para: 模块的内部参数

举例: mpos_lws. paraCFGet (&paraCFReg)

6.4.4 mpos_lws.paraCFSave

存储 CF 参数到模块 flash 中

void mpos_lws.paraCSave (U_CROCF *para);

para: CF 参数

举例：mpos_lws.paraCFSave (&paraCFReg)

6.4.5 mpos_lws.paraCFmodify

将修改后的 CF 参数装载到协议栈中。

void mpos_lws. paraCFmodify (U_CROFW *para);

para: 模块的内部参数

举例：mpos_lws. paraCFmodify (&paraCFReg)

6.5 发送和接收射频数据

6.5.1 mpos_lws.LW_TxData

按照 LoRaWAN 协议的格式组包发送数据

LWERRO_t mpos_lws. LW_TxData (uint8_t * txBuffer,u1_t lenth,u1_t port,LWOP_t mode)

txBuffer: 用户需要发送数据的首地址

lenth: 需要发送数据的字节数

port: 发送数据的端口号

mode:

LWOP_LTC 表示发送 confirm 数据包。

LWOP_LTU 表示发送 unconfirm 数据包。

LWOP_SWT 表示发送 SW 模式的数据包，SW 模式下，port 必须为 223 或者<200.其他端口为系统保留端口。

Return value: 返回执行发送函数的结果

举例: mpos_lws. LW_TxData ((unsigned char*)(Sensor.Bytes),INFOR_LEN,SENSOR_PORT, LWOP_LTC);

6.6 LWS 的事件

LoRaWAN 协议中会触发大量的事件，通过 HookUserEvent（mt_ev_t ev，u1_t port，u1_t * Buffer，u2_t len）函数通知用户。所以，用户需要编写一个 Hook 函数并链接到 Hook_Event.function 函数，如果发生事件，系统会通知用户,事件类型如下：

6.6.1 事件的列表

事件	描述
MT_EV_TXDONE	射频数据发送完成
MT_EV_JOINED	成功加入到网络
MT_EV_JOIN_FAILED	入网失败
MT_EV_REJOIN_FAILED	重新入网失败
MT_EV_TXOVER_NOPORT	发送完成并且没有下行数据
MT_EV_TXOVER_NACK	发送完成需要一个回复，但是，并没有回复
MT_EV_TXOVER_DNW1	设备在接收窗口 1 收到了下行数据
MT_EV_TXOVER_DNW2	设备在接收窗口 2 收到了下行数据
MT_EV_TXOVER_PING	设备在 ClassB 模式下的一个 PING 槽接收到了一个下行数据
MT_EV_BEACON_TRACKED	设备跟踪到了 Beacon
MT_EV_BEACON_MISSED	设备收不到 Beacon 了
MT_EV_LOST_TSYNC	设备失去了时间同步
MT_EV_LINK_ALIVE	设备和服务器的连接是正常的
MT_EV_LINK_DEAD	设备和服务器失去了连接
MT_EV_MICWRONG	设备收到了一个数据完整性校验错的数据
MT_EV_NTRX	没有发送而直接收到的数据（ClassB 或者 ClassC 下行数据）
MT_EV_SWRXDONE	休眠唤醒模式下收到的数据

6.6.2 Hook_Event.function

这是一个用于处理 LoRaWAN 事件的指针，用户需要写他们自己的处理这些事件的函数并且链接到此指针

Hook_Event.function(mt_ev_t ev ,u1_t port,u1_t * Buffer, u2_t len);

Ev：定义了 LoRaWAN 协议的事件

Port: 如果事件是收到了服务器下行数据包，其含义是数据包的端口

Buffer：如果事件是收到了服务器下行数据包，其含义是数据包的起始地址

len：如果事件是收到了服务器下行数据包，其含义是数据包的长度

7. 快速使用指南

LoRaWAN 是一个复杂的协议栈，可以实现多种功能，但对于大多数应用来说，协议栈非常易于使用。 如果要快速启动，只需保留默认参数，然后使用 mpos_lws.LW_TxData 实现数据发送，并通过 Hook_Event.function 接收服务器的下行数据，然后处理数据，这对于用户来说是快速和容易的。

如果用户不想知道每个参数的细节，可以联系 ManThink 的技术支持来获取默认参数的设置方法，并使用 mpos_lws.LW_TxData 和 Hook_Event.function 来实现他们的 LoRaWAN 应用程序。

8.其它

如需更多支持，请与北京门思科技有限公司联系

联系电话：+ 86-010-56229170

邮箱：info@manthink.cn

地址：北京市亦庄经济技术开发区地盛北街 1 号经开大厦 B 座 904