如何使用 STM32CubeWL 构建 LoRa®应用程序

引言

本应用笔记将指导用户完成基于 STM32WL 系列微控制器构建特定 LoRa®应用程序所需的所有步骤。

LoRa®是一种无线通信网络，旨在以极低的比特率进行远距离通信，并延长电池供电型传感器的寿命。LoRaWAN®定义了通信和安全协议，此协议会确保与 LoRa®网络的互操作性。

STM32CubeWL MCU 软件包中的固件兼容 LoRa Alliance®规范协议 LoRaWAN®，并具有以下主要特性：

- 可直接集成应用程序
- 低功耗 LoRa®解决方案的简易附加组件
- CPU 负载极低
- 无延迟要求
- STM32 存储器占用空间小
- 低功耗定时服务

STM32CubeWL MCU 软件包的固件基于 STM32Cube HAL 驱动程序。

本文提供了有关带 STM32WL55JC 的 NUCLEO-WL55JC 开发板（适于高频段的订购代码为 NUCLEO-WL55JC1，适于低频段的订购代码为 NUCLEO-WL55JC2）和带 STM32WL5M 的 B-WL5M-SUB1 连接扩展板的客户应用程序示例。

为了充分利用本应用笔记中的信息并创建应用程序，用户必须熟悉 STM32 系列微控制器、LoRa®技术，并了解低功耗管理和任务排序等系统服务。

1 概述

STM32CubeWL 在基于 Arm® Cortex®-M 处理器的 STM32WL 系列微控制器上运行。

注意： *Arm 是 Arm Limited（或其子公司）在美国和/或其他地区的注册商标。*

arm

表 1. 缩略语和术语

缩略语	定义
ABP	个性化激活
ADR	自适应数据率
BSP	板级支持包
DC/DC	直流-直流转换器
FHSS	跳频扩频
FSK	频移键控
HAL	硬件抽象层
IoT	物联网
IPCC	处理器间通信控制器
IRQ	中断请求
LBT	先听后说
LoRa	远距离无线电技术
LoRaWAN	LoRa 广域网
LPWAN	低功耗广域网
LR-FHSS	远距离跳频扩频
MAC	介质访问控制
MCPS	MAC 公共部分子层
MIB	MAC 信息库
MLME	MAC 子层管理实体
MSC	消息序列图
OTAA	无线激活
PA	功率放大器
PER	数据包错误率
PRBS	伪随机比特序列
RSSI	接收的信号强度指示
接收	接收
SWD	串行线调试
<target>	STM32WL Nucleo 板（NUCLEO-WL55JC）
Tx	传输

参考文档

[1] LoRaWAN® 1.0.3 链路层规范 - 2018 年 1 月
[2] LoRaWAN® 1.0.3 区域参数规范 - 2018 年 7 月
[3] TS001-1.0.4 LoRaWAN®链路层 1.0.4 规范 - 2020 年 10 月
[4] RP002-1.0.1 LoRaWAN®区域参数规范 - 2020 年 2 月
[5] STM32CubeWL 的 LoRaWAN® AT 指令应用笔记（AN5481）
[6] STM32WL HAL 和底层驱动程序的用户手册说明（UM2642）
[7] IEEE 802.15.4TM 标准 - 2011。低速率无线个人局域网（LR-WPAN）
[8] 涉及 STM32CubeWL 的长数据包应用笔记（AN5687）
[9] STM32CubeWL 上的 SBSFU 集成指南应用笔记（包括 KMS）（AN5544）
[10] 如何使用 STM32CubeWL 保护 LoRaWAN®和 Sigfox™的应用笔记（AN5682）

LoRa 标准

有关 LoRa 和 LoRaWAN 建议的更多详情，请参阅文档[1]、[2]、[3]和[4]。

STM32Cube_FW_WL 固件包

下表列出了使用 NUCLEO-WL55JC1、NUCLEO-WL55JC2 和/或 B-WL5M-SUB1 板的应用程序。

表 2. LoRaWAN 和 SubGHz_Phy 工程列表

工程模块 = 应用程序

模块名称	工程模块	NUCLEO-WL55JC1(1)	NUCLEO-WL55JC2(2)	B-WL5M-SUB1
LoRaWAN	LoRaWAN_AT_Slave	有		
	LoRaWAN_End_Node	有		
SubGhz_Phy	SubGHz_Phy_AT_Slave	有		
	SubGhz_Phy_PingPong	有		
	SubGhz_Phy_LrFhss	有		无
	SubGhz_Phy_Per	有		

1. *高频段*
2. *低频段*

2 STM32CubeWL 概述

STM32CubeWL MCU 软件包的固件包括以下资源（参见图 1）：

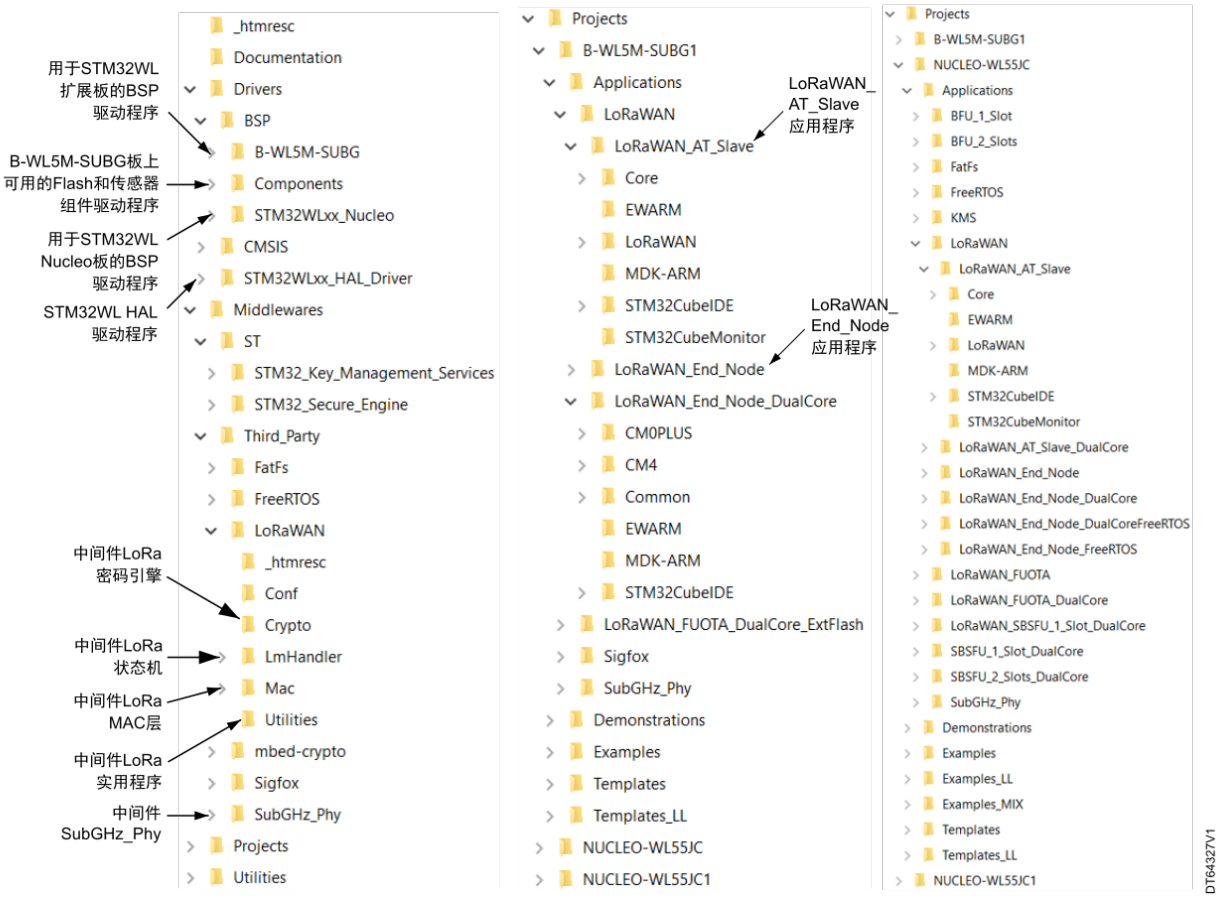
- 板级支持包：
  - STM32WL_Nucleo 驱动程序
  - B-WL5M-SUBG1 驱动程序
- STM32WLxx_HAL_Driver
- 中间件：
  - LoRaWAN 包含：
    - LoRaWAN 层
    - LoRa 实用程序
    - LoRa 软件密码引擎
    - LoRa 状态机
  - 包含无线电和 radio_driver 接口的 SubGHz_Phy 层中间件
- LoRaWAN 应用程序：
  - LoRaWAN_AT_Slave（单核及双核）
  - LoRaWAN_End_Node（单核、双核、使用 FreeRTOS 的单核以及使用 FreeRTOS 的双核）
- SubGHz_Phy 应用程序：
  - SubGHz_Phy_PingPong（单核及双核）
  - SubGHz_Phy_Per（单核）
  - SubGHz_Phy_AT_Slave（单核）
  - SubGHz_Phy_LrFhss（单核）

另外，此应用程序还提供与以下设备的高效系统集成：

- 调度器，用于在后台执行任务并在没有活动时进入低功耗模式
- 定时器服务，为应用提供在 RTC 上运行的虚拟定时器（在停止和待机模式下）

若需更多信息，请参见第 9 节。

图 1. 项目文件结构


用于STM32WL
扩展板的BSP
驱动程序
B-WL5M-SUBG板上
可用的Flash和传感器
组件驱动程序
用于STM32WL
Nucleo板的BSP
驱动程序
STM32WL HAL
驱动程序
中间件LoRa
密码引擎
中间件LoRa
状态机
中间件LoRa
MAC层
中间件LoRa
实用程序
中间件
SubGHz_Phy
_htmresc
Documentation
Drivers
BSP
B-WL5M-SUBG
Components
STM32WLxx_Nucleo
CMSIS
STM32WLxx_HAL_Driver
Middlewares
ST
STM32_Key_Management_Services
STM32_Secure_Engine
Third_Party
FatFs
FreeRTOS
LoRaWAN
_htmresc
Conf
Crypto
LmHandler
Mac
Utilities
mbed-crypto
Sigfox
SubGHz_Phy
Projects
Utilities
Projects
B-WL5M-SUBG1
Applications
LoRaWAN
LoRaWAN_AT_Slave
LoRaWAN_
AT_Slave
应用程序
Core
EWARM
LoRaWAN
MDK-ARM
STM32CubeIDE
STM32CubeMonitor
LoRaWAN_End_Node
LoRaWAN_
End_Node
应用程序
LoRaWAN_End_Node_DualCore
CM0PLUS
CM4
Common
EWARM
MDK-ARM
STM32CubeIDE
LoRaWAN_FUOTA_DualCore_ExtFlash
Sigfox
SubGHz_Phy
Demonstrations
Examples
Templates
Templates_LL
NUCLEO-WL55JC
NUCLEO-WL55JC1
Projects
B-WL5M-SUBG1
NUCLEO-WL55JC
Applications
BFU_1_Slot
BFU_2_Slots
FatFs
FreeRTOS
KMS
LoRaWAN
LoRaWAN_AT_Slave
Core
EWARM
LoRaWAN
MDK-ARM
STM32CubeIDE
STM32CubeMonitor
LoRaWAN_AT_Slave_DualCore
LoRaWAN_End_Node
LoRaWAN_End_Node_DualCore
LoRaWAN_End_Node_DualCoreFreeRTOS
LoRaWAN_End_Node_FreeRTOS
LoRaWAN_FUOTA
LoRaWAN_FUOTA_DualCore
LoRaWAN_SBSFU_1_Slot_DualCore
SBSFU_1_Slot_DualCore
SBSFU_2_Slots_DualCore
SubGHz_Phy
Demonstrations
Examples
Examples_LL
Examples_MIX
Templates
Templates_LL
NUCLEO-WL55JC1
DT64327V1

3 SubGHz HAL 驱动程序

本节主要介绍 SubGHz HAL（未详述其他 HAL 功能，如定时器或 GPIO）。

SubGHz HAL 直接位于 sub-GHz 无线电外设上方（参见图 3）。

SubGHz HAL 驱动程序基于简单的一次性面向指令架构（无完整过程）。因此，未定义 LL 驱动程序。

此 SubGHz HAL 驱动程序由以下主要部分组成：

- 句柄、初始化和配置数据结构
- 初始化 API
- 配置和控制 API
- MSP 和事件回调
- 基于 SUBGHZ_SPI 的总线 I/O 操作（固有服务）

由于 HAL API 主要基于总线服务通过一次性操作发送指令，因此，除了复位/就绪 HAL 状态外，不使用功能状态机。

3.1 SubGHz 资源

初始化无线电时会调用以下 HAL SubGHz API：

- 声明 SUBGHZ_HandleTypeDef 句柄结构。
- 通过调用 HAL_SUBGHZ_Init(&hUserSubghz) API，初始化 sub-GHz 无线电外设。
- 通过实现 HAL_SUBGHZ_MspInit() API，初始化 SubGHz 低级资源：
  - PWR 配置：启用 sub-GHz 无线电外设的唤醒信号。
  - NVIC 配置：
    - 启用 NVIC 无线电 IRQ 中断。
    - 配置 sub-GHz 无线电中断优先级。

在 stm32wlxx_it.c 文件中会调用以下 HAL 无线电中断：

- SUBGHZ_Radio_IRQHandler 中的 HAL_SUBGHZ_IRQHandler。

3.2 SubGHz 数据传输

Set 指令操作在轮询模式下使用 HAL_SUBGHZ_ExecSetCmd(); API 执行。

Get Status 操作在轮询模式下使用 HAL_SUBGHZ_ExecGetCmd(); API 执行。

读/写寄存器访问操作在轮询模式下使用以下 API 执行：

- HAL_SUBGHZ_WriteRegister();
- HAL_SUBGHZ_ReadRegister();
- HAL_SUBGHZ_WriteRegisters();
- HAL_SUBGHZ_ReadRegisters();
- HAL_SUBGHZ_WriteBuffer();
- HAL_SUBGHZ_ReadBuffer();

4 BSP STM32WL 板

本 BSP 驱动程序提供了一系列无线电射频服务管理功能，如射频开关设置和控制、TCXO 设置以及 DC/DC 设置。

注意： *无线电中间件（SubGHz_Phy）通过 radio_board_if.c/h 接口文件连接无线电 BSP。当使用定制用户板时，建议执行一个以下选项：*

- *第一个选项*
  - *复制 BSP/STM32WLxx_Nucleo/目录。*
  - *使用以下信息重命名并更新用户 BSP API：*
    - *用户射频开关配置和控制（如引脚控制或端口编号）*
    - *用户 TCXO 配置*
    - *用户 DC/DC 配置*
  - *将 IDE 项目中的 STM32WLxx_Nucleo BSP 文件替换为用户 BSP 文件。*
- *第二个选项*
  - *禁用 Core/Inc/platform.h 中的 USE_BSP_DRIVER，并在 radio_board_if.c 中直接实现 BSP 功能。*

4.1 频段

STM32WL 系列上可以使用两种 Nucleo 板：

- NUCLEO-WL55JC1：高频段，适于 865 MHz 至 930 MHz 之间的频率
- NUCLEO-WL55JC2：低频段，适于 470 MHz 至 520 MHz 之间的频率

如果用户试图在低频段板上运行在 868 MHz 下编译的固件，预计射频性能会非常差。

固件不会检查其所在的板的频段。

4.2 RF 开关

STM32WL Nucleo 板内置一个射频 3 端口开关（SP3T），使用同一个板即可实现以下模式：

- 高功率输出
- 低功率输出
- 接收

表 3. BSP 无线电开关

函数	说明
int32_t BSP_RADIO_Init(void)	初始化 RF 开关。
int32_t BSP_RADIO_ConfigRFSwitch(BSP_RADIO_Switch_TypeDef Config)	配置 RF 开关。
int32_t BSP_RADIO_DeInit (void)	取消 RF 开关初始化。
int32_t BSP_RADIO_GetTxConfig(void)	返回板配置：高功率、低功率或两者。

RF 状态与开关配置如下表所示。

表 4. RF 状态与开关配置

RF 状态	FE_CTRL1	FE_CTRL2	FE_CTRL3
高功率输出	低	高	高
低功率输出	高	高	高
接收	高	低	高

4.3 RF 唤醒时间

sub-GHz 无线电唤醒时间通过以下 API 恢复。

表 5. BSP 无线电唤醒时间

函数	说明
uint32_t BSP_RADIO_GetWakeUpTime(void)	返回 RF_WAKEUP_TIME 值。

用户必须通过设置指令 RADIO_SET_TCXOMODE 启动 TCXO，超时取决于应用程序。

超时值可在 radio_conf.h 中更新。默认模板值如下：

```
#define RF_WAKEUP_TIME                  1U
```

4.4 TCXO

在用户应用程序中可以安装各种类型的振荡器。在 STM32WL Nucleo 板上，使用温度补偿晶体振荡器（TCXO）实现更高的频率精度。

表 6. BSP 无线电 TCXO

函数	说明
uint32_t BSP_RADIO_IsTCXO (void)	返回 IS_TCXO_SUPPORTED 值。

通过选择 Core/Inc/platform.h 中的 USE_BSP_DRIVER，由 STM32WL Nucleo BSP 定义 TCXO 模式。

如果用户想要更新此值（不兼容 NUCLEO 板），或者如果不存在 BSP，则可在 radio_board_if.h 中更新 TXCO 模式。默认模板值如下：

```
#define IS_TCXO_SUPPORTED               1U
```

4.5 功率调节

根据用户应用程序，采用 LDO 或 SMPS（亦称 DC/DC）进行功率调节。STM32WL Nucleo 板上采用 SMPS。

表 7. BSP 无线电 SMPS

函数	说明
uint32_t BSP_RADIO_IsDCDC (void)	返回 IS_DCDC_SUPPORTED 值。

通过选择 Core/Inc/platform.h 中的 USE_BSP_DRIVER，由 STM32WL Nucleo BSP 定义 DC/DC 模式。

如果用户想要更新此值（不兼容 NUCLEO 板），或者如果不存在 BSP，则可在 radio_board_if.h 中更新 DC/DC 模式。默认模板值定义如下：

```
#define IS_DCDC_SUPPORTED                    1U
```

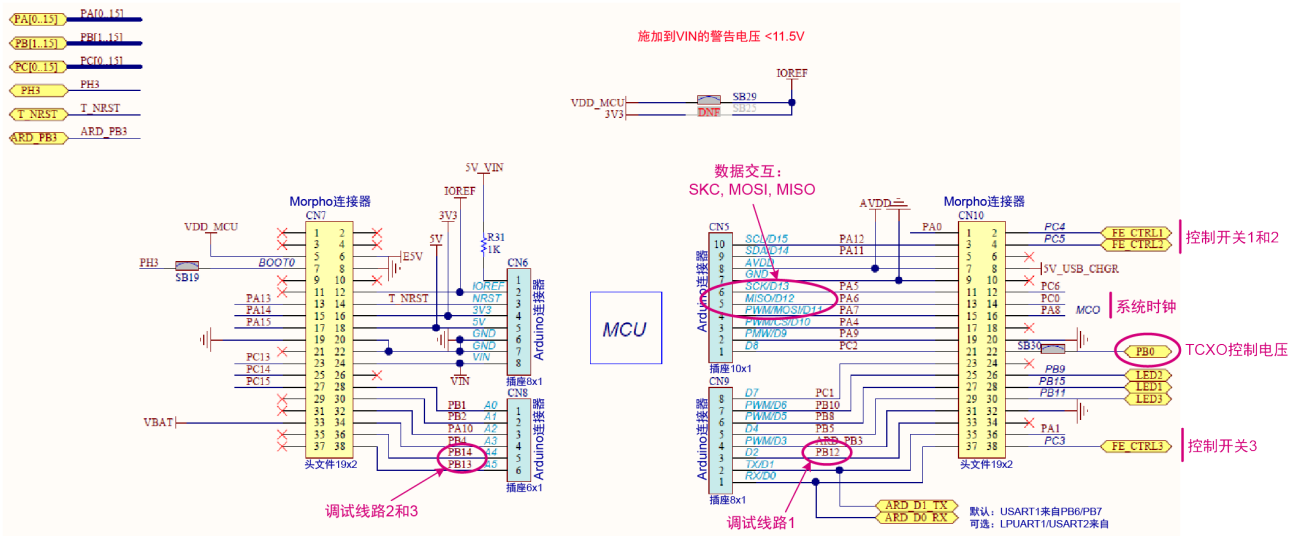
通过设置 IS_DCDC_SUPPORTED = 0，可禁用板上的 SMPS。

4.6 STM32WL Nucleo 板件原理图

下图详细展示了 STM32WL Nucleo 板（MB1389 参考板）原理图，并着重标明了一些有用信号：

- PC4、PC5 和 PC3 上的控制开关
- PB0 上的 TCXO 控制电压引脚
- PB12、PB13 和 PB14 上的调试线路
- PA8 上的系统时钟
- PA5 上的 SCK
- PA6 上的 MISO
- PA7 上的 MOSI

图 2. NUCLEO-WL55JC 原理图



5 BSP B-WL5M-SUBG1 板

BSP 驱动程序提供了一组无线电 RF 开关设置管理功能。另外，它还提供了 B-WL5M-SUBG1 上可用的附加元件，如外部 Flash 存储器、LED 和传感器（环境、运动等）。

5.1 RF 开关

B-WL5M-SUBG1 板在 STM32WL5MOC 中集成了一个 RF 开关。仅有一个 API 来返回板配置。

表 8. BSP 无线电开关

函数	说明
int32_t BSP_RADIO_GetTxConfig(void)	返回板配置：高功率、低功率或两者。

5.2 外部元件

B-WL5M-SUBG1 板内置了多个元件：

- 一个用户按钮
- 3 个 LED
- 温度/气压计传感器
- 加速度计、陀螺仪和磁力计传感器
- 外部 Flash 存储器

有关所定义 API 的更多详情，请参考 BSP 目录中的用户手册文档。

6 LoRaWAN 协议栈说明

STM32CubeWL MCU 软件包的固件包括 STM32WL 资源，如：

- STM32WLxx_Nucleo 驱动程序
- B-WL5M-SUBG1 驱动程序
- STM32WLxx HAL 驱动程序
- LoRaWAN 中间件
- SubGHz 物理层中间件
- LoRaWAN 应用程序示例
- 实用程序

STM32 系列微控制器的 LoRaWAN 协议栈中间件分为几个模块：

表 9. LoRaWAN 协议栈说明

模块	说明	位置
LoRaMAC 层	实现链路层规范	Middlewares\Third_Party\LoRaWAN\Mac
区域层	实现区域参数规范，作为 LoRaMAC 层模块的独立接口	Middlewares\Third_Party\LoRaWAN\Mac\Region
LoRa 密码	使用 SecureEngine 元素实现 AES/CMAC 算法和接口	Middlewares\Third_Party\LoRaWAN\Crypto
LmHandler	实现 LoRaMac Handler 公共接口、认证规范和 FUOTA 包	Middlewares\Third_Party\LoRaWAN\LmHandler
LoRa 实用程序	实现公共实用程序功能	Middlewares\Third_Party\LoRaWAN\Utilities

实现多个符合 LoRa 联盟协议规范的 LoRaWAN 特性：

- 链路层规范方面：
  - 板载 LoRaWAN A、B 和 C 级协议栈
  - 通过 OTAA 或个性化激活（ABP）实现终端设备激活
  - 支持自适应数据率
- 区域参数规范方面：
  - 符合 EU 868 MHz ISM 频段 ETSI
  - 符合 EU 433 MHz ISM 频段 ETSI
  - 符合 US 915 MHz ISM 频段 FCC
  - 韩国政府定义的 KR 920 MHz ISM 频段
  - 俄罗斯法规定义的 RU 864 MHz ISM 频段
  - 中国政府定义的 CN 779 MHz 和 CN470Mhz ISM 频段
  - 亚洲各国政府定义的 AS 923 MHz ISM 频段
  - 澳大利亚政府定义的 AU 915 MHz ISM 频段
  - 印度政府定义的 IN 865 MHz ISM 频段

另外，LoRaWAN 协议栈还集成了：

- 符合下述规范的认证解决方案
- NVM 上下文管理，以防断电时丢失上下文
- 通过待机/睡眠无线电状态集成低功耗模式

6.1 LoRaWAN 规范版本

链路层规范和区域参数规范均由 LoRa 联盟制定。LoRaWAN 协议栈基于 Semtech 协议栈交付项实现了 2 种不同的版本：

- LoRaWAN 链路层 1.0.3 规范 + LoRaWAN 1.0.3 区域参数规范
- LoRaWAN 链路层 1.0.4 规范（TS001-1.0.4）+ LoRaWAN 2-1.0.1 区域参数规范（RP002-1.0.1）

必须选择具有预期 LoRaWAN 服务器配置的改编版协议栈。

6.2 LoRaWAN 调试

根据使用的 LoRaWAN 版本，必须个性化设置每个终端设备，并使用一些唯一标识符以及与首选 LoRaWAN 网络共享的一些网络密钥激活。

终端设备可以采用两种方式通过以下途径激活：

- 当部署或复位终端设备时通过无线激活（OTAA）。
- 个性化激活（ABP），采用这种方式时将一步完成终端设备的个性化设置和激活两个步骤

对于 ABP 方式，需要个性化设置终端设备的以下配置：

- 设备地址（DevAddr）
- 应用程序/加入标识符（JoinEUI）
- 网络会话密钥（NwkSKey）
- 应用程序会话密钥（AppSKey）

对于 OTAA 方式，需要个性化设置终端设备的以下配置：

- 设备唯一标识符（DevEUI）
- 应用程序/加入标识符（JoinEUI）
- 网络根密钥（NwkKey）
- 应用程序根密钥（AppKey）

NwkSKey 加密 Fport = 0 的有效负载，AppSKey 加密 Fport = 0 的有效负载。根密钥用于在加入后派生会话密钥。

警告： *由于 LoRaWAN 集成了多个协议栈版本，因此应用程序密钥和网络密钥不会按照预期由 Link Layer v1.0 定义使用，但是符合 Link Layer v1.1.x 定义。因此，AppS 和 NwkS 密钥源自网络根密钥，而应用程序根密钥仅用于管理附加数据包。*

6.3 LoRaWAN 认证

包括 NUCLEO-WL55JC 板和 STM32CubeWL 固件调制解调器的应用程序已经过 LoRaWAN 测试机构验证，并通过了 EU868、IN865、KR920、AS923 和 US915 频段的认证。

LoRaWAN 认证实现取决于所使用的 LoRaWAN 规范版本：

表 10. LoRaWAN 认证

LoRaWAN 规范版本	LoRaWAN 认证规范
LoRaWAN 链路层规范 1.0.3	面向 AS923MHz ISM 频段设备的 LoRa 联盟终端设备认证要求 v1.1.0
	面向 EU863-870 MHz ISM 频段设备的 LoRa 联盟终端设备认证要求 v1.6.0
	面向印度 865-867 MHz ISM 频段设备的 LoRa 联盟终端设备认证要求 v1.1.0
	面向韩国 920-923MHz ISM 频段设备的 LoRa 联盟终端设备认证要求 v1.2.0
	面向美国和加拿大 902-928 MHz ISM 频段设备的 LoRa 联盟终端设备认证要求 v1.5.0
LoRaWAN 链路层 1.0.4 规范（TS001-1.0.4）	LoRaWAN 1.0.4 全区域终端设备认证要求 v1.6.0
	LoRaWAN 认证协议 1.0.0 规范（TS009-1.0.0）

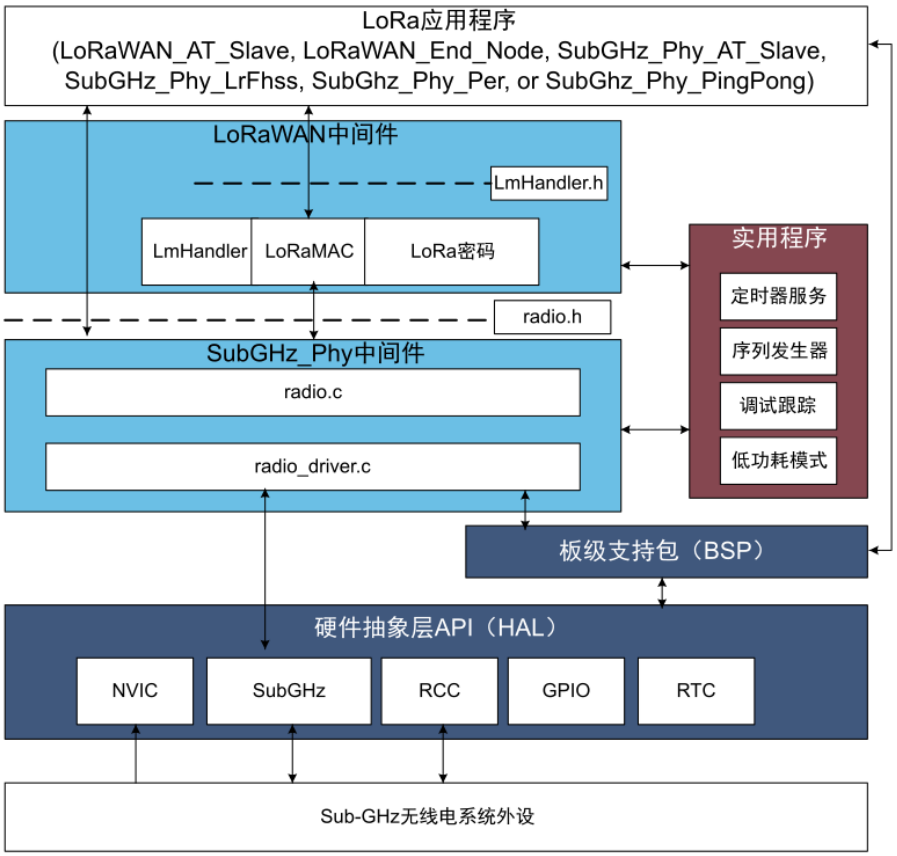
有关认证环境的更多详情，请访问 https://lora-alliance.org/lorawan-certification/。

6.4 架构

6.4.1 静态视图

下图展示了 LoRa 应用程序固件的主要设计。

图 3. 静态 LoRa 架构



HAL 采用 STM32Cube API 驱动应用程序所需的 MCU 硬件。LoRa 中间件中仅包含运行 LoRa 应用程序所必须的特定硬件。

RTC 提供一个集中时间单元，即使在低功耗模式（“停止 2”模式）下，它也可以继续运行。RTC 报警用于在定时器服务器管理的特定时间唤醒系统。

SubGHz_Phy 中间件利用 HAL SubGHz 控制无线电（参见上图）。若需更多信息，请参见第 8 节。

MAC 使用 802.15.4 模型控制 SubGHz_Phy。MAC 连接 SubGHz_Phy 驱动程序，并利用定时器服务器添加或删除定时任务。

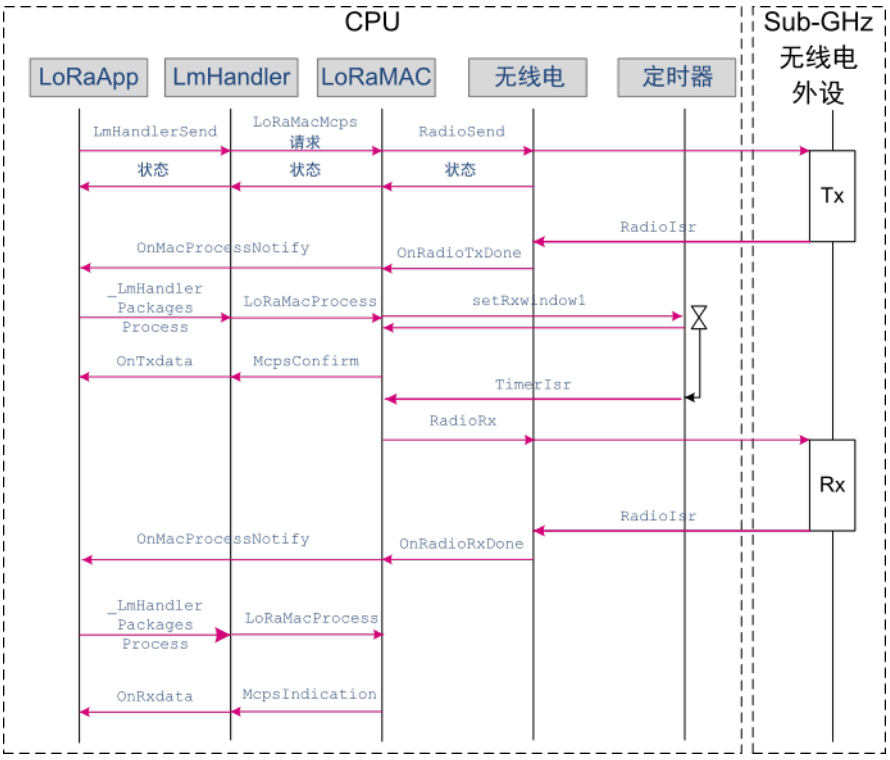
由于控制 Class A 类设备 LoRa 的状态机十分敏感，所以在 MAC 和应用程序之间插入一个中间级软件（LmHandler.c）（请参考上图中的 LoRaMAC 驱动程序）。通过使用一组有限的 API，用户可以轻松在应用程序级实现 A 类状态机。若需更多信息，请参见第 7 节。

基于无限循环构建的应用程序会管理低功耗模式，运行中断处理程序（报警或 GPIO），并在必须完成任何任务时调用 Class A 类设备 LoRa。

6.4.2 动态视图

下图所示的 MSC（消息序列图）描述了发送应用程序数据并从服务器接收应用程序数据的 Class A 类设备。

图 4. Class A 类设备 Tx 和 Rx 处理 MSC



无线电完成应用程序数据发送后，异步 RadioIRQ 会唤醒系统。RadioIsr 此时会在处理程序模式下调用 txDone。

所有 RadioIsr 和 MAC 定时器将调用 LoRaMacProcessNotify 回调，请求应用层更新 LoRaMAC 状态并在需要时进行进一步处理。

例如，在接收结束时，ISR（处理程序）中会调用 rxDone，但不得在 ISR 中进行所有 Rx 数据包处理，包括解密。本案例是一个调用序列示例。如果在 Rx1 窗口内未接收到数据，则启动另一个 Rx2 窗口。

6.4.3 驱动无线电所需的 STM32 外设

Sub-GHz 无线电

sub-GHz 无线电外设通过 stm32wlxx_hal_subghz HAL 进行访问。

sub-GHz 无线电通过 SUBGHZ_Radio_IRQHandler NVIC 发出中断，以通知 TxDone 或 RxDone 事件。更多事件请参阅产品参考手册。

RTC

RTC（实时时钟）日历用作在 32 kHz 外部振荡器提供的所有功率模式下运行的 32 位计数器。默认情况下，RTC 通过编程设置为每秒提供 1024 个节拍（亚秒）。每次硬件初始化时（MCU 首次启动时），都会对 RTC 进行一次编程。RTC 输出受对应的 32 位定时器限制，周期大约为 48 天。

注意： 更改节拍时长时，用户必须保持其低于 1ms。

7 LoRaWAN 中间件说明

7.1 LoRaWAN 中间件初始化

LoRaMAC 层通过 LoRaMacInitialization API 完成初始化，该 API 会初始化 LoRaMAC 层的前导码运行时间以及 MCPS 和 MLME 服务的回调原语（参见下表）。

表 11. LoRaWAN 中间件初始化

函数	说明
LoRaMacStatus_t LoRaMacInitialization (LoRaMacPrimitives_t *primitives, LoRaMacCallback_t *callback, LoRaMacRegion_t region)	初始化 LoRaMAC 层模块 （参见第 7.3 节）

7.2 中间件 MAC 层 API

提供的 API 遵循 IEEE802.15.4-2011 中规定的“原语”定义（请参阅文档[7]）。

通过请求-确认和指示-响应架构实现与 LoRaMAC 的连接。应用层可以请求 LoRaMAC 层通过确认原语进行确认。相反，如果发生任何事件，LoRaMAC 层则通过指示原语通知应用层。

应用层可通过响应原语对指示做出回应。因此，所有确认或指示均使用回调实现。

LoRaMAC 层提供以下服务：

- **MCPS** 服务
  通常，LoRaMAC 层使用 MCPS 服务进行数据发送和数据接收。

表 12. MCPS 服务

函数	说明
LoRaMacStatus_t LoRaMacMcpsRequest (McpsReq_t* mcpsRequest, bool allowDelayedTx)	请求发送 Tx 数据。

- **MLME** 服务
  LoRaMAC 层使用 MLME 服务管理 LoRaWAN 网络。

表 13. MLME 服务

函数	说明
LoRaMacStatus_t LoRaMacMlmeRequest (MlmeReq_t *mlmeRequest)	生成加入请求或请求进行链路检查。

- **MIB** 服务
  MIB 存储重要的运行时信息（如 MIB_NETWORK_ACTIVATION 或 MIB_NET_ID）并保留 LoRaMAC 层的配置（例如 MIB_ADR、MIB_APP_KEY）。

表 14. MIB 服务

函数	说明
LoRaMacStatus_t LoRaMacMibSetRequestConfirm (MibRequestConfirm_t *mibSet)	设置 LoRaMAC 层的属性。
LoRaMacStatus_t LoRaMacMibGetRequestConfirm (MibRequestConfirm_t *mibGet)	获取 LoRaMAC 层的属性。

7.3 中间件 MAC 层回调

应用程序实现的 LoRaMAC 用户事件函数原语（亦称为回调）如下：

表 15. LoRaMacPrimitives_t 结构说明

函数	说明
void (*MacMcpsConfirm) (McpsConfirm_t *McpsConfirm)	响应 McpsRequest
Void (*MacMcpsIndication) (McpsIndication_t* McpsIndication, LoRaMacRxStatus_t* RxStatus)	通知应用程序已接收数据包。
void (*MacMlmeConfirm) (MlmeConfirm_t *MlmeConfirm)	管理 LoRaWAN 网络。
void (*MacMlmeIndication) (MlmeIndication_t* MlmeIndication, LoRaMacRxStatus_t* RxStatus)	通知 MAC 层具有 MAC 响应。

7.4 中间件 MAC 层定时器

表 16. MAC 定时器事件

函数	说明
void OnRxWindow1TimerEvent (void* context)	通过 RxWindow1Delay 在第一个 Rx 窗口定时器事件时执行 正常帧：RxWindowXDelay = ReceiveDelayX – RADIO_WAKEUP_TIME 合并帧：RxWindowXDelay = JoinAcceptDelayX – RADIO_WAKEUP_TIME
void OnRxWindow2TimerEvent (void* context)	通过 RxWindow2Delay 在第二个 Rx 窗口定时器事件时执行
void OnTxDelayedTimerEvent (void* context)	在占空比延迟 Tx 定时器事件时执行。
void OnAckTimeoutTimerEvent (void* context)	通过确认超时定时器在 AckTimeout 定时器事件时执行。 用于数据包重传（仅限 LoRaWAN 版本 v1.0.3）。

函数	说明
void OnRetransmitTimeoutTimerEvent (void* context)	通过确认超时定时器在 AckTimeout 定时器事件时执行。 用于数据包重传（仅限 LoRaWAN 版本 v1.0.4）。

7.5 中间件 LmHandler 应用程序函数

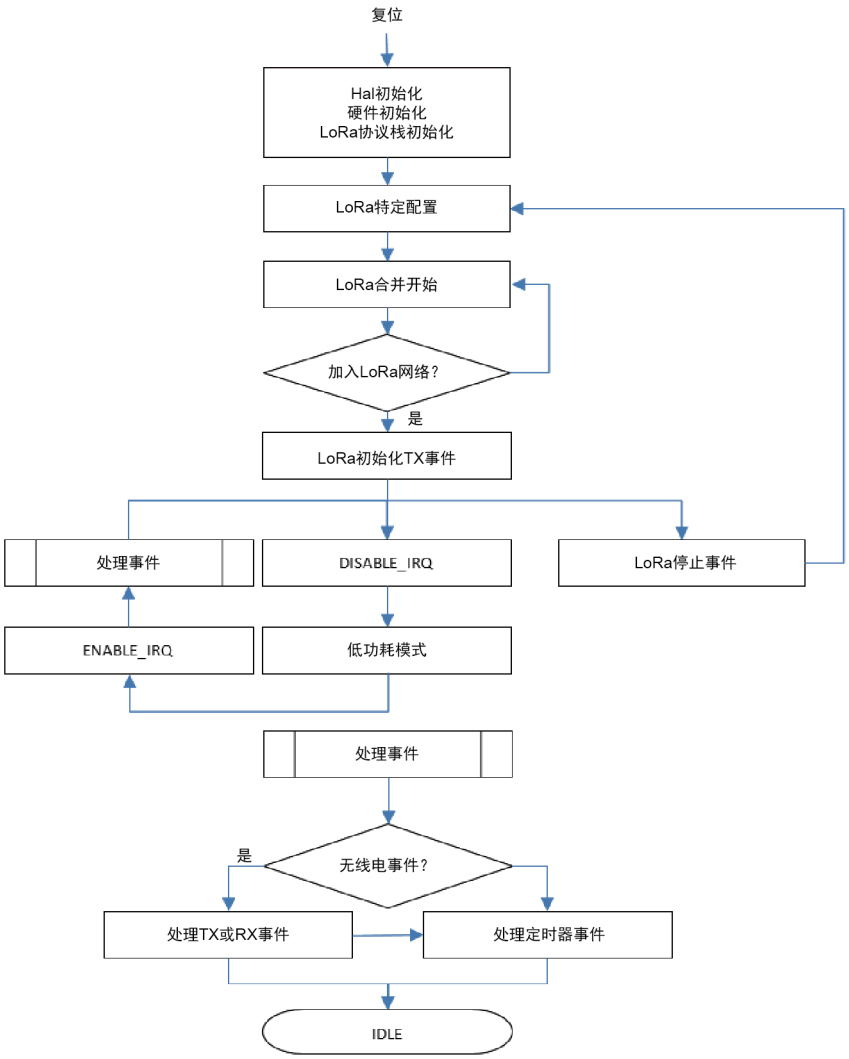
在一个以下模式下，通过 MAC 接口 LoRaMac.h 文件连接到 MAC：

- 标准模式
  提供接口文件（LoRaMAC 驱动程序，参见图 3），以便用户执行启动，而无需担忧 LoRa 状态机。此文件位于 Middlewares\Third_Party\LoRaWAN\LmHandler\LmHandler.c 中，并实现了：
  - 一组用以访问 LoRaMAC 服务的 API
  - 应用层不可见的 LoRa 认证测试案例
- 高级模式
  用户将 MAC 纳入到用户文件中，从而直接访问 MAC 层。

7.5.1 工作模型

针对 LoRaWAN_End_Node 提出的工作模型基于“事件驱动”范型，包括“时间驱动”（参见下图）。LoRa 系统的行为由定时器事件或由无线电事件加上保护转换触发。

图 5. 工作模型



下一节详细说明用于访问 LoRaMAC 服务的 LoRaWAN_End_Node 和 LoRaWAN_AT_Slave API。相应的接口文件位于 Middlewares\Third_Party\LoRaWAN\LmHandler\LmHandler.c 中

用户必须使用这些 API 实现应用程序。

LoRaWAN_End_Node 应用程序示例参见

\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.c.

LoRaWAN_AT_Slave 应用程序示例参见

\Projects\<target>\Applications\LoRaWAN\LoRaWAN_AT_Slave\LoRaWAN\App\lora_app.c.

7.5.2 主要应用程序函数定义

表 17. LmHandler 主要函数

函数	说明
LmHandlerErrorStatus_t LmHandlerInit (LmHandlerCallbacks_t *handlerCallbacks, uint32_t fwVersion)	初始化 LoRa 有限状态机
LmHandlerErrorStatus_t LmHandlerDeInit(void)	取消 LoRa 状态机初始化，停止所有定时器、复位 MAC 参数、关断无线电和删除所有现有回调参考
LmHandlerErrorStatus_t LmHandlerConfigure (LmHandlerParams_t *handlerParams)	配置所有应用参数
bool LmHandlerIsBusy(void)	指示 LoRaMacHandler 是否忙碌。
void LmHandlerProcess(void)	处理 LoRaMac 和无线电事件。
void LmHandlerJoin (ActivationType_t mode, bool forceRejoin)	在 OTAA 或 ABP 模式下请求加入到网络。即使 LoRaWAN 上下文可以恢复，forceRejoin 标志也会强制重新加入。
LmHandlerFlagStatus_t LmHandlerJoinStatus (void)	检查设备是否已加入网络
LmHandlerErrorStatus_t LmHandlerStop (void)	停止 LoRa 过程并在重新加入动作前等待新配置。
LmHandlerErrorStatus_t LmHandlerHalt (void)	通过中断当前过程停止 LoRa 协议栈
LmHandlerErrorStatus LmHandlerRequestClass (DeviceClass_t newClass)	请求 MAC 层更改 LoRaWAN 类别。
LmHandlerErrorStatus_t LmHandlerSend (LmHandlerAppData_t *appData, LmHandlerMsgTypes_t isTxConfirmed, bool allowDelayedTx)	发送上行链路帧。此帧可以是未确认的空帧，也可以是未确认/已确认的有效负载帧。
TimerTime_t LmHandlerGetDutyCycleWaitTime (void)	获取当前的占空比等待时间
LmHandlerErrorStatus_t LmHandlerGetVersion (LmHandlerVersionType_t lmhType, uint32_t *featureVersion)	返回当前的 LoRaWAN 规范版本
LmHandlerErrorStatus_t LmHandlerNvmDataStore (void)	启动 NVM 数据存储过程（更多详情参见第 15 节）。

7.6 应用程序回调

下表中的回调用于 LoRaWAN_End_Node 和 LoRaWAN_AT_Slave 应用程序。

表 18. LmHandlerCallbacks_t callback 结构说明

函数	说明
uint8_t GetBatteryLevel (void)	获取电池电量。
int16_t GetTemperature (void)	以 q7.8 格式获取设备的当前温度（°C）。
void GetUniqueId (uint8_t *id)	获取板的 64 位唯一 ID。
void GetDevAddr (uint32_t *devAddr)	获取板的 32 位唯一 ID（LSB）。

函数	说明
void OnRestoreContextRequest (void *nvm, uint32_t nvm_size)	从 Flash 中恢复 NVM 数据上下文。
void OnStoreContextRequest (void *nvm, uint32_t nvm_size)	将 NVM 数据上下文存储到 Flash 中。
void OnMacProcess (void)	当收到无线电 IRQ 时调用 LmHandler 进程。
void OnNvmDataChange (LmHandlerNvmContextStates_t state)	通知上层已更改 NVM 上下文。
void OnNetworkParametersChange (CommissioningParams_t *params)	通知上层已设置网络参数。
void OnJoinRequest (LmHandlerJoinParams_t *params)	通知上层已加入网络。
void OnTxData (LmHandlerTxParams_t *params)	通知上层已传输帧。
void OnRxData (LmHandlerAppData_t *appData, LmHandlerRxParams_t *params)	通知上层已接收应用帧。
void OnClassChange (DeviceClass_t deviceClass)	确认 LoRaWAN 设备类别变更。
void OnBeaconStatusChange (LmHandlerBeaconParams_t *params)	通知上层已更改信标状态。
void OnBeaconStatusChange (LmHandlerBeaconParams_t *params)	通知上层已更改信标状态。
void OnSysTimeUpdate (void)	通知上层已更新系统时间。
void OnTxPeriodicityChanged(uint32_t periodicity)	调用此函数以更改应用 Tx 帧周期。 当定义 TS001-1.0.4 + TS009 1.0.0 时使用的合规性测试协议回调。
void OnTxFrameCtrlChanged(LmHandlerMsgTypes_t isTxConfirmed)	调用此函数以更改应用 Tx 帧控制。 当定义 TS001-1.0.4 + TS009 1.0.0 时使用的合规性测试协议回调。
void OnPingSlotPeriodicityChanged(uint8_t pingSlotPeriodicity)	调用此函数以更改 ping 周期。 当定义 TS001-1.0.4 + TS009 1.0.0 时使用的合规性测试协议回调。
void OnSystemReset(void)	调用此函数以复位系统。 当定义 TS001-1.0.4 + TS009 1.0.0 时使用的合规性测试协议回调。

7.7 扩展应用程序函数

这些回调用于 LoRaWAN_End-Node 和 LoRaWAN_AT-Slave 应用程序。

表 19. Getter/setter 函数

函数	说明
LmHandlerErrorStatus_t LmHandlerGetCurrentClass(DeviceClass_t *deviceClass)	获取当前的 LoRaWAN 类别。
LmHandlerErrorStatus_t LmHandlerGetDevEUI(uint8_t *devEUI)	获取 LoRaWAN 设备 EUI。
LmHandlerErrorStatus_t LmHandlerSetDevEUI(uint8_t *devEUI)	设置 LoRaWAN 设备 EUI（如果 OTAA）。
LmHandlerErrorStatus_t LmHandlerGetAppEUI(uint8_t *appEUI)	获取 LoRaWAN 应用程序 EUI。
LmHandlerErrorStatus_t LmHandlerSetAppEUI(uint8_t *appEUI)	设置 LoRaWAN 应用程序 EUI。
LmHandlerErrorStatus_t LmHandlerGetNetworkID(uint32_t *networkId)	获取 LoRaWAN 网络 ID。
LmHandlerErrorStatus_t LmHandlerSetNetworkID uint32_t networkId)	设置 LoRaWAN 网络 ID。
LmHandlerErrorStatus_t LmHandlerGetDevAddr(uint32_t *devAddr)	获取 LoRaWAN 设备地址。
LmHandlerErrorStatus_t LmHandlerSetDevAddr(uint32_t devAddr)	设置 LoRaWAN 设备地址（如果 ABP）。
LmHandlerErrorStatus_t LmHandlerGetKey(KeyIdentifier_t keyID, uint8_t *key)	获取 LoRaWAN 密钥。
LmHandlerErrorStatus_t LmHandlerSetKey(KeyIdentifier_t keyID, uint8_t *key)	设置 LoRaWAN 密钥。
LmHandlerErrorStatus_t LmHandlerGetActiveRegion(LoRaMacRegion_ t *region)	获取有效区域。
LmHandlerErrorStatus_t LmHandlerSetActiveRegion(LoRaMacRegion_ t region)	设置有效区域。
LmHandlerErrorStatus_t LmHandlerGetAdrEnable(bool *adrEnable)	获取自适应数据率状态。
LmHandlerErrorStatus_t LmHandlerSetAdrEnable(bool adrEnable)	设置自适应数据率状态。
LmHandlerErrorStatus_t LmHandlerGetTxDatarate(int8_t *txDatarate)	获取当前的 Tx 数据率。
LmHandlerErrorStatus_t LmHandlerSetTxDatarate(int8_t txDatarate)	设置 Tx 数据率（如果自适应数据率禁用）。
LmHandlerErrorStatus_t LmHandlerGetDutyCycleEnable (bool *dutyCycleEnable)	获取当前的 Tx 占空比状态。

函数	说明
LmHandlerErrorStatus_t LmHandlerSetDutyCycleEnable (bool dutyCycleEnable)	设置 Tx 占空比状态。
LmHandlerErrorStatus_t LmHandlerGetRX2Params (RxChannelParams_t *rxParams)	获取当前的 Rx2 数据率和频率配置。
LmHandlerErrorStatus_t LmHandlerSetRX2Params (RxChannelParams_t *rxParams)	设置 Rx2 数据率和频率配置。
LmHandlerErrorStatus_t LmHandlerGetTxPower(int8_t *txPower)	获取当前的 Tx 功率值。
LmHandlerErrorStatus_t LmHandlerSetTxPower(int8_t txPower)	设置 Tx 功率值。
LmHandlerErrorStatus_t LmHandlerGetRx1Delay(uint32_t *rxDelay)	获取当前的 Rx1 延迟（Tx 窗口后）。
LmHandlerErrorStatus_t LmHandlerSetRx1Delay(uint32_t rxDelay)	设置 Rx1 延迟（Tx 窗口后）。
LmHandlerErrorStatus_t LmHandlerGetRx2Delay(uint32_t *rxDelay)	获取当前的 Rx2 延迟（Tx 窗口后）。
LmHandlerErrorStatus_t LmHandlerSetRx2Delay(uint32_t rxDelay)	设置 Rx2 延迟（Tx 窗口后）。
LmHandlerErrorStatus_t LmHandlerGetJoinRx1Delay(uint32_t *rxDelay)	获取当前的加入 Rx1 延迟（Tx 窗口后）。
LmHandlerErrorStatus_t LmHandlerSetJoinRx1Delay(uint32_t rxDelay)	设置加入 Rx1 延迟（Tx 窗口后）。
LmHandlerErrorStatus_t LmHandlerGetJoinRx2Delay(uint32_t *rxDelay)	获取当前的加入 Rx2 延迟（Tx 窗口后）
LmHandlerErrorStatus_t LmHandlerSetJoinRx2Delay(uint32_t rxDelay)	设置加入 Rx2 延迟（Tx 窗口后）。
LmHandlerErrorStatus_t LmHandlerGetPingPeriodicity (uint8_t pingPeriodicity)	获取当前的 Rx Ping 时隙周期 （如果 LORAMAC_CLASSB_ENABLED）。
LmHandlerErrorStatus_t LmHandlerSetPingPeriodicity (uint8_t pingPeriodicity)	设置 Rx Ping 时隙周期 （如果 LORAMAC_CLASSB_ENABLED）。
LmHandlerErrorStatus_t LmHandlerGetBeaconState (BeaconState_t *beaconState)	获取信标状态 （如果 LORAMAC_CLASSB_ENABLED）。
LmHandlerErrorStatus_t LmHandlerDeviceTimeReq(void)	请求网络服务器时间更新。
LmHandlerErrorStatus_t LmHandlerLinkCheckReq(void)	请求链路连接检查。

函数	说明
LmHandlerErrorStatus_t LmHandlerPingSlotReq(uint8_t periodicity)	通知服务器将使用的 ping 时隙周期。

8 SubGHz_Phy 层中间件说明

无线电抽象层由两层组成：

- 高级层（radio.c）
  此层为协议栈中间件提供高级无线电接口。另外，还维护无线电状态、处理中断和管理超时。此层还记录回调并在发生无线电事件时调用这些回调。
- 低级无线电驱动程序
  它是 RF 接口的抽象层。 该层知道寄存器名称和结构，以及详细的序列。 但它不知道硬件接口。

SubGHz_Phy 层中间件包含无线电抽象层，此层直接连接到 BSP 提供的硬件接口上方（请参考第 4 节）。

SubGHz_Phy 中间件目录分为两部分：

- radio.c：包含调用 radio_driver 函数的所有无线电通用回调。这组 API 对于所有无线电均通用且相同。
- radio_driver.c：低级无线电驱动程序

radio_conf.h 包含无线电应用程序配置，如 RF_WAKEUP_TIME、DC/DC 动态设置、XTAL_FREQ。

8.1 中间件无线电驱动程序结构

定义了无线电通用结构（*struct Radio_s Radio {};*），以调用所有回调，详细字段如下表所示：

表 20. Radio_s 结构回调

回调	说明
void Init (RadioEvents_t *events	初始化无线电。
RadioState_t GetStatus (void	返回当前的无线电状态。
void SetModem (RadioModems_t modem)	使用给定调制解调器配置无线电。
void SetChannel (uint32_t freq)	设置通道频率。
bool IsChannelFree (uint32_t freq, uint32_t rxBandwidth, int16_t rssiThresh, uint32_t maxCarrierSenseTime)	检查通道是否在给定时间内处于空闲状态。
uint32_t Random (void)	基于 RSSI 读数生成 32 位随机值。
void SetRxConfig (RadioModems_t modem, uint32_t bandwidth, uint32_t datarate, uint8_t coderate, uint32_t bandwidthAfc, uint16_t preambleLen, uint16_t symbTimeout, bool fixLen, uint8_t payloadLen, bool crcOn, bool freqHopOn, uint8_t hopPeriod, bool iqInverted, bool rxContinuous)	设置接收参数。
void SetTxConfig (RadioModems_t modem, int8_t power, uint32_t fdev, uint32_t bandwidth, uint32_t datarate, uint8_t coderate, uint16_t preambleLen, bool fixLen, bool crcOn, bool freqHopOn, uint8_t hopPeriod, bool iqInverted, uint32_t timeout)	设置发送参数。
bool CheckRfFrequency (uint32_t frequency)	检查硬件是否支持给定 RF 频率。

回调	说明
uint32_t TimeOnAir (RadioModems_t modem, uint32_t bandwidth, uint32_t datarate, uint8_t coderate, uint16_t preambleLen, bool fixLen, uint8_t payloadLen, bool crcOn)	计算给定有效负载的空中数据包时间（ms）。
radio_status_t Send (uint8_t *buffer, uint8_t size)	准备需要发送的数据包并启动无线电传输。
void Sleep (void)	将无线电设置为睡眠模式。
void Standby (void)	将无线电设置为待机模式。
void Rx (uint32_t timeout)	将无线电设置为接收模式保持给定时间。
void StartCad (void)	启动 CAD（通道活动检测）。
void SetTxContinuousWave (uint32_t freq, int8_t power, uint16_t time)	将无线电设置为连续波传输模式。
int16_t Rssi (RadioModems_t modem)	读取当前的 RSSI 值。
void Write (uint16_t addr, uint8_t data)	将无线电寄存器写入到指定地址。
uint8_t Read (uint16_t addr)	读取指定地址的无线电寄存器。
void WriteRegisters (uint16_t addr, uint8_t *buffer, uint8_t size)	写入多个以地址开头的无线电寄存器。
void ReadRegisters (uint16_t addr, uint8_t *buffer, uint8_t size)	读取多个以地址开头的无线电寄存器。
void SetMaxPayloadLength (RadioModems_t modem, uint8_t max))	设置最大有效负载长度。
void SetPublicNetwork (bool enable)	将网络设为公共或私有，并更新同步字节。
uint32_t GetWakeupTime (void)	获取无线电退出睡眠模式所需的时间。
void IrqProcess (void)	处理无线电 IRQ。
void RxBoosted (uint32_t timeout)	将无线电设置为接收模式并具有最大 LNA 增益保持给定时间。
void SetRxDutyCycle (uint32_t rxTime, uint32_t sleepTime)	设置 Rx 占空比管理参数。
void TxPrbs (void)	将发射器设置为连续 PRBS 模式。
void TxCw (int8_t power)	将发射器设置为连续非调制载波模式。
int32_t RadioSetRxGenericConfig (GenericModems_t modem, RxConfigGeneric_t* config, uint32_t rxContinuous, uint32_t symbTimeout)	使用多个配置字段设置接收参数。
int32_t RadioSetTxGenericConfig (GenericModems_t modem, TxConfigGeneric_t* config, int8_t power, uint32_t timeout)	使用多个配置字段设置发送参数。
int32_t TransmitLongPacket (uint16_t payload_size, uint32_t timeout,void (*TxLongPacketGetNextChunkCb) (uint8_t** buffer, uint8_t buffer_size))	开始发送长数据包，数据包可能较短。

回调	说明
int32_t ReceiveLongPacket (uint8_t boosted_mode, uint32_t timeout, void (*RxLongStorePacketChunkCb) (uint8_t* buffer, uint8_t chunk_size))	开始接收长数据包，数据包可能较短。
radio_status_t LrFhssSetCfg (const radio_lr_fhss_cfg_params_t *cfg_params)	配置无线电 LR-FHSS 调制解调器参数。
radio_status_t LrFhssGetTimeOnAirInMs (const radio_lr_fhss_time_on_air_params_t *params, uint32_t *time_on_air_in_ms);	获取 LR-FHSS 数据包的空中时间（ms）。

8.2 无线电 IRQ 中断

可能的 sub-GHz 无线电中断源详见下表。

表 21. 无线电 IRQ 位映射和定义

位	源	说明	数据包类型	操作
0	txDone	数据包发送完成	LoRa 和 GFSK	Tx
1	rxDone	数据包接收完成	LoRa 和 GFSK	接收
2	PreambleDetected	检测到前导码	LoRa 和 GFSK	接收
3	SyncDetected	同步字有效	GFSK	接收
4	HeaderValid	头文件有效	LoRa	接收
5	HeaderErr	头文件错误	LoRa	接收
6	Err	前导码、同步字、地址、CRC 或长度错误	GFSK	接收
	CrcErr	CRC 错误	LoRa	接收
7	CadDone	通道活动检测完成	LoRa	CAD
8	CadDetected	检测到通道活动	LoRa	CAD
9	超时	Rx 或 Tx 超时	LoRa 和 GFSK	Rx 和 Tx
10	-	RFU	-	-
11	-	RFU	-	-
12	-	RFU	-	-
13	-	RFU	-	-
14	LrFhssHop	在远距离 FHSS 中，PA 再次增加后，在每次跳频时置位	LR-FHSS	Tx
15	-	RFU	-	-

有关详细信息，请参见产品参考手册。

9 实用程序说明

实用程序位于\Utilities 目录中。

主要 API 如下。有关辅助 API 和附加信息，请参考与驱动程序相关的头文件。

9.1 序列发生器

序列发生器提供一个稳健且简单的框架，以便在后台执行任务，并在没有更多活动时进入低功耗模式。序列发生器实现了一种防止竞态条件的机制。

此外，序列发生器还提供事件特性，允许任何函数等待事件（其中特定事件由中断设置），并且可以在任何实施“运行到完成”命令的应用程序中轻松节省 MIPS 和功耗。

位于工程子文件夹中的 utilitties_def .h 文件用于配置任务和事件 ID。已列出的任务和事件 ID 不可删除。

序列发生器并不是操作系统。任何任务均运行到完成，除非任务通过调用 UTIL_SEQ_WaitEvt 自行挂起，否则无法像 RTOS 在 RTOS 节拍上那样切换到另一个任务。此外，还使用一个单存储器栈。序列发生器是以任务和事件位图标志为中心的高级“while 循环”。

序列发生器具有以下特性：

- 封装的高级 while 循环系统
- 支持多达 32 个任务和 32 个事件
- 任务注册和执行
- 等待事件和设置事件
- 任务优先级设置
- 竞态条件安全进入低功耗

要使用序列发生器，应用程序必须执行以下操作：

- 通过定义 UTIL_SEQ_CONF_TASK_NBR 值来设置支持的最大函数数量。
- 通过 UTIL_SEQ_RegTask()注册调度器要支持的函数。
- 通过调用 UTIL_SEQ_Run()运行后台 while 循环来启动调度器。
- 在需要执行某个函数时调用 UTIL_SEQ_SetTask()。

序列发生器实用程序位于 Utilities\sequencer\stm32_seq.c 中。

表 22. 序列发生器 API

函数	说明
void UTIL_SEQ_Idle(void)	在没有可执行的代码时调用（临界区 - PRIMASK）。
void UTIL_SEQ_Run(UTIL_SEQ_bm_t mask_bm)	请求调度器执行挂起的函数并在 mask_bm 中启用。
void UTIL_SEQ_RegTask(UTIL_SEQ_bm_t task_id_bm, uint32_t flags, void (*task) (void))	注册与调度器中的信号（task_id_bm）关联的函数（任务）。task_id_bm 必须有一位置位。
void UTIL_SEQ_SetTask(UTIL_SEQ_bm_t taskId_bm, uint32_t task_Prio)	请求执行与 task_id_bm 关联的函数。调度器只在函数执行完毕时评估 task_prio。 如果任一时刻有多个挂起的函数，将执行优先级最高（0）的函数。
void UTIL_SEQ_WaitEvt(UTIL_SEQ_bm_t EvtId_bm);	等待需要设置的特定事件。
void UTIL_SEQ_SetEvt(UTIL_SEQ_bm_t EvtId_bm);	使用 UTIL_SEQ_WaitEvt()设置等待事件。

下图比较了标准 while 循环实现与序列发生器 while 循环实现。

表 23. While 循环标准与序列发生器实现

标准方式	序列发生器方式

While(1)
{
  if(flag1)
  {
  flag1=0;
  Fct1();
  }
  if(flag2)
  {
  flag2=0;
  Fct2();
  }
  /*Flags are checked in critical section to avoid race conditions*/ /*Note: in the critical section, NVIC records Interrupt source and system will wake up if asleep */
__disable_irq();
  if (!(flag1 || flag2))
  {
  /*Enter LowPower if nothing else to do*/
LPM_EnterLowPower( );
  }
  __enable_irq();
  /*Irq executed here*/
}
 	
Void some_Irq(void) /*handler context*/
{
  flag2=1; /*will execute Fct2*/
}
	/*Flag1 and Flag2 are bitmasks*/
	UTIL_SEQ_RegTask(flag1, Fct1());
	UTIL_SEQ_RegTask(flag2, Fct2());
	
	While(1)
	{
	  UTIL_SEQ_Run();
	}
	
	void UTIL_SEQ_Idle(void)
	{
	  LPM_EnterLowPower( );
	}
	
	Void some_Irq(void) /*handler context*/
	{
	 UTIL_SEQ_SetTask(flag2); /*will execute Fct2*/
	}

9.2 定时器服务

定时器服务器使用户可以请求执行定时任务。由于硬件定时器基于 RTC，因此，即使在低功耗模式下，仍始终计时。

定时器服务器为用户和协议栈提供可靠的时钟。用户可以请求应用程序所需数量的定时器。

定时器服务器位于 Utilities\timer\stm32_timer.c 中。

表 24. 定时器服务器 API

函数	说明
UTIL_TIMER_Status_t UTIL_TIMER_Init(void)	初始化定时器服务。
UTIL_TIMER_Status_t UTIL_TIMER_Create (UTIL_TIMER_Object_t *TimerObject, uint32_t PeriodValue, UTIL_TIMER_Mode_t Mode, void (*Callback) (void*), void *Argument)	创建定时器对象并在定时器结束时关联回调函数。
UTIL_TIMER_Status_t UTIL_TIMER_SetPeriod(UTIL_TIMER_Object_t *TimerObject, uint32_t NewPeriodValue)	更新周期并启动具有超时值的定时器（ms）。
UTIL_TIMER_Status_t UTIL_TIMER_Start (UTIL_TIMER_Object_t *TimerObject)	启动定时器对象并添加到定时器事件列表中。
UTIL_TIMER_Status_t UTIL_TIMER_Stop (UTIL_TIMER_Object_t *TimerObject)	停止定时器对象并从定时器事件列表中删除。

9.3 低功耗函数

低功耗实用程序集中了固件所实现的独立模块的低功耗要求，并在系统进入空闲模式时管理低功耗进入。例如，当使用 DMA 将数据打印到控制台时，如果系统处于睡眠模式，则不得进入低功耗模式，因为在停止模式下会关闭 DMA 时钟

下表中的 API 用于管理核心 MCU 的低功耗模式。低功耗实用程序位于 Utilities\lpm\tiny_lpm\stm32_lpm.c 中。

表 25. 低功耗 API

函数	说明
void UTIL_LPM_EnterLowPower(void)	进入选定的低功耗模式。通过系统的空闲状态调用
void UTIL_LPM_SetStopMode(UTIL_LPM_bm_t lpm_id_bm, UTIL_LPM_State_t state);	设置停止模式。id 定义所请求的进程模式：UTIL_LPM_ENABLE 或 UTIL_LPM_DISABLE。(1)
void UTIL_LPM_SetOffMode(UTIL_LPM_bm_t lpm_id_bm, UTIL_LPM_State_t state);	设置停止模式。id 定义所请求的进程模式：UTIL_LPM_ENABLE 或 UTIL_LPM_DISABLE。
UTIL_LPM_Mode_t UTIL_LPM_GetMode(void)	返回当前选定的低功耗模式。

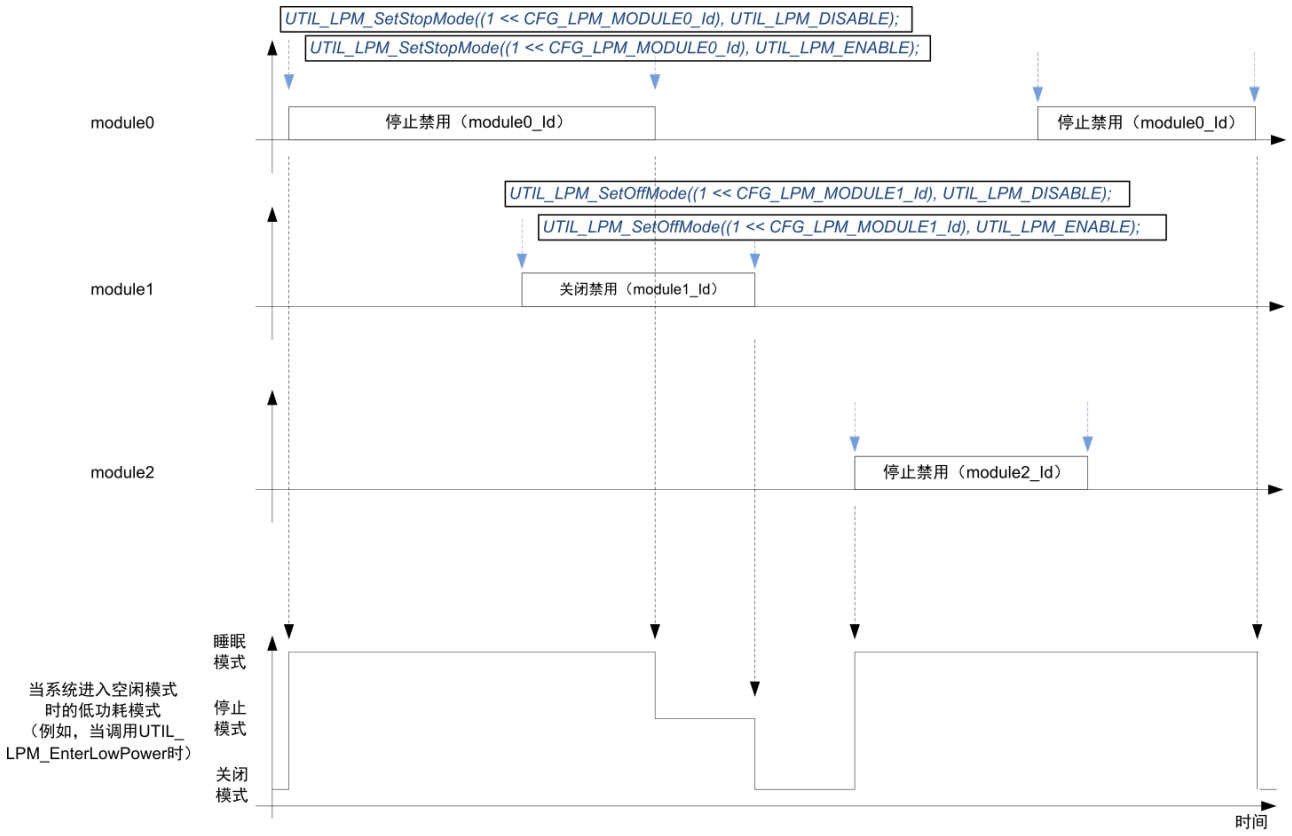
1. *偏移值的位图定义参见 utilities_def.h。*

默认低功耗模式为关闭模式，这可以是待机或关断模式（定义参见表 26 中的 void PWR_EnterOffMode (void)）：

- 如果至少一个固件模块禁用停止模式并进入低功耗模式，则选择睡眠模式。
- 如果任何固件模块均未禁用停止模式，至少一个固件模块禁用关闭模式并进入低功耗模式。则选择停止模式。
- 如果任何固件模块均未禁用停止模式，任何固件模块未禁用关闭模式并进入低功耗模式。则选择关闭模式。

图 6 描述了当系统进入低功耗模式时三个不同固件模块在各自的低功耗要求和低功耗模式设置下的行为。

图 6. 低功耗模式动态视图示例



必须实施底层 API 来定义系统进入/退出低功耗模式必须执行的操作，这些函数在工程子文件夹的 stm32_lpm_if.c 中实现。

表 26. 低级 API

函数	说明
void PWR_EnterSleepMode (void)	在进入睡眠模式前调用的 API
void PWR_ExitSleepMode (void)	在退出睡眠模式时调用的 API
void PWR_EnterStopMode (void)	在进入停止模式前调用的 API
void PWR_ExitStopMode (void)	在退出停止模式时调用的 API
void PWR_EnterOffMode (void)	在进入关闭模式前调用的 API
void PWR_ExitOffMode(void)	在退出关闭模式时调用的 API

在睡眠模式下，内核时钟停止。每个外设时钟可以选择是否设置门控。所有外设保持供电。

在“停止 2”模式下，大多数外设时钟停止。大多数外设电源关闭。外设的某些寄存器不再保留，必须在退出“停止 2”模式时重新初始化。存储器以及内核寄存器仍保留。

在待机模式下，LSI 和 LSE 以外的所有时钟均关闭。所有外设电源关闭（BOR、备份寄存器、GPIO 拉取和 RTC 除外），无保留（仅保留附加 SRAM2），必须在退出待机模式时重新初始化。内核寄存器不再保留，必须在退出待机模式时重新初始化。

注意： *sub-GHz 无线电电源独立于系统其他部分。参见产品参考手册了解更多信息。*

9.4 系统时间

MCU 复位时会参考 MCU 时间。系统时间可以记录 UNIX®纪元时间。

下表中的 API 用于管理内核 MCU 的系统时间。系统实用程序位于 Utilities\misc\stm32_systime.c 中。

表 27. 系统时间函数

函数	说明
void SysTimeSet (SysTime_t sysTime)	根据以秒和亚秒为单位的 UNIX 纪元输入，与 MCU 时间的差值会存储在备份寄存器中（即使在待机模式下也会保留）。(1)
SysTime_t SysTimeGet (void)	获取当前的系统时间。(1)
uint32_t SysTimeMkTime (const struct tm* localtime)	将本地时间转换为 UNIX 纪元时间。(2)
void SysTimeLocalTime (const uint32_t timestamp, struct tm *localtime)	将 UNIX 纪元时间转换为本地时间。(2)

1. *系统时间参考是从 1970 年 1 月 1 日开始的 UNIX 纪元。*
2. *还提供了 SysTimeMkTime 和 SysTimeLocalTime，用以将纪元转换为 time .h 接口指定的 tm 结构。*

为了将 UNIX 时间转换为本地时间，必须添加时区并且必须删除闰秒。必须删除 2018 年中的 18 闰秒。巴黎的夏季时间与格林威治时间相差两个小时。假定已设置时间，即可利用以下代码在终端上打印本地时间。

```
{
SysTime_t UnixEpoch = SysTimeGet();
struct tm localtime;
UnixEpoch.Seconds-=18; /*removing leap seconds*/
UnixEpoch.Seconds+=3600*2; /*adding 2 hours*/
SysTimeLocalTime(UnixEpoch.Seconds, & localtime);
PRINTF (“it's %02dh%02dm%02ds on %02d/%02d/%04d\n\r”,
localtime.tm_hour, localtime.tm_min, localtime.tm_sec,
localtime.tm_mday, localtime.tm_mon+1, localtime.tm_year + 1900);
}
```

9.5 跟踪

跟踪模块可以利用 DMA 通过 COM 端口打印数据。下表中的 API 用于管理跟踪函数。

跟踪实用程序位于 Utilities\trace\adv_trace\stm32_adv_trace.c 中。

表 28. 跟踪函数

函数	说明
UTIL_ADV_TRACE_Status_t UTIL_ADV_TRACE_Init(void)	初始化应用程序时必须调用 TraceInit。在 DMA 模式下初始化 com 或 vcom 硬件，并在 DMA 传输完成时注册待处理的回调。
UTIL_ADV_TRACE_Status_t UTIL_ADV_TRACE_COND_FSend(uint32_t VerboseLevel, uint32_t Region, uint32_t TimeStampState, const char *strFormat, ...)	将字符串格式转换为缓冲区，并将其提交到循环队列以进行打印。
UTIL_ADV_TRACE_Status_t UTIL_ADV_TRACE_COND_Send(uint32_t VerboseLevel, uint32_t Region, uint32_t TimeStampState, const uint8_t *pdata, uint16_t length)	发布数据长度 = len，并将其提交到循环队列以进行打印。
UTIL_ADV_TRACE_Status_t UTIL_ADV_TRACE_COND_ZCSend_Allocation(ui nt32_t VerboseLevel, uint32_t Region, uint32_t TimeStampState, uint16_t length,uint8_t **pData, uint16_t *FifoSize, uint16_t *WritePos)	将用户格式化的数据直接写入到 FIFO（Z-Cpy）。

跟踪函数的状态值定义参见以下 UTIL_ADV_TRACE_Status_t 结构。

```
typedef enum {
  UTIL_ADV_TRACE_OK                 = 0,    /*Operation terminated successfully*/
  UTIL_ADV_TRACE_INVALID_PARAM      = -1,   /*Invalid Parameter*/
  UTIL_ADV_TRACE_HW_ERROR           = -2,   /*Hardware Error*/
  UTIL_ADV_TRACE_MEM_ERROR          = -3,   /*Memory Allocation Error*/
  UTIL_ADV_TRACE_UNKNOWN_ERROR      = -4,   /*Unknown Error*/
  UTIL_ADV_TRACE_GIVEUP             = -5,   /*!< trace give up*/
  UTIL_ADV_TRACE_REGIONMASKED       = -6    /*!< trace region masked*/
} UTIL_ADV_TRACE_Status_t;
```

在以下模式下，可以使用 UTIL_ADV_TRACE_COND_FSend (..)函数：

- 在无实时限制的轮询模式下：例如，在应用程序初始化期间

```
#define APP_PPRINTF(...) do{ } while( UTIL_ADV_TRACE_OK \
!= UTIL_ADV_TRACE_COND_FSend(VLEVEL_ALWAYS, T_REG_OFF, TS_OFF, __VA_ARGS__) )
/* Polling Mode */
```

- 在实时模式下：当循环队列中没有空间时，不增加字符串，也不通过 com 端口打印

```
#define APP_LOG(TS,VL,...)do{
{UTIL_ADV_TRACE_COND_FSend(VL, T_REG_OFF, TS, __VA_ARGS__);} }while(0);)
```

其中：

– VI 是跟踪的详细级别。

– TS 允许在跟踪中添加时间戳（TS_ON 或 TS_OFF）。

应用程序详细级别通过以下函数在 Core\Inc\sys_conf.h 中设置：

```
#define VERBOSE_LEVEL <VLEVEL>
```

其中 VLEVEL 可以是 VLEVEL_OFF、VLEVEL_L、VLEVEL_M 或 VLEVEL_H。

只有 VLEVEL ≥ 详细级别时，才显示 UTIL_ADV_TRACE_COND_FSend (..)。

如果已饱和，则可使用以下函数在 Core\Inc\utilities_conf.h 中增加缓冲区长度：

```
#define UTIL_ADV_TRACE_TMP_BUF_SIZE 256U
```

该实用程序提供了钩子函数，可以实现在 DMA 活动期间禁止系统进入停止或低功耗模式。

- ```
  void UTIL_ADV_TRACE_PreSendHook (void)
  {UTIL_LPM_SetStopMode((1 << CFG_LPM_UART_TX_Id) , UTIL_LPM_DISABLE);}
  ```

- ```
  void UTIL_ADV_TRACE_PostSendHook (void)
  {UTIL_LPM_SetStopMode((1 << CFG_LPM_UART_TX_Id) , UTIL_LPM_ENABLE);}
  ```

10 LoRaWAN_End_Node 应用程序

此应用程序检测电池电量和 MCU 温度。这些值会利用 868 MHz 的 Class A 类 LoRa 无线电定期发送到 LoRa 网络。

适用性参见表 2。

要启动 LoRaWAN_End_Node 项目，访问\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node，并选择最喜欢的工具链文件夹（在 IDE 环境中）。从适当的目标板中选择工程。

请注重下述配置，以设置应用程序。

10.1 LoRaWAN 用户代码段说明

定义了四个主要函数，作为实现和使用\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.c 中的 LoRaWAN 协议栈的示例。

这些函数包含用户代码段下方的示例代码，这些代码可以重写，以处理应用层的具体特性。

表 29. LoRaWAN 用户函数

函数	说明
void LoRaWAN_Init (void)	按照图 5 所示初始化 LoRaWAN 应用程序。
static void SendTxData(void)	生成通用内容有效负载以及通过临时有效负载生成的缓冲区调用 LmHandlerSend(...)的 LoRaWAN Tx 进程示例。
static void OnTxData(LmHandlerTxParams_t *params)	当 LoRaWAN 应用程序成功传输一帧时的回调实现示例。 • **params** 参数包含最后一次 Tx 的状态
static void OnRxData(LmHandlerAppData_t *appData, LmHandlerRxParams_t *params)	当 LoRaWAN 应用程序接收一帧时的回调实现示例。 • **appData** 参数包含最后一次 Rx 时接收的数据 • **params** 参数包含最后一次 Rx 的状态

10.2 设备配置

10.2.1 激活方法和密钥

可以采用两种方式来激活网络上的设备：通过 OTAA 或通过 ABP。

必须调整应用程序中的全局变量“ActivationType”，以在所选模式下激活设备。

```
static ActivationType_t ActivationType = LORAWAN_DEFAULT_ACTIVATION_TYPE;
```

位于\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.c

且

```
#define LORAWAN_DEFAULT_ACTIVATION_TYPE ACTIVATION_TYPE_OTAA
```

位于\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h

其中 ActivationType_t enum 定义如下：

```
typedef enum eActivationType {
        ACTIVATION_TYPE_NONE = 0, /* None */
        ACTIVATION_TYPE_ABP = 1, /* Activation by personalization */
        ACTIVATION_TYPE_OTAA = 2, /* Over the Air Activation */
```

\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\se-identity.h 文件包含有助于设备激活的调试数据。

10.2.2 LoRa 设备类别的激活

默认定义 Class A 类设备。要更改类别激活（Class A 类设备、Class B 类设备或 Class C 类设备），用户必须：

- 设置代码

```
#define LORAWAN_DEFAULT_CLASS CLASS_B;
```

in
\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h

- 设置代码

```
#define LORAMAC_CLASSB_ENABLED 1
```

in
\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lorawan_co nf.h

10.2.3 Tx 触发

使用以下位置的 EventType 全局变量，可以通过两种方式生成上行链路动作
\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.c:

- 通过定时器
- 通过外部事件

使用代码

```
static TxEventType_t EventType = TX_ON_TIMER;
```

其中 TxEventType_t enum 定义如下：

```
typedef enum TxEventType_e {
        TX_ON_TIMER = 0, /* App data transmission issue based on timer */
        TX_ON_EVENT = 1, /* App data transmission by external event */
}TxEventType_t;
```

TX_ON_EVENT 特性采用按键 1 作为 LoRaWAN_End_Node 应用程序中的事件。

10.2.4 占空比

将用于应用程序的占空比值（ms）在
\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 中使用以下代码（示例）定义：

```
#define APP_TX_DUTYCYCLE                            10000 /* 10s duty cycle */
```

10.2.5 应用程序端口

将用于应用程序的应用程度端口在
\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 中使用以下代码（示例）定义：

```
#define LORAWAN_APP_PORT
```

注意： *LORAWAN_APP_PORT 不得使用为认证预留的端口 224。*

10.2.6 确认/未确认模式

将用于应用程序的确认/未确认模式在

\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 中使用以下代码定义：

```
#define LORAWAN_DEFAULT_CONFIRMED_MSG_STATE    LORAMAC_HANDLER_UNCONFIRMED_MSG
```

10.2.7 数据缓存大小

发送到网络的缓存大小在

\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 中使用以下代码定义：

```
#define LORAWAN_APP_DATA_BUFFER_MAX_SIZE                242
```

此值定义设备可以发送的最大有效负载大小。但是，上行链路/下行链路有效负载大小还取决于所使用的数据率。有关更多详细信息，请参考区域参数规范的最大有效负载大小表。

10.2.8 自适应数据率（ADR）

ADR 在

\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 中使用以下代码启用：

```
#define LORAWAN_ADR_STATE                               LORAMAC_HANDLER_ADR_ON
```

当 ADR 禁用时，则在

\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 中使用以下代码设置默认数据率：

```
#define LORAWAN_DEFAULT_DATA_RATE                       DR_0
```

其中预期值必须在 0-15 范围内（取决于所选的区域配置）。有关更多详细信息，请参考区域参数规范的 TX 数据率表。

10.2.9 Tx 功率

加入时和第一个上行链路处使用的默认 Tx 功率（直至收到 LinkADRReq）可以在 \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 中使用以下代码进行修改：

```
#define LORAWAN_DEFAULT_TX_POWER                        TX_POWER_0
```

其中预期值必须在 0-15 范围内（取决于所选的区域配置）。有关更多详细信息，请参考区域参数规范的 TX 功率表。

10.2.10 Ping 周期

如果设备能够切换为 Class B 类设备，则必须在

\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 中使用以下代码启用默认的 Rx Ping 时隙周期。

```
#define LORAWAN_DEFAULT_PING_SLOT_PERIODICITY 4
```

其中预期值必须在 0-7 范围内。

产生的周期时间通过以下设置进行定义：

```
period = 2^LORAWAN_DEFAULT_PING_SLOT_PERIODICITY
```

10.2.11 LoRa 频段选择

区域及其相应的频段选择在 \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\Target\lorawan_conf.h 中使用以下代码定义：

```
#define REGION_AS923
#define REGION_AU915
#define REGION_CN470
#define REGION_CN779
#define REGION_EU433
#define REGION_EU868
#define REGION_KR920
#define REGION_IN865
#define REGION_US915
#define REGION_RU864
```

注意： *编译相同应用程序时可以启用多个区域。*

必须根据区域在\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h 中使用以下代码（以欧洲为例）定义默认有效区域：

```
#define ACTIVE_REGION LORAMAC_REGION_EU868
```

当 REGION_AS923 启用时，还可以使用以下代码选择相关的通道计划：

```
#define REGION_AS923_DEFAULT_CHANNEL_PLAN                CHANNEL_PLAN_GROUP_AS923_1
```

可能的选择：

- CHANNEL_PLAN_GROUP_AS923_1 (Default configuration.Freq offset = 0.0 MHz / Freq range = 915-928 MHz)
- CHANNEL_PLAN_GROUP_AS923_2（频率偏移 = -1.80 MHz /频率范围 = 915-928 MHz）
- CHANNEL_PLAN_GROUP_AS923_3（频率偏移 = -6.60 MHz /频率范围 = 915-928 MHz）
- CHANNEL_PLAN_GROUP_AS923_4（频率偏移 = -5.90 MHz /频率范围 = 917-920 MHz）
- CHANNEL_PLAN_GROUP_AS923_1_JP（频率偏移 = 0.0 MHz /频率范围 = 920.6-923.4 MHz）

10.2.12 上下文管理

上下文管理在

\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lorawan_conf.h 中使用以下代码定义：

```
#define CONTEXT_MANAGEMENT_ENABLED 1
```

有关此特性的更多详情，请参见第 15 节。

10.2.13 LoRaWAN 版本

可以选择 LoRaWAN 版本，以使用预期特性或者符合网络服务器定义的所需版本。此版本在\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lorawan_conf.h 中使用以下代码定义：

```
#define LORAMAC_SPECIFICATION_VERSION 0x01000400
```

可能的值：0x01000300 或 0x01000400.若需更多信息，请参见第 6.1 节。

10.2.14 LoRaWAN 包

LoRaWAN 包提供一些管理时钟同步、多播以及碎片化特性的附加功能：

- 应用层时钟同步（包 ID：1，默认端口：202）
- 远程多播设置（包 ID：2，默认端口：200）
- 碎片数据块传输（包 ID：3，默认端口：201）
- 固件管理协议（包 ID：4，默认端口：203）

这些包在\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\Lo RaWAN\App\lorawan_conf.h 中使用以下代码启用：

```
#define LORAWAN_DATA_DISTRIB_MGT 0
```

另外，还需要涵盖 Middlewares\Third_Party\LoRaWAN\LmHan dler\Packages 中可用的所有源文件

10.2.15 LoRaWAN 包版本

可以选择 LoRaWAN 包版本，以使用网络服务器定义的所需版本。此版本在 \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lorawan_conf.h 中使用以下代码定义：

```
#define LORAWAN_PACKAGES_VERSION 1
```

可能的值：

- v1.0.0 包涵盖：
  - 应用层时钟同步 v1.0.0
  - 远程多播设置 v1.0.0
  - 碎片数据块传输 v1.0.0
- v2.0.0 包涵盖：
  - 应用层时钟同步 v2.0.0
  - 远程多播设置 v2.0.0
  - 碎片数据块传输 v2.0.0
  - 固件管理协议 v1.0.0

10.2.16 调试开关

调试模式在

\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h 中使用以下代码启用：

```
#define DEBUGGER_ENABLED            1 /* 开=1，关=0 */
```

即使 MCU 进入低功耗模式时，调试模式也会启用 SWD 引脚。

注意： *为了启用真正的低功耗，#define DEBUGGER_ENABLED 必须复位。*

一些额外的定义可以激活某些内部 RF 信号的监视（探针），以进行调试。

```
#define DEBUG_SUBGHZSPI_MONITORING_ENABLED    0
#define DEBUG_RF_NRESET_ENABLED_ENABLED       0
#define DEBUG_RF_HSE32RDY_ENABLED_ENABLED     0
#define DEBUG_RF_SMPSRDY_ENABLED              0
#define DEBUG_RF_LDORDY_ENABLED               0
#define DEBUG_RF_DTB1_ENABLED                 0
#define DEBUG_RF_BUSY_ENABLED                 0
```

10.2.17 低功耗开关

当系统空闲时，则会进入低功耗“停止 2”模式。

这种进入“停止 2”模式的行为可在\Projects\<target>

\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h 中使用以下代码禁用：

```
#define LOW_POWER_DISABLE       0 /* 低功耗启用 = 0，低功耗禁用 = 1 */
```

其中：

- 低功耗启用 = 0 表示 MCU 进入“停止 2”模式
  “停止 2”是指低功耗稳压器和 VDD12I 可中断数字核心域电源均关闭的停止模式。与低功耗“停止 1”模式下相比，激活的外设更少，以降低功耗。如需更详细信息，请参考[6]。
- 低功耗禁用 = 1 表示 MCU 仅进入睡眠模式。

10.2.18 跟踪级别

跟踪模式在
\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h 中使用以下代码启用：

```
#define APP_LOG_ENABLED                 1
```

跟踪级别在
\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h 中使用以下代码选择：

```
#define VERBOSE_LEVEL VLEVEL_M
```

建议使用以下跟踪级别：

- VLEVEL_OFF：禁用全部跟踪
- VLEVEL_L：启用功能跟踪
- VLEVEL_M：启用调试跟踪
- VLEVEL_H：启用全部跟踪

10.3 LoRaWAN_End_Node 应用程序的设备配置总结

表 30. LoRaWAN_End_Node 应用程序配置的开关选项

项目模块	详细信息	开关选项	定义	位置
LoRa 协议栈	唯一设备标识	LORAWAN_DEVICE_EUI	终端设备 IEEE 扩展唯一标识符（EUI）	se-identity.h
	加入 EUI	LORAWAN_JOIN_EUI	应用程序或加入服务器 IEEE EUI（仅在 OTAA 中使用）	
	终端设备地址	LORAWAN_DEVICE_ADDRESS	网络上的终端设备地址（仅在 ABP 中使用，由 OTAA 中的网络服务器生成）	
	应用程序根密钥	LORAWAN_APP_KEY	用于派生应用程序会话密钥的应用程序根密钥	
	网络根密钥	LORAWAN_NWK_KEY	用于在加入 OTAA 时派生会话密钥的网络根密钥	
	应用程序会话密钥	LORAWAN_APP_S_KEY	用于通过应用程序服务器加密/解密有效负载的应用程序会话密钥（仅在 ABP 中使用，由 OTAA 中的根密钥生成）	
	网络会话密钥	LORAWAN_NWK_S_KEY	用于通过网络服务器加密/解密有效负载的网络会话密钥（仅在 ABP 中使用，由 OTAA 中的根密钥生成）	
	支持的区域	REGION_EU868	设备支持的区域	lorawan_conf.h
		REGION_EU433		
		REGION_US915		
		REGION_AS923		
		REGION_AU915		
		REGION_CN470		
		REGION_CN779		
		REGION_IN865		
		REGION_RU864		
		REGION_KR920		
	AS923 通道计划	REGION_AS923_DEFAULT_CHANNEL_PLAN	用于区域 AS923 的通道计划	
	限制通道	HYBRID_ENABLED	限制 AU915、CN470 和 US915 区域的默认可用通道数量。	
	读取密钥	KEY_EXTRACTABLE	定义存储器中的密钥的读取访问。	
	可选类别	LORAMAC_CLASSB_ENABLED	Class B 类终端设备	
	上下文管理	CONTEXT_MANAGEMENT_ENABLED	存储和恢复 LoRaWAN 协议栈上下文。	
应用	Tx 触发	EventType = TX_ON_TIMER	Tx 触发方式	lora_app.c
	类别选择	LORAWAN_DEFAULT_CLASS	设置设备类别。	lora_app.h
	占空比	APP_TX_DUTYCYCLE	两次发送 Tx 之间的时间	

项目 模块	详细信息	开关选项	定义	位置
应用	应用程序端口	LORAWAN_USER_APP_PORT	Tx 数据帧使用的 LoRa 端口	lora_app.h
	确认模式	LORAWAN_DEFAULT_CONFIRMED_MSG_STATE	确认模式选择	
	自适应数据率	LORAWAN_ADR_STATE	ADR 选择	
	默认数据率	LORAWAN_DEFAULT_DATA_RATE	ADR 禁用时的数据率	
	最大数据缓存大小	LORAWAN_APP_DATA_BUFFER_MAX_SIZE	缓存大小定义	
	Tx 功率	LORAWAN_DEFAULT_TX_POWER	启动时的默认 Tx 输出功率	
	Ping 周期	LORAWAN_DEFAULT_PING_SLOT_PERIODICITY	Rx ping 时隙周期	
	网络加入激活	LORAWAN_DEFAULT_ACTIVATION_TYPE	激活程序默认选择	
	初始区域	ACTIVE_REGION	设备启动时使用的区域	
	准备加入	LORAWAN_FORCE_REJOIN_AT_BOOT	即使 NVM 上下文已恢复仍强制加入	
	调试	DEBUGGER_ENABLED	启用 SWD 引脚。	sys_conf.h
	低功耗	LOW_POWER_DISABLE	禁用低功耗模式。	
	跟踪启用	APP_LOG_ENABLED	启用跟踪模式。	
	跟踪级别	VERBOSE_LEVEL	设置跟踪级别。	

11 SubGhz_Phy_PingPong 应用程序

此应用程序展示了两个 PingPong 设备（一个称为 Ping，另一个称为 Pong）之间的简单 Rx/Tx RF 链路。

默认情况下，每个 PingPong 设备作为主设备启动，传输‘Ping’消息并等待回复。启动时，每个 PingPong 设备都会有两个 LED 闪烁。当板同步时（一个板的 Tx 窗口与另一个板的 Rx 窗口一致），Ping 设备（接收‘Ping’消息的板）会使绿色 LED 闪烁，Pong 设备（接收‘Pong’消息的板）会使红色 LED 闪烁。收到‘Ping’消息的第一个 PingPong 设备会成为从设备并以‘Pong’消息回复主设备。

适用性参见表 2。

要启动 SubGhz_Phy_PingPong 工程，用户必须进入

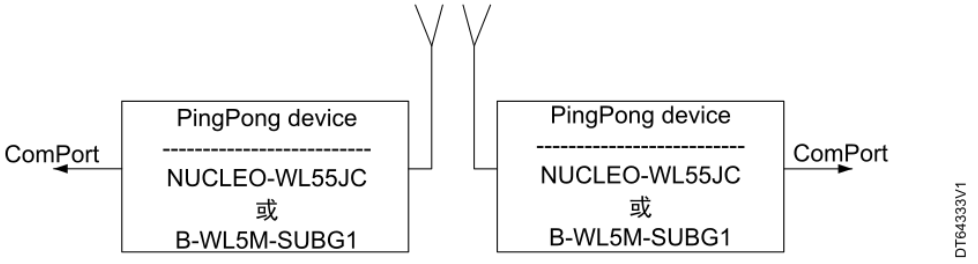
\Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong，并遵循与 LoRaWAN_End_Node 项目启动首选工具链时相同的步骤。从适当的目标板中选择工程。

11.1 SubGhz_Phy_PingPong 硬件/软件环境设置

要设置 STM32WL 板（NUCLEO-WL55JC 或 B-WL5M-SUBG1），使用 USB Type-A 转 Mini-B 线缆将此板连接到计算机以及集成 ST-LINK 连接器（CN1）（对于 NUCLEO-WL55JC）或连接至调试连接器（CN3）上的外部 ST-LINK-V3（对于 B-WL5M-SUBG1）。

编译并将工程加载到两个板中，然后打开各板串行端口上的首选超级终端，如下图所示：

图 7. **SubGhz_Phy_PingPong application setup**



11.2 设备配置

11.2.1 调制定义

此应用程序建议使用两种调制：LoRa 或 FSK。要配置一种调制，用户必须在 \Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h 中更新下表所示的这些定义：

表 31. **SubGHz_Phy_PingPong 调制配置**

LoRa		FSK	
#define USE_MODEM_LORA	1	#define USE_MODEM_LORA	0
#define USE_MODEM_FSK	0	#define USE_MODEM_FSK	1

11.2.2 有效负载长度

每个 Tx 有效负载通过字符串 PING 或 PONG 与后面的 0 序列定义。此有效负载的长度在 \Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h 中使用以下代码定义：

```
#define PAYLOAD_LEN        64
```

典型有效负载大小通常在 51 至 242 之间。

必须使用以下代码设置等于或小于 \Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.c 中定义的最大缓存大小的值：

```
#define MAX_APP_BUFFER_SIZE        255
```

11.2.3 区域和频率

将用于此应用程序的频率值在 \Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h 中使用以下代码（示例）定义：

```
#define RF_FREQUENCY      868000000 /* Hz */
```

11.2.4 带宽、扩频因子和数据率

根据所选的调制，可以在 \Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h 中定义带宽、扩频因子和数据率，如下表所示：

表 32. SubGHz_Phy_PingPong 带宽、SF 和 DR 配置

LoRa		FSK	
#define LORA_BANDWIDTH	0	#define FSK_BANDWIDTH	50000
#define LORA_SPREADING_FACTOR	7	#define FSK_DATARATE	50000

LORA_BANDWIDTH 的预期值必须在 0-2 范围内，对应等效频率：[0:125 kHz, 1:250 kHz, 2:500 kHz].

LORA_SPREADING_FACTOR 的预期值必须在 7-12 范围内。扩频因子会影响发送一帧所需的时间以及传输功率。

FSK_BANDWIDTH 的预期值必须在 4800-500000 范围内（单位：Hz）。

相当于比特率值的 FSK_DATARATE 预期值通常定义为 50 kbps。

11.2.5 前导码长度

所有传输/接收的数据包均包含前导码（通常具有 8 个字符）、头文件、有效负载（在第 10.2.2 节中定义大小）和 CRC 字段。

前导码字段大小可在 \Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h 中通过下表所示的这些定义进行更新：

表 33. SubGHz_Phy_PingPong 前导码配置

LoRa		FSK	
#define LORA_PREAMBLE_LENGTH	8	#define FSK_PREAMBLE_LENGTH	5
/* default LoRa preamble size */		/* default FSK preamble size */	

11.3 SubGhz_Phy_PingPong 应用程序的设备配置总结

表 34. SubGhz_Phy_PingPong 应用程序配置的开关选项

项目模块 = 应用程序

详细信息	开关选项	定义	位置
Rx/Tx configuration	RX_TIMEOUT_VALUE	Rx 窗口超时	subghz_phy_app.c
	TX_TIMEOUT_VALUE	Tx 窗口超时	
	MAX_APP_BUFFER_SIZE	最大数据缓存大小	
	RX_TIME_MARGIN	Rx 结束与 Tx 开始之间的时间	
	FSK_AFC_BANDWIDTH	AFC 带宽（Hz）	
	LED_PERIOD_MS	LED 闪烁周期	
调制配置	USE_MODEM_LORA	选择的 LoRa 调制解调器	subghz_phy_app.h
	USE_MODEM_FSK	选择的 FSK 调制解调器	
LoRa/FSK 通用参数	RF_FREQUENCY	收发器使用的频率	
	TX_OUTPUT_POWER	RF 输出功率：-17 至 22 dBm	
	PAYLOAD_LEN	数据缓存大小	
LoRa 特定参数	LORA_BANDWIDTH	带宽： • 0：125 kHz • 1：250 kHz • 2：500 kHz • 3：保留	
	LORA_SPREADING_FACTOR	扩频因子：SF7 至 SF12	
	LORA_CODINGRATE	编码率： • 1：4/5 • 2：4/6 • 3：4/7 • 4：4/8	
	LORA_PREAMBLE_LENGTH	Tx/Rx 前导码的长度	
	LORA_SYMBOL_TIMEOUT	超时前检查的符号数	
	LORA_FIX_LENGTH_PAYLOAD_ON	固定/动态长度有效负载选项	
	LORA_IQ_INVERSION_ON	IQ 反转选项	
FSK 特定参数	FSK_FDEV	频率偏移（Hz）	
	FSK_DATARATE	数据速率（bit/s）	
	FSK_BANDWIDTH	带宽（Hz）	
	FSK_PREAMBLE_LENGTH	Tx/Rx 前导码的长度	
	FSK_FIX_LENGTH_PAYLOAD_ON	固定/动态长度有效负载选项	
调试	DEBUGGER_ENABLED	启用 SWD 引脚。	sys_conf.h
低功耗	LOW_POWER_DISABLE	禁用低功耗模式。	
跟踪启用	APP_LOG_ENABLED	启用跟踪模式。	
跟踪级别	VERBOSE_LEVEL	设置跟踪级别。	

12 SubGHz_Phy_LrFhss 应用程序

SubGHz_Phy_LrFhss 应用程序是一项基于从设备到网关仅产生 Tx 的快速跳频扩频（FHSS）调制测试。传输从 LR-FHSS 开始，有效负载为 64 字节。此应用程序需要集成了 LR-FHSS 调制的网关解决方案。

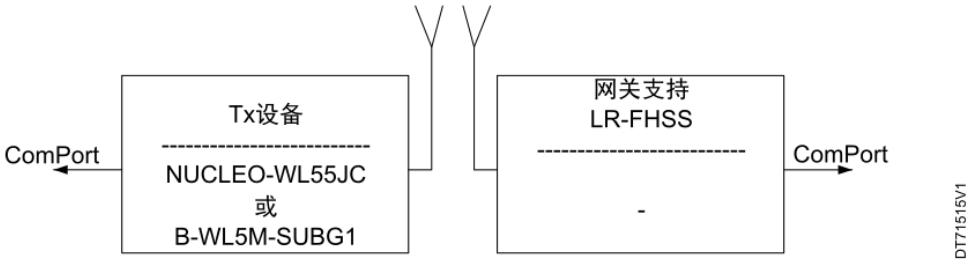
适用性参见表 2。

要启动 SubGhz_Phy_LrFhss 项目，用户必须进入 \Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_LrFhss，并遵循与 LoRaWAN_End_Node 工程启动首选工具链时相同的步骤。

12.1 SubGhz_Phy_LrFhss 硬件/软件环境设置

要设置 STM32WL Nucleo 板（NUCLEO-WL55JC），使用 USB Type-A 转 Mini-B 线缆将此板连接至计算机和 ST-LINK 连接器（CN1），如下图所示：

图 8. SubGhz_Phy_LrFhss 应用程序设置



12.2 SubGhz_Phy_LrFhss 应用程序的设备配置总结

表 35. SubGhz_Phy_LrFhss 应用程序配置的开关选项

项目模块 = 应用程序

详细信息	开关选项	定义	位置
Rx/Tx configuration	TX_TIMEOUT_VALUE	Tx 窗口超时	subghz_phy_app.c
	MAX_APP_BUFFER_SIZE	最大数据缓存大小	
通用参数	PAYLOAD_LEN	数据缓存大小	subghz_phy_app.h
调试	DEBUGGER_ENABLED	启用 SWD 引脚。	sys_conf.h
低功耗	LOW_POWER_DISABLE	禁用低功耗模式。	
跟踪启用	APP_LOG_ENABLED	启用跟踪模式。	
跟踪级别	VERBOSE_LEVEL	设置跟踪级别。	

13 SubGhz_Phy_Per 应用程序

SubGHz_Phy_Per 应用程序是一项利用 IBM®白化在一个 Tx 设备与一个 Rx 设备之间进行的数据包错误率测试的示例。

Tx 设备

更新/SubGHz_Phy/App/subghz_phy_app.c 中的#define TEST_MODE to RADIO_TX。编译并加载。

数据包内容包括 前导码 | 同步 | 有效负载长度 | 有效负载 | crc，其中：

- crc 基于有效负载长度和有效负载计算得出。
- 白化通过有效负载长度 | 有效负载 | crc 计算得出。

传输默认以 GFSK 50 Kbit/s 开始，有效负载为 64 字节。这些参数可在 /SubGHz_Phy/App/subghz_phy_app.h 中使用与第 11.2 节所述相同的定义进行修改。

此外，还可以通过用户按键在运行时修改应用程序：

- 用户按键 1 可将数据包长度增加 16 字节。
- 用户按键 2 可将数据包长度增加 1 字节。
- 用户按键 3 将数据包有效负载模式从斜坡（0x00、0x01..）切换到 prbs9。

当无线电处于 Tx 模式时，蓝色 LED 点亮。

Rx 设备

更新/SubGHz_Phy/App/subghz_phy_app.c 中的#define TEST_MODE to RADIO_RX。编译并加载。当 Rx 正常时，绿色 LED 点亮。当 Rx 异常时，红色 LED 点亮。

适用性参见表 2。

要启动 SubGhz_Phy_Per 工程，用户必须进入

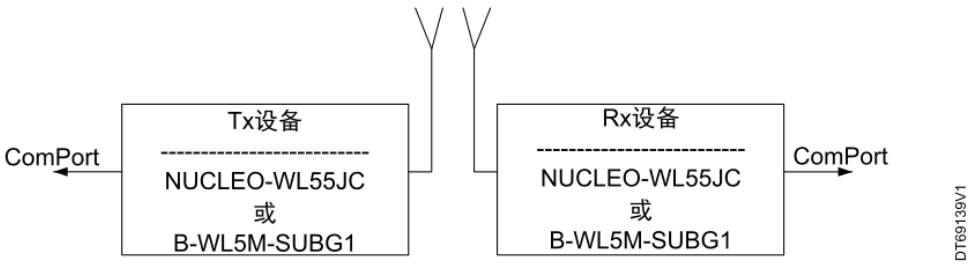
\Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_Per，并遵循与 LoRaWAN_End_Node 工程启动首选工具链时相同的步骤。

13.1 SubGhz_Phy_Per 硬件/软件环境设置

要设置 STM32WL 板（NUCLEO-WL55JC 或 B-WL5M-SUBG1），使用 USB Type-A 转 Mini-B 线缆将此板连接到计算机以及集成 ST-LINK 连接器（CN1）（对于 NUCLEO-WL55JC）或连接至调试连接器（CN3）上的外部 ST-LINK-V3（对于 B-WL5M-SUBG1）。

编译具有每个模式（TX 和 RX）的工程并加载到 2 个板中，然后打开各板串行端口上的首选超级终端，如下图所示：

图 9. SubGhz_Phy_Per 应用程序设置



13.2 SubGhz_Phy_Per 应用程序的设备配置总结

表 36. SubGhz_Phy_Per 应用程序配置的开关选项

项目模块 = 应用程序

详细信息	开关选项	定义	位置
Rx/Tx configuration	RX_TIMEOUT_VALUE	Rx 窗口超时	subghz_phy_app.c
	TX_TIMEOUT_VALUE	Tx 窗口超时	
	MAX_APP_BUFFER_SIZE	最大数据缓存大小	
	RX_CONTINUOUS_ON	RX 模式持续或超时	
	TEST_MODE	设备模式： RADIO_TX：无限地发送数据包 RADIO_RX：无限地接收数据包	
	APP_LONG_PACKET	长数据包选项(1)(2)	
调制配置	USE_MODEM_LORA	选择的 LoRa 调制解调器	subghz_phy_app.h
	USE_MODEM_FSK	选择的 FSK 调制解调器	
LoRa/FSK 通用参数	RF_FREQUENCY	收发器使用的频率	
	TX_OUTPUT_POWER	RF 输出功率：-17 至 22 dBm	
	PAYLOAD_LEN	数据缓存大小	
LoRa 特定参数	LORA_BANDWIDTH	带宽： • 0：125 kHz • 1：250 kHz • 2：500 kHz • 3：保留	
	LORA_SPREADING_FACTOR	扩频因子：SF7 至 SF12	
	LORA_CODINGRATE	编码率： • 1：4/5 • 2：4/6 • 3：4/7 • 4：4/8	
	LORA_PREAMBLE_LENGTH	Tx/Rx 前导码的长度	
	LORA_SYMBOL_TIMEOUT	超时前检查的符号数	
	LORA_FIX_LENGTH_PAYLOAD_ON	固定/动态长度有效负载选项	
	LORA_IQ_INVERSION_ON	IQ 反转选项	
FSK 特定参数	FSK_FDEV	频率偏移（Hz）	
	FSK_DATARATE	数据速率（bit/s）	
	FSK_BANDWIDTH	带宽（Hz）	
	FSK_PREAMBLE_LENGTH	Tx/Rx 前导码的长度	
	FSK_FIX_LENGTH_PAYLOAD_ON	固定/动态长度有效负载选项	
调试	DEBUGGER_ENABLED	启用 SWD 引脚。	sys_conf.h
低功耗	LOW_POWER_DISABLE	禁用低功耗模式。	
跟踪启用	APP_LOG_ENABLED	启用跟踪模式。	
跟踪级别	VERBOSE_LEVEL	设置跟踪级别。	

1. *仅在 FSK 调制时可用。*
2. *如需更详细信息，请参考[8]。*

14 AT_Slave 应用程序

此应用程序旨在实现由外部主机利用 AT 指令接口通过 UART 控制 SubGHz_Phy 调制解调器。

外部主机可以是嵌入此应用程序和 AT 驱动程序的主机微控制器，也可以只是一台执行终端功能的计算机。STM32WL 与外部主机之间利用 AT 指令通过 LPUART 进行通信。对于此类应用程序，STM32WL 设备始终处于“停止 2”模式，除非将其唤醒，以处理来自外部主机的 AT 指令。

适用性参见表 2。

有两种应用程序可供选择：

- SubGHz_Phy_AT_Slave 允许外部主机（利用 AT 指令）在 STM32WL 上运行一些 RF 测试。
- LoRaWAN_AT_Slave 允许外部主机（利用 AT 指令）通过内置 LoRa 无线电驱动 LoRaWAN 协议栈。

注意： *LoRaWAN_AT_Slave 还可用于运行与 SubGHz_Phy_AT_Slave 相同的 RT 测试指令。*

因此，再次使用 LoRaWAN 协议栈 AT 指令前必须复位设备。

要启动所选的 AT_Slave 工程，用户必须进入 \Projects\<target>\Applications\LoRaWAN\LoRaWAN_AT_Slave 或 \Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_AT_Slave，并遵循与 LoRaWAN_End_Node 工程启动首选工具链时相同的步骤。从适当的目标板中选择工程。

下表总结了可用于所选应用程序的全部 AT 指令。

表 37. AT 指令

指令	说明	LoRaWAN	SubGHz_PHY
一般指令			
AT	检查接口是否可用	有	
AT?	帮助所有支持的指令。	有	
ATZ	复位。	有	
AT+VER	获取应用程序/中间件版本。	有	
AT+VL	获取 UTC 格式的本地时间。	有	
LoRaWAN 协议栈上下文管理指令			
AT+RFS	擦除 Flash 存储器中的 LoRaWAN 上下文并在复位后恢复出厂设置。	有	无
AT+CS	将当前的 LoRaWAN 上下文恢复到 Flash 存储器扇区。	有	无
LoRaWAN 密钥、ID 和 EUI 管理指令			
AT+APPEUI	设置/获取应用程序 EUI。	有	无
AT+NWKKEY	设置/获取网络根密钥。	有	无
AT+APPKEY	设置/获取应用程序根密钥。	有	无
AT+APPSKEY	设置/获取应用程序会话密钥。	有	无
AT+NWKSKEY	设置/获取网络会话密钥。	有	无
AT+DADDR	设置/获取设备地址。	有	无
AT+DEUI	设置/获取模块唯一 ID。	有	无
AT+NWKID	设置/获取网络 ID。	有	无
LoRaWAN 加入并发送数据指令			
AT+JOIN	加入网络。	有	无
指令	说明	LoRaWAN	SubGHz_PHY
AT+LINKC	捎带链路检查发送至下一上行链路的 MAC 指令请求。	有	无

指令	说明	LoRaWAN	SubGHz_PHY
AT+SEND	将数据包发送到网络。	有	无
LoRaWAN 网络管理指令			
AT+ADR	设置/获取自适应数据率功能。	有	无
AT+DR	设置/获取数据率。	有	无
AT+BAND	设置/获取有效区域。	有	无
AT+CLASS	设置/获取 LoRa 类别。	有	无
AT+DCS	设置/获取占空比设置。	有	无
AT+JN1DL	设置/获取 Rx 窗口 1 的加入延迟。	有	无
AT+JN2DL	设置/获取 Rx 窗口 2 的加入延迟。	有	无
AT+RX1DL	设置/获取 Rx 窗口 1 的延迟。	有	无
AT+RX2DL	设置/获取 Rx 窗口 2 的延迟。	有	无
AT+RX2DR	设置/获取 Rx 窗口 2 的数据率。	有	无
AT+RX2FQ	设置/获取 Rx 窗口 2 的频率。	有	无
AT+TXP	设置/获取传输功率。	有	无
AT+PGSLOT	设置/获取单播 ping 时隙周期。	有	无
无线电测试指令			
AT+TTONE	设置 RF 调准测试。	有	
AT+TRSSI	设置/获取配置 LoRa RF 测试。	有	
AT+TTX	设置执行 PER RF Tx 测试将发送的数据包数。	有	
AT+TRX	设置执行 PER RF Rx 测试将接收的数据包数。	有	
AT+TTH	开始从 Fstart 到 Fstop（Hz 或 MHz）的 RF Tx 跳频测试，Fdelta 单位为 Hz。	有	
AT+TOFF	停止 RF 测试。	有	
LoRaWAN 认证指令			
AT+CERTIF	在联合模式下设置模块 LoRaWAN 认证。	有	无
信息指令			
AT+BAT	获取电池电量。	有	无

有关 AT 指令及其说明详情，请参考文档[5]。

下表总结了 AT_Slave 应用程序配置的主要选项。

表 38. AT_Slave 应用程序配置的开关选项

项目模块	详细信息	开关选项	定义	位置
LoRaWAN 协议栈(1)	唯一设备标识	LORAWAN_DEVICE_EUI	终端设备 IEEE 扩展唯一标识符（EUI）	se-identity.h
	加入 EUI	LORAWAN_JOIN_EUI	应用程序或加入服务器 IEEE EUI（仅在 OTAA 中使用）	
	终端设备地址	LORAWAN_DEVICE_ADDRESS	网络上的终端设备地址（仅在 ABP 中使用，由 OTAA 中的网络服务器生成）	
	应用程序根密钥	LORAWAN_APP_KEY	用于派生应用程序会话密钥的应用程序根密钥	
	网络根密钥	LORAWAN_NWK_KEY	用于在加入 OTAA 时派生会话密钥的网络根密钥	
	应用程序会话密钥	LORAWAN_APP_S_KEY	用于通过应用程序服务器加密/解密有效负载的应用程序会话密钥（仅在 ABP 中使用，由 OTAA 中的根密钥生成）	
	网络会话密钥	LORAWAN_NWK_S_KEY	用于通过网络服务器加密/解密有效负载的网络会话密钥（仅在 ABP 中使用，由 OTAA 中的根密钥生成）	
	支持的区域	REGION_EU868	设备支持的区域	lorawan_conf.h
		REGION_EU433		
		REGION_US915		
		REGION_AS923		
		REGION_AU915		
		REGION_CN470		
		REGION_CN779		
		REGION_IN865		
		REGION_RU864		
		REGION_KR920		
	AS923 通道计划	REGION_AS923_DEFAULT_CHANNEL_PLAN	用于区域 AS923 的通道计划	
	限制通道	HYBRID_ENABLED	限制 AU915、CN470 和 US915 区域的默认可用通道数量。	
	读取密钥	KEY_EXTRACTABLE	定义存储器中的密钥的读取访问。	
	可选类别	LORAMAC_CLASSB_ENABLED	Class B 类终端设备	
	上下文管理	CONTEXT_MANAGEMENT_ENABLED	存储和恢复 LoRaWAN 协议栈上下文	

项目 模块	详细信息	开关选项	定义	位置
LoRaWAN 应用程序(1)	自适应数据率	LORAWAN_ADR_STATE	ADR 选择	lora_app.h
	默认数据率	LORAWAN_DEFAULT_DATA_RATE	ADR 禁用时的数据率	
	Tx 功率	LORAWAN_DEFAULT_TX_POWER	启动时的默认 Tx 输出功率。	
	Ping 周期	LORAWAN_DEFAULT_PING SLOT_PERIODICITY	Rx ping 时隙周期	
	初始区域	ACTIVE_REGION	设备启动时使用的区域	
全局应用程序	调试	DEBUGGER_ENABLED	启用 SWD 引脚。	sys_conf.h
	低功耗	LOW_POWER_DISABLE	禁用低功耗模式。	
	跟踪启用	APP_LOG_ENABLED	启用跟踪模式。	
	跟踪级别	VERBOSE_LEVEL	设置跟踪级别。	

1. *仅在 LoRaWAN_AT_Slave 中可用。*

15 LoRaWAN 上下文管理说明

NVM 上下文管理用于在断电或复位板之前将当前的 LoRaWAN 协议栈配置存储到 ROM 中。
提议的解决方案会将 1484 字节的 LoRaMacNvmData_t 结构存储到预分配的 2 KB ROM 页中。
此结构的定义如下：

表 39. LoRaWAN NVM 上下文结构

位域	类型	大小（字节）	说明
密码	LoRaMacCryptoNvmData_t	40	与密码层相关的参数。每次 TX/RX 时改变。
MacGroup1	LoRaMacNvmDataGroup1_t	32	与 MAC 相关的、每次 TX/RX 后变化概率较高的参数。
MacGroup2	LoRaMacNvmDataGroup2_t	260	与 MAC 相关的、不太可能随着每次 TX/RX 而变化的参数。
SecureElement	SecureElementNvmData_t	216	与安全元素相关的参数（密钥、标识符等）
RegionGroup1	RegionNvmDataGroup1_t	20	与区域实现相关的、每次 TX/RX 过程后变化概率较高的参数。
RegionGroup2	RegionNvmDataGroup2_t	892	与区域实现相关的、不太可能随着每次 TX/RX 过程而变化的参数。
ClassB	LoRaMacClassBNvmData_t	24	与 Class B 类设备相关的参数。

15.1 NVM 上下文管理数据 API 定义

表 40. LoRaWAN 上下文管理 API 和回调

函数	说明
LmHandlerErrorStatus_t LmHandlerNvmDataStore (void)	• 中止 LoRaMAC。 • 准备将存储到 FLASH 中的 NVM 数据。 • 调用 OnStoreContextRequest() 回调，以执行 FLASH_Write 操作。 • 恢复 LoRaMAC。
void OnRestoreContextRequest (void *nvm, uint32_t nvm_size)	将 FLASH 中存储的 NVM 数据恢复到 RAM 中的备用缓冲区。
void OnStoreContextRequest (void *nvm, uint32_t nvm_size)	将 RAM 中的 NVM 数据存储到 FLASH 中。
void OnNvmDataChange (LmHandlerNvmContextStates_t state)	当存储或恢复操作完成时通过 LmHandler 调用。

如果未定义回调，则此功能视为禁用。此功能可通过第 10.2.12 节中所述的定义启用。

16 双核管理

STM32WL5x 设备嵌入两个 Cortex：

- Cortex-M4（称为 CPU1）
- Cortext-M0+（称为 CPU2）

在双核应用程序中，CPU1 上映射的应用程序部分与 CPU2 上映射的协议栈和固件底层分开。

在建议的双核模型中，会生成两个单独的二进制：CPU1 二进制位于 0x0800 0000, CPU2 二进制位于 0x0802 0000。

两个二进制都不知道彼此的函数地址：这就是必须建立通信模型的原因。该模型旨在确保用户可以更改 CPU1 上的应用程序，而不影响 CPU2 上的内核栈特性。但是，ST 仍在开源中实现了两个 CPU。

两个内核之间通过 IPCC 外设（处理器间通信控制器）与核间存储器实现连接，如第 16.1 节所述。

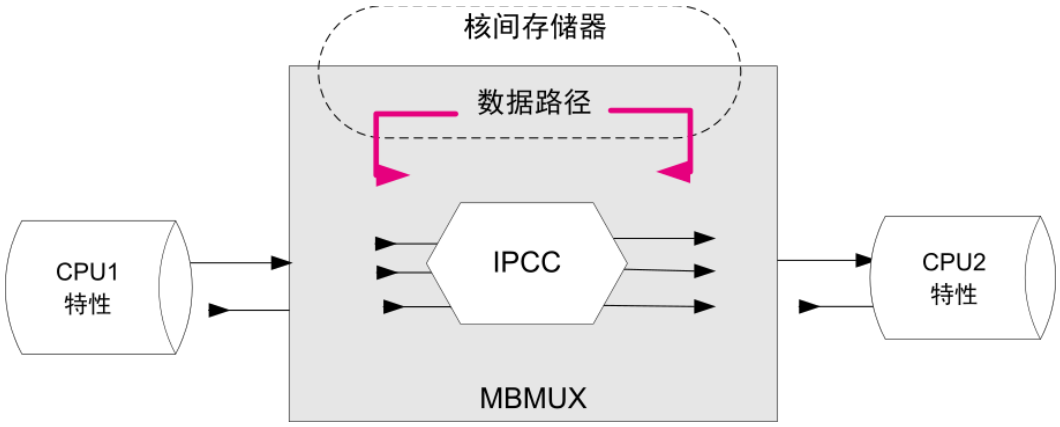
这种双核实现设计通过邮箱机制处理消息阻塞，因此与单核程序执行具相同的行为方式。

16.1 邮箱机制

邮箱是一种在两个处理器之间实现交换数据的服务。如下图所示，邮箱基于两个资源构建：

- **IPCC**：此硬件外设用于触发远程 CPU 的中断，并在它完成通知时接收中断。IPCC 高度可配置，每个中断通知都可以禁用/启用。IPCC 内部没有存储器管理。
- **核间存储器**：两个 CPU 均可对此共享存储器执行读取/写入操作。此存储器用于存储所有包含将在两个 CPU 之间交换的数据的缓冲区。

图 10. 邮箱概述



指定邮箱允许缓冲区定义的一定程度改变，而不破坏向后兼容性。

16.1.1 邮箱多路复用器（MBMUX）

如图 11 所述，待交换的数据需要通过 12 个可用的 IPCC 通道（每个方向各 6 个）进行通信。此通信将通过 MBMUX（邮箱多路复用器）完成，它是一个负责消息路由的固件组件。这些通道会标识 1 - 6。附加通道 0 专门用于系统功能。

数据类型分为不同的功能组。每个功能通过各自的 MBMUXIF（MBUX 接口）连接 MBMUX。

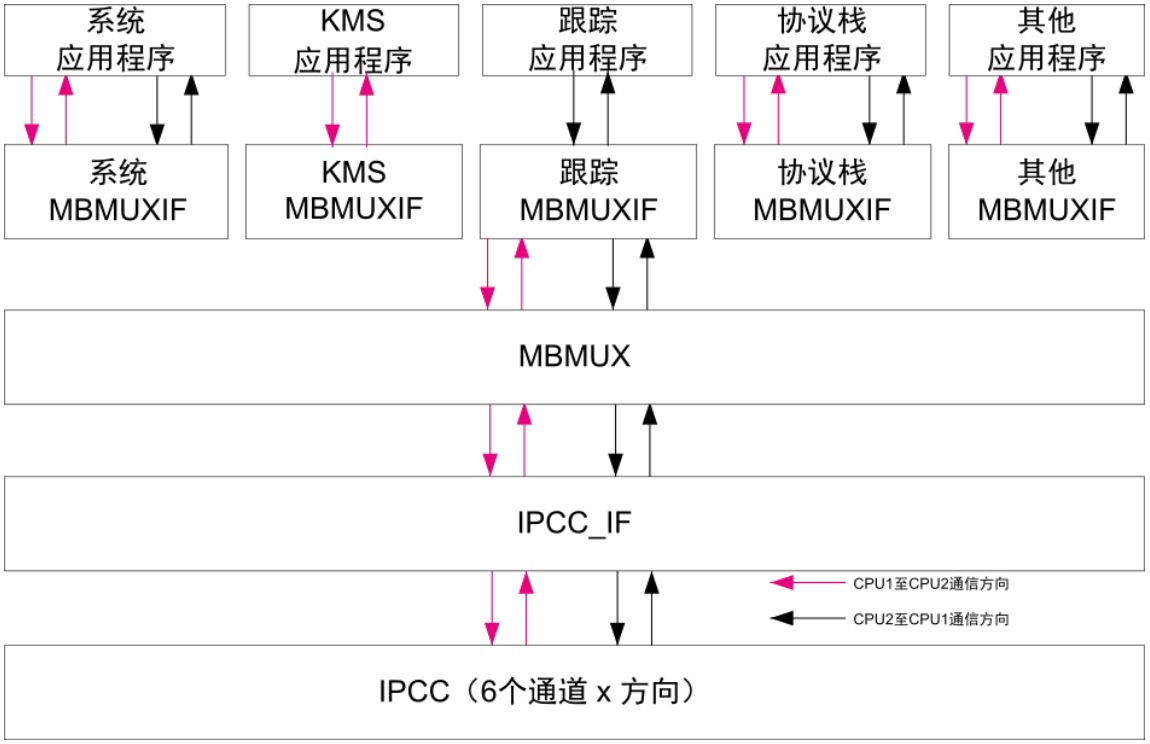
邮箱用于提取另一个内核执行的函数。

16.1.2 邮箱功能

在 STM32WL5x 设备中，MBMUX 具有以下功能：

- **系统**，支持与系统相关的所有通信
  这包括与所支持的协议栈之一相关的、或完全不相关的消息。CPU1 通道 0 用于通知 CPU2 已发布一条指令，并接收来自 CPU2 对该指令的回应。CPU2 通道 0 用于通知 CPU1 已发布异步事件。
  以下服务会映射到系统通道：
  - 系统初始化
  - IPCC 通道与功能注册
  - 有关功能属性和能力的信息交换
  - 可能用于高优先级操作（如 RTC 通知）的附加系统通道
- **跟踪**
  CPU2 会在循环队列中添加通过 IPCC 发送到 CPU1 的信息或调试。CPU1 负责处理此信息，将其输出到用于 CPU1 日志的相同通道（如 USART）。
- **KMS**（密钥管理服务）
- **无线电**
  可直接连接 sub-GHz 无线电，无需绕过 CPU2 栈。但是应使用专门的邮箱通道。
- **协议栈**
  此通道用于连接所有协议栈指令（如初始化或请求），以及与协议栈实现的协议相关的事件（响应/指示）。

图 11. MBMUX - 功能与 IPCC 通道之间的多路复用器



为了使用 MBMUX，需要注册一个功能（默认注册并始终映射到 IPCC 通道 0 的系统功能除外）。注册会动态分配到此功能，请求的 IPCC 通道数：通常每个方向（CPU1 至 CPU2 以及 CPU2 至 CPU1）各 1 个。

在以下情况下，此功能仅需要每个方向各 1 个通道：

- 跟踪功能仅用于将调试信息从 CPU2 发送到 CPU1。
- 仅 CPU1 使用 KMS 请求 CPU2 执行功能。

注意：

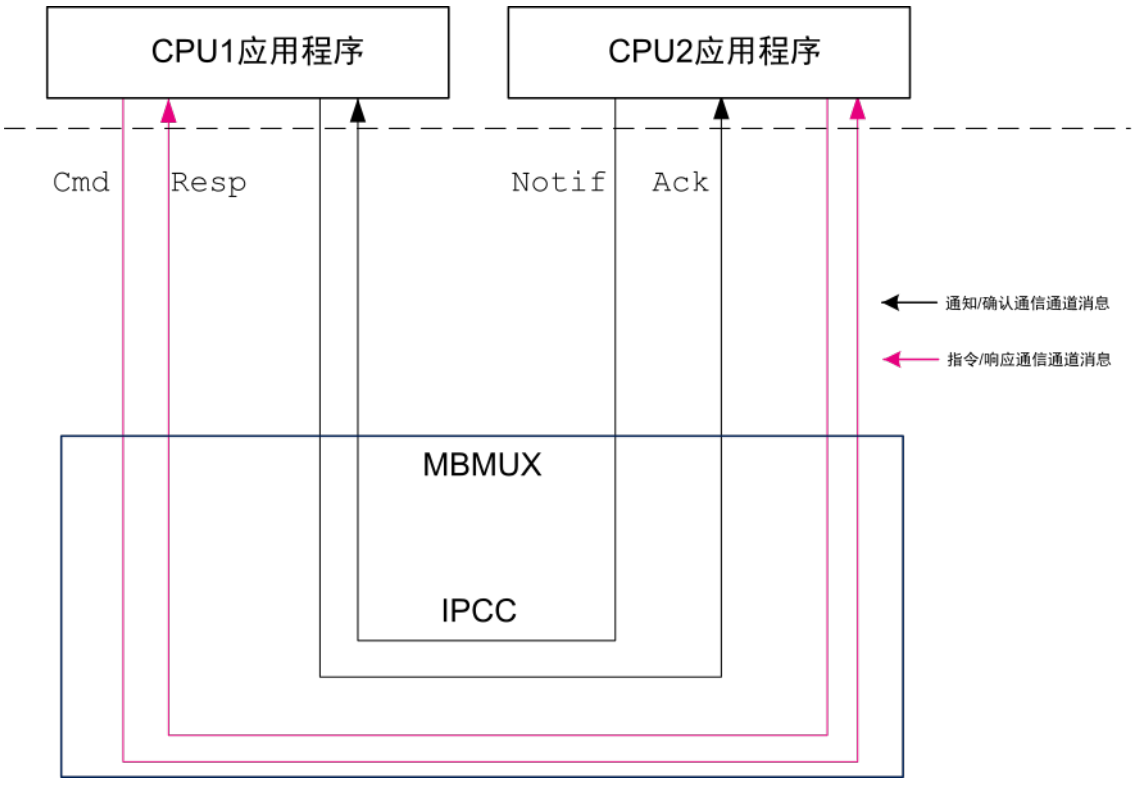
- *RTC 报警 A 使用一个不属于功能的 IPCC IRQ 传输中断。*
- *用户必须考虑添加 KMS 封装器，以便能够将其用作一个功能。*

16.1.3 MBMUX 消息

邮箱使用以下消息类型：

- CPU1 发送到 CPU2 的 Cmd 指令，包括：
  - Msg ID 标识由 CPU1 调用、但在 CPU2 上实现的函数。
  - Ptr buffer params 指向包含以上函数参数的缓冲区
  - Number of params
- Resp，CPU2 发送至 CPU1 的响应，包括：
  - Msg ID（与 Cmd Msg ID 的值相同）
  - Return value 包含以上函数的返回值。
- Notif，CPU2 发送到 CPU1 的通知，包括：
  - Msg ID 标识由 CPU2 调用、但在 CPU1 上实现的回调函数。
  - Ptr buffer params 指向包含以上函数参数的缓冲区。
  - Number of params
- Ack，CPU1 发送到 CPU2 的确认，包括：
  - Msg ID（与 Notif Msg ID 的值相同）
  - Return value 包含以上回调函数的返回值。

图 12. 通过 MBMUX 和 IPCC 通道传输的邮箱消息



16.2 核间存储器

核间存储器是两个内核都可以访问，并供两个内核用以交换数据、函数参数和返回值的集中式存储器。

16.2.1 CPU2 能力

多个 CPU2 能力必须告知 CPU1，以详细说明其支持的功能（如在 CPU2 上实现的协议栈、每个协议栈的版本号或者支持的区域）。

这些 CPU2 能力存储在 features_info 表中。CPU1 初始化时会请求此表中的数据，以展现 RAM 映射中所示的 CPU2 能力。

features_info 表包括：

- Feat_Info_Feature_Id：功能名称
- Feat_Info_Feature_Version：当前实现中使用的功能版本号

MB_MEM2 用于存储这些 CPU2 能力。

16.2.2 通过 CPU1 调用执行 CPU2 函数的邮箱序列

当 CPU1 需要调用 CPU2 feature_func_X()时，必须在 CPU1 上实现具有相同 API 的 feature_func_X()：

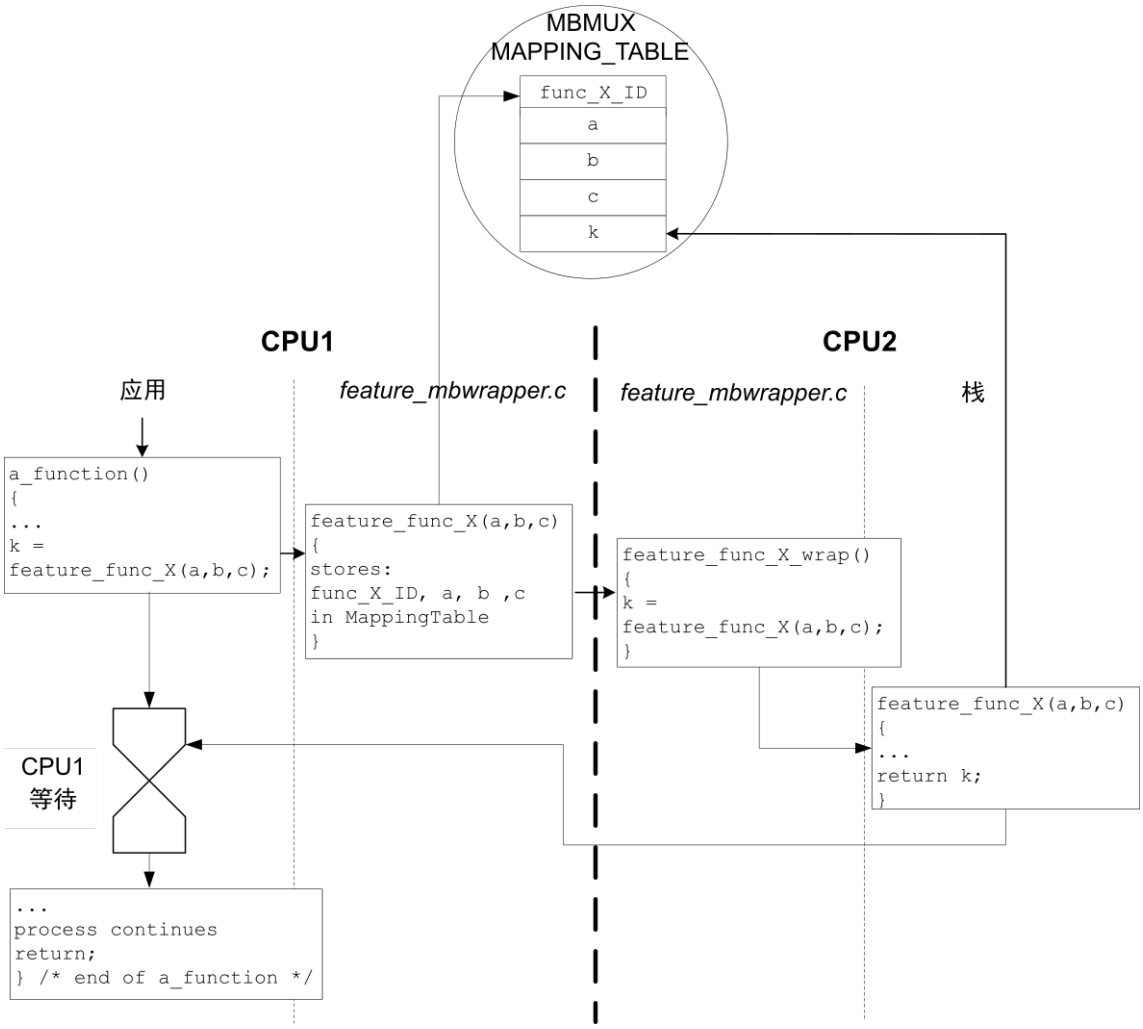
1. CPU1 发送在映射表中包含 feature_func_X()参数的指令：
   a. 注册期间执行初始化时会将与 feature_func_X()相关的 func_X_ID 添加到映射表。两个内核必须均了解 func_X_ID：这在编译时固定不变。
   b. CPU1 等待 CPU2 执行 feature_func_X()并进入低功耗模式。
   c. 如果处于低功耗模式下，则 CPU2 唤醒并执行 feature_func_X()。
2. CPU2 发送响应并将返回值添加到映射表中：
   a. IPCC 中断唤醒 CPU1。
   b. CPU1 检索映射表中的返回值。

相反地，当 CPU2 需要调用 CPU1 feature_func_X_2()时，则必须在 CPU2 上实现具有相同 API 的 feature_func_X_2()：

1. CPU2 发送在映射表中包含 feature_func_X_2()的通知。
2. CPU1 发送确认并将返回值添加到映射表中。

完整流程如下图所示。

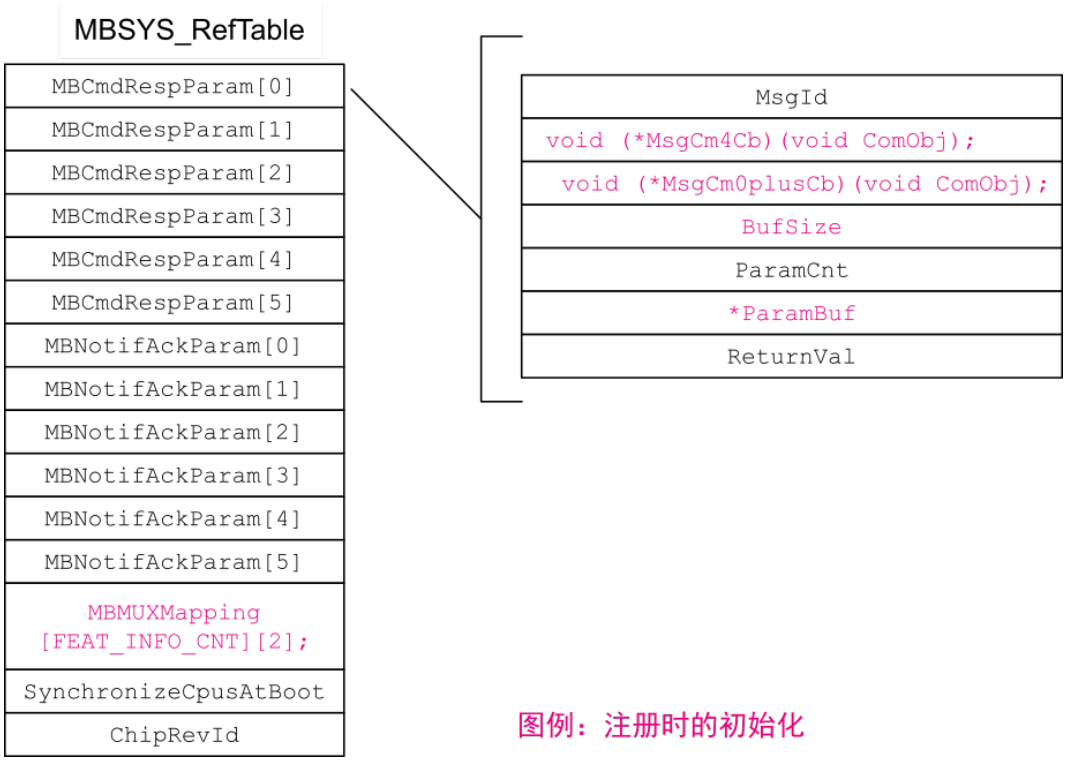
图 13. CPU1 至 CPU2 的 feature_func_X()过程



16.2.3 映射表

映射表是图 13 的 MBMUX 区的通用结构。在 RAM 映射中，存储器映射称为 MAPPING_TABLE。MBMUX 通信表（MBSYS_RefTable）说明如下图所示。

图 14. MBMUX 通信表



此 MBSYS_RefTable 包括：

- 用于每个 sic IPCC 通道的指令/响应和通知/确认参数的两个通信参数结构。
  图 13 的 MBMUX 映射表区中所示的每个通信参数包括：
  - MsgId：feature_func_X()的消息 ID
  - *MsgCm4Cb：CPU1 回调 feature_func_X()指针
  - *MsgCm0plusCb：CPU2 回调 feature_func_X()指针
  - BufSize：缓存大小
  - ParamCnt：消息参数编号
  - ParamBuf：参数的消息指针
  - ReturnVal：feature_func_X()的返回值
- MBMUXMapping：用于将通道映射到功能的图表
  注册期间执行 MBMUX 初始化时会填充此表。例如，如果无线电功能关联 Cmd/Response channel number = 1，则 MBMUXMapping 必须关联[FEAT_INFO_RADIO_ID][1]。
- SynchronizeCpusAtBoot：用于同步 CPU1 和 CPU2 处理的标志，如图 15 序列图所示。
- ChipRevId：存储硬件版本 ID。

MB_MEM1 用于发送 command/response set ()参数以及获取 CPU1 的返回值。

16.2.4 选项字节警告

代码中会设置陷阱，以避免错误的选项字节加载（参见产品数据手册中的高 MSI 系统时钟频率下的选项字节加载失败章节）。如果系统时钟设置低于或等于 16 MHz，则可删除陷阱。

16.2.5 RAM 映射

下表详细说明了 CPU1 和 CPU2 RAM 区以及核间存储器的映射。

表 41. STM32WL5x RAM 映射

页索引[1]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
分配区域/部分	CPU1 RAM																CPU1 RAM2_Shared		CPU2 RAM2_Shared		CPU2 RAM2											

1. *每页 2 KB。*

表 42. STM32WL5x RAM 分配和共享缓冲区

CPU 区	章节	模块	分配的符号	大小（字节）	总大小（字节）
CPU1 RAM	readwrite	-			
	CSTACK				
	HEAP				
CPU1 RAM2_Shared	MAPPING_TABLE	MBMUX_SYSTEM	MBMUX_ComTable_t MBSYS_RefTable	316	316
	MB_MEM1	MBMUX_LORAWAN	uint32_t aLoraCmdRespBuff[]	60	524
			uint32_t aLoraNotifAckBuff[]	20	
		MBMUX_RADIO	uint32_t aRadioCmdRespBuff[]	60	
			uint32_t aRadioNotifAckBuff[]	16	
		MBMUX_TRACE	uint32_t aTraceNotifAckBuff[]	44	
		MBMUX_SYSTEM	uint32_t aSystemCmdRespBuff[]	28	
			uint32_t aSystemNotifAckBuff[]	20	
			uint32_t aSystemPrioACmdRespBuff[]	4	
			uint32_t aSystemPrioANotifAckBuff[]	4	
			uint32_t aSystemPrioBCmdRespBuff[]	4	
			uint32_t aSystemPrioBNotifAckBuff[]	4	
		LMH_MBWRAPPER	uint8_t aLoraMbWrapShareBuffer[]	260	

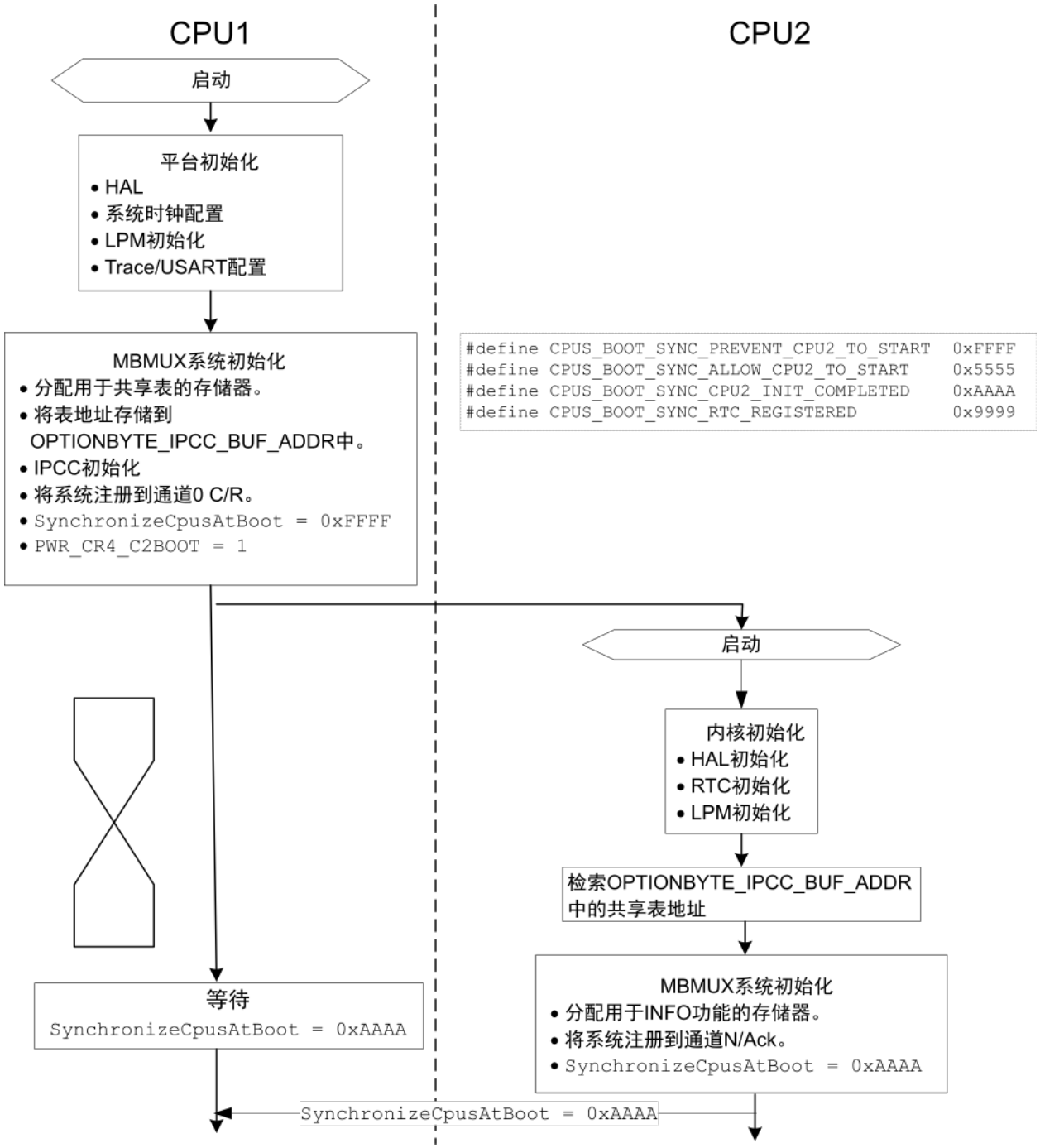
CPU 区	章节	模块	分配的符号	大小（字节）	总大小（字节）
CPU2 RAM2 _Shared	MB_MEM2	MBMUX_LORAWAN	FEAT_INFO_Param_t Feat_Info_Table	80	88
		MBMUX_LORAWAN	FEAT_INFO_List_t Feat_Info_List	8	
	MB_MEM3(1)	MBMUX_TRACE	uint8_t ADV_TRACE_Buffer[]	1024	1608
		MBMUX_LORAWAN	LoraInfo_t loraInfo	16	
		LMH_MBWRAPPER	uint8_t aLoraMbWrapShare2Buffer[]	312	
		RADIO_MBWRAPPER	uint8_t aRadioMbWrapRxBuffer[]	256	
CPU2 RAM2	readwrite	-			
	CSTACK				
	HEAP				

1. *本节将防止过度使用 Flash，以便通过 STM32CubeIDE 初始化这些 BSS RAM 变量。*

16.3 启动序列

CPU1 和 CPU2 的启动序列详见下图。

图 15. 启动序列

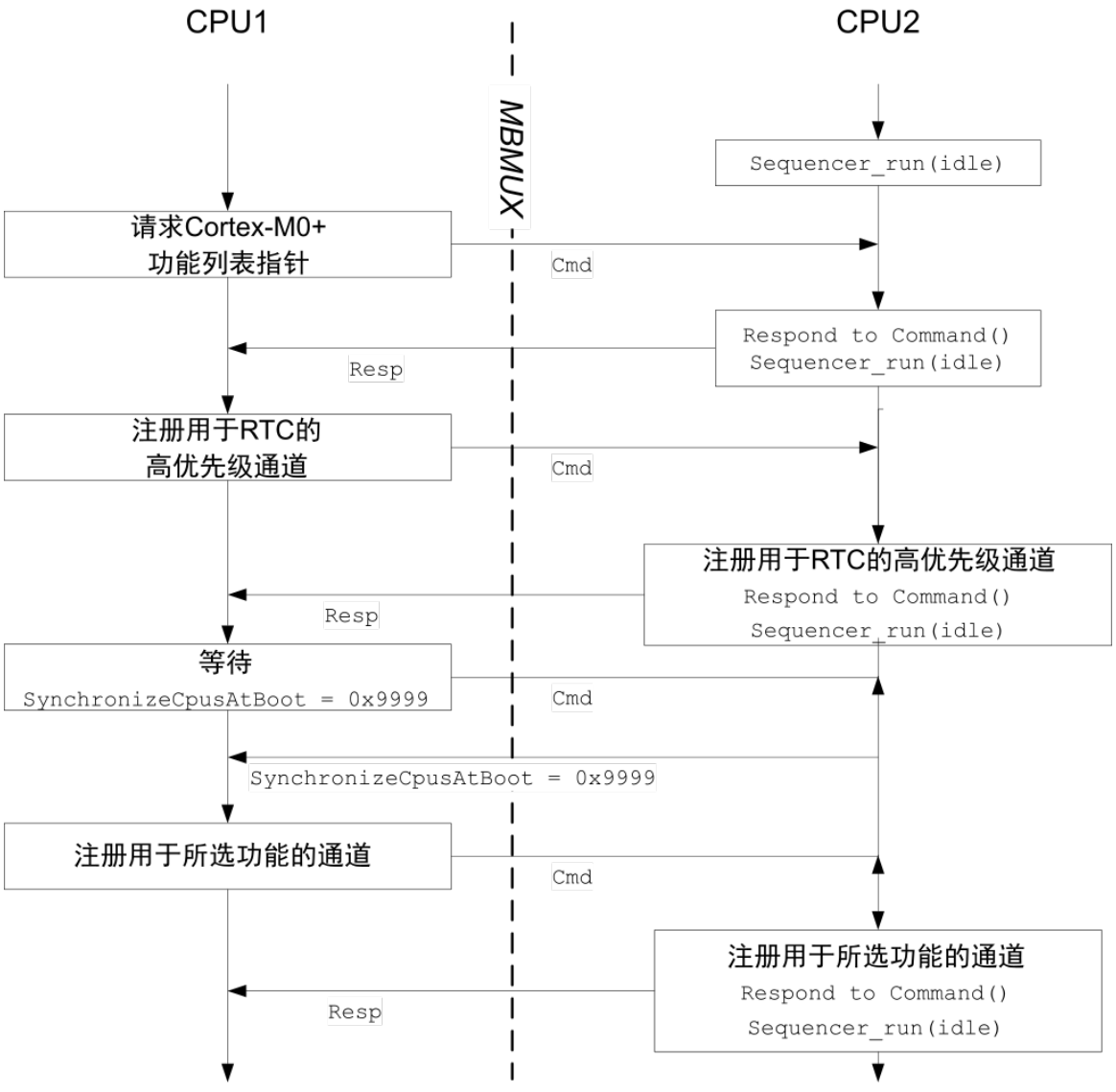


以下是所需的各个步骤：

1. CPU1 是此初始化序列中的主处理器：
   a. 执行平台初始化。
   b. 初始化 MBMUX 系统。
   c. 将 PWR_CR4_C2BOOT 标志设为 1，即启动 CPU2。
   d. 等待 CPU2 将 SynchronizeCpusAtBoot 标志设为 0xAAAA。
2. CPU2 启动并：
   a. 执行内核初始化。
   b. 检索共享表地址。
   c. 初始化 MBMUX 系统。
   d. 将 SynchronizeCpusAtBoot 设为 0xAAAA，以通知 CPU1 已结束初始化序列并准备就绪。
3. CPU1 确认此 CPU2 通知。

然后初始化两个内核，初始化通过 MBMUX 继续，如下图所示。

图 16. MBMUX 初始化

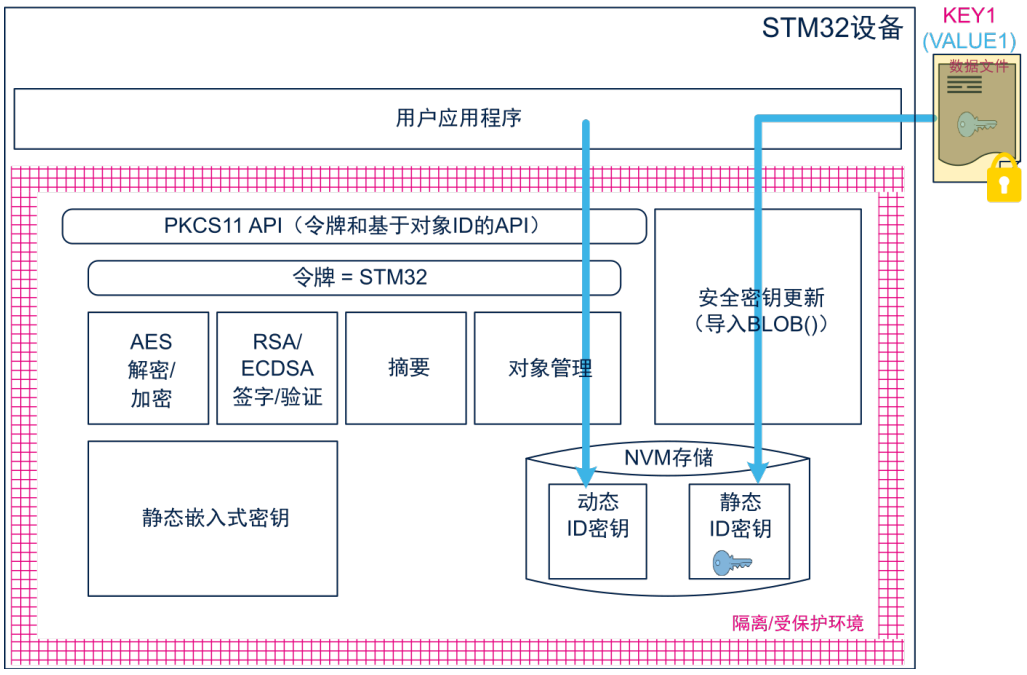


17 密钥管理服务（KMS）

密钥管理服务（KMS）通过标准 PKCS#11 API（由 OASIS 开发）提供加密服务，用于将密钥值提取给调用方（使用对象 ID 而非直接使用密钥值）。

KMS 可以在受保护/隔离环境内执行，以确保无法通过在受保护/隔离环境外运行的未授权代码访问密钥值，如下图所示。

图 17. **KMS 总体架构**



有关更多详细信息，请参考 STM32CubeWL 的 SBSFU 入门用户手册的 KMS 章节（UM2767）。

要激活 KMS 模块，必须在 C/C++编译器项目选项中将 KMS_ENABLE 设为 1。

KMS 仅支持下列 PKCS #11 API：

- 对象管理功能（创建/更新/删除）
- AES 加密/解密功能（CBC、CCM、ECB、GCM、CMAC 算法）
- 摘要功能
- RSA 和 ECDSA 签名/验证功能
- 密钥管理功能（密钥生成/派生）

17.1 KMS 密钥类型

KMS 管理三种密钥类型，但是仅使用以下两种：

- 静态嵌入式密钥
  - 代码内嵌入的预定义密钥，不可修改
  - 不可改变的密钥
- NVM_DYNAMIC 密钥：
  - 运行时密钥
  - 可在使用 KMS（DeriveKey()或 CreateObject()）创建密钥时定义的密钥 ID
  - 可以删除或定义为可变的密钥

17.2 KMS 密钥定义

协议栈使用的静态和动态密钥具有不同的大小。

静态密钥

每个静态密钥由以下两个元素组成：

- blob 头文件：五个 4 字节字段（总计 = 20 字节）：version、configuration、blob_size、blob_count 和 object_id。
- blob 缓冲区：元素列表中的一些必要和可选 blob 元素，定义如下：

表 43. **全局 KMS blob 元素**

特性区	值	必要	大小（字节）	说明
CKA_CLASS	CKO_SECRET_KEY	有	12	blob 元素类型
CKA_KEY_TYPE	CKK_AES	有	12	密钥类型
CKA_VALUE	“KEY_VALUE”	有	24	密钥值（uint32_t 格式）
CKA_DERIVE	真 / 假	无	12	默认用于启用/禁用功能的可选参数。如果未定义，则这些字段均为 TRUE。
CKA_ENCRYPT	真 / 假	无	12	
CKA_DECRYPT	真 / 假	无	12	
CKA_COPYABLE	真 / 假	无	12	
CKA_EXTRACTABLE	真 / 假	无	12	
CKA_LABEL	“UNIQUE LABEL”	有	静态密钥为 12 动态密钥为 16	唯一标签

示例：

由 blob 头文件及 8 个 blob 元素（CKA_CLASS、CKA_KEY_TYPE、CKA_VALUE、CKA_LABEL、CKA_DERIVE、CKA_DECRYPT、CKA_COPYABLE 和 CKA_EXTRACTABLE）组成的静态密钥的总大小为 128 字节（blob 头文件 = 20 字节，blob 缓冲区 =（12 × 7 + 24）= 108 字节）。

动态密钥

每个动态密钥由数据头文件、blob 头文件和 blob 缓冲区 3 个元素组成，其中：

- 数据头文件：八个 4 字节字段（总计 = 32 字节）：magic1、magic2、slot、instance、next、data_type、size 和 checksum。

示例：

由数据头文件、blob 头文件和 5 个 blob 元素（CKA_CLASS、CKA_KEY_TYPE、CKA_VALUE、CKA_LABEL 和 CKA_EXTRACTABLE）组成的动态密钥的总大小为 128 字节（数据头文件 = 32 字节，blob 头文件 = 20 字节，blob 缓冲区 =（12 + 12 + 24 + 12 + 16）= 76 字节。

注意：

- *NVM 动态存储器始终从初始数据头文件元素开始。*
- *每次‘删除’动态密钥（如新密钥值代替废弃密钥）时，会将附加数据头文件写入到存储器，以声明不再使用之前的实例。*

17.3 LoRaWAN 密钥

在 STM32CubeWL 应用程序中，仅在双核应用程序的 CPU2 上使用 KMS。根密钥采用静态嵌入式密钥。所有派生密钥均为 NVM_DYNAMIC 密钥。

对于 LoRaWAN 协议栈，不可改变的根密钥详见下表。

表 44. 具有 blob 属性的 LoRaWAN 静态密钥

特性区	密钥				
	LoRaWAN_Zero_Key	LoRaWAN_APP_Key	LoRaWAN_NWK_Key	LoRaWAN_NWK_S_Key（仅 ABP）	LoRaWAN_APP_S_Key（仅 ABP）
CKA_CLASS	CKO_SECRET_KEY				
KA_KEY_TYPE	CKK_AES				
CKA_VALUE	“KEY_VALUE”				
CKA_DERIVE	FALSE	TRUE	TRUE	TRUE	TRUE
CKA_ENCRYPT	TRUE	FALSE	FALSE	TRUE	TRUE
CKA_DECRYPT	FALSE	FALSE	FALSE	TRUE	TRUE
CKA_COPYABLE	FALSE				
CKA_EXTRACTABLE	FALSE	TRUE/FALSE defined by #define KEY_EXTRACTABLE			
CKA_ LABEL	“UNIQUE LABEL”				

所有其他密钥均为可改变的 NVM_DYNAMIC 生成密钥，详见下表。

表 45. 具有 blob 属性的 LoRaWAN 动态密钥

特性区	密钥							
	LoRaWAN_NWK_S_Key（仅 OTAA）	LoRaWAN_APP_S_Key（仅 OTAA）	MC_ROOT_Key	MC_KE_Key	MC_KEY_0	MC_APP_S_Key_0	MC_NWK_S_Key_0	SLOT_RAND_ZERO_Key
CKA_CLASS	CKO_SECRET_KEY							
KA_KEY_TYPE	CKK_AES							
CKA_VALUE	“KEY_VALUE”							
CKA_DERIVE	TRUE							
CKA_ENCRYPT	TRUE							
CKA_DECRYPT	TRUE							
CKA_COPYABLE	TRUE							
CKA_EXTRACTABLE	TRUE/FALSE defined by #define KEY_EXTRACTABLE							
CKA_ LABEL	“UNIQUE LABEL”							

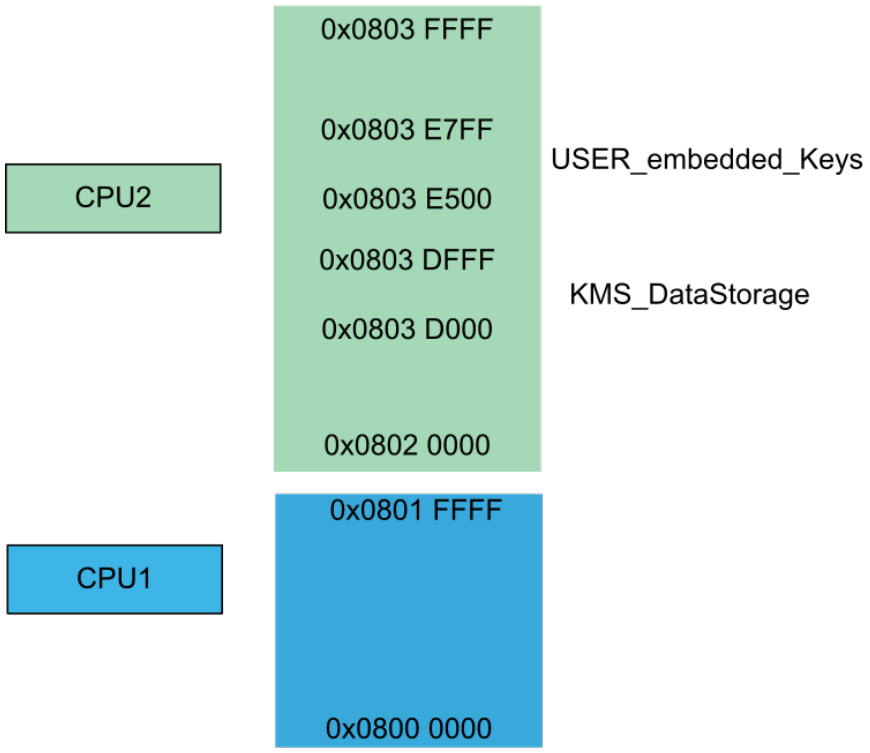
17.4 用户应用程序的 KMS 密钥存储器映射

静态嵌入式密钥对应 USER_embedded_Keys（用于根密钥）。这些密钥位于 Flash 存储器/ROM 中的专门数据存储区。用户应用程序的链接器文件将其放置到 0x0803 E500 至 0x0803 E7FF 之间，如下图所示。

NVM_DYNAMIC 密钥置于 KMS 密钥数据存储区，KMS_DataStorage。

总数据存储区必须为 4 KB，如第 17.5 节所述。密钥置于：0x0803 D000 至 0x0803 DFFF 区域，如下图所示。如果需要更多密钥，则可能增加此空间。

图 18. ROM 存储器映射



17.5 如何确定用于 KMS 数据存储的 NVM 大小

NVM 由 2 KB 页面组成。由于双重缓冲（触发器 EEPROM 仿真机制），每页都需要“双重”配置。因此至少应为 NVM 分配 4 KB。分配大小在链接器文件中定义。

应用程序提议的链接器文件使用最小允许大小（2 × 2 KB）。相关限制/不足如下所述。用户必须根据应用程序特定需求确定 NVM 大小。

应用程序仅使用 NVM 存储动态密钥。包含数据头文件、blob 头文件以及 5 个元素的 blob 缓冲区的 LoRaWAN 动态密钥占用 108 字节。空 NVM 初始化后将包含全局 32 字节数据头文件，对于每个废弃密钥，会写入附加 32 字节数据头文件，以声明以前的实例不再可用。

根据以上值，可以评估 2 KB 字节空间中可以存储的密钥数量：

（2048 - 32）/（108 + 32）= 14.4 ==> 执行触发器操作前可将 14 个动态密钥存储到 2 KB 存储器页面中。

在用户应用程序配置中，仅使用 NVM_DYNAMIC。NVM_STATIC 可通过 blob 填充，但是用户应用程序未涵盖。

NVM_DYNAMIC 可托管派生密钥（通过 C_DeriveKey()）和根密钥（通过 C_CreateObject()）。

LoRaWAN 应用程序：

- 每次在 ABP 模式下加入时生成两个派生密钥
- 每次在 OTAA 模式下加入时生成四个派生密钥

在较为复杂的场景（如多播设置）中，最多可以生成 10 个同时激活的派生密钥。如果用户想要编写一个使用超过 14 个密钥的应用程序，则必须为链接器文件分配附加 NVM 页面。

NVM 空间越小，写入和擦除的 NVM 越多，其预期寿命越短。

销毁密钥并不意味着删除此密钥，而是将此密钥标记为已销毁，并在下次触发器切换时不复制此密钥。销毁标志也占用一些 NVM 字节：销毁 8 个密钥后，剩余空间会小于 4 个密钥。

对于每次加入时生成 4 个密钥的场景，销毁前一个加入密钥后，预期寿命估计如下：

- 在第 3 个加入会话中，派生 4 个新密钥，但页面 1 中没有用于最后一个密钥的位置。所有 4 个密钥（仍处于激活状态）均放在页面 2 中。由于 NVM 页面仅可完全擦除，所以会立即擦除页面 1。
- 在第 5 次加入时，擦除页面 2，并将密钥存储回页面 1。40,000 次加入后，两个 NVM 页面被擦除 10,000 次，这就是 Flash 扇区的估计生命周期。
- 如果用户应用程序应非常频繁地加入（例如每 2 小时加入一次），那么预期 NVM 寿命为 80,000 小时（约 9 年）。如果每天执行一次加入过程，则生命周期将远远超过十年。

请求的同时激活（未销毁）的派生密钥数量越多，触发器机制的效率就越低。

总之，对于需要保持 NVM 生命周期的应用程序，建议确保 NVM 大小远远超过同时激活（未销毁）的密钥数量。

注意： *废弃密钥必须销毁，否则，如果页面 1 中完全填满激活密钥，触发器将无法切换，并会产生错误。*

17.6 构建应用程序的 KMS 配置文件

在 LoRaWAN 应用程序中，通过设置以下代码使用 KMS

```
#define LORAWAN_KMS      1
```

位于 CM0PLUS/LoRaWAN/Target/lorawan_conf.h

以下文件必须包含 SubGHz_Phy 协议栈密钥的信息：

- CM0PLUS/Core/Inc/kms_platf_objects_config.h 中定义的嵌入式密钥结构
- 与 SubGHz_Phy 协议栈密钥相关的嵌入式对象句柄，使用 CM0PLUS/Core/Inc/kms_platf_objects_interface.h 中定义的 KMS 模块

17.7 嵌入式密钥

SubGHz_Phy 协议栈的嵌入式密钥必须存储在由安全的附加软件（如 SBSFU、Secure Boot 和 Firmware Update）确保数据机密性和完整性的 ROM 区域中。有关 SBSFU 的更多详情，请参考 STM32CubeWL 上的 SBSFU 的集成指南应用笔记（AN5544）。

这些嵌入式密钥置于图 18 所示的 ROM 中。

18 如何保护 LoRaWAN 应用程序

文档[10]阐述了如何使用 SBSFU 框架保护双核 LoRaWAN 应用程序。

19 系统性能

19.1 内存占用

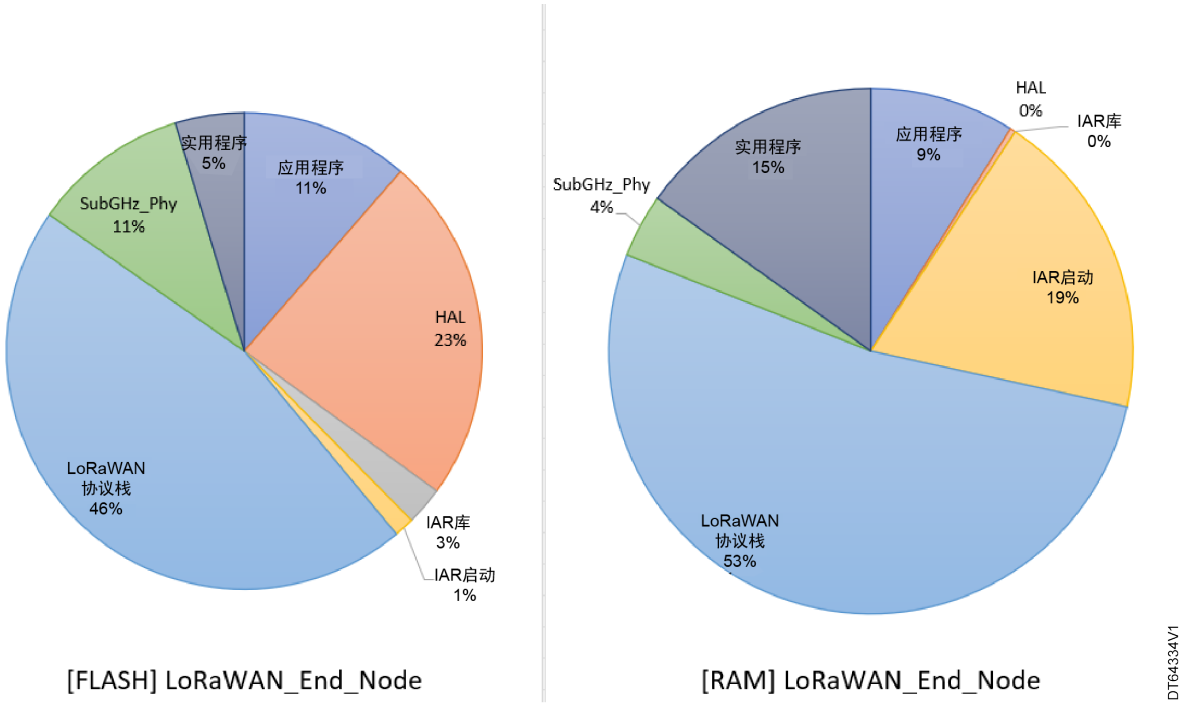
下表中的值基于以下配置的 IAR Embedded Workbench®编译器（EWARM 版本 9.20.1）测得：

- 大小优化级别 3
- 调试选项关闭
- 跟踪选项：VLEVEL_M（启用调试跟踪）
- 目标：NUCLEO-WL55JC1
- LoRaWAN_End_Node 应用程序
- LoRaMAC Class A 类设备
- LoRaMAC 区域 EU868 和 US915

表 46. LoRaWAN_End_Node 应用程序的内存占用值

项目模块	Flash 存储器（字节）	RAM（字节）	说明
应用	7189	958	内核、应用程序和目标组件
LoRaWAN 协议栈	28789	5630	中间件 Lmhandler 接口、密码、MAC 和区域
HAL	14831	36	STM32WL HAL 和 LL 驱动程序
实用程序	2947	1620	所有 STM32 服务（序列发生器、时间服务器、低功耗管理器、跟踪、存储器）
SubGHz_Phy	6676	417	中间件无线电接口
IAR 库	1630	0	专用 IAR 库
IAR 启动	858	2048	Int_vect、init routines、init table、CSTACK 和 HEAP
总应用程序	62920	10709	LoRaWAN_End_Node 应用程序的内存占用

图 19. Flash 存储器和 RAM 占用

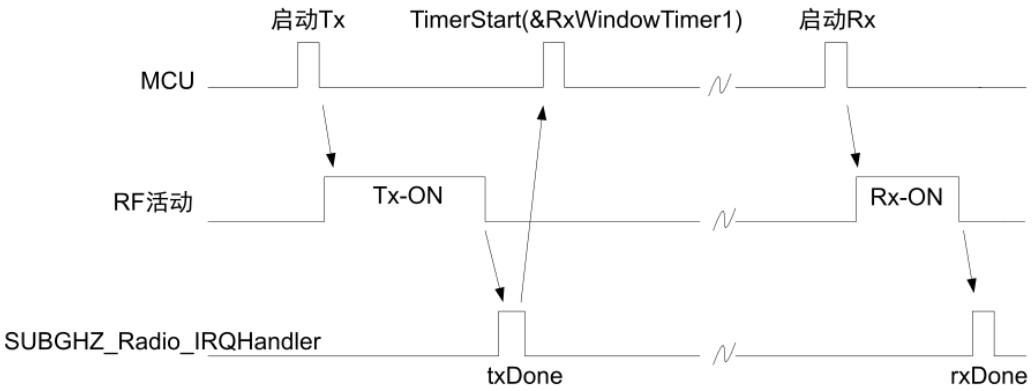


19.2 实时限制

LoRa RF 异步协议意味着遵循严格的 Tx/Rx 时序建议（参见下图）。

STM32WL Nucleo 板（NUCLEO-WL55JC）经过优化，实现了用户透明的低锁定时间和快速的自动校准操作。BSP 结合了发射器启动时间和接收器启动时间限制（参见第 4 节）。

图 20. Rx/Tx 时间图



Rx 窗口通道启动。Rx1 窗口在 txDone 下降沿后打开 1 秒（±20 µs）。Rx2 窗口在 txDone 下降沿后打开 1 秒（±20 µs）。

JOIN_ACCEPT 在上行链路调制结束后使用 5 秒（±20 µs）和 6 秒延迟。

必须遵循当前的调度中断级别优先级。换言之，所有新用户中断的优先级必须高于 Radio IRQ_interrupt，以避免收到的启动时间出现延迟。

19.3 功耗

测量了 STM32WL Nucleo 板 NUCLEO-WL55JC1 的功耗。

测量设置：

- 无调试
- 跟踪级别 VLEVEL_OFF（无跟踪）
- no SENSOR_ENABLED

测量结果：

- “停止 2”模式下的典型功耗为 = 2 µA（参见图 22）。
- TCXO 处于 Tx 模式时的典型功耗 = 23 mA（参见图 21）。
- TCXO 处于 Rx 模式时的典型功耗 = 7 mA（参见图 21）。

测量图：30 秒的瞬时功耗。

图 21. NUCLEO-WL55JC1 当前功耗与时间

图 22. NUCLEO-WL55JC1 在“停止 2”模式下的当前功耗

版本历史

表 47. 文档版本历史

日期	版本	变更
2019 年 12 月 10 日	1	初始版本。
2020 年 4 月 27 日	2	全面更新文档结构和内容。
2020 年 11 月 17 日	3	更新了： • *图 1. 项目文件结构* • *第 4 节“BSP STM32WL Nucleo-73 板”*中的注释 • *第 6 节“LoRaWAN 中间件说明”*简介 • *第 6.6 节“中间件 LmHandler 应用程序函数”*的标题和简介 • *表 22 和表 23* • *第 8.1.1 节“激活方式和密钥”* • *表 38. End_Node 应用程序配置的开关选项* • *表 39. AT_Slave 应用程序配置的开关选项* • *表 40. PingPong 应用程序配置的开关选项* • *第 13.1 节 内存占用* 增加了： • *表 26.LmHandler 过程* • *第 11 节 双核管理* • *第 12 节“密钥管理服务（KMS）”* 删除了*第 6.7 节*中的“板唯一 ID”和“板随机种子”表格。
2021 年 2 月 18 日	4	更新了： • 第 8.1.2 节、第 8.1.4 节、第 8.1.5 节、第 8.1.6 节、第 8.1.7 节、第 8.1.8 节和第 8.1.9 节中的 lora_app.c 重命名为 lora_app.h • 表 38.End_Node 应用程序配置的开关选项 • 表 39.AT_Slave 应用程序配置的开关选项
2021 年 7 月 9 日	5	更新了： • “概述”章节重命名为第 1 节“概述” • 第 11 节“SubGhz_Phy_PingPong 应用程序”简介 • 图 7.SubGhz_Phy_PingPong application setup • 第 14 节“双核管理”简介 • 第 14.1.3 节“MBMUX 消息” • 第 14.2.5 节“RAM 映射” • 第 15.2 节“KMS 密钥定义” • 第 15.3 节“LoRaWAN 密钥” • 图 17.ROM 存储器映射 • 第 15.5 节“如何确定用于 KMS 数据存储的 NVM 大小” • 第 15.6 节“构建应用程序的 KMS 配置文件” • 第 17.1 节 内存占用 • 第 11 节“SubGhz_Phy_PingPong 应用程序”简介 • 图 7.SubGhz_Phy_PingPong application setup • 第 14 节“双核管理”简介 • 第 14.1.3 节“MBMUX 消息” • 第 14.2.5 节“RAM 映射” • 第 15.2 节“KMS 密钥定义” • 第 15.3 节“LoRaWAN 密钥” • 图 17.ROM 存储器映射 • 第 15.5 节“如何确定用于 KMS 数据存储的 NVM 大小” • 第 15.6 节“构建应用程序的 KMS 配置文件” • 第 17.1 节“内存占用” 增加了： • 第 6.6 节“应用程序回调”中的 7 个表格 • 第 12 节“SubGhz_Phy_Per 应用程序” • 第 16 节“如何保护 LoRaWAN 应用程序”

日期	版本	变更
2022 年 2 月 21 日	6	更新了： • 表 1.缩略语和术语 • 第 2 节“STM32CubeWL 概述” • 第 6 节“LoRaWAN 中间件说明” 增加了： • 第 5 节“LoRaWAN 协议栈说明” • 第 5.4.3 节“驱动无线电所需的 STM32 外设” • 第 9.1 节“LoRaWAN 用户代码段说明” • 第 9.2.12 节“上下文管理” • 第 13 节“LoRaWAN 上下文管理说明” • 第 13.1 节“NVM 上下文管理数据 API 定义” • 第 10.2 节“设备配置” • 第 10.2.1 节“调制定义” • 第 10.2.2 节“有效负载长度” • 第 10.2.3 节“区域和频率” • 第 10.2.4 节“带宽、扩频因子和数据率” • 第 10.2.5 节“前导码长度” 删除了： • STM32CubeWL 架构

日期	版本	变更
2022 年 11 月 17 日	7	更新了： • 章节介绍 • 第 1 节“概述” • 第 2 节“STM32CubeWL 概述” • 图 1.项目文件结构 • 第 4.2 节“RF 开关” • 第 6 节“LoRaWAN 协议栈说明” • 第 6.3 节 LoRaWAN 认证 • 图 3.静态 LoRa 架构 • 第 7.4 节“中间件 MAC 层定时器” • 第 7.5.2 节“主要应用程序函数定义” • 第 7.6 节“应用程序回调” • 第 7.7 节“扩展应用程序函数” • 第 8.1 节“中间件无线电驱动程序结构” • 第 8.2 节“无线电 IRQ 中断” • 第 10.2.7 节“数据缓存大小” • 第 10.2.8 节“自适应数据率（ADR）” • 第 10.2.11 节“LoRa 频段选择” • 第 10.2.16 节“调试开关” • 第 10.3 节“LoRaWAN_End_Node 应用程序的设备配置总结” • 第 11 节“SubGhz_Phy_PingPong 应用程序” • 第 11.1 节“SubGhz_Phy_PingPong 硬件/软件环境设置” • 第 11.2.3 节“区域和频率” • 第 13 节“SubGhz_Phy_Per 应用程序” • 第 13.1 节“SubGhz_Phy_Per 硬件/软件环境设置” • 第 13.2 节“SubGhz_Phy_Per 应用程序的设备配置总结” • 第 14 节“AT_Slave 应用程序” • 第 15 节“LoRaWAN 上下文管理说明” • 第 19.1 节 内存占用 • 图 19.Flash 存储器和 RAM 占用 增加了： • 第 5 节“BSP B-WL5M-SUBG1 板” • 第 5.1 节“RF 开关” • 第 5.2 节“外部组件” • 第 6.2 节“LoRaWAN 调试” • 第 10.2.9 节“Tx 功率” • 第 12 节“SubGHz_Phy_LrFhss 应用程序” • 第 12.1 节“SubGhz_Phy_LrFhss 硬件/软件环境设置” • 第 12.2 节“SubGhz_Phy_LrFhss 应用程序的设备配置总结”

目录

表格索引

图片目录

重要通知 - 请仔细阅读

意法半导体公司及其子公司（“意法半导体”）保留随时对 ST 产品和/或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于意法半导体产品的最新信息。意法半导体产品的销售依照订单确认时的相关意法半导体销售条款。

买方自行负责对意法半导体产品的选择和使用，意法半导体概不承担与应用协助或买方产品设计相关的任何责任。

意法半导体不对任何知识产权进行任何明示或默示的授权或许可。

转售的意法半导体产品如有不同于此处提供的信息的规定，将导致意法半导体针对该产品授予的任何保证失效。

ST 和 ST 标志是意法半导体的商标。关于意法半导体商标的其他信息，请访问 www.st.com/trademarks。其他所有产品或服务名称是其各自所有者的财产。本文档中的信息取代本文档所有早期版本中提供的信息。