Как создать приложение LoRa® с помощью STM32CubeWL

Вступление

В этом примечании к приложению пользователь проходит через все шаги, необходимые для создания конкретных приложений LoRa® на основе микроконтроллеров серии STM32WL.

LoRa® - это тип беспроводной телекоммуникационной сети, предназначенный для обеспечения связи на большие расстояния с очень низкой скоростью передачи данных и для использования датчиков с длительным сроком службы батарей. LoRaWAN® определяет протокол связи и безопасности, который обеспечивает взаимодействие с сетью LoRa®.

Прошивка в пакете MCU STM32CubeWL соответствует протоколу спецификации LoRa Alliance® под названием LoRaWAN® и имеет следующие основные функции:

- Готовность к интеграции приложений
- Простое добавление маломощного решения LoRa®
- Чрезвычайно низкая загрузка процессора
- Нет требований к задержке
- Небольшой объем памяти STM32
- Услуги хронометража с низким энергопотреблением

Прошивка пакета MCU STM32CubeWL основана на драйверах STM32Cube HAL.

В этом документе представлены примеры приложений заказчика на платах STM32WL Nucleo NUCLEO_WL55JC (коды заказа NUCLEO WL55JC1 для диапазона высоких частот и NUCLEO-WL55JC2 для диапазона низких частот).

Чтобы в полной мере воспользоваться информацией в этой заметке по применению и создать приложение, пользователь должен быть знаком с микроконтроллерами STM32, технологией LoRa® и понимать системные службы, такие как управление с низким энергопотреблением и последовательность задач.

1 Общая информация

STM32CubeWL работает на микроконтроллерах серии STM32WL на базе процессора Arm® Cortex®-M.

Таблица 1. Аббревиатуры и термины

Acronym	Определение
ABP	Активация путем персонализации
ADR	Адаптивная скорость передачи данных
BSP	Пакет поддержки платы
DC/DC	Преобразователь постоянного тока в постоянный ток
HAL	Уровень аппаратной абстракции
IoT	Интернет вещей
IPCC	Контроллер межпроцессорной связи
IRQ	Запрос на прерывание
LBT	Слушать, прежде чем говорить
LoRa	Радиотехнология дальнего действия
LoRaWAN	Глобальная сеть LoRa
LPWAN	Глобальная сеть с низким энергопотреблением
MAC	Контроль доступа к среде
MCPS	Подуровень общей части MAC
MIB	Информационная база MAC
MLME	Объект управления подуровнем MAC
MSC	Диаграмма последовательности сообщений
OTAA	Активация по воздуху
PA	Усилитель мощности
PER	Частота ошибок пакета
PRBS	Режим псевдослучайная двоичная последовательность
RSSI	Индикатор уровня принимаемого сигнала
Rx	Прием
SWD	Отладка последовательного кабеля
<target>	Платы STM32WL Nucleo (NUCLEO-WL55JC)
Tx	Передача

Справочные документы

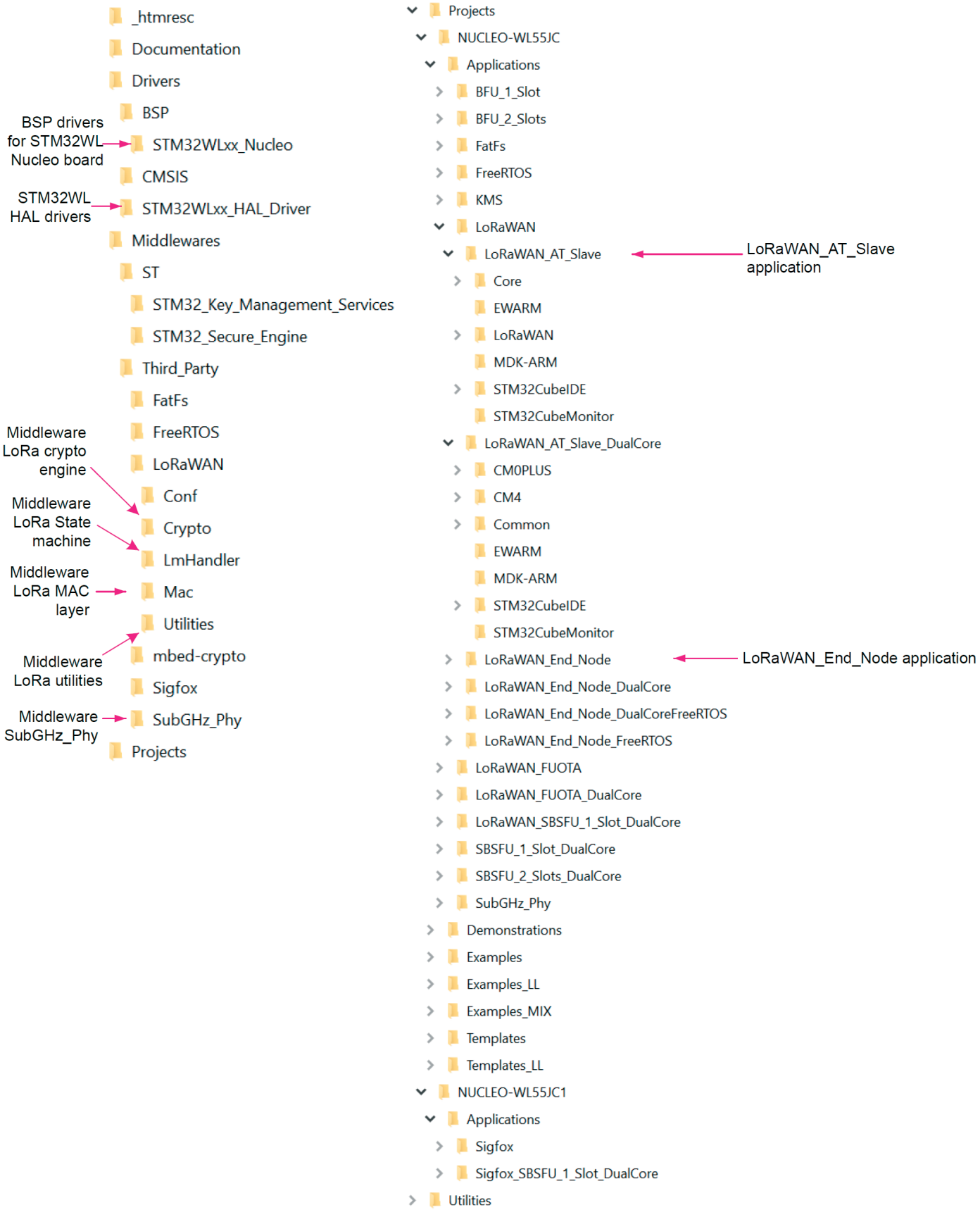
[1] LoRaWAN 1.0.3 Specification by LoRa Alliance Specification Protocol - январь 2018 г.
[2] Примечание по применению LoRaWAN AT-команды для STM32CubeWL (AN5481)
[3] Руководство пользователя Описание STM32WL HAL и низкоуровневых драйверов (UM2642)
[4] IEEE Std 802.15.4TM - 2011. Низкоскоростные беспроводные персональные сети (LR-WPAN)
[5] Примечание по применению Длинный пакет GFSK с STM32CubeWL (AN5687)
[6] Примечание по применению Руководство по интеграции SBSFU на STM32CubeWL (AN5544)
[7] Примечание по применению Как защитить LoRaWAN и Sigfox с помощью STM32CubeWL (AN5682)

Стандарт LoRa

См. Документ [1] для получения более подробной информации о рекомендациях LoRa и LoRaWAN.

2 Обзор STM32CubeWL

Прошивка пакета MCU STM32CubeWL включает следующие ресурсы:



- Пакет поддержки платы: драйверы STM32WLxx_Nucleo.
- STM32WLxx_HAL_Driver
- Промежуточное ПО:
  - LoRaWAN, содержащий:
    - ·Слой LoRaWAN

·Утилиты LoRa
·Криптовалютный движок программного обеспечения LoRa
·Конечный автомат LoRa

– Промежуточное ПО уровня SubGHz_Phy, содержащее интерфейсы радио и radio_driver.

• Приложения LoRaWAN:
– LoRaWAN_AT_Slave (одноядерный и двухъядерный)
– LoRaWAN_End_Node (SingleCore, DualCore, SingleCore с FreeRTOS и DualCore с FreeRTOS)

• Приложение SubGHz_Phy:
– SubGHz_Phy_PingPong (одноядерный и двухъядерный)
– SubGHz_Phy_Per (одноядерный)

Кроме того, это приложение обеспечивает эффективную системную интеграцию со следующим:

• секвенсор для выполнения задач в фоновом режиме и перехода в режим пониженного энергопотребления при отсутствии активности

• сервер таймера для предоставления приложению виртуальных таймеров, работающих в режиме RTC (в режимах остановки и ожидания). Для получения более подробной информации см. Раздел 8 Описание утилит.

3 SubGHz драйвер HAL

В этом разделе основное внимание уделяется SubGHz HAL (другие функции HAL, такие как таймеры или GPIO, не описаны).

SubGHz HAL находится непосредственно над периферийным радиоустройством, работающим в диапазоне менее ГГц (см. Рисунок 2. Статическая архитектура LoRa).

Драйвер SubGHz HAL основан на простой одноразовой командно-ориентированной архитектуре (без полных процессов).

Следовательно, драйвер LL не определен.

Этот драйвер HAL SubGHz состоит из следующих основных частей:

• Обработка, инициализация и конфигурация структур данных
• API инициализации
• API настройки и управления
• MSP и обратные вызовы событий
• Операция ввода-вывода шины на основе SUBGHZ_SPI (внутренние услуги)

Поскольку API HAL в основном основаны на сервисах шины для отправки команд в одноразовых операциях, не используется никакой функциональный конечный автомат, кроме состояний RESET / READY HAL.

3.1 Ресурсы SubGHz

Следующие API HAL SubGHz вызываются при инициализации радио:

• Объявите структуру дескриптора SUBGHZ_HandleTypeDef.

• Инициализируйте периферийное радиоустройство с частотой менее ГГц, вызвав API HAL_SUBGHZ_Init (& hUserSubghz).

• Инициализируйте низкоуровневые ресурсы SubGHz путем реализации API HAL_SUBGHZ_MspInit ():

- Конфигурация PWR: включение сигнала пробуждения периферийного радиоустройства с частотой менее ГГц.
- Конфигурация NVIC:
  · Включите прерывания IRQ радио NVIC.
  · Сконфигурируйте приоритет прерывания радиосвязи на суб-ГГц.

Следующее радио прерывание HAL вызывается в файле stm32wlxx_it.c:

• HAL_SUBGHZ_IRQHandler в SUBGHZ_Radio_IRQHandler.

3.2 Передача данных на SubGHz

Операция команды Set выполняется в режиме опроса с помощью API HAL_SUBGHZ_ExecSetCmd ();

Операция получения статуса выполняется в режиме опроса с API HAL_SUBGHZ_ExecGetCmd ();

Доступ к регистру чтения / записи осуществляется в режиме опроса с помощью следующих API:

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

4 BSP платы STM32WL Nucleo

Этот драйвер BSP предоставляет набор функций для управления РЧ-службами радиосвязи, такими как настройки и управление переключателем РЧ, настройки TCXO и настройки DCDC.

Примечание. Промежуточное программное обеспечение радиосвязи (SubGHz_Phy) взаимодействует с BSP радиосвязи через интерфейсный файл radio_board_if.c / h. Когда используется настраиваемая пользовательская доска, рекомендуется выполнить одно из следующих действий:

• Первый вариант
  - Скопируйте каталог BSP / STM32WLxx_Nucleo /.
  - Переименуйте и обновите пользовательские API BSP с помощью:
    · настройка и управление пользовательским радиочастотным переключателем (например, управление контактами или номер порта)
    · пользовательская конфигурация TCXO
    · пользовательская конфигурация DC / DC
  - заменить в проекте IDE файлы BSP STM32WLxx_Nucleo на файлы BSP пользователя.
• Второй вариант
  - Отключите USE_BSP_DRIVER в Core / Inc / platform.h и реализуйте функции BSP непосредственно в radio_board_if.c.

4.1 Диапазон частот

Для серии STM32WL доступны два типа плат Nucleo:

• NUCLEO-WL55JC1: высокочастотный диапазон, настроенный на частоту от 865 МГц до 930 МГц

• NUCLEO-WL55JC2: низкочастотный диапазон, настроенный на частоту от 470 МГц до 520 МГц

Если пользователь попытается запустить микропрограмму, скомпилированную на частоте 868 МГц на плате с низкочастотным диапазоном, ожидаются очень плохие радиочастотные характеристики.

Прошивка не проверяет полосу платы, на которой работает.

4.2 Радиочастотный переключатель

Плата STM32WL Nucleo включает 3-портовый радиочастотный коммутатор (SP3T) для работы с той же платой в следующих режимах:

• передача большой мощности

• передача малой мощности

• прием

Таблица 2. Радиопереключатель BSP

Функция	Описание
int32_t BSP_RADIO_Init(void)	Инициализирует переключатель RF.
BSP_RADIO_ConfigRFSwitch (BSP_RADIO_Switch_TypeDef Config)	Настраивает переключатель RF.
int32_t BSP_RADIO_DeInit (void)	Деинициализирует переключатель RF.
int32_t BSP_RADIO_GetTxConfig(void)	Возвращает конфигурацию платы: высокая мощность, низкая мощность или и то, и другое.

Состояния RF в зависимости от конфигурации переключателя приведены в таблице ниже.

Таблица 3. RF состояния в зависимости от конфигурации коммутатора

Состояние RF	FE_CTRL1	FE_CTRL2	FE_CTRL3
Передача высокой мощности	Low	High	High
Передача малой мощности	High	High	High
Прием	High	Low	High

4.3 Время пробуждения по радиочастоте

Время пробуждения на частоте ниже ГГц восстанавливается с помощью следующего API.

Таблица 4. BSP время пробуждения радиостанции

Функция	Описание
uint32_tBSP_RADIO_GetWakeUpTime(void)	Возвращает значение RF_WAKEUP_TIME

Пользователь должен запустить TCXO, установив для команды RADIO_SET_TCXOMODE время ожидания, зависящее от приложения.

Значение тайм-аута можно обновить в radio_conf.h. Значение шаблона по умолчанию следующее:

```
#define RF_WAKEUP_TIME                  1U
```

4.4 TCXO

В пользовательском приложении могут быть установлены различные типы генераторов. На платах STM32WL Nucleo используется TCXO (кварцевый генератор с температурной компенсацией) для достижения большей точности частоты.

Таблица 5. BSP TCXO радиомодуля

Функция	Описание
uint32_t BSP_RADIO_IsTCXO (void)	Возвращает значение IS_TCXO_SUPPORTED

Режим TCXO определяется STM32WL Nucleo BSP путем выбора USE_BSP_DRIVER в Core / Inc / platform.h.

Если пользователь хочет обновить это значение (плата NUCLEO не совместима) или если BSP отсутствует, режим TXCO можно обновить в radio_board_if.h. Значение шаблона по умолчанию следующее:

```
#define IS_TCXO_SUPPORTED               1U
```

4.5 Регулировка мощности

В зависимости от приложения пользователя для регулирования мощности используется LDO или SMPS (также называемый DCDC). SMPS используется на платах STM32WL Nucleo.

Таблица 6. BSP радио SMPS

Функция	Описание
uint32_t BSP_RADIO_IsDCDC (void)	Возвращает значение IS_DCDC_SUPPORTED.

Режим DCDC определяется STM32WL Nucleo BSP путем выбора USE_BSP_DRIVER в Core / Inc / platform.h.

Если пользователь хочет обновить это значение (плата NUCLEO не совместима) или если BSP отсутствует, режим DCDC можно обновить в radio_board_if.h. Значение шаблона по умолчанию определено ниже:

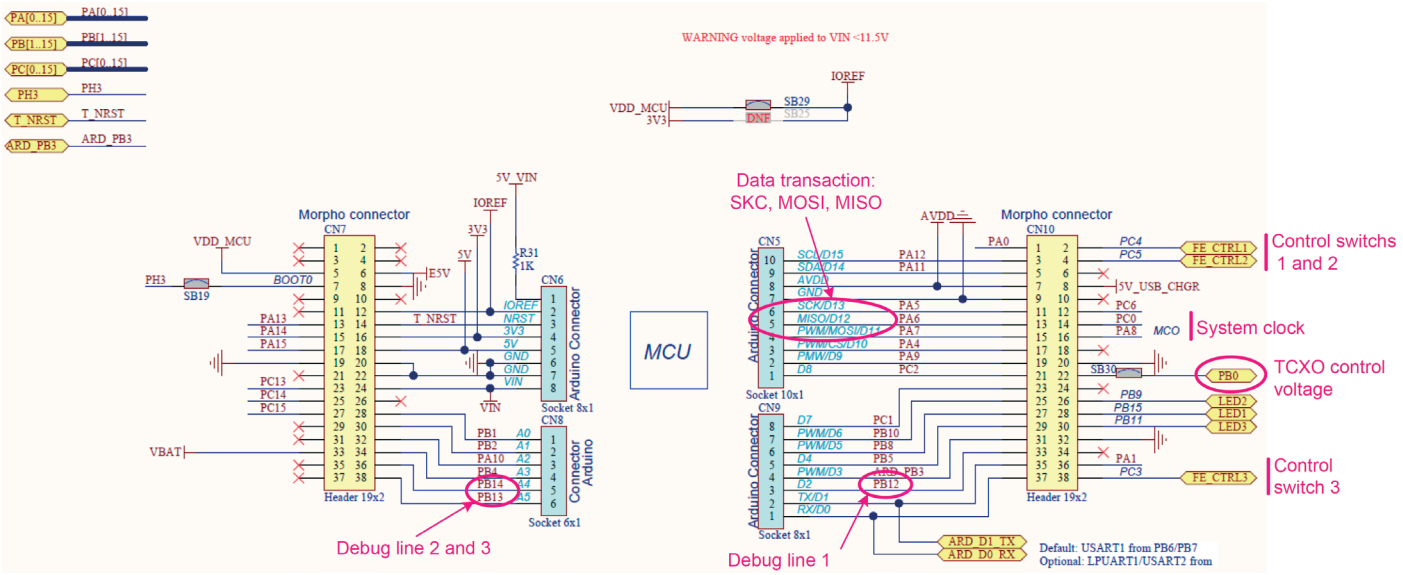
```
#define IS_DCDC_SUPPORTED               1U
```

SMPS на плате можно отключить, установив IS_DCDC_SUPPORTED = 0

4.6 Схема платы Nucleo STM32WL

На рисунке ниже представлена схема платы STM32WL Nucleo (эталонная плата MB1389) с выделением некоторых полезных сигналов:

- переключатели управления на PC4, PC5 и PC3
- вывод управляющего напряжения TCXO на PB0
- строки отладки на PB12, PB13 и PB14
- системные часы на PA8
- SCK на PA5
- MISO на PA6
- MOSI на PA7



5 Описание стека LoRaWAN

Прошивка пакета MCU STM32CubeWL включает ресурсы STM32WL, такие как:

- Драйверы STM32WLxx Nucleo
- Драйверы STM32WLxx HAL
- Промежуточное ПО LoRaWAN
- Промежуточное ПО физического уровня SubGHz
- Пример приложения LoRaWAN
- Утилиты

Промежуточное ПО стека LoRaWAN для микроконтроллеров STM32 разделено на несколько модулей.

Табл. 7. Описание стека LoRaWAN

Модуль	Описание	Расположение
LoRaMAC layer	реализует спецификацию канального уровня	Middlewares\Third_Party\LoRaWAN\Mac
Region layer	реализует спецификацию региональных параметров в качестве зависимого интерфейса для модуля уровня LoRaMAC	Middlewares\Third_Party\LoRaWAN\Mac\Region
LoRa crypto	реализует алгоритмы AES/CMAC и интерфейс с элементом SecureEngine	Middlewares\Third_Party\LoRaWAN\Crypto
LmHandler	реализует общедоступный интерфейс LoRaMac Handler, спецификации сертификации и пакеты FUOTA	Middlewares\Third_Party\LoRaWAN\LmHandler
LoRa utilities	реализует общие полезные функции	Middlewares\Third_Party\LoRaWAN\Utilities

Некоторые функции LoRaWAN реализованы в соответствии со спецификациями протокола LoRa Allicance:

- Из спецификации канального уровня:

– Встроенный стек протоколов LoRaWAN класса A, класса B и класса C

– Активация на конечном устройстве либо через OTAA, либо через активацию посредством персонализации (ABP).

– Адаптивная поддержка скорости передачи данных

- Из спецификации региональных параметров:

– Диапазон ISM 868 МГц EC, совместимый с ETSI

– Диапазон ISM 433 МГц EC, совместимый с ETSI
– Американский диапазон ISM 915 МГц, соответствующий требованиям FCC
– Диапазон KR 920 МГц ISM определен правительством Кореи
– RU 864 МГц ISM диапазон, определенный российским законодательством
– диапазоны ISM CN 779 МГц и CN470 МГц, определенные правительством Китая
– Диапазон ISM AS 923 МГц, определенный правительствами Азии
– Диапазон ISM AU 915 МГц, определенный правительством Австралии
– В диапазоне ISM 865 МГц, определенном правительством Индии.

Кроме того, стек LoRaWAN объединяет:

• Решение по сертификации в соответствии со спецификациями, описанными ниже.

• Управление контекстом NVM для предотвращения потери контекста при отключении питания.

• Интеграция с низким энергопотреблением в режимах ожидания/сна

5.1 Версия спецификаций LoRaWAN

Спецификация канального уровня и спецификации региональных параметров определяются LoRa-Alliance.

Стек LoRaWAN реализует 2 разные версии, основанные на поставках стека Semtech:

• Спецификация уровня канала LoRaWAN 1.0.3 + Спецификация региональных параметров LoRaWAN 1.0.3

• Спецификация канального уровня LoRaWAN 1.0.4 (TS001-1.0.4) + региональные параметры LoRaWAN 2-1.0.1

Спецификация (RP002-1.0.1)

Обязателен выбор адаптированной версии стека с ожидаемой конфигурацией LoRaWAN Server.

5.2 Сертификация LoRaWAN

Система, включающая плату NUCLEO-WL55JC и модемное приложение STM32CubeWL, была проверена LoRaWAN TestHouse и прошла сертификацию для диапазонов EU868, IN865, KR920, AS923 и US915.

Реализация сертификации LoRaWAN зависит от используемой версии спецификации LoRaWAN:

Таблица 8. Сертификация LoRaWAN

Версия спецификации LoRaWAN	Спецификация сертификации LoRaWAN
Спецификация канального уровня LoRaWAN 1.0.3	Требования к сертификации конечных устройств LoRa Alliance для устройств AS923MHz ISM Band v1.1.0
	Требования к сертификации оконечных устройств LoRa Alliance для устройств EU863-870 MHz ISM Band v1.6.0
	Требования к сертификации конечных устройств LoRa Alliance для Индии, диапазон ISM 865–867 МГц, версия 1.1.0
	Требования к сертификации конечных устройств LoRa Alliance для устройств диапазона ISM 920–923 МГц в Южной Корее v1.2.0
	Требования к сертификации конечных устройств LoRa Alliance для США и Канады, диапазон ISM 902–928 МГц v1.5.0

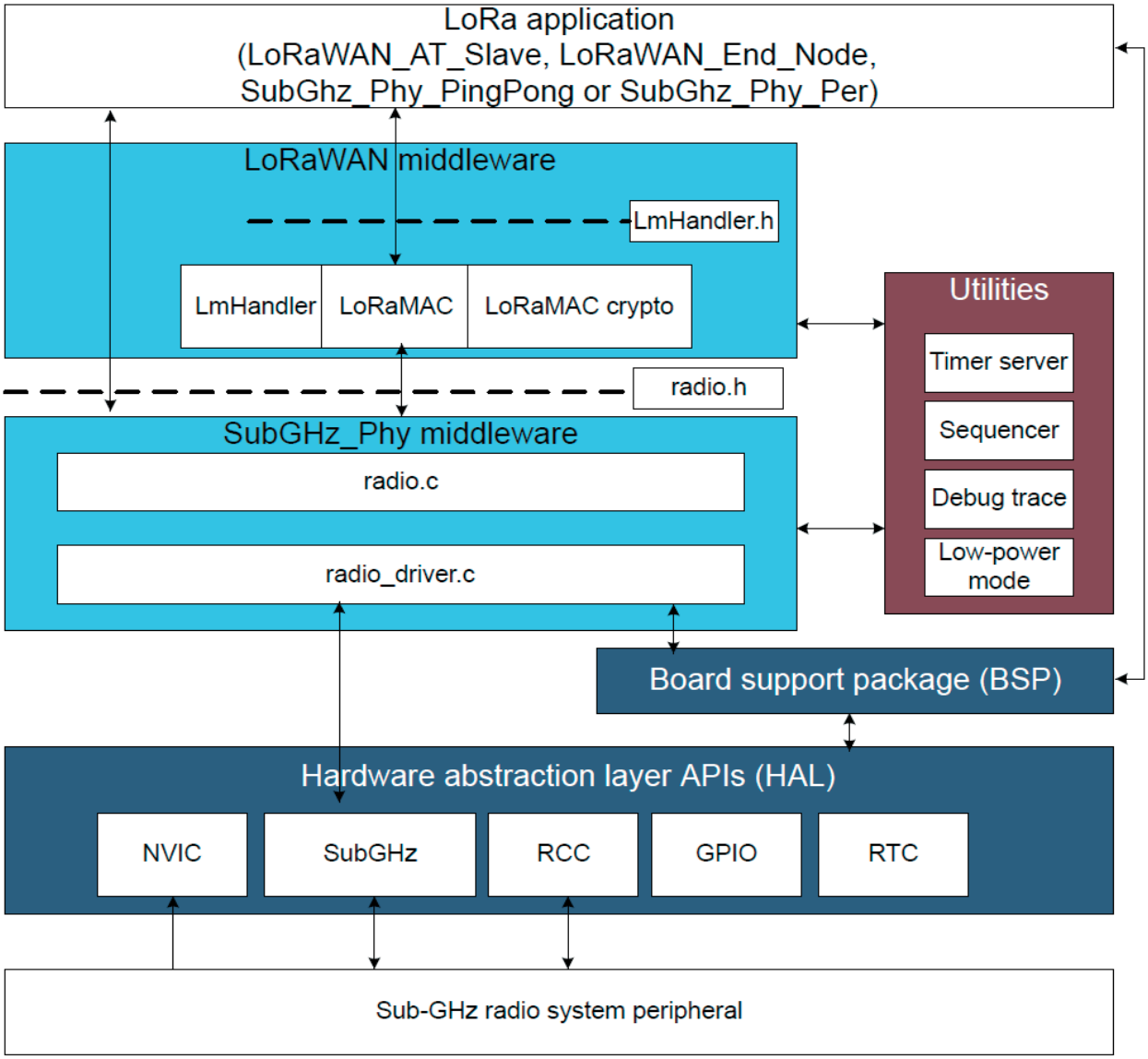
Спецификация канального уровня LoRaWAN 1.0.4 (TS001-1.0.4)	LoRaWAN 1.0.4 Требования к сертификации конечных устройств для всех регионов
	Спецификация протокола сертификации LoRaWAN 1.0.0 (TS009-1.0.0)

5.3 Архитектура

5.3.1 Статический вид

На рисунке ниже описан основной дизайн прошивки для приложения LoRa.

Figure 2. Static LoRa architecture



HAL использует API-интерфейсы STM32Cube для управления оборудованием MCU, необходимым для приложения. В промежуточное ПО LoRa включено только определенное оборудование, поскольку оно является обязательным для запуска приложения LoRa.

RTC обеспечивает централизованную единицу времени, которая продолжает работать даже в режиме пониженного энергопотребления (режим Stop 2). Аварийный сигнал RTC используется для пробуждения системы в определенное время, управляемое сервером таймера.

Промежуточное ПО SubGHz_Phy использует HAL SubGHz для управления радио (см. Рисунок выше). Для получения дополнительной информации обратитесь к Разделу 5. MAC управляет SubGHz_Phy, используя модель 802.15.4.

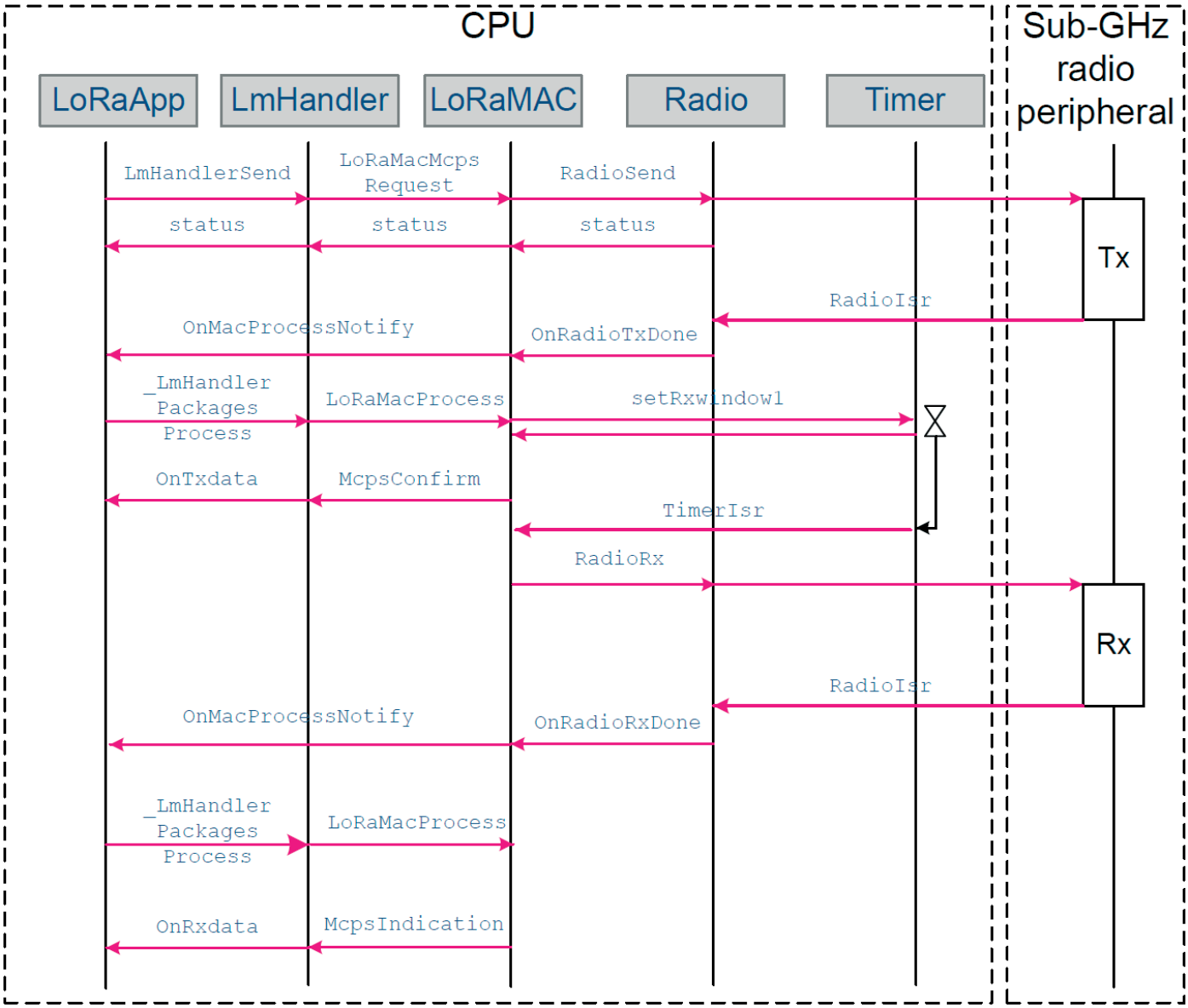
MAC взаимодействует с драйвером SubGHz_Phy и использует сервер таймера для добавления или удаления синхронизированных задач.

Поскольку конечный автомат, который управляет LoRa класса A, является чувствительным, промежуточный уровень программного обеспечения вставлен (LmHandler.c) между MAC и приложением (см. Драйвер LoRaMAC на приведенном выше рисунке). Имея ограниченный набор API, пользователь может реализовать конечный автомат класса A на уровне приложения. Для получения более подробной информации обратитесь к Разделу 6.

Приложение, построенное на основе бесконечного цикла, управляет режимом с низким энергопотреблением, запускает обработчики прерываний (сигнализация или GPIO) и вызывает LoRa класса A, если какая-либо задача должна быть выполнена.

5.3.2 Динамический просмотр

Figure 3. Class A Tx and Rx processing MSC



MSC (диаграмма последовательности сообщений), показанная на рисунке ниже, изображает устройство класса A, передающее данные приложения и получающее данные приложения с сервера.

Как только радио завершит передачу данных приложения, асинхронный RadioIRQ будит систему. RadioIsr здесь вызывает txDone в режиме обработчика.

Все таймеры RadioIsr и MAC вызывают обратный вызов LoRaMacProcessNotify, чтобы запросить уровень приложения для обновления состояния LoRaMAC и выполнения дальнейшей обработки, когда это необходимо.

Например, в конце приема rxDone вызывается в ISR (обработчике), но вся обработка пакетов Rx, включая дешифрование, не должна обрабатываться в ISR. Этот случай является примером последовательности вызовов. Если в окно Rx1 не поступают данные, запускается другое окно Rx2.

5.3.3 Необходимые периферийные устройства STM32 для управления радио

Sub-GHz радио

Доступ к периферийному радиоустройству суб-ГГц осуществляется через HAL stm32wlxx_hal_subghz.

Радиочастота субгигагерца выдает прерывание через SUBGHZ_Radio_IRQHandler NVIC, чтобы уведомить о событии TxDone или RxDone. Другие события перечислены в справочном руководстве по продукту.

RTC

Календарь RTC (часы реального времени) используется как 32-битный счетчик, работающий во всех режимах питания от внешнего генератора 32 кГц. По умолчанию RTC запрограммирован на 1024 тика (субсекунды) в секунду. RTC программируется один раз при аппаратной инициализации (когда MCU запускается в первый раз). Выход RTC ограничен 32-битным таймером, что соответствует примерно 48-дневному периоду.

Внимание: при изменении длительности тика пользователь должен удерживать ее менее 1 мс.

6 Описание промежуточного программного обеспечения LoRaWAN

6.1 Инициализация промежуточного программного обеспечения LoRaWAN

Инициализация уровня LoRaMAC выполняется через API LoRaMacInitialization, который инициализирует как время выполнения преамбулы уровня LoRaMAC, так и примитивы обратного вызова служб MCPS и MLME (см. таблицу ниже).

Табл. 9. Инициализация промежуточного ПО LoRaWAN

Функция	Описание
LoRaMacStatus_t LoRaMacInitialization (LoRAMacPrimitives_t *primitives, LoRaMacCallback_t *callback, LoRaMacRegion_t region)	Инициализирует модуль уровня LoRaMAC (см. Раздел 6.3 Обратные вызовы уровня MAC промежуточного слоя)

6.2 API уровня MAC промежуточного слоя

Предоставляемые API соответствуют определению «примитивов», определенному в IEEE802.15.4-2011 (см. Документ [4]). Взаимодействие с LoRaMAC осуществляется через архитектуру запрос-подтверждение и указание-ответ. Прикладной уровень может выполнять запрос, который уровень LoRaMAC подтверждает примитивом подтверждения. И наоборот, уровень LoRaMAC уведомляет прикладной уровень с помощью примитива индикации в случае любого события. Прикладной уровень может ответить на указание примитивом ответа. Следовательно, все подтверждения или индикации реализуются с помощью обратных вызовов.

Уровень LoRaMAC предоставляет следующие услуги:

- Услуги **MCPS**

Как правило, уровень LoRaMAC использует службы MCPS для передачи и приема данных.

Таблица 10. Услуги MCPS

Функция	Описание
LoRaMacStatus_t LoRaMacMcpsRequest (McpsReq_t* mcpsRequest, bool allowDelayedTx)	Запросы на отправку данных Tx .

- Услуги **MLME**

Уровень LoRaMAC использует службы MLME для управления сетью LoRaWAN.

Таблица 11. Услуги MMLE

Функция	Описание
LoRaMacStatus_t LoRaMacMlmeRequest (MlmeReq_t *mlmeRequest)	Создает запрос на присоединение или запросы на проверку ссылки

- Услуги **MIB**

MIB хранит важную информацию времени выполнения (такую как MIB_NETWORK_ACTIVATION или MIB_NET_ID) и содержит конфигурацию уровня LoRaMAC (например, MIB_ADR, MIB_APP_KEY).

Таблица 12. Услуги MIB

Функция	Описание
LoRaMacStatus_t LoRaMacMibSetRequestConfirm (MibRequestConfirm_t *mibSet)	Устанавливает атрибуты слоя LoRaMAC.
LoRaMacStatus_t LoRaMacMibGetRequestConfirm (MibRequestConfirm_t *mibGet)	Получает атрибуты слоя LoRaMAC.

6.3 Обратные вызовы уровня MAC промежуточного слоя

Примитивы функций пользовательских событий LoRaMAC (также называемые обратными вызовами), которые должны быть реализованы приложением, следующие:

Таблица 13. Примитив MCPS

Функция	Описание
void (*MacMcpsConfirm) (McpsConfirm_t *McpsConfirm)	Ответ на McpsRequest
Void (*MacMcpsIndication) (McpsIndication_t* McpsIndication, LoRaMacRxStatus_t* RxStatus)	Уведомляет приложение о том, что полученный пакет доступен.
void (*MacMlmeConfirm) (MlmeConfirm_t *MlmeConfirm)	Управляет сетью LoRaWAN.
void (*MacMlmeIndication) (MlmeIndication_t* MlmeIndication, LoRaMacRxStatus_t* RxStatus)	Уведомить уровень MAC о наличии ответа MAC.

6.4 Таймеры уровня MAC промежуточного слоя

Таблица 14. События MAC-таймера

Функция	Описание
void OnRxWindow1TimerEvent (void* context)	Выполняется RxWindow1Delay при первом событии таймера окна Rx. Обычный кадр: RxWindowXDelay = ReceiveDelayX — RADIO_WAKEUP_TIME Кадр соединения: RxWindowXDelay = JoinAcceptDelayX — RADIO_WAKEUP_TIME
void OnRxWindow2TimerEvent (void* context)	Выполняется RxWindow2Delay при первом событии таймера окна Rx.
void OnTxDelayedTimerEvent (void* context)	Выполняется по событию таймера Tx с задержкой рабочего цикла. (Устанавливает таймер для передачи кадра Tx.)
void OnAckTimeoutTimerEvent (void* context)	Выполняется по событию таймера AckTimeout таймером тайм-аута подтверждения. Используется для ретрансляции пакетов (Устанавливает тайм-аут для подтверждения приема кадра.) (только для версии LoRaWAN v1.0.3).
void OnRetransmitTimeoutTimerEvent (void* context)	Выполняется по событию таймера AckTimeout таймером тайм-аута подтверждения. Используется для ретрансляции пакетов (только для версии LoRaWAN v1.0.4).

6.5 Функция приложения Middleware LmHandler

Интерфейс к MAC осуществляется через файл LoRaMac.h интерфейса MAC в одном из следующих режимов:

• Стандартный режим

Предоставляется интерфейсный файл (драйвер LoRaMAC, см. Рис. 2), позволяющий пользователю начать работу, не беспокоясь о конечном автомате LoRa. Этот файл находится в

Middlewares \ Third_Party \ LoRaWAN \ LmHandler \ LmHandler.c и реализует:

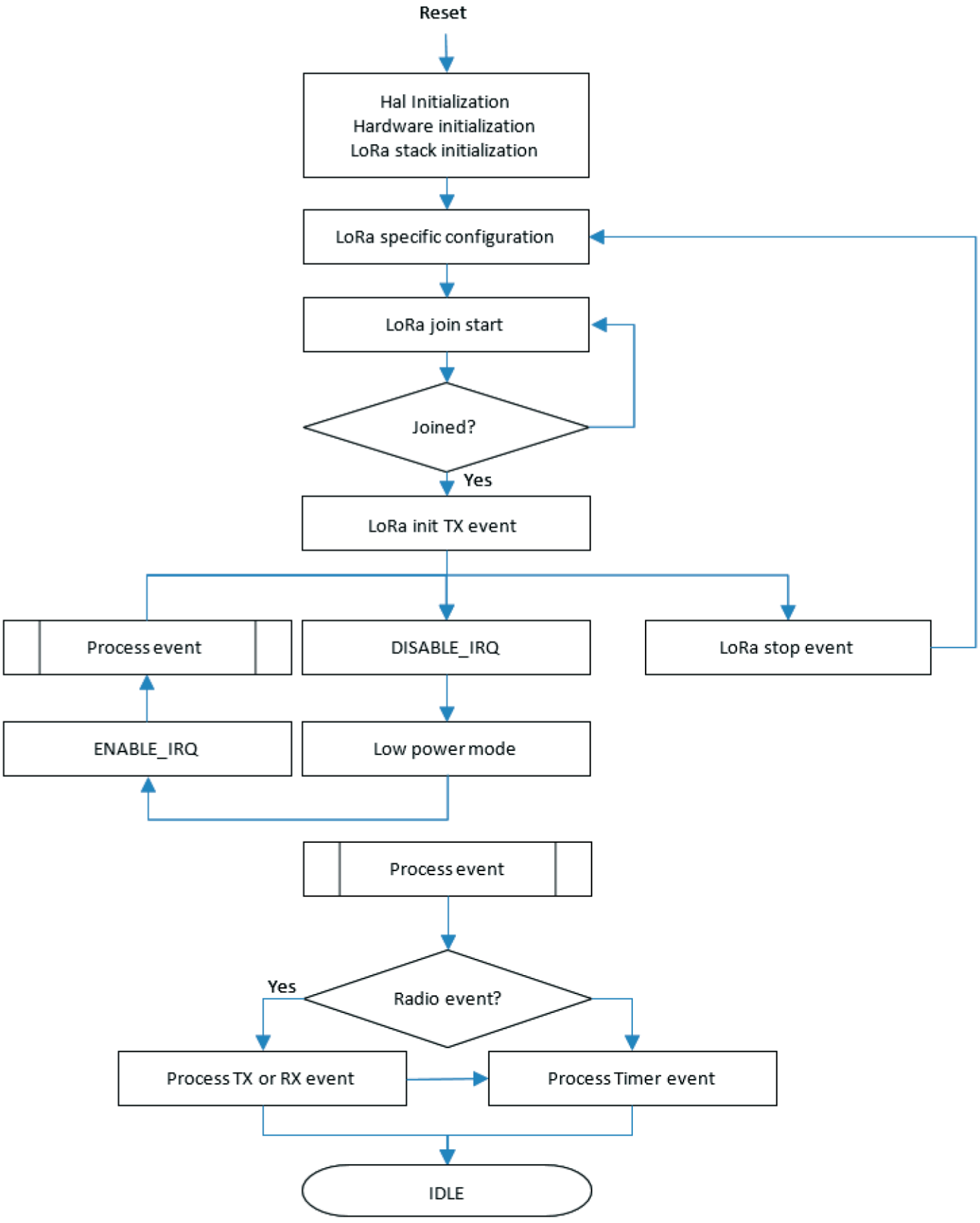
- набор API для доступа к сервисам LoRaMAC

- тестовые примеры сертификации LoRa, которые не видны на уровне приложения

• Расширенный режим

Пользователь получает прямой доступ к уровню MAC, включая MAC в файл пользователя.

Figure 5. Operation model



6.5.1 Операционная модель

Операционная модель, предложенная для LoRaWAN_End_Node, основана на парадигме «управляемой событиями», включая «управляемую временем» (см. Рисунок ниже). Поведение системы LoRa запускается либо событием таймера, либо событием радио, а также защитным переходом.

В следующих разделах подробно описаны API-интерфейсы LoRaWAN_End_Node и LoRaWAN_AT_Slave, используемые для доступа к службам LoRaMAC. Соответствующие файлы интерфейса находятся в

Middlewares \ Third_Party \ LoRaWAN \ LmHandler \ LmHandler.c

Пользователь должен реализовать приложение с этими API.

Пример приложения LoRaWAN_End_Node приведен в

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

Пример приложения LoRaWAN_AT_Slave представлен в

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

6.5.2 Определение основных функций приложения

Таблица 15. Основные функции LmHandler

Функция	Описание
LmHandlerErrorStatus_t LmHandlerInit (LmHandlerCallbacks_t *handlerCallbacks, uint32_t fwVersion)	Инициализация конечного автомата LoRa
LmHandlerErrorStatus_t LmHandlerDeInit(void)	Деинициализировать конечный автомат LoRa, остановить все таймеры, сбросить параметры MAC, отключить радио и удалить все существующие ссылки на обратные вызовы.
LmHandlerErrorStatus_t LmHandlerConfigure (LmHandlerParams_t *handlerParams)	Конфигурация всех параметров приложения
void LmHandlerJoin (ActivationType_t mode, bool forceRejoin)	Запрос на присоединение к сети в режиме OTAA или ABP. "forceRejoin" - флаг для принудительного повторного присоединения, даже если контекст LoRaWAN может быть восстановлен.
LmHandlerFlagStatus_t LmHandlerJoinStatus (void)	Проверяет, подключено ли Устройство к сети
void LmHandlerStop (void)	Останавливает процесс LoRa и ожидает новой конфигурации перед повторным присоединением.
LmHandlerErrorStatus_t LmHandlerHalt (void)	Останавливает стек LoRa с прерыванием текущего процесса
LmHandlerErrorStatus LmHandlerRequestClass (DeviceClass_t newClass)	Запрашивает уровень MAC для изменения класса LoRaWAN.

LmHandlerErrorStatus_t LmHandlerSend (LmHandlerAppData_t *appData, LmHandlerMsgTypes_t isTxConfirmed, bool allowDelayedTx)	Отправляет кадр восходящей линии связи. Этот кадр может быть либо неподтвержденным пустым кадром, либо неподтвержденным/подтвержденным кадром полезной нагрузки.
TimerTime_t LmHandlerGetDutyCycleWaitTime (void)	Получает текущее время ожидания рабочего цикла
LmHandlerErrorStatus_t LmHandlerGetVersion (LmHandlerVersionType_t lmhType, uint32_t *featureVersion)	Возвращает текущую версию спецификации LoRaWAN
LmHandlerErrorStatus_t LmHandlerNvmDataStore (void)	Запускает процесс хранилища данных NVM (подробнее см. в разделе 13 «Описание управления контекстом LoRaWAN»).

6.6 Обратные вызовы приложений

Обратные вызовы в таблицах ниже используются как для приложений LoRaWAN_End_Node, так и для приложений LoRaWAN_AT_Slave.

Таблица 16. Описание структуры обратного вызова LmHandlerCallbacks_t

Функция	Описание
uint8_t GetBatteryLevel (void)	*Получает уровень заряда батареи.*
int16_t GetTemperature (void)	*Получает текущую температуру (в °C) устройства в формате q7.8.*
*void GetUniqueId (uint8_t *id)*	*Получает 64-битный уникальный идентификатор платы.*
uint32 GetDevAddr (void)	*Получает 32-битный уникальный идентификатор платы (LSB).*
*void OnRestoreContextRequest (void *nvm, uint32_t nvm_size)*	*Восстанавливает контекст данных NVM из флэш-памяти.*
*void OnStoreContextRequest (void *nvm, uint32_t nvm_size)*	*Сохраняет контекст данных NVM во флэш-памяти.*
void OnMacProcess (void)	*Вызывает процесс LmHandler при получении IRQ радио.*
void OnNvmDataChange (LmHandlerNvmContextStates_t state)	*Уведомляет верхний уровень об изменении контекста NVM.*
*void OnNetworkParametersChange (CommissioningParams_t *params)*	*Уведомляет верхний уровень о том, что сетевые параметры установлены.*
*void OnJoinRequest (LmHandlerJoinParams_t *params)*	*Уведомляет верхний уровень о присоединении к сети.*
*void OnTxData (LmHandlerTxParams_t *params)*	*Уведомляет верхний уровень о том, что кадр был передан.*
*void OnRxData (LmHandlerAppData_t *appData, LmHandlerRxParams_t *params)*	*Уведомляет верхний уровень о получении аппликативного кадра.*

void OnClassChange (DeviceClass_t deviceClass)	*Подтверждает изменение класса устройства LoRaWAN.*
*void OnBeaconStatusChange (LmHandlerBeaconParams_t *params)*	*Уведомляет верхний уровень об изменении состояния маяка.*
*void OnBeaconStatusChange (LmHandlerBeaconParams_t *params)*	*Уведомляет верхний уровень об изменении состояния маяка.*
void OnSysTimeUpdate (void)	*Уведомляет верхний уровень об обновлении системного времени.*
void OnTxPeriodicityChanged (uint32_t periodicity)	*Вызывается для изменения периода аппликативного кадра Tx. Обратные вызовы протокола проверки соответствия, используемые, когда определены TS001-1.0.4 + TS009 1.0.0.*
void OnTxFrameCtrlChanged (LmHandlerMsgTypes_t isTxConfirmed)	*Вызывается для изменения аппликативного управления фреймом Tx. Обратные вызовы протокола проверки соответствия, используемые, когда определены TS001-1.0.4 + TS009 1.0.0.*
void OnPingSlotPeriodicityChanged (uint8_t pingSlotPeriodicity)	*Вызывается для изменения периода пинга. Обратные вызовы протокола проверки соответствия, используемые, когда определены TS001-1.0.4 + TS009 1.0.0.*
void OnSystemReset(void)	*Вызывается для сброса системы. Обратные вызовы протокола проверки соответствия, используемые, когда определены TS001-1.0.4 + TS009 1.0.0.*

6.7 Расширенные функции приложения

Эти обратные вызовы используются как для приложений LoRaWAN_End-Node, так и для приложений LoRaWAN_AT-Slave.

(См. таблицу 17 на обороте).

Таблица 17. Функции получения / установки

Функция	Описание
LmHandlerErrorStatus_t LmHandlerGetCurrentClass (DeviceClass_t *deviceClass)	Получает текущий класс LoRaWAN.
LmHandlerErrorStatus_t LmHandlerGetDevEUI (uint8_t *devEUI)	Получает EUI устройства LoRaWAN.
LmHandlerErrorStatus_t LmHandlerSetDevEUI (uint8_t *devEUI)	Устанавливает EUI устройства LoRaWAN (если OTAA).
LmHandlerErrorStatus_t LmHandlerGetAppEUI (uint8_t *appEUI)	Получает EUI для приложения LoRaWAN.
LmHandlerErrorStatus_t LmHandlerSetAppEUI (uint8_t *appEUI)	Устанавливает EUI для приложения LoRaWAN.
LmHandlerErrorStatus_t LmHandlerGetNetworkID (uint32_t *networkId)	Получает идентификатор сети LoRaWAN.
LmHandlerErrorStatus_t LmHandlerSetNetworkID uint32_t networkId)	Устанавливает идентификатор сети LoRaWAN.
LmHandlerErrorStatus_t LmHandlerGetDevAddr (uint32_t *devAddr)	Получает адрес устройства LoRaWAN.
LmHandlerErrorStatus_t LmHandlerSetDevAddr (uint32_t devAddr)	Устанавливает адрес устройства LoRaWAN (если ABP).
LmHandlerErrorStatus_t LmHandlerGetAppKey (uint8_t *appKey)	Получает корневой ключ приложения LoRaWAN.
LmHandlerErrorStatus_t LmHandlerSetAppKey (uint8_t *appKey)	Устанавливает корневой ключ приложения LoRaWAN.
LmHandlerErrorStatus_t LmHandlerGetNwkKey (uint8_t *nwkKey)	Получает корневой ключ сети LoRaWAN.
LmHandlerErrorStatus_t LmHandlerSetNwkKey (uint8_t *nwkKey)	Устанавливает корневой ключ сети LoRaWAN.
LmHandlerErrorStatus_t LmHandlerGetNwkSKey (uint8_t *nwkSKey)	Получает сетевой сеансовый ключ LoRaWAN.
LmHandlerErrorStatus_t LmHandlerSetNwkSKey (uint8_t *nwkSKey)	Устанавливает ключ сеанса сети LoRaWAN.
LmHandlerErrorStatus_t LmHandlerGetAppSKey (uint8_t *appSKey)	Получает ключ сеанса приложения LoRaWAN.
LmHandlerErrorStatus_t LmHandlerSetAppSKey (uint8_t *appSKey)	Устанавливает ключ сеанса приложения LoRaWAN.
LmHandlerErrorStatus_t LmHandlerGetActiveRegion (LoRaMacRegion_t *region)	Получает активную область.
LmHandlerErrorStatus_t LmHandlerSetActiveRegion (LoRaMacRegion_t region)	Устанавливает активную область.
LmHandlerErrorStatus_t LmHandlerGetAdrEnable (bool *adrEnable)	Получает состояние адаптивной скорости передачи данных.
LmHandlerErrorStatus_t LmHandlerSetAdrEnable (bool adrEnable)	Устанавливает состояние адаптивной скорости передачи данных.

LmHandlerErrorStatus_t LmHandlerGetTxDatarate (int8_t *txDatarate)	Получает текущую скорость передачи данных.
LmHandlerErrorStatus_t LmHandlerSetTxDatarate (int8_t txDatarate)	Устанавливает скорость передачи данных (если адаптивный DR отключен)
LmHandlerErrorStatus_t LmHandlerGetDutyCycleEnable (bool *dutyCycleEnable)	Получает текущее состояние рабочего цикла Tx.
LmHandlerErrorStatus_t LmHandlerSetDutyCycleEnable (bool dutyCycleEnable)	Устанавливает состояние рабочего цикла Tx.
LmHandlerErrorStatus_t LmHandlerGetRX2Params (RxChannelParams_t *rxParams)	Получает текущую скорость передачи данных и частоту Rx2.
LmHandlerErrorStatus_t LmHandlerSetRX2Params (RxChannelParams_t *rxParams)	Устанавливает скорость передачи данных и частоту Rx2.
LmHandlerErrorStatus_t LmHandlerGetTxPower (int8_t *txPower)	Получает текущее значение мощности Tx.
LmHandlerErrorStatus_t LmHandlerSetTxPower (int8_t txPower)	Устанавливает значение мощности Tx.
LmHandlerErrorStatus_t LmHandlerGetRx1Delay (uint32_t *rxDelay)	Получает текущую задержку Rx1 (после окна Tx).
LmHandlerErrorStatus_t LmHandlerSetRx1Delay (uint32_t rxDelay)	Устанавливает задержку Rx1 (после окна Tx).
LmHandlerErrorStatus_t LmHandlerGetRx2Delay (uint32_t *rxDelay)	Получает текущую задержку Rx2 (после окна Tx).
LmHandlerErrorStatus_t LmHandlerSetRx2Delay (uint32_t rxDelay)	Устанавливает задержку Rx2 (после окна Tx).
LmHandlerErrorStatus_t LmHandlerGetJoinRx1Delay (uint32_t *rxDelay)	Получает текущую задержку соединения Rx1 (после окна Tx).
LmHandlerErrorStatus_t LmHandlerSetJoinRx1Delay (uint32_t rxDelay)	Устанавливает задержку соединения Rx1 (после окна Tx).
LmHandlerErrorStatus_t LmHandlerGetJoinRx2Delay (uint32_t *rxDelay)	Получить текущую задержку соединения Rx2 (после окна Tx)
LmHandlerErrorStatus_t LmHandlerSetJoinRx2Delay (uint32_t rxDelay)	Устанавливает задержку соединения Rx2 (после окна Tx).
LmHandlerErrorStatus_t LmHandlerGetPingPeriodicity (uint8_t pingPeriodicity)	Получает текущую периодичность Rx Ping Slot (если LORAMAC_CLASSB_ENABLED)
LmHandlerErrorStatus_t LmHandlerSetPingPeriodicity (uint8_t pingPeriodicity)	Устанавливает периодичность слота Rx Ping (если LORAMAC_CLASSB_ENABLED)
LmHandlerErrorStatus_t LmHandlerGetBeaconState (BeaconState_t *beaconState)	Получает состояние маяка (если LORAMAC_CLASSB_ENABLED)
LmHandlerErrorStatus_t LmHandlerDeviceTimeReq (void)	Запрашивает обновление времени сетевого сервера.
LmHandlerErrorStatus_t LmHandlerLinkCheckReq (void)	Запросы на проверку подключения ссылки
LmHandlerErrorStatus_t LmHandlerPingSlotReq(uint8_t periodicity)	Информирует сервер о периодичности использования ping-слот.

7 Описание промежуточного программного обеспечения уровня SubGHz_Phy

Уровень радиоабстракции состоит из двух уровней:

• уровень высокого уровня (radio.c)

Он обеспечивает высокоуровневый радиоинтерфейс для промежуточного программного обеспечения стека. Он также поддерживает состояние радио, обрабатывает прерывания и управляет таймаутами. Он записывает обратные вызовы и вызывает их при возникновении радио событий.

• низкоуровневые радиодрайверы

Это уровень абстракции для интерфейса RF. Этот уровень знает имя и структуру регистра, а также подробную последовательность. Он не знает об аппаратном интерфейсе.

Промежуточное программное обеспечение уровня SubGHz_Phy содержит уровень абстракции радио, который взаимодействует непосредственно поверх аппаратного интерфейса, предоставляемого BSP (см. Раздел 4).

Каталог промежуточного программного обеспечения SubGHz_Phy разделен на две части:

• radio.c: содержит набор всех общих обратных вызовов радио, вызывающих функции radio_driver. Этот набор API должен быть общим и идентичным для всех радиостанций.

• radio_driver.c: низкоуровневые радиодрайверы

radio_conf.h содержит конфигурацию радио-приложения,
такую как RF_WAKEUP_TIME, динамические настройки DCDC, XTAL_FREQ.

7.1 Структура радиодрайвера промежуточного программного обеспечения

Общая структура радио (struct Radio_s Radio {};) определена для регистрации всех обратных вызовов с полями, подробно описанными в таблице ниже.

Таблица 18. Обратные вызовы структуры Radio_s

Callback	Описание
RadioInit	Инициализирует радио.
RadioGetStatus	Возвращает текущий статус радио.
RadioSetModem	Настраивает радио с данным модемом.
RadioSetChannel	Устанавливает частоту канала.
RadioIsChannelFree	Проверяет, свободен ли канал в данное время.
RadioRandom	Создает 32-битное случайное значение на основе показаний RSSI.
RadioSetRxConfig	Устанавливает параметры приема.
RadioSetTxConfig	Устанавливает параметры передачи.
RadioCheckRfFrequenc	Проверяет, поддерживается ли данная частота RF оборудованием.
RadioTimeOnAir	Вычисляет время передачи пакета в эфире (в мс) для заданной полезной нагрузки.
RadioSend	Подготавливает пакет к отправке и запускает передачу по радио.
RadioSleep	Переводит радио в спящий режим.

RadioStandby	Переводит радио в режим ожидания.
RadioRx	Устанавливает радио в режим приема на заданное время.
RadioStartCad	Запускает CAD (обнаружение активности канала).
RadioSetTxContinuousWave	Устанавливает радио в режим непрерывной передачи.
RadioRssi	Считывает текущее значение RSSI.
RadioWrite	Записывает в радиорегистратор по указанному адресу.
RadioRead	Читает радиорегистратор по указанному адресу.
RadioSetMaxPayloadLength	Устанавливает максимальную длину полезной нагрузки.
RadioSetPublicNetwork	Устанавливает общедоступную или частную сеть и обновляет байт синхронизации.
RadioGetWakeUpTime	Получает время, необходимое радиостанции для выхода из спящего режима.
RadioIrqProcess	Обрабатывает радио IRQ.
RadioRxBoosted	Устанавливает радиостанцию в режим приема с максимальным усилением LNA на заданное время.
RadioSetRxDutyCycle	Устанавливает параметры управления рабочим циклом приемника.
RadioTxPrbs	Устанавливает передатчик в непрерывный режим PRBS.
RadioTxCw	Устанавливает передатчик в режим непрерывной немодулированной несущей.

7.2 IRQ прерываний радиосвязи

Возможные источники прерываний радиосвязи в диапазоне частот ниже ГГц подробно описаны в таблице ниже.

Таблица 19. Битовое отображение и определение IRQ радиосвязи

Bit	Источник	Описание	Тип пакета	Операция
0	txDone	Передача пакета завершена	LoRa and GFSK	Tx
1	rxDone	Прием пакета завершен		Rx
2	PreambleDetected	Обнаружена преамбула		
3	SyncDetected	Слово синхронизации допустимо	GFSK	
4	HeaderValid	Заголовок действителен	LoRa	
5	HeaderErr	Ошибка заголовка		
6	Err	Преамбула, синхронизирующее слово, адрес, CRC или ошибка длины	GFSK	
	CrcErr	Ошибка CRC	LoRa	
7	CadDone	Обнаружение активности канала завершено		CAD
8	CadDetected	Обнаружена активность канала		
9	Timeout	Таймаут Rx или Tx	LoRa and GFSK	Rx and Tx

Дополнительные сведения см. в reference manual.

8 Описание утилит

Утилиты находятся в каталоге \ Utilities.

Основные API описаны ниже. Вторичные API и дополнительную информацию можно найти в файлах заголовков, связанных с драйверами.

8.1 Sequencer (Секвенсор)

Секвенсор обеспечивает надежную и простую структуру для выполнения задач в фоновом режиме и переходит в режим пониженного энергопотребления, когда больше нет активности. В секвенсоре реализован механизм предотвращения состояния гонки. Вдобавок, секвенсор предоставляет функцию события, позволяющую любой функции ожидать события (где конкретное событие устанавливается прерыванием), а количество операций в секунду и мощность легко сохраняются в любом приложении, которое реализует команду «выполнить до завершения».

Файл utilities_def.h, расположенный в подпапке проекта, используется для настройки идентификаторов задач и событий. Уже перечисленные нельзя удалять.

Секвенсор - это не ОС. Любая задача выполняется до завершения и не может переключиться на другую задачу, как это может сделать RTOS по тику RTOS, если только задача не приостанавливает себя, вызывая UTIL_SEQ_WaitEvt. Причем используется один стек с одной памятью. Секвенсор представляет собой усовершенствованный «цикл while», централизующий флаги задач и событий.

Секвенсор предоставляет следующие возможности:

- Усовершенствованная и упакованная система цикла while
- Поддержка до 32 задач и 32 событий
- Регистрация и выполнение задач
- Ожидающее событие и установленное событие
- Установка приоритета задачи
- Безопасный вход с низким энергопотреблением в условиях гонки

Чтобы использовать секвенсор, приложение должно выполнить следующее:

- Установите максимальное количество поддерживаемых функций, задав значение для UTIL_SEQ_CONF_TASK_NBR.
- Зарегистрируйте функцию, которая будет поддерживаться секвенсором, с помощью UTIL_SEQ_RegTask ().
- Запустите секвенсор, вызвав UTIL_SEQ_Run (), чтобы запустить цикл while в фоновом режиме.
- Вызовите UTIL_SEQ_SetTask (), когда функция должна быть выполнена.

Утилита секвенсора находится в папке Utilities \ sequencecer \ stm32_seq.c.

Таблица 20. API-интерфейсы секвенсора

Функция	Описание
void UTIL_SEQ_Idle (void)	Вызывается (в критическом разделе - PRIMASK), когда нечего выполнять.
void UTIL_SEQ_Run (UTIL_SEQ_bm_t mask_bm)	Запрашивает, чтобы секвенсор выполнял функции, ожидающие обработки и включенные в маске **mask_bm**.
void UTIL_SEQ_RegTask (UTIL_SEQ_bm_t task_id_bm, uint32_t flags, void (*task)(void))	Регистрирует функцию (задачу), связанную с сигналом (**task_id_bm**) в секвенсоре. В **task_id_bm** должен быть установлен один бит.

void UTIL_SEQ_SetTask (UTIL_SEQ_bm_t taskId_bm, uint32_t task_Prio)	Запрашивает выполнение функции, связанной с **task_id_bm**. **Task_prio** оценивается секвенсором только после завершения функции. Если несколько функций ожидают одновременного выполнения, выполняется функция с наивысшим приоритетом (0).
void UTIL_SEQ_WaitEvt (UTIL_SEQ_bm_t EvtId_bm);	Ожидает установки определенного события.
void UTIL_SEQ_SetEvt (UTIL_SEQ_bm_t EvtId_bm);	Устанавливает событие, ожидающее с помощью **UTIL_SEQ_WaitEvt ()**.

На рисунке ниже стандартная реализация цикла while сравнивается с циклом while секвенсора.

Table 21. While-loop standard vs. sequencer implementation

Standard way

```
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*/
}
```

Sequencer way

```
/*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*/
}
```

8.2 Сервер таймера

Сервер таймера позволяет пользователю запрашивать выполнение заданий по времени. Поскольку аппаратный таймер основан на часах реального времени, время всегда считается, даже в режимах с низким энергопотреблением.

Сервер таймера обеспечивает надежные часы для пользователя и стека. Пользователь может запросить столько таймеров, сколько требуется приложению.

Сервер таймера находится в папке Utilities \ timer \ stm32_timer.c.

Таблица 39. API сервера таймера

Функция	Описание
UTIL_TIMER_Status_t UTIL_TIMER_Init (void)	Инициализирует сервер таймера.
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)	Создает объект таймера и связывает функцию обратного вызова по истечении времени таймера.

UTIL_TIMER_Status_t UTIL_TIMER_SetPeriod (UTIL_TIMER_Object_t *TimerObject, uint32_t NewPeriodValue)	Обновляет период и запускает таймер со значением тайм-аута (миллисекунды).
UTIL_TIMER_Status_t UTIL_TIMER_Start (UTIL_TIMER_Object_t *TimerObject)	Запускает и добавляет объект таймера в список событий таймера.
UTIL_TIMER_Status_t UTIL_TIMER_Stop (UTIL_TIMER_Object_t *TimerObject)	Останавливает и удаляет объект таймера из списка событий таймера.

7.3 Функции с низким энергопотреблением

Утилита с низким энергопотреблением централизует требования к низкому энергопотреблению отдельных модулей, реализованных с помощью встроенного ПО, и управляет входом с низким энергопотреблением, когда система переходит в режим ожидания. Например, когда DMA используется для печати данных на консоли, система не должна переходить в режим пониженного энергопотребления ниже спящего режима, потому что часы DMA отключены в режиме остановки.

API-интерфейсы, представленные в таблице ниже, используются для управления режимами пониженного энергопотребления основного MCU. Утилита с низким энергопотреблением находится в папке Utilities \ lpm \ tiny_lpm \ stm32_lpm.c.

Таблица 40. API с низким энергопотреблением

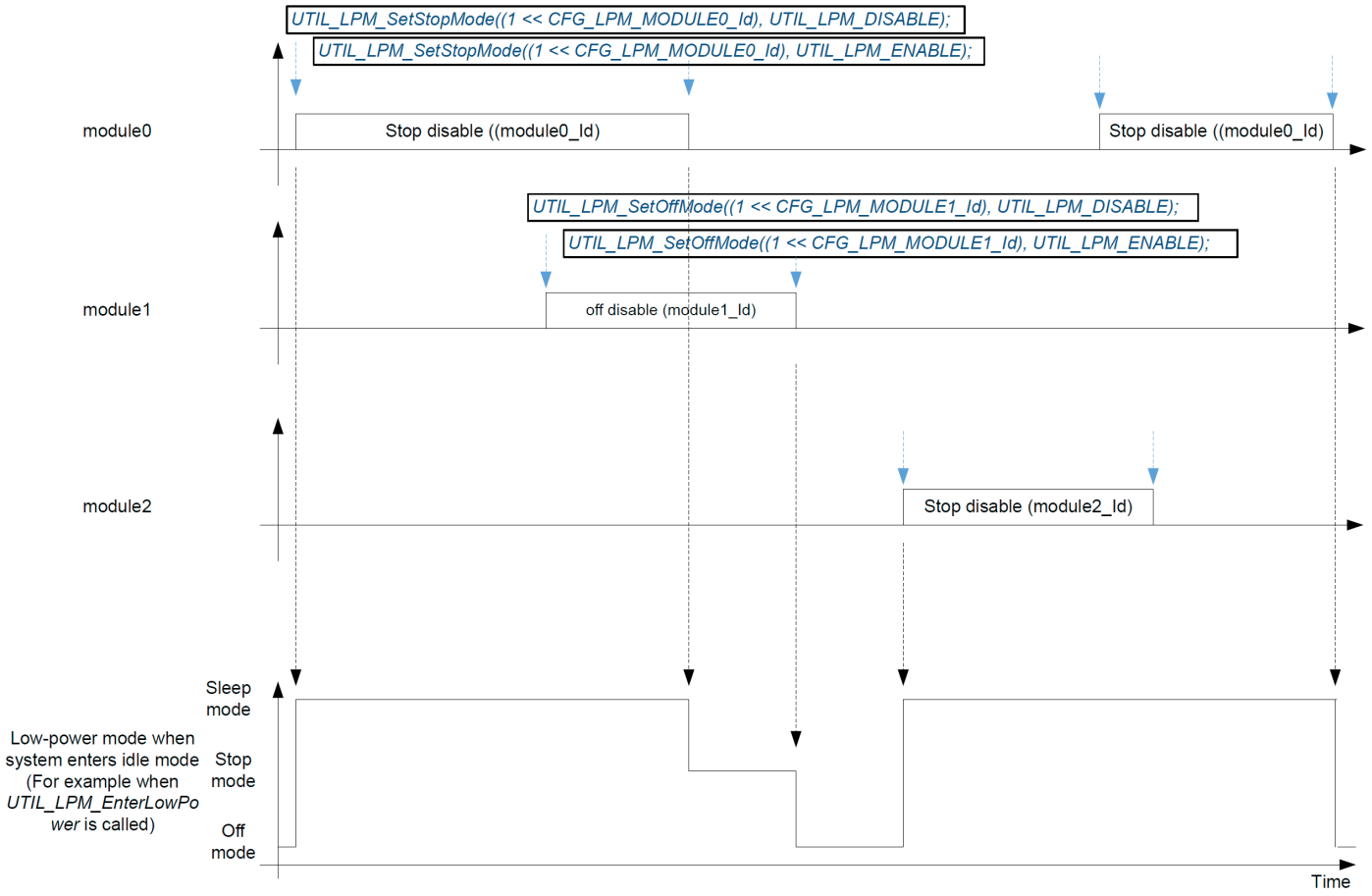
Функция	Описание
void UTIL_LPM_EnterLowPower(void)	Переход в выбранный режим пониженного энергопотребления. Вызывается незанятым состоянием системы
void UTIL_LPM_SetStopMode(UTIL_LPM_bm_t lpm_id_bm, UTIL_LPM_State_t state);	Устанавливает режим остановки. **id** определяет запрошенный режим процесса: UTIL_**LPM_ ENABLE или UTIL_LPM_DISABLE**. (1)
void UTIL_LPM_SetOffMode(UTIL_LPM_bm_t lpm_id_bm, UTIL_LPM_State_t state);	Устанавливает режим остановки. **id** определяет запрашиваемый режим процесса: **UTIL_LPM_ENABLE или UTIL_LPM_DISABLE**.
UTIL_LPM_Mode_t UTIL_LPM_GetMode(void)	Возвращает текущий выбранный режим пониженного энергопотребления.

[1]. Битовые карты, для которых значения сдвига определены в файлеutils_def.h.

Режим пониженного энергопотребления по умолчанию - это режим Off, который может быть режимом ожидания или выключения (определенным в void PWR_ EnterOffMode (void) из таблицы 41):

- Если режим остановки отключен хотя бы одним модулем микропрограммы и перешел в режим пониженного энергопотребления, выбирается спящий режим.
- Если режим остановки не отключен ни одним модулем встроенного ПО, режим выключения отключается по крайней мере одним модулем встроенного ПО и переходит в режим пониженного энергопотребления. Выбран режим остановки.
- Если режим остановки не отключен ни одним модулем встроенного ПО, режим выключения не отключен ни одним модулем встроенного ПО, и переходит в режим пониженного энергопотребления. Выбран режим выключения.

На приведенном ниже рисунке показано поведение трех различных модулей микропрограмм, устанавливаемых в зависимости от их требований к низкому энергопотреблению и режима пониженного энергопотребления, который выбирается, когда система переходит в режим пониженного энергопотребления.



Должны быть реализованы низкоуровневые API-интерфейсы, чтобы определить, что система должна делать для входа / выхода из режима низкого энергопотребления. Эти функции реализованы в stm32_lpm_if.c подпапки проекта.

Таблица 41. API низкого уровня

Функция	Описание
void PWR_EnterSleepMode (void)	API вызывается перед переходом в спящий режим
void PWR_ExitSleepMode (void)	API вызывается при выходе из спящего режима
void PWR_EnterStopMode (void)	API вызывается перед режимом остановки
void PWR_ExitStopMode (void)	API вызывается при выходе из режима остановки
void PWR_EnterOffMode (void)	API вызывается перед переходом в режим Off
void PWR_ExitOffMode(void)	API вызывается при выходе из режима Off

В спящем режиме тактовая частота ядра остановлена. Часы каждого периферийного устройства могут быть стробированы или нет. Питание поддерживается на всех периферийных устройствах.

В режиме Stop 2 остановлены большинство периферийных часов. Большинство периферийных устройств отключено. Некоторые регистры периферийных устройств не сохраняются и должны быть повторно инициализированы при выходе из режима Stop 2. Регистры памяти и ядра сохраняются.

В режиме ожидания все часы выключены, кроме LSI и LSE. Все периферийные источники питания выключены (кроме BOR, резервных регистров, GPIO pull и RTC) без сохранения (кроме дополнительной SRAM2 с удержанием) и должны быть повторно инициализированы при выходе из режима ожидания. Регистры

ядра не сохраняются и должны быть повторно инициализированы при выходе из режима ожидания.

Примечание. Радиоприемник на частоте ниже ГГц не зависит от остальной системы. См. Справочное руководство по продукту для получения более подробной информации.

8.4 Системное время

Время MCU относится к сбросу MCU. Системное время может записывать время эпохи UNIX®. API-интерфейсы, представленные в таблице ниже, используются для управления системным временем основного MCU. Утилита systime находится в папке Utilities \ misc \ stm32_systime.c.

Таблица 42. Функции системного времени

Функция	Описание
void SysTimeSet (SysTime_t sysTime)	На основе введенного периода UNIX в секундах и субсекундах разница со временем MCU сохраняется в резервном регистре (сохраняется даже в режиме ожидания). (1)
SysTime_t SysTimeGet (void)	Получает текущее системное время. (1)
uint32_t SysTimeMkTime (const struct tm* localtime)	Преобразует местное время в эпоху UNIX. (2)
void SysTimeLocalTime (const uint32_t timestamp, struct tm *localtime)	Преобразует время эпохи UNIX в местное время. (2)

[1]. Эталоном системного времени является эпоха UNIX, начинающаяся 1 января 1970 года.

[2]. SysTimeMkTime и SysTimeLocalTime также предоставляются для преобразования эпохи в структуру tm, как указано в интерфейсе time.h.

Чтобы преобразовать время UNIX в местное, необходимо добавить часовой пояс и удалить дополнительные секунды. В 2018 году необходимо убрать 18 дополнительных секунд. Летнее время в Париже отличается от гринвичского на два часа. Предполагая, что время установлено, местное время может быть напечатано на терминале с помощью кода ниже

```
{
SysTime_t UnixEpoch = SysTimeGet();
struct tm localtime;
UnixEpoch.Seconds-=18; /*removing leap seconds  удаление дополнительных секунд */
UnixEpoch.Seconds+=3600*2; /*adding 2 hours  добавив 2 часа */
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);
}
```

8.5 Trace (Трассировка)

Модуль трассировки позволяет выводить данные на COM-порт с помощью DMA. API, представленные в таблице ниже, используются для управления функциями трассировки.

Утилита трассировки находится в папке Utilities \ trace \ adv_trace \ stm32_adv_trace.c.

Таблица 26. Функции трассировки

Функция	Описание
UTIL_ADV_TRACE_Status_t UTIL_ADV_TRACE_Init(void)	TraceInit должен вызываться при инициализации приложения. Инициализирует оборудование com или vcom в режиме DMA и регистрирует обратный вызов для обработки по завершении передачи DMA.
UTIL_ADV_TRACE_Status_t UTIL_ADV_TRACE_COND_FSend(uint32_t VerboseLevel, uint32_t Region, uint32_t TimeStampState, const char *strFormat, ...)	Преобразует строковый формат в буфер и отправляет его в круговую очередь для печати.
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)	Отправляет данные length = **len** и отправляет их в круговую очередь для печати.
UTIL_ADV_TRACE_Status_t UTIL_ADV_TRACE_COND_ZCSend_Allocation(uint32_t VerboseLevel, uint32_t Region, uint32_t TimeStampState, uint16_t length, uint8_t **pData, uint16_t *FifoSize, uint16_t *WritePos)	Записывает отформатированные пользователем данные непосредственно в FIFO (Z-Cpy).

Значения статуса функций трассировки определены в структуре UTIL_ADV_TRACE_Status_t следующим образом.

```
typedef enum {
        UTIL_ADV_TRACE_OK                = 0,    /*Операция успешно завершена */
        UTIL_ADV_TRACE_INVALID_PARAM     = -1,   /*Неверный параметр*/
        UTIL_ADV_TRACE_HW_ERROR          = -2,   /*Аппаратная ошибка */
        UTIL_ADV_TRACE_MEM_ERROR         = -3,   /*Ошибка выделения памяти */
        UTIL_ADV_TRACE_UNKNOWN_ERROR = -4,   /*Неизвестная ошибка*/
        UTIL_ADV_TRACE_GIVEUP            = -5,   /*!< область трассировки оставлена*/
        UTIL_ADV_TRACE_REGIONMASKED      = -6    /*!< область трассировки замаскирована*/
} UTIL_ADV_TRACE_Status_t;
```

Функцию UTIL_ADV_TRACE_COND_FSend (..) можно использовать:

• в режиме опроса, когда не применяются ограничения реального времени: например, во время инициализации приложения

```
#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 */
```

• в режиме реального времени: когда в кольцевой очереди не осталось места, строка не добавляется и не распечатывается в com-порту

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

где:

- VL - подробный уровень трассировки.

- TS позволяет добавить метку времени к трассе (TS_ON или TS_OFF).

Уровень детализации приложения устанавливается в Core \ Inc \ sys_conf.h с помощью:

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

где VLEVEL может быть VLEVEL_OFF, VLEVEL_L, VLEVEL_M или VLEVEL_H.

UTIL_ADV_TRACE_COND_FSend (..) отображается, только если VLEVEL ≥ VerboseLevel.

Длина буфера может быть увеличена в случае его насыщения в Core \ Inc \ utilities_conf.h с помощью:

```
#define UTIL_ADV_TRACE_TMP_BUF_SIZE      256U
```

Утилита предоставляет хуки, которые должны быть реализованы, чтобы запретить системе переходить в режим Stop или более низкий, пока DMA активен:

• 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);}

9 Приложение LoRaWAN_End_Node

Это приложение измеряет уровень заряда батареи и температуру микроконтроллера. Эти значения периодически отправляются в сеть LoRa с использованием радио LoRa в классе A на частоте 868 МГц.

Чтобы запустить проект LoRaWAN_End_Node, перейдите в

\ Projects \ <target> \ Applications \ LoRaWAN \ LoRaWAN_End_Node и выберите папку любимой цепочки инструментов (в среде IDE). Выберите проект LoRa на соответствующей целевой доске.

Сосредоточьтесь на конфигурации, описанной ниже, чтобы настроить приложение.

9.1 Описание разделов пользовательского кода LoRaWAN

Четыре основные функции определены в качестве примера для реализации и использования стека LoRaWAN в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.c.

Эти функции содержат пример кода в разделах USER CODE, который можно перезаписать для обработки специфических функций прикладного уровня.

Табл. 27. Пользовательские функции LoRaWAN

Функция	Описание
void LoRaWAN_Init (void)	Инициализируйте приложение LoRaWAN, как описано на рисунке 5. Модель работы.
static void SendTxData(void)	Пример процесса LoRaWAN Tx с генерацией универсальной полезной нагрузки и вызовом LmHandlerSend(...) с буфером, сгенерированным временной полезной нагрузкой.
static void OnTxData(LmHandlerTxParams_t *params)	Пример реализации обратного вызова, когда приложение LoRaWAN успешно передало кадр. • параметр params содержит статус последнего Tx
static void OnRxData(LmHandlerAppData_t *appData, LmHandlerRxParams_t *params)	Пример реализации обратного вызова, когда приложение LoRaWAN получило кадр. • Параметр appData содержит данные, полученные в последнем Rx. • параметр params содержит статус последнего Rx

9.2 Конфигурация устройства

9.2.1 Способы активации и ключи

Есть два способа активировать устройство в сети: через OTAA или через ABP.

Глобальная переменная ActivationType в приложении должна быть настроена для активации устройства в выбранном режиме.

static ActivationType_t ActivationType = LORAWAN_DEFAULT_ACTIVATION_TYPE;

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

и

```
#define LORAWAN_DEFAULT_ACTIVATION_TYPE ACTIVATION_TYPE_OTAA
```

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

где перечисление ActivationType_t определяется следующим образом:

```
typedef enum eActivationType {
    ACTIVATION_TYPE_NONE = 0, /* Никак */
    ACTIVATION_TYPE_ABP  = 1, /* Активация путем персонализации */
    ACTIVATION_TYPE_OTAA = 2, /* Активация по воздуху */
```

\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\se-identity.h файл содержит данные ввода в эксплуатацию, полезные для активации устройства.

9.2.2 Активация класса LoRa

По умолчанию определен класс A. Чтобы изменить активацию класса (класс A, класс B или класс C), пользователь должен:

- установить код

```
#define LORAWAN_DEFAULT_CLASS CLASS_B;
```

в

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

- установить код

```
#define LORAMAC_CLASSB_ENABLED 1
```

в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lorawan_conf.h

9.2.3 Tx триггер

Существует два способа создания действия восходящего канала с глобальной переменной EventType в

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

- по таймеру
- по внешнему событию

с кодом

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

где перечисление TxEventType_t определяется следующим образом:

```
typedef enum TxEventType_e {
   TX_ON_TIMER = 0, /* Передача данных приложения по таймеру */
   TX_ON_EVENT = 1, /* Передача данных приложения по внешнему событию */
}TxEventType_t;
```

Функция TX_ON_EVENT использует кнопку 1 как событие в приложении LoRaWAN_End_Node.

9.2.4 Рабочий цикл

Значение рабочего цикла (в мс), которое будет использоваться для приложения, определено в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h, с кодом ниже (например) :

```
#define APP_TX_DUTYCYCLE            10000 /* Рабочий цикл 10 с */
```

9.2.5 Порт приложения

Порт приложения, который будет использоваться для приложения, определен в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h, с кодом ниже (например) :

```
#define LORAWAN_APP_PORT            2
```

*Примечание. **LORAWAN_APP_PORT не должен использовать порт 224**, зарезервированный для сертификации.*

9.2.6 Подтвержденный / неподтвержденный режим

Режим подтверждения / неподтверждения, который будет использоваться для приложения, определен в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h, с кодом ниже :

```
#define LORAWAN_DEFAULT_CONFIRMED_MSG_STATE  LORAMAC_HANDLER_UNCONFIRMED_MSG
```

9.2.7 Размер буфера данных

Размер буфера, отправляемого в сеть, определяется в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h, с кодом ниже :

```
#define LORAWAN_APP_DATA_BUFFER_MAX_SIZE            242
```

9.2.8 Адаптивная скорость передачи данных (ADR)

ADR включен в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h, с кодом ниже :

```
#define LORAWAN_ADR_STATE                LORAMAC_HANDLER_ADR_ON
```

Когда ADR отключен, ставка по умолчанию устанавливается в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h, с кодом ниже :

```
#define LORAWAN_DEFAULT_DATA_RATE            DR_0
```

9.2.9 Периодичность проверки связи

Если устройство может переключаться в Класс B, периодичность слота Rx Ping по умолчанию должна быть включена в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h с кодом ниже.

```
#define LORAWAN_DEFAULT_PING_SLOT_PERIODICITY        4
```

где ожидаемое значение должно быть в диапазоне 0-7.

Результирующий период времени определяется как :

```
period = 2^LORAWAN_DEFAULT_PING_SLOT_PERIODICITY
```

9.2.10 Выбор диапазона LoRa

Выбор региона и соответствующего ему диапазона определяется в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\Target\lorawan_conf.h с кодом ниже:

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

Примечание. В одном приложении можно определить несколько регионов.

В зависимости от региона активная область по умолчанию должна быть определена в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h с кодом ниже

(пример для Европы)

```
#define ACTIVE_REGION LORAMAC_REGION_EU868
```

9.2.11 Управление контекстом

Управление контекстом определяется в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lorawan_conf.h с кодом ниже:

```
#define CONTEXT_MANAGEMENT_ENABLED 1
```

Дополнительные сведения об этой функции см. в разделе 13 «Описание управления контекстом LoRaWAN».

9.2.12 Переключатель отладки

Режим отладки включен в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h с кодом ниже:

```
#define DEBUGGER_ENABLED    1  /* ON=1, OFF=0 */
```

Режим отладки включает контакты SWD, даже когда MCU переходит в режим пониженного энергопотребления.

Примечание. Чтобы включить действительно низкое энергопотребление, необходимо сбросить #define DEBUGGER_ENABLED.

Некоторые дополнительные определения активируют мониторинг (пробы) некоторого внутреннего радиочастотного сигнала для отладки:

```
#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
```

9.2.13 Переключатель энергосбережения

Когда система находится в режиме ожидания, она переходит в режим остановки 2 с низким энергопотреблением.

Эта запись в режиме Stop 2 может быть отключена в \Projects\<target> \Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h с кодом ниже :

```
#define LOW_POWER_DISABLE  0
/* Низкое энергопотребление включено = 0, низкое энергопотребление отключено = 1 */
```

где:

- Низкое энергопотребление включено = 0 означает, что MCU переходит в режим Stop 2.

Stop 2 — это режим Stop с регулятором малой мощности и отключаемым питанием домена цифрового ядра VDD12I.

Активируется меньше периферийных устройств, чем в режиме Stop 1 с низким энергопотреблением, чтобы снизить энергопотребление. См. документ [3] для более подробной информации.

• Низкое энергопотребление отключено = 1 означает, что MCU переходит только в спящий режим.

9.2.14 Уровень трассировки

Режим трассировки включен в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h с кодом ниже:

```
#define APP_LOG_ENABLED 1
```

Уровень трассировки выбирается в \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h с кодом ниже:

```
#define VERBOSE_LEVEL VLEVEL_M
```

Предлагаются следующие уровни трассировки:

- VLEVEL_OFF: все трассировки отключены
- VLEVEL_L: функциональные трассировки включены
- VLEVEL_M: трассировка отладки включена
- VLEVEL_H: все трассировки включены

9.3 Сводка конфигурации устройства для приложения LoRaWAN_End_Node

Таблица 28. Параметры переключения для конфигурации приложения LoRaWAN_End_Node

	Деталь	Вариант переключения	Определение	Местоположение
LoRa stack	Identification	STATIC_DEVICE_EUI	Статическая или динамическая идентификация конечного устройства	se-identity.h
	Address	STATIC_DEVICE_ADDRESS	Статический или динамический адрес конечного устройства	
	Supported regions	REGION_EU868	Регионы, поддерживаемые устройством	lorawan_conf.h
		REGION_EU433		
		REGION_US915		
		REGION_AS923		
		REGION_AU915		
		REGION_CN470		
		REGION_CN779		
		REGION_IN865		
		REGION_RU864		
		REGION_KR920		
	Limited channels	HYBRID_ENABLED	По умолчанию ограничивает количество используемых каналов для регионов AU915, CN470 и US915	
	Read keys	KEY_EXTRACTABLE	Определяет доступ для чтения ключей в памяти.	
	Optional class	LORAMAC_CLASSB_ENABLED	Возможность конечного устройства класса B	
	Context Management	CONTEXT_MANAGEMENT_ENABLED	Сохраняет и восстанавливает контекст стека LoRaWAN.	

Application	Tx trigger	EventType = TX_ON_TIMER	Метод включения передачи	lora_app.c
	Class choice	LORAWAN_DEFAULT_CLASS	Устанавливает класс устройства	lora_app.h
	Duty cycle	APP_TX_DUTYCYCLE	Период времени между двумя отправленными Tx	
	App port	LORAWAN_USER_APP_PORT	Порт LoRa, используемый фреймом данных Tx	
	Confirmed mode	LORAWAN_DEFAULT_CONFIRMED_MSG_STATE	Подтвержденный выбор режима	
	Adaptive data rate	LORAWAN_ADR_STATE	Выбор ADR	
	Default data rate	LORAWAN_DEFAULT_DATA_RATE	Скорость передачи данных, если ADR отключен	
	Max data buffer size	LORAWAN_APP_DATA_BUFFER _MAX_SIZE	Определение размера буфера	
	Ping period	LORAWAN_DEFAULT_PING_SLOT_PERIODICITY	Период слота пинга приема	
	Network Join activation	LORAWAN_DEFAULT _ACTIVATION_TYPE	Выбор процедуры активации по умолчанию	
	Initial region	ACTIVE_REGION	Регион, используемый при запуске устройства	
	Debug	DEBUGGER_ENABLED	Включает выводы SWD	sys_conf.h
	Low power	LOW_POWER_DISABLE	Отключает режим пониженного энергопотребления	
	Trace enable	APP_LOG_ENABLED	Включает режим трассировки	
	Trace level	VERBOSE_LEVEL	Включает уровень трассировки	

10. Приложение LoRaWAN_AT_Slave

Цель этого примера - реализовать модем LoRa, управляемый через интерфейс AT-команд через UART внешним хостом.

Внешний хост может быть микроконтроллером хоста, встраивающим приложение и драйвер AT, или просто компьютером, выполняющим терминал.

Это приложение предназначено для платы STM32WL Nucleo (NUCLEO-WL55JC).

Пример LoRaWAN_AT_Slave реализует стек LoRaWAN, управляющий встроенным радио LoRa. Стек управляется через интерфейс AT-команд через UART. Модем всегда находится в режиме Stop 2, если он не обрабатывает AT-команду от внешнего хоста.

Чтобы запустить проект LoRaWAN_AT_Slave, пользователь должен перейти в \ Projects \ <target> \ Applications \ LoRaWAN \ LoRaWAN_AT_Slave и выполнить ту же процедуру, что и для проекта LoRaWAN_End_Node, чтобы запустить предпочитаемый набор инструментов.

В документе [2] приводится список AT-команд и их описание.

В таблице ниже приведены основные параметры конфигурации приложения LoRaWAN_AT_Slave.

Таблица 29. Параметры переключения для конфигурации приложения LoRaWAN_AT_Slave

	Деталь	Вариант переключения	Определение	Местоположение
LoRa stack	Identification	STATIC_DEVICE_EUI	Статическая или динамическая идентификация конечного устройства	se-identity.h
	Address	STATIC_DEVICE_ADDRESS	Статический или динамический адрес конечного устройства	
	Supported regions	REGION_EU868	Регионы, поддерживаемые устройством	lorawan_conf.h
		REGION_EU433		
		REGION_US915		
		REGION_AS923		
		REGION_AU915		
		REGION_CN470		
		REGION_CN779		
		REGION_IN865		
		REGION_RU864		
		REGION_KR920		
	Limited channels	HYBRID_ENABLED	По умолчанию ограничивает количество используемых каналов для регионов AU915, CN470 и US915	
	Read keys	KEY_EXTRACTABLE	Определяет доступ для чтения ключей в памяти.	
	Optional class	LORAMAC_CLASSB_ENABLED	Возможность конечного устройства класса B	
	Context Management	CONTEXT_MANAGEMENT_ENABLED	Сохраняет и восстанавливает контекст стека LoRaWAN.	
Application	Adaptive data rate	LORAWAN_ADR_STATE	Выбор ADR	lora_app.h
	Default data rate	LORAWAN_DEFAULT_DATA_RATE	Скорость передачи данных, если ADR отключен	
	Ping period	LORAWAN_DEFAULT_PING_SLOT_PERIODICITY	Период слота пинга приема	
	Initial region	ACTIVE_REGION	Регион, используемый при запуске устройства	
	Debug	DEBUGGER_ENABLED	Включает выводы SWD	sys_conf.h
	Low power	LOW_POWER_DISABLE	Отключает режим пониженного энергопотребления	
	Trace enable	APP_LOG_ENABLED	Включает режим трассировки	
	Trace level	VERBOSE_LEVEL	Включает уровень трассировки	

11 Приложение SubGhz_Phy_PingPong

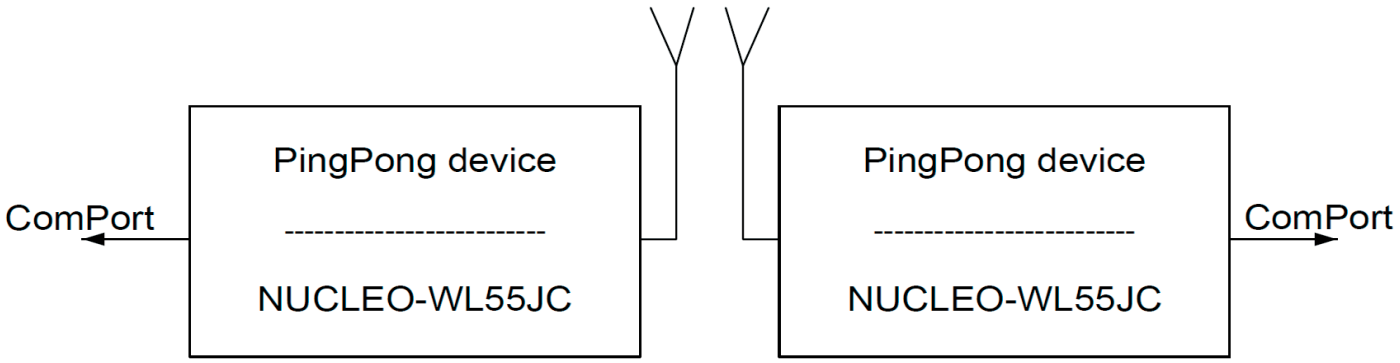
Это приложение показывает простой канал Rx / Tx RF между двумя устройствами PingPong (одно называется Ping, другое - Pong).

По умолчанию каждое устройство PingPong запускается как ведущее, передает сообщение «Ping» и ожидает ответа. При запуске на каждом устройстве PingPong мигают два светодиода. Когда платы синхронизированы (окно Tx одной платы совмещено с окном Rx другой платы), устройство Ping (плата принимает сообщение Ping) заставляет мигать зеленый светодиод, а устройство Pong (плата принимает сообщение Pong) заставляет мигать красный светодиод. Первое устройство PingPong, которое получает сообщение «Ping», становится ведомым и отвечает сообщением «Pong» ведущему устройству.

Чтобы запустить проект SubGhz_Phy_PingPong, пользователь должен перейти в \ Projects \ <target> \ Applications \ SubGHz_Phy \ SubGHz_Phy_PingPong и выполнить ту же процедуру, что и для проекта LoRaWAN_End_Node, чтобы запустить предпочтительный набор инструментов.

11.1 Настройка аппаратной / программной среды SubGhz_Phy_PingPong

Чтобы настроить плату STM32WL Nucleo (NUCLEO-WL55JC), подключите эту плату к компьютеру с помощью кабеля USB типа A - Mini B к разъему ST-LINK (CN1), как показано на рисунке ниже.



11.2 Конфигурация устройства

11.2.1 Определение модуляции

Это приложение предлагает использовать две модуляции: LoRa или FSK. Чтобы настроить одну модуляцию, пользователь должен обновить эти определения в \Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h, как показано в таблице ниже:

Таблица 30. Конфигурация модуляции SubGHz_Phy_PingPong

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 Длина полезной нагрузки

Каждая полезная нагрузка Tx определяется строкой PING или PONG, за которой следует последовательность 0. Длина этой полезной нагрузки определяется в \Projects \<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h с кодом ниже:

```
#define PAYLOAD_LEN 64
```

Типичный размер полезной нагрузки обычно составляет от 51 до 242.

Обязательно установите значение, равное или меньше максимального размера буфера, определенного в \Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.c с кодом ниже:

```
#define MAX_APP_BUFFER_SIZE 255
```

11.2.3 Регион и частота

Значение частоты, которое будет использоваться для приложения, определяется в \Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h с кодом ниже (например):

```
#define RF_FREQUENCY 868000000 /* Гц */
```

Дополнительное определение предлагает использовать некоторые предопределенные значения частоты для каждого плана каналов с кодом ниже.

(одновременно можно раскомментировать только одно определение):

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

11.2.4 Полоса пропускания, коэффициент расширения и скорость передачи данных

В зависимости от выбранной модуляции можно определить полосу пропускания, коэффициент расширения и скорость передачи данных в \Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h, как показано в таблице ниже:

Таблица 31. Полоса пропускания SubGHz_Phy_PingPong, конфигурация SF и DR

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

Ожидаемое значение LORA_BANDWIDTH должно находиться в диапазоне 0-2, что соответствует эквивалентной частоте:

[0: 125 кГц, 1: 250 кГц, 2: 500 кГц].

Ожидаемое значение LORA_SPREADING_FACTOR должно находиться в диапазоне 7–12. Коэффициент расширения влияет на время, необходимое для передачи кадра, и на мощность передачи.

Ожидаемое значение FSK_BANDWIDTH должно находиться в диапазоне 4800-500000 (в Гц).

Ожидаемое значение FSK_DATARATE, эквивалентное значению битрейта, обычно определяется как 50 кбит/с.

11.2.5 Длина преамбулы

Все переданные/полученные пакеты содержат преамбулу (обычно из восьми символов), заголовок, полезную нагрузку (размер определяется в разделе 10.2.2) и поле CRC.

Размер поля преамбулы может быть обновлен в \Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h с помощью этих определений, как показано в таблице ниже:

Таблица 32. Конфигурация преамбулы SubGHz_Phy_PingPong

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

11.3 Сводная информация о конфигурации устройства для приложения SubGhz_Phy_PingPong

Табл. 33. Опции переключателей для конфигурации приложения SubGhz_Phy_PingPong

Project module = Application

Detail	Switch option	Definition	Location
Rx/Tx configuration	RX_TIMEOUT_VALUE	Тайм-аут окна Rx	subghz_phy_app.c
	TX_TIMEOUT_VALUE	Тайм-аут окна Tx	
	MAX_APP_BUFFER_SIZE	Максимальный размер буфера данных	
	RX_TIME_MARGIN	Время между окончанием Rx и началом Tx	
	FSK_AFC_BANDWIDTH	Полоса пропускания АЧХ (в Гц)	
	LED_PERIOD_MS	период мигания светодиода	
Modulation configuration	USE_MODEM_LORA	Выбран модем LORA	subghz_phy_app.h
	USE_MODEM_FSK	Выбран модем FSK	
LoRa/FSK common parameters	REGION_XXyyy	Активный регион LoRa: AS923, AU915, CN470, CN779, EU433, EU868, KP920, IN865, US915, RU864.	
	RF_FREQUENCY	Частота, используемая трансивером	
	TX_OUTPUT_POWER	Выходная мощность RF: от -17 до 22 дБм	
	PAYLOAD_LEN	Размер буфера данных	
LoRa specific parameters	LORA_BANDWIDTH	Пропускная способность: • 0: 125 kHz • 2: 500 kHz • 1: 250 kHz • 3: Зарезервирован	
	LORA_SPREADING_FACTOR	Коэффициент расширения: от SF7 до SF12	
	LORA_CODINGRATE	Скорость кодирования: • 1: 4/5 • 2: 4/6 • 3: 4/7 • 4: 4/8	
	LORA_PREAMBLE_LENGTH	Длина преамбулы Tx/Rx	
	LORA_SYMBOL_TIMEOUT	Количество проверенных символов до истечения времени ожидания	
	LORA_FIX_LENGTH_PAYLOAD_ON	Опция фиксированной/динамической длины полезной нагрузки	
	LORA_IQ_INVERSION_ON	Вариант инверсии IQ	
FSK specific parameters	FSK_FDEV	Отклонение частоты (в Гц)	
	FSK_DATARATE	Скорость передачи данных (в бит / с)	
	FSK_BANDWIDTH	Полоса пропускания (в Гц)	
	FSK_PREAMBLE_LENGTH	Длина преамбулы Tx / Rx	
	FSK_FIX_LENGTH_PAYLOAD_ON	Параметр полезной нагрузки фиксированной / динамической длины	
Debug	DEBUGGER_ENABLED	Включает выводы SWD	sys_conf.h
Low power	LOW_POWER_DISABLE	Отключает режим пониженного энергопотребления	
Trace enable	APP_LOG_ENABLED	Включает режим трассировки	
Trace level	VERBOSE_LEVEL	Включает уровень трассировки	

12 Приложение SubGhz_Phy_Per

Приложение SubGHz_Phy_Per - это тест на частоту ошибок пакетов с отбеливанием IBM® между одним устройством Tx и одним Rx устройством.

Tx device

Обновите #define TEST_MODE до RADIO_TX в /SubGHz_Phy/App/subghz_phy_app.c. Скомпилируйте и загрузите.

Содержимое пакета - преамбула | синхронизация | длина полезной нагрузки | полезная нагрузка | crc где:

- crc рассчитывается с использованием длины полезной нагрузки и полезной нагрузки.
- Отбеливание рассчитывается по длине полезной нагрузки | полезная нагрузка | crc.

Передача начинается с постоянной скоростью в GFSK 50 Кбит/с и полезной нагрузкой 64 байта. Пользовательская кнопка 1 увеличивает длину пакета на 16 байтов. Пользовательская кнопка 2 увеличивает длину пакета на 1 байт. Пользовательская кнопка 3 переключает режим полезной нагрузки пакета с ramp (0x00, 0x01 ..) на prbs9.

Синий светодиод горит, когда радио находится в режиме Tx.

Rx device

Обновите #define TEST_MODE до RADIO_RX в /SubGHz_Phy/App/subghz_phy_app.c. Скомпилируйте и загрузите.

Зеленый светодиод горит, когда Rx в порядке. Красный светодиод горит, когда Rx находится в положении KO.

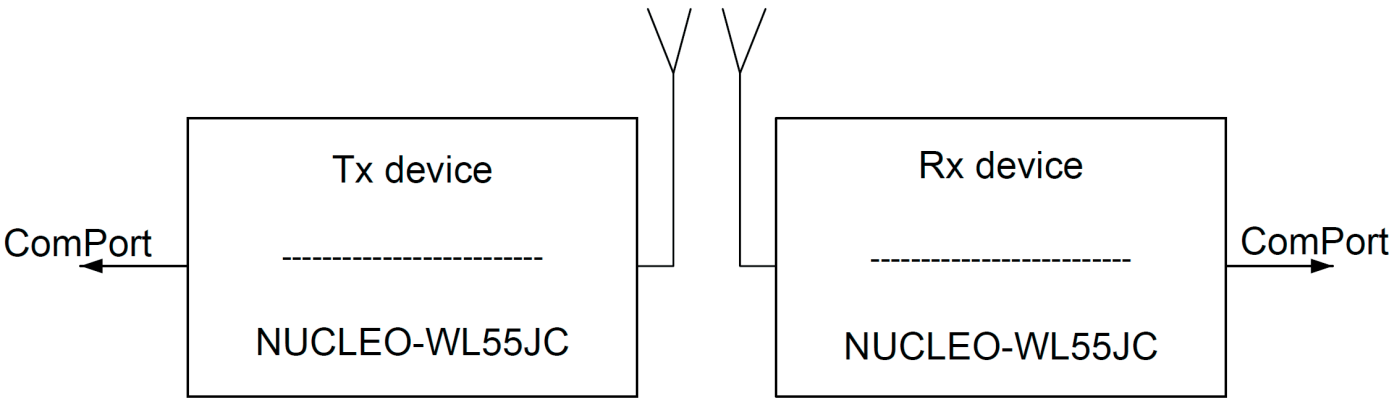
Чтобы запустить проект SubGhz_Phy_Per, пользователь должен перейти в

\ Projects \ <target> \ Applications \ SubGHz_Phy \ SubGHz_Phy_Per и выполните ту же процедуру, что и для проекта LoRaWAN_End_Node, чтобы запустить предпочтительный набор инструментов.

12.1 Настройка аппаратной / программной среды SubGhz_Phy_Per

Чтобы настроить плату STM32WL Nucleo (NUCLEO-WL55JC), подключите эту плату к компьютеру с помощью USB

Кабель типа - A - Mini - B к разъему ST-LINK (CN1), как показано на рисунке ниже.



12.2 Сводка конфигурации устройства для приложения SubGhz_Phy_Per

Таблица 34. Параметры переключения для конфигурации приложения SubGhz_Phy_Per

Project module = Application

Detail	Switch option	Definition	Location
Rx/Tx configuration	RX_TIMEOUT_VALUE	Тайм-аут окна Rx	subghz_phy_app.c
	TX_TIMEOUT_VALUE	Тайм-аут окна Tx	
	RX_CONTINUOUS_ON	Режим приема непрерывный или с тайм-аутом	
	TEST_MODE	Режим устройства: • RADIO_TX: отправлять пакет на неопределенный срок • RADIO_RX: принимать пакет бесконечно	
	APP_LONG_PACKET	Вариант длинного пакета (1)	
	MAX_APP_BUFFER_SIZE	Максимальный размер буфера данных	
Modulation configuration	USE_MODEM_FSK	Выбран модем FSK	subghz_phy_app.h
LoRa/FSK common parameters	REGION_XXyyy	Активный регион LoRa: AS923, AU915, CN470, CN779, EU433, EU868, KP920, IN865, US915, RU864.	
	RF_FREQUENCY	Частота, используемая трансивером	
	TX_OUTPUT_POWER	Выходная мощность RF: от -17 до 22 дБм	
	PAYLOAD_LEN	Размер буфера данных	
FSK specific parameters	FSK_FDEV	Отклонение частоты (в Гц)	
	FSK_DATARATE	Скорость передачи данных (в бит / с)	
	FSK_BANDWIDTH	Полоса пропускания (в Гц)	
	FSK_PREAMBLE_LENGTH	Длина преамбулы Tx / Rx	
	FSK_FIX_LENGTH_PAYLOAD_ON	Параметр полезной нагрузки фиксированной / динамической длины	
Debug	DEBUGGER_ENABLED	Включает выводы SWD	sys_conf.h
Low power	LOW_POWER_DISABLE	Отключает режим пониженного энергопотребления	
Trace enable	APP_LOG_ENABLED	Включает режим трассировки	
Trace level	VERBOSE_LEVEL	Включает уровень трассировки	

1. *Подробную информацию см. в документе [5].*

13 Описание управления контекстом LoRaWAN

Управление контекстом NVM используется для сохранения текущей конфигурации стека LoRaWAN в ПЗУ перед отключением питания или перезагрузкой платы.

Предлагаемое решение состоит в том, чтобы хранить структуру LoRaMacNvmData_t размером 1420 байт на предварительно выделенной странице ПЗУ размером 2 Кбайт.

Эта структура определяется следующим образом:

Табл. 35. Структура контекста LoRaWAN NVM

Поле	Тип	Размер (байты)	Описание
Crypto	LoRaMacCryptoNvmData_t	40	Параметры, относящиеся к криптослою. Меняются с каждым TX/RX.

MacGroup1	LoRaMacNvmDataGroup1_t	24	Параметры, относящиеся к MAC, которые с высокой вероятностью изменяются после каждой TX/RX.
MacGroup2	LoRaMacNvmDataGroup2_t	228	Параметры, относящиеся к MAC, которые, скорее всего, не меняются при каждой передаче/приеме.
SecureElement	SecureElementNvmData_t	192	Параметры, относящиеся к защищенному элементу (ключи, идентификаторы и т. д.)
RegionGroup1	RegionNvmDataGroup1_t	20	Параметры, относящиеся к региональной реализации, которые с высокой вероятностью меняются после каждой процедуры TX/RX.
RegionGroup2	RegionNvmDataGroup2_t	892	Параметры, относящиеся к региональной реализации, которые, скорее всего, не меняются при каждой процедуре TX/RX.
ClassB	LoRaMacClassBNvmData_t	24	Параметры, относящиеся к классу b.

13.1 Определение API данных управления контекстом NVM

Табл. 36. API управления контекстом LoRaWAN и обратные вызовы

Функция	Описание
LmHandlerErrorStatus_t mHandlerNvmDataStore (void)	• Останавливает LoRaMAC. • Подготавливает данные NVM для сохранения во флэш-памяти. • Вызывает обратный вызов OnStoreContextRequest() для выполнения операции FLASH_Write. • Возобновляет LoRaMAC.
void OnRestoreContextRequest (void *nvm, uint32_t nvm_size)	Восстанавливает данные NVM, хранящиеся во флэш-памяти, в резервный буфер в ОЗУ.
void OnStoreContextRequest (void *nvm, uint32_t nvm_size)	Сохраняет данные NVM из ОЗУ во флэш-память.
void OnNvmDataChange (LmHandlerNvmContextStates_t state)	Вызывается LmHandler после завершения действия сохранения или восстановления.

Если обратные вызовы не определены, функция считается отключенной. Функция, включенная определением, описанным в Разделе 9.2.11 Управление контекстом Управление контекстом

14 Двухъядерное управление

В устройства STM32WL5x встроены два Cortex:

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

В двухъядерных приложениях часть приложения, отображаемая на CPU1, отделена от стека, а нижние уровни микропрограммного обеспечения отображаются на CPU2.

В предлагаемой двухъядерной модели генерируются два отдельных двоичных файла: двоичный файл CPU1 помещается в 0x0800 0000, а двоичный файл CPU2 помещается в 0x0802 0000.

Адрес функции из одного двоичного файла неизвестен из другого двоичного файла: поэтому необходимо создать модель связи. Цель этой модели состоит в том, чтобы пользователь мог изменить приложение на CPU1, не влияя на поведение основного стека на CPU2. Однако ST по-прежнему предоставляет реализацию двух процессоров с открытым исходным кодом. Интерфейс между ядрами осуществляется периферийным устройством IPCC (контроллером межпроцессорной связи) и межъядерной памятью, как описано в разделе 12.1.

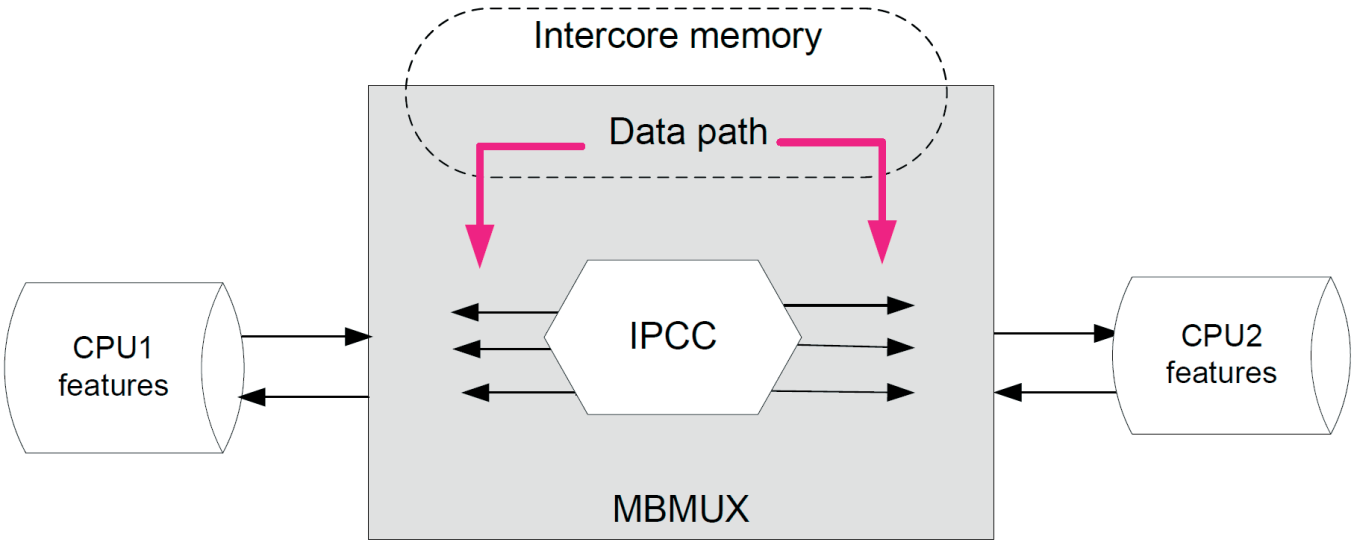
Эта двухъядерная реализация была разработана так, чтобы вести себя так же, как выполнение одноядерной программы, благодаря обработке блокировки сообщений через механизм почтового ящика.

14.1 Механизм почтового ящика

Почтовый ящик - это служба, реализующая способ обмена данными между двумя процессорами. Как показано на рисунке ниже, почтовый ящик состоит из двух ресурсов:

- **IPCC**: это аппаратное периферийное устройство используется для запуска прерывания для удаленного ЦП и для получения прерывания, когда он завершил уведомление. IPCC легко настраивается, и каждое уведомление о прерывании может быть отключено / включено. Внутри IPCC нет управления памятью.
- **Intercore memory (Межядерная память)**: эта общая память может считываться / записываться обоими процессорами. Он используется для хранения всех буферов, содержащих данные, которыми должны обмениваться два ЦП.

Figure 10. Mailbox overview



Почтовый ящик указывается так, чтобы разрешить изменение определения буфера в некоторой степени без нарушения обратной совместимости.

14.1.1 Мультиплексор почтовых ящиков (MBMUX)

Как показано на рисунке 11, данные, подлежащие обмену, должны передаваться через 12 доступных каналов IPCC (шесть для каждого направления). Это осуществляется через MBMUX (мультиплексор почтовых ящиков), который является компонентом встроенного ПО, отвечающим за маршрутизацию сообщений. Эти каналы обозначаются от 1 до 6. Дополнительный канал 0 выделен для функции системы.

Тип данных разделен на группы, называемые функциями. Каждая функция взаимодействует с MBMUX через собственный MBMUXIF (интерфейс MBUX).

Почтовый ящик используется для абстрагирования функции, выполняемой другим ядром.

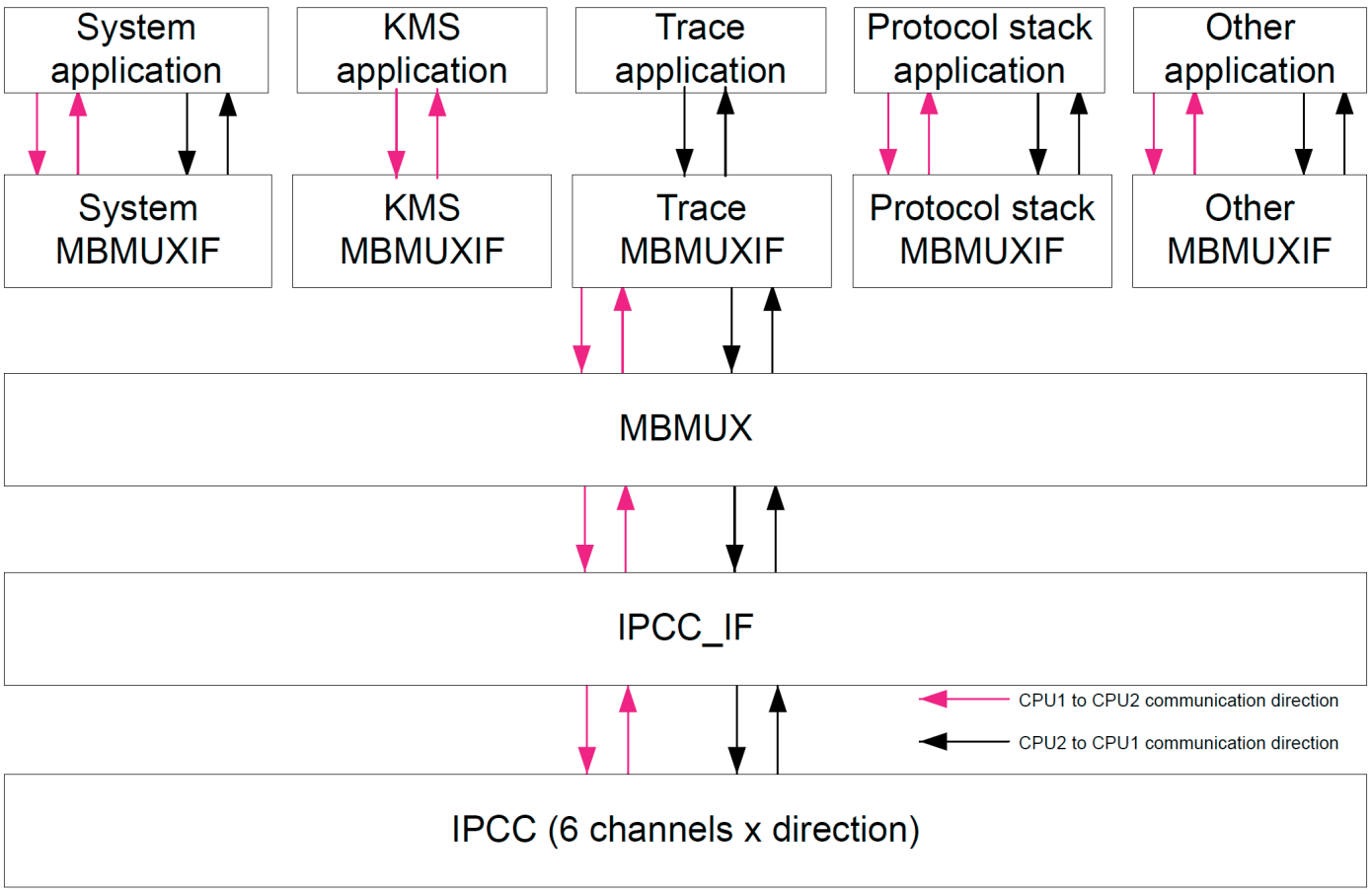
14.1.2 Функции почтового ящика

В устройствах STM32WL5x MBMUX имеет следующие функции:

• Система, поддерживающая все коммуникации, связанные с системой

Сюда входят сообщения, которые либо относятся к одному из поддерживаемых стеков, либо не относятся ни к одному из них. Канал 0 CPU1 используется для уведомления CPU2 о том, что команда отправлена, и для получения ответа на эту команду от CPU2. Канал 0 CPU2 используется для уведомления CPU1 о том, что было отправлено асинхронное событие.

Figure 11. MBMUX - Multiplexer between features and IPCC channels



Следующие службы отображаются на системном канале:

- инициализация системы

- Каналы IPCC по сравнению с регистрацией функций
- обмен информацией об атрибутах и возможностях функций
- возможные дополнительные системные каналы для высокоприоритетных операций (например, уведомлений RTC)

• **Trace**

CPU2 заполняет круговую очередь для информации или отладки, которая отправляется на CPU1 через IPCC. CPU1 отвечает за обработку этой информации, выводя ее по тому же каналу, который используется для журналов CPU1 (например, USART).

• **KMS** (службы управления ключами)

• **Radioo**

Радиомодуль с частотой менее ГГц может подключаться напрямую, минуя стек CPU2. Используется выделенный канал почтового ящика.

• **Protocol stack** Этот канал используется для взаимодействия всех команд стека протоколов (таких как инициализация или запрос) и событий (ответ / индикация), связанных с протоколом, реализованным в стеке.

Для использования MBMUX необходимо зарегистрировать функцию (кроме системной функции, которая регистрируется по умолчанию и всегда отображается на канале 0 IPCC). Регистрация динамически присваивает функции запрошенное количество каналов IPCC: обычно по одному для каждого направления (CPU1 - CPU2 и CPU2 - CPU1).

В следующих случаях функции нужен только канал в одном направлении:

• Функция трассировки предназначена только для отправки отладочной информации от CPU2 к CPU1.

• KMS используется CPU1 только для запроса выполнения функций CPU2.

Примечание:

• Аварийный сигнал RTC A передает прерывание с использованием одного IRQ IPCC, что не рассматривается как функция.

• Пользователь должен рассмотреть возможность добавления оболочки KMS, чтобы иметь возможность использовать ее в качестве функции.

14.1.3 Сообщения MBMUX

В почтовом ящике используются следующие типы сообщений:

• Cmd command, отправляемая CPU1 в CPU2, состоит из:
  - Msg ID идентифицирует функцию, вызываемую CPU1, но реализованную на CPU2.
  - Ptr buffer params указывает на буфер, содержащий параметры вышеуказанной функции
  - Number of params (Количество параметров)

• Resp, ответ, отправленный CPU2 на CPU1, состоящий из:
  - Msg ID (то же значение, что и Cmd Msg ID)
  - Return value содержит возвращаемое значение указанной выше функции.

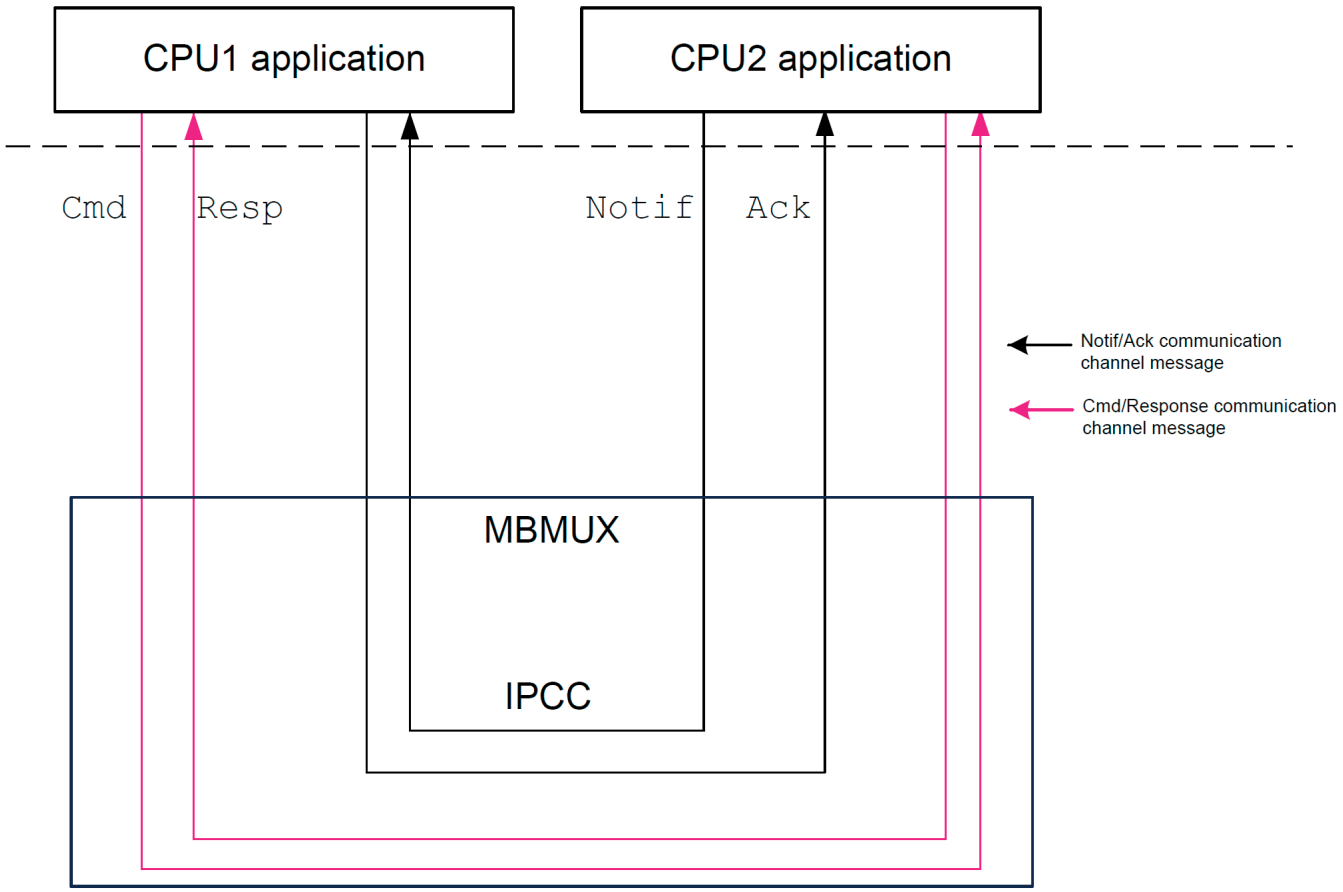
• Notif - уведомление, отправляемое CPU2 на CPU1, состоящее из:
  - Msg ID идентифицирует функцию обратного вызова, вызываемую CPU2, но реализованную на CPU1.

- Ptr buffer params указывает на буфер, содержащий параметры вышеуказанной функции.
- Number of params (Количество параметров)

• Ack, подтверждение, отправленное CPU1 на CPU2, состоит из:

- Msg ID (то же значение, что и Notif Msg ID)
- Return value (Возвращаемое значение) содержит возвращаемое значение указанной выше функции обратного вызова.

Figure 12. Mailbox messages through MBMUX and IPCC channels



14.2 Межядерная память

Межъядерная память - это централизованная память, доступная для обоих ядер и используемая ядрами для обмена данными, параметрами функций и возвращаемыми значениями.

14.2.1 Возможности CPU2

CPU1 должен знать несколько возможностей CPU2, чтобы подробно описать его поддерживаемые функции (например, стек протоколов, реализованный на CPU2, номер версии каждого стека или поддерживаемые регионы).

Эти возможности CPU2 хранятся в таблице features_info. Данные из этой таблицы запрашиваются при инициализации CPU1, чтобы раскрыть возможности CPU2, как показано в отображении RAM.

Таблица features_info состоит из:

• Feat_Info_Feature_Id: имя функции

• Feat_Info_Feature_Version: номер версии функции, используемый в текущей реализации.

MB_MEM2 используется для хранения этих возможностей CPU2.

14.2.2 Последовательность почтового ящика для выполнения функции CPU2 из вызова CPU1

Когда CPU1 необходимо вызвать CPU2 feature_func_X (), feature_func_X () с тем же API должна быть реализована на CPU1:

1. CPU1 отправляет ***команду***, содержащую параметры feature_func_X () в таблице сопоставления:

a. func_X_ID, который был связан с feature_func_X () при инициализации во время регистрации, добавляется в таблицу сопоставления. func_X_ID должен быть известен обоим ядрам: это фиксируется во время компиляции.

b. CPU1 ожидает от CPU2 выполнения функции feature_func_X () и переходит в режим пониженного энергопотребления.

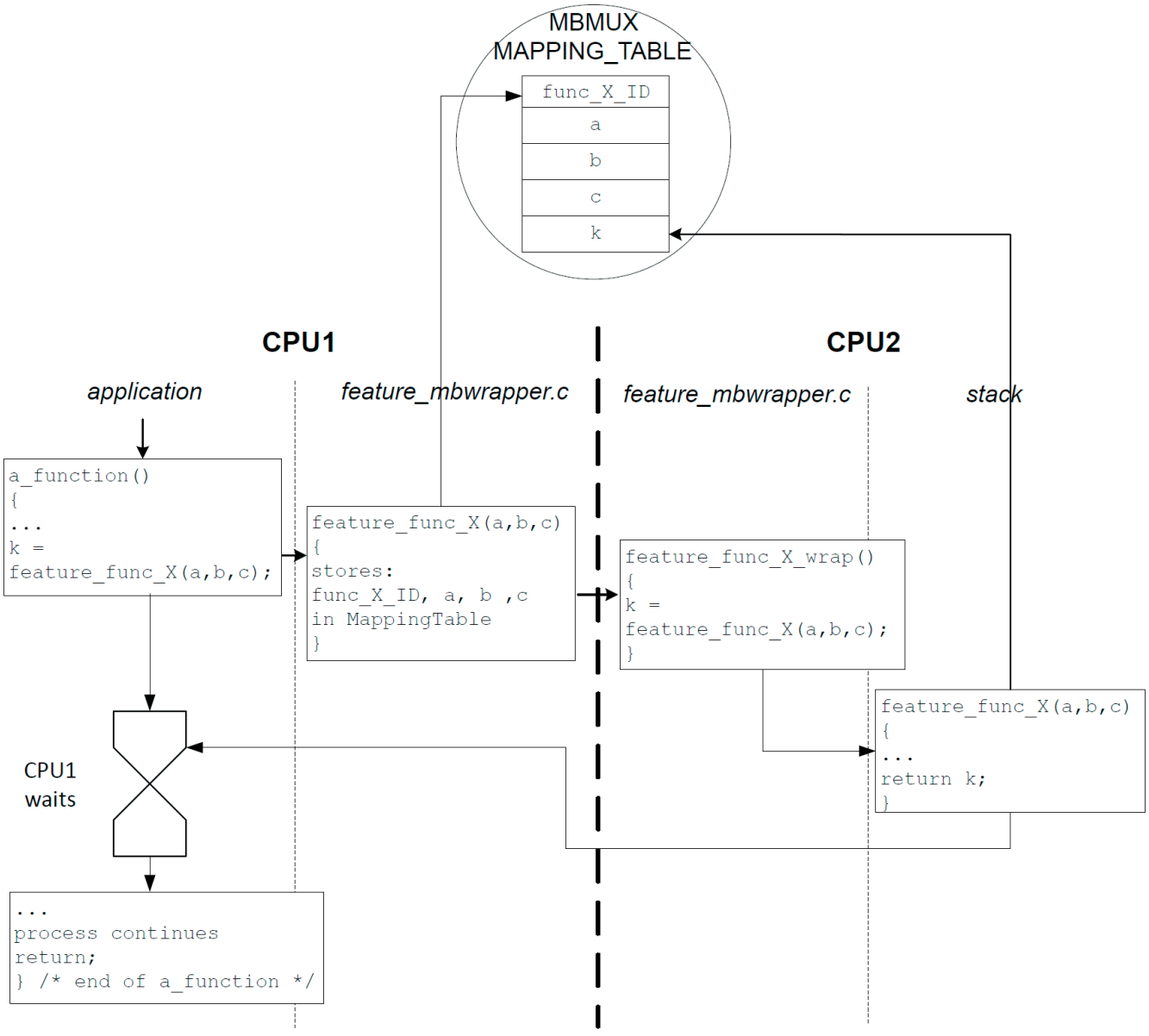
c. CPU2 выходит из спящего режима, если он был в режиме пониженного энергопотребления, и выполняет функцию feature_func_X ().

2. CPU2 отправляет ***ответ*** и заполняет таблицу сопоставления возвращаемым значением:

a. Прерывание IPCC пробуждает CPU1.

b. CPU1 получает возвращаемое значение из таблицы сопоставления.

Figure 13. CPU1 to CPU2 feature_func_X() process



И наоборот, когда CPU2 необходимо вызвать функцию CPU1 feature_func_X_2 (), функция feature_func_X_2 () с тем же API должна быть реализована на CPU2:

1. CPU2 отправляет ***уведомление***, содержащее feature_func_X_2 () в таблице сопоставления.

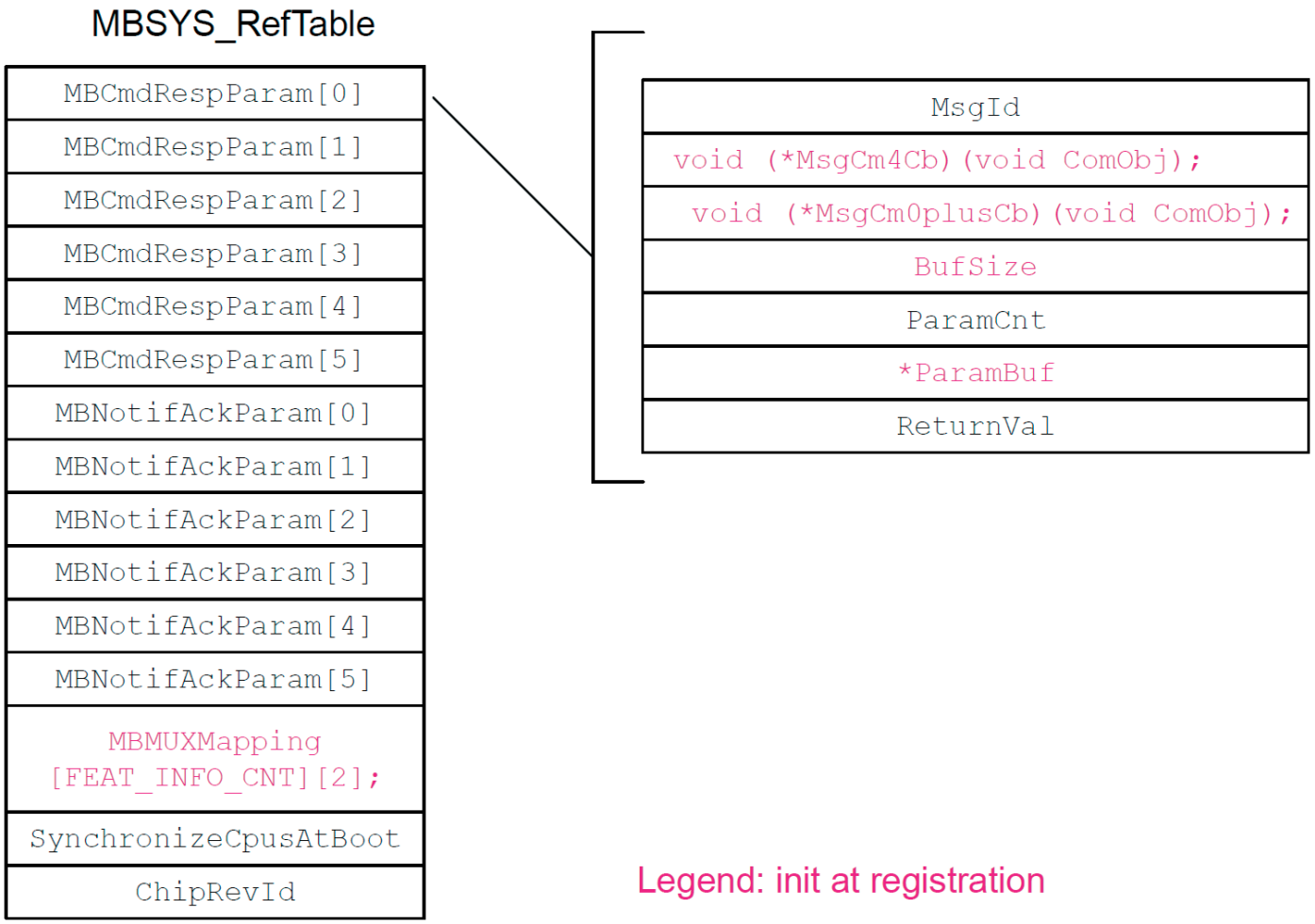
2. CPU1 отправляет ***подтверждение*** и заполняет таблицу сопоставления возвращаемым значением.

14.2.3 Таблица сопоставления

Таблица отображения является общей структурой в области MBMUX на рисунке 13. При отображении RAM отображение памяти обозначается как MAPPING_TABLE.

Таблица связи MBMUX (MBSYS_RefTable) описана на рисунке ниже.

Figure 14. MBMUX communication table



Эта MBSYS_RefTable включает:

• две структуры параметров связи для параметров команды / ответа и уведомления / подтверждения для каждого из основных каналов IPCC.

Каждый параметр связи, как показано в области таблицы сопоставления MBMUX на рисунке 13, состоит из:

- MsgId: идентификатор сообщения feature_func_X ()
- * MsgCm4Cb: указатель на обратный вызов CPU1 feature_func_X ()
- * MsgCm0plusCb: указатель на функцию обратного вызова CPU2 feature_func_X ()

- BufSize: размер буфера
- ParamCnt: номер параметра сообщения
- ParamBuf: указатель сообщения на параметры
- ReturnVal: возвращаемое значение feature_func_X ()

• MBMUXMapping: диаграмма, используемая для сопоставления каналов с объектами.

Этот график заполняется при инициализации MBMUX во время регистрации. Например, если функция радиосвязи связана с Cmd/Response channel number = 1, то MBMUXMapping должен связать [FEAT_INFO_RADIO_ID] [1].

• SynchronizeCpusAtBoot: флаги, используемые для синхронизации обработки CPU1 и CPU2, как показано на диаграмме последовательности на Рисунке 15.

• ChipRevId: хранит идентификатор версии оборудования.

MB_MEM1 используется для отправки параметра command/response set () и для получения возвращаемых значений для CPU1.

14.2.4 Предупреждение о необязательном байте

В код помещается ловушка, чтобы избежать ошибочной загрузки опционного байта (см. Раздел «Ошибка загрузки опционального байта при высокой системной тактовой частоте MSI» в списке ошибок продукта). Ловушку можно удалить, если системные часы установлены ниже или равны 16 МГц.

14.2.5 Отображение RAM

В таблицах ниже подробно показано сопоставление областей RAM CPU1 и CPU2, а также межядерной памяти.

Page index (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
Allocation region/section	CPU1 RAM																CPU1 RAM2_Shared		CPU2 RAM2_Shared		CPU2 RAM2											

1. 2 Kbytes for each page.

Таблица 49. Распределение RAM и общий буфер STM32WL5x

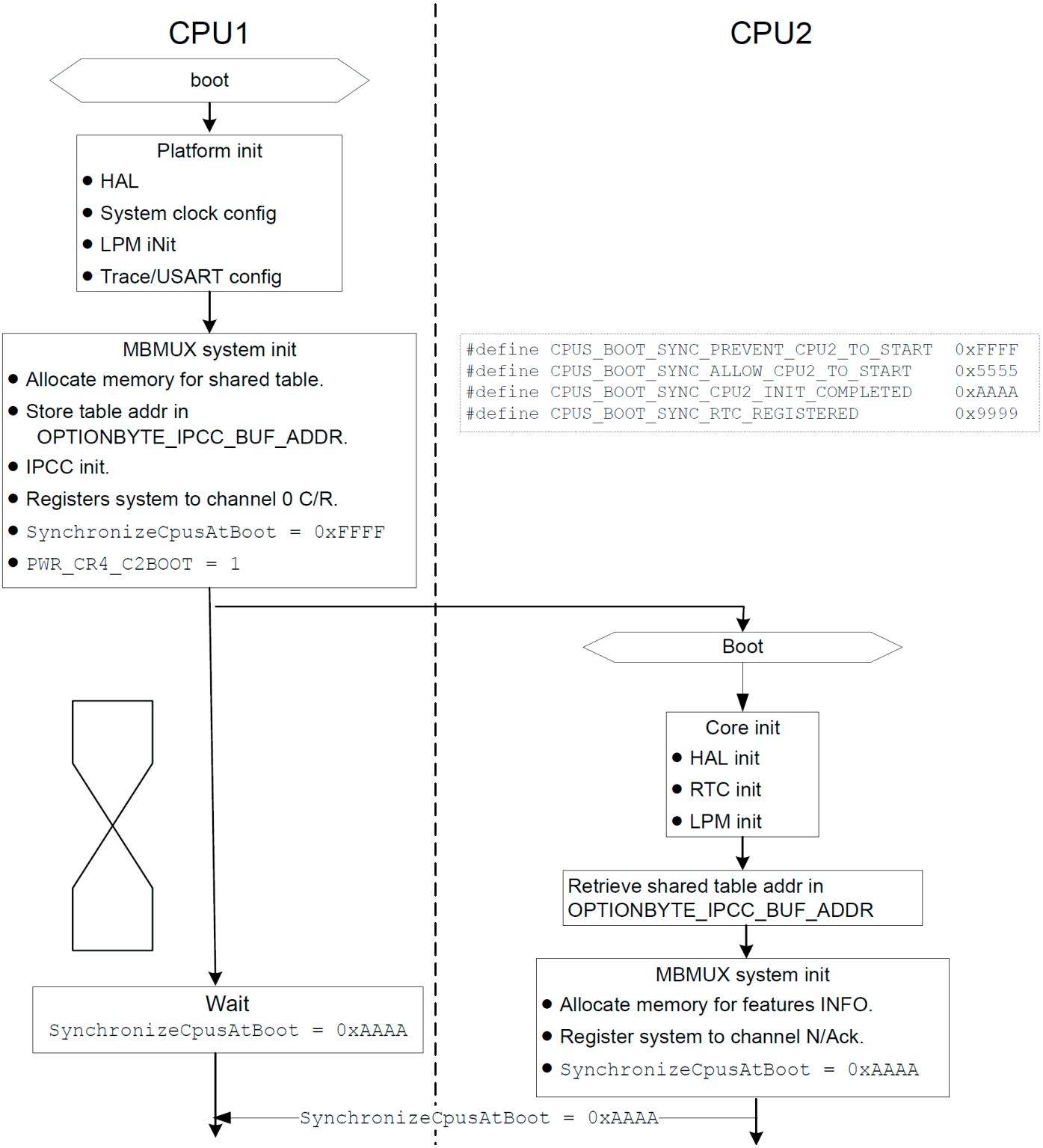
Область процессора	Раздел	Модуль	Выделенный символ	Размер (bytes)	Всего (bytes)
CPU1 RAM	readwrite	-			
	CSTACK				
	HEAP				
CPU1 RAM2 _Shared	MAPPING_TABLE	MBMUX_SYSTEM	MBMUX_ComTable_t MBSYS_RefTable	316	316
	MB_MEM1	MBMUX_LORAWAN	uint32_t aLoraCmdRespBuff[]	60	524
			uint32_t aLoraNotifAckBuff[]	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	
CPU2 RAM2_Shared	MB_MEM2	MBMUX_LORAWAN	FEAT_INFO_Param_t Feat_Info_Table	80	-
		MBMUX_LORAWAN	FEAT_INFO_List_t Feat_Info_List	8	
	MB_MEM3(1)	MBMUX_TRACE	uint8_t ADV_TRACE_Buffer[]	1024	-
		MBMUX_LORAWAN	LoraInfo_t loraInfo	16	
		LMH_MBWRAPPER	uint8_t aLoraMbWrapShare2Buffer[]	280	
		RADIO_MBWRAPPER	uint8_t aRadioMbWrapRxBuffer[]	256	
CPU2 RAM2	readwrite	-			
	CSTACK				
	HEAP				

1. *Этот новый раздел предотвращает чрезмерное использование флэш-памяти для инициализации этих переменных ОЗУ BSS с помощью STM32CubeIDE.*

14.3 Последовательность запуска

Последовательность запуска CPU1 и CPU2 подробно описана на рисунке ниже. Последовательность шагов следующия:

Figure 15. Startup sequence

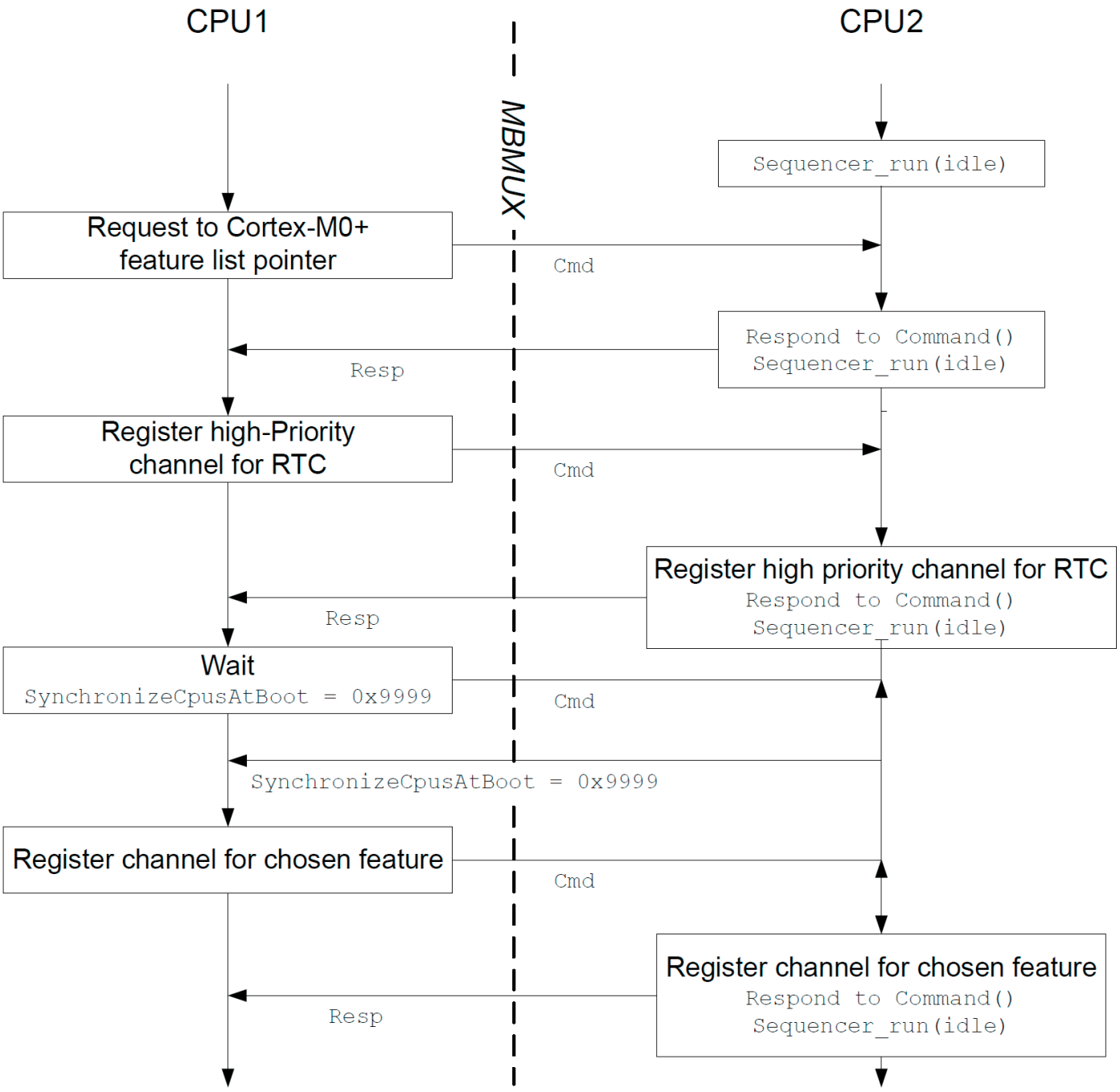


1. CPU1, то есть главный процессор в этой последовательности инициализации:
   a. выполняет инициализацию платформы.
   b. инициализирует систему MBMUX.
   c. устанавливает флаг PWR_CR4_C2BOOT в 1, который запускает CPU2.
   d. ожидает, что CPU2 установит флаг SynchronizeCpusAtBoot в 0xAAAA.

2. CPU2 загружается и:
   a. выполняет инициализацию ядра.
   b. получает адрес общей таблицы.
   c. инициализирует систему MBMUX.
   d. устанавливает SynchronizeCpusAtBoot на 0xAAAA, чтобы сообщить CPU1, что он завершил свою последовательность инициализации и что он готов.
3. CPU1 подтверждает это уведомление CPU2.

Затем инициализируются оба ядра, и инициализация продолжается через MBMUX, как показано на рисунке ниже.

Figure 16. MBMUX initialization

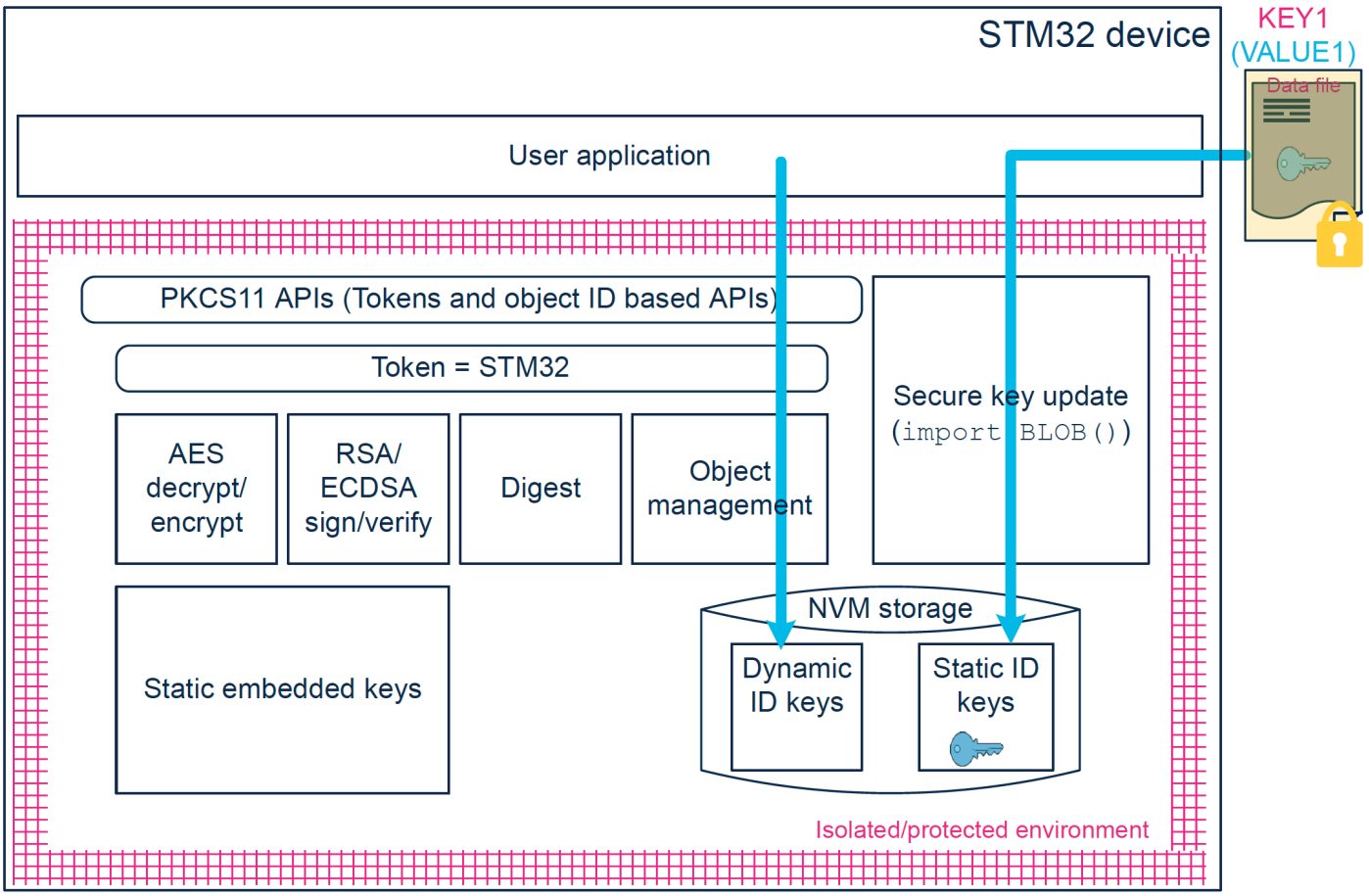


15 Службы управления ключами (KMS)

Службы управления ключами (KMS) предоставляют криптографические службы через стандартные API PKCS # 11 (разработанные OASIS), которые используются для абстрагирования значения ключа для вызывающей стороны (с использованием идентификатора объекта, а не непосредственно значения ключа).

KMS может выполняться внутри защищенной / изолированной среды, чтобы гарантировать, что значение ключа не может быть доступно для неавторизованного кода, работающего вне защищенной / изолированной среды, как показано на рисунке ниже.

Figure 17. KMS overall architecture



Дополнительные сведения см. в разделе KMS в руководстве пользователя «Начало работы с SBSFU» STM32CubeWL (UM2767).

Чтобы активировать модуль KMS, для параметра KMS_ENABLE необходимо установить значение 1 в параметрах проекта компилятора C / C ++.

KMS поддерживает только API PKCS # 11, перечисленные ниже:

- Функции управления объектами (создание / обновление / удаление)
- Функции шифрования / дешифрования AES (алгоритмы CBC, CCM, ECB, GCM, CMAC)
- Функции переваривания
- Функции подписи / проверки RSA и ECDSA
- Ключевые функции управления (генерация / вывод ключей)

15.1 Типы ключей KMS

KMS управляет тремя типами ключей, но используются только два следующих:

• Статические встроенные ключи

- предопределенные ключи, встроенные в код, которые не могут быть изменены
- неизменяемые ключи

• Ключи NVM_DYNAMIC:

- ключи времени выполнения
- идентификаторы ключей, которые могут быть определены при создании ключей с использованием KMS (DeriveKey () или CreateObject ())
- ключи, которые можно удалить или определить как изменяемые

15.2 Определение ключа KMS

Статические и динамические ключи, используемые стеком, имеют разные размеры.

Статический ключ

Каждый статический ключ состоит из двух следующих элементов:

• заголовок большого двоичного объекта: пять 4-байтовых полей (всего = 20 байтов): version, configuration, blob_size, blob_count и object_id.

• буфер больших двоичных объектов: некоторые обязательные и дополнительные элементы больших двоичных объектов из списка элементов, определенных следующим образом:

Таблица 50. Глобальные элементы BLOB-объектов KMS

Attribute	Value	Необходимость	Size (bytes)	Описание
CKA_CLASS	CKO_SECRET_KEY	Yes	12	Тип элемента blob
CKA_KEY_TYPE	CKK_AES	Yes	12	Тип ключа
CKA_VALUE	"KEY_VALUE"	Yes	24	Значение ключа (формат uint32_t)
CKA_DERIVE	TRUE/FALSE	No	12	Необязательные параметры для включения / отключения возможностей по умолчанию. Эти поля имеют значение ИСТИНА, если не определены.
CKA_ENCRYPT	TRUE/FALSE	No	12	
CKA_DECRYPT	TRUE/FALSE	No	12	
CKA_COPYABLE	TRUE/FALSE	No	12	
CKA_EXTRACTABLE	TRUE/FALSE	No	12	
CKA_LABEL	"UNIQUE LABEL"	Yes	12 for static key 16 for dynamic key	Уникальная метка

Пример:

Статический ключ, состоящий из заголовка большого двоичного объекта и восьми элементов большого двоичного объекта (CKA_CLASS, CKA_KEY_TYPE, CKA_VALUE, CKA_LABEL, CKA_DERIVE, CKA_DECRYPT, CKA_COPYABLE, и CKA_EXTRACTABLE), использует общий размер 128 байт (заголовок большого двоичного объекта = 20 байтов, а буфер большого двоичного объекта = (12 x 7 + 24) = 108 байтов).

Динамический ключ

Каждый динамический ключ состоит из трех элементов, заголовка данных, заголовка большого двоичного объекта и буфера большого двоичного объекта, с:

• заголовок данных: восемь 4-байтовых полей (всего = 32 байта): magic1, magic2, slot, instance, next, data_type, size, и контрольная сумма.

Пример:

Динамический ключ, состоящий из заголовка данных, заголовка большого двоичного объекта и пяти элементов большого двоичного объекта(CKA_CLASS, CKA_KEY_TYPE, CKA_VALUE, CKA_LABEL, и CKA_EXTRACTABLE) использует общий размер 128 байтов (заголовок данных = 32 байта, заголовок большого двоичного объекта = 20 байтов, а буфер большого двоичного объекта = (12 + 12 + 24 + 12 + 16) = 76 байтов.

Примечание:

• Динамическая память NVM всегда начинается с элемента заголовка начальных данных.

• При каждом «удалении» динамического ключа (например, устаревший ключ заменяется новым значением ключа) в память записывается дополнительный заголовок данных, чтобы объявить, что предыдущий экземпляр больше не может использоваться.

15.3 Ключи LoRaWAN

В приложениях STM32CubeWL KMS используются на CPU2 только в двухъядерных приложениях. Корневые ключи выбираются как встроенные статические ключи. Все производные ключи являются ключами NVM_DYNAMIC.

Для стека LoRaWAN неизменяемые корневые ключи подробно описаны в таблице ниже.

Таблица 40. Статические ключи LoRaWAN с атрибутами blob

Attribute	Key				
	LoRaWAN_Zero_Key	LoRaWAN_APP_Key	LoRaWAN_NWK_Key	LoRaWAN_NWK_S_Key (ABP only)	LoRaWAN_APP_S_Key (ABP only)
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"				

Все остальные ключи являются изменяемыми ключами, генерируемыми NVM_DYNAMIC, подробно описанными в таблице ниже.

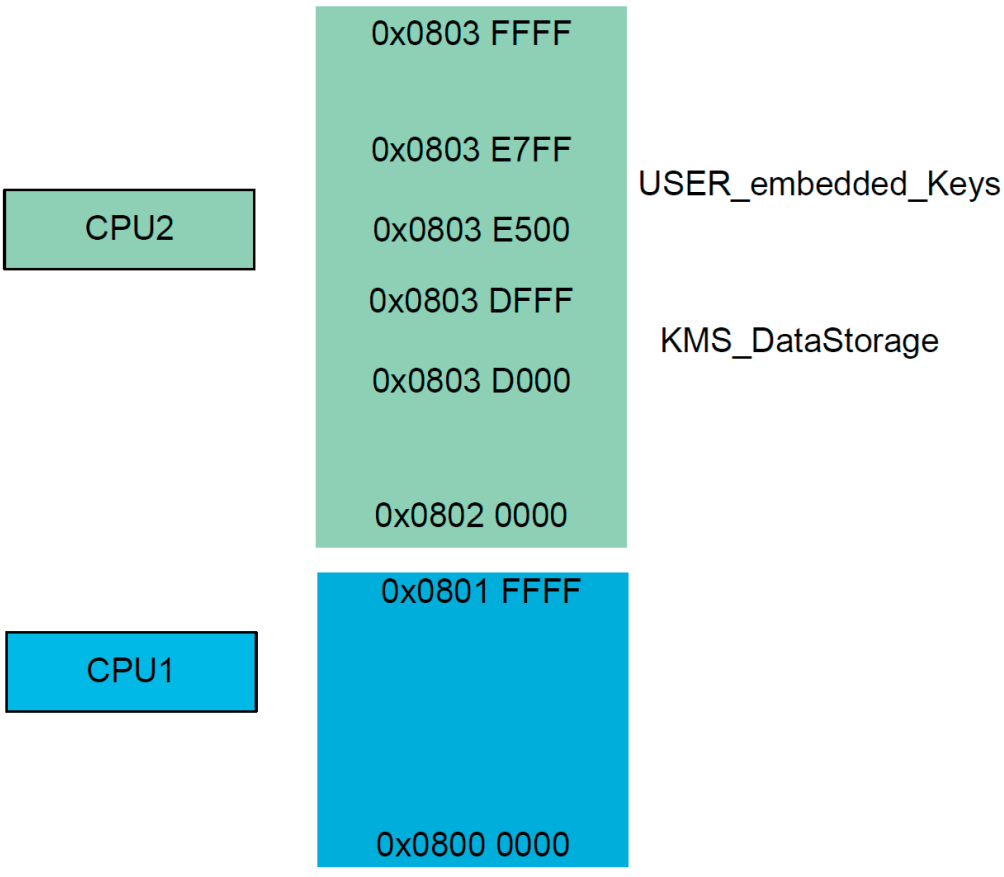
Таблица 41. Динамические ключи LoRaWAN с атрибутами blob

Attribute	Key							
	LoRaWAN_NWK_S_Key (OTAA only)	LoRaWAN_APP_S_Key (OTAA only)	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"							

15.4 Отображение памяти ключа KMS для пользовательских приложений

Статические встроенные ключи соответствуют USER_embedded_Keys (используются для корневых ключей). Они размещаются в специальном хранилище данных во флэш-памяти / ПЗУ. Файлы компоновщика для пользовательских приложений располагаются от 0x0803 E500 до 0x0803 E7FF, как показано на рисунке ниже.

Figure 18. ROM memory mapping



Ключи NVM_DYNAMIC помещаются в область хранения данных ключей KMS, KMS_DataStorage.

Общая область хранения данных должна составлять 4 Кбайта, как описано в Разделе 13.5. Они были размещены от: 0x0803 D000 до 0x0803 DFFF, как показано на рисунке ниже. Этот размер может быть увеличен, если потребуется больше ключей.

15.5 Как определить размер NVM для хранилища данных KMS

NVM состоит из 2-килобайтных страниц. Из-за двойной буферизации (механизм эмуляции flip / flop EEPROM) каждой странице нужен «двойник». Таким образом, минимум для NVM составляет 4 Кбайта. Размер выделения определяется в файле компоновщика.

Файлы компоновщика, предлагаемые приложениями, используют минимально допустимый размер (2 x 2 Кбайта). Связанные с этим ограничения / недостатки поясняются ниже. Пользователь должен установить размер NVM в зависимости от потребностей конкретного приложения.

Приложения используют NVM только для хранения динамических ключей. Динамический ключ LoRaWAN с заголовком данных, заголовком большого двоичного объекта и буфером большого двоичного объекта из пяти элементов занимает 108 байтов. Пустой NVM инициализируется глобальным 32-байтовым заголовком данных, и для каждого устаревшего ключа записывается дополнительный 32-байтовый заголовок данных, чтобы объявить предыдущий экземпляр непригодным для использования.

Учитывая приведенные выше значения, можно оценить, сколько ключей может быть сохранено в 2 Кбайтах: (2048 - 32) / (108 + 32) = 14,4 ==> 14 динамических ключей могут быть сохранены на странице памяти размером 2 Кбайта до того, как операция провалится.

В конфигурации пользовательского приложения используется только NVM_DYNAMIC. NVM_STATIC можно заполнить через BLOB-объект, но он не поддерживается пользовательскими приложениями.

NVM_DYNAMIC может содержать производные ключи (через C_DeriveKey ()) и корневые ключи (через C_CreateObject ()).

Приложение LoRaWAN генерирует:

- два производных ключа, каждый из которых соединяется в режиме ABP
- четыре производных ключа, каждый из которых соединяется в режиме OTAA

До десяти одновременно активных производных ключей могут быть сгенерированы в более сложных сценариях (например, при настройке многоадресной рассылки). Если пользователь хочет написать одно приложение, которое использует более 14 ключей, дополнительные страницы NVM должны быть выделены файлу компоновщика.

Меньше размер NVM, чем больше записывается и стирается NVM, тем короче становится его ожидаемый срок службы.

Уничтожение ключа не означает, что этот ключ стирается, но что этот ключ помечен как уничтоженный и не копируется при следующем переключении триггера. Флаг уничтожения также занимает несколько байтов NVM: после уничтожения восьми ключей оставшееся место меньше четырех ключей.

Для сценария, в котором при каждом соединении генерируются четыре ключа, а после уничтожения предыдущего ключа соединения ожидаемый срок службы оценивается следующим образом:

• На третьем сеансе соединения выводятся четыре новых ключа, но на странице 1 нет места для последнего ключа. Все четыре клавиши (которые все еще активны) помещаются на страницу 2. Страница 1 стирается сразу, так как страницу NVM можно только полностью стереть.

• При 5-м соединении также стирается страница 2, а ключи сохраняются обратно на страницу 1. После 40 000 объединений две страницы NVM были стерты 10.000 раз, что является расчетным сроком службы сектора Flash.

• Если предполагается, что пользовательское приложение будет подключаться слишком часто (например, каждые 2 часа), ожидаемое время жизни NVM составит 80 000 часов (около девяти лет). Если процесс присоединения выполняется один раз в день, срок службы намного превышает десять лет.

Чем больше количество запрошенных производных ключей, одновременно активных (не уничтоженных), тем менее эффективен механизм триггера.

В заключение, для приложений, которым необходимо сохранять продолжительность жизни NVM, рекомендуется, чтобы размер NVM был больше, чем количество одновременно активных ключей (не уничтоженных).

Примечание:

Устаревшие ключи должны быть уничтожены, в противном случае, если страница 1 полностью заполнена активными ключами, переключение триггера не может быть выполнено и генерируется ошибка.

15.6 Файлы конфигурации KMS для сборки приложения

KMS используются в приложении LoRaWAN путем установки кода
#define LORAWAN_KMS 1
в CM0PLUS / LoRaWAN / Target / lorawan_conf.h

Следующие файлы должны быть заполнены информацией о ключах стека SubGHz_Phy:

• структуры встроенных ключей, определенные в CM0PLUS / Core / Inc / kms_platf_objects_config.h

• встроенные дескрипторы объектов, связанные с ключами стека SubGHz_Phy, использование модулей KMS, определенных в CM0PLUS / Core / Inc / kms_platf_objects_interface.h

15.7 Встроенные ключи

Встроенные ключи стека протокола SubGHz_Phy должны храниться в области ПЗУ, в которой безопасное дополнительное программное обеспечение (такое как SBSFU, Secure Boot и Firmware Update) обеспечивает конфиденциальность и целостность данных. Дополнительные сведения о SBSFU см. в руководстве по интеграции SBSFU на STM32CubeWL (AN5544).

Эти встроенные ключи расположены в ПЗУ, как показано на рисунке 18. Отображение памяти ПЗУ.

16 Как защитить приложение LoRaWAN

В документе [7] описывается, как защитить двухъядерное приложение LoRaWAN с помощью фреймворка SBSFU. (Application note How to secure LoRaWAN and Sigfox with STM32CubeWL (AN5682))

17 Системные характеристики

17.1 Объем памяти

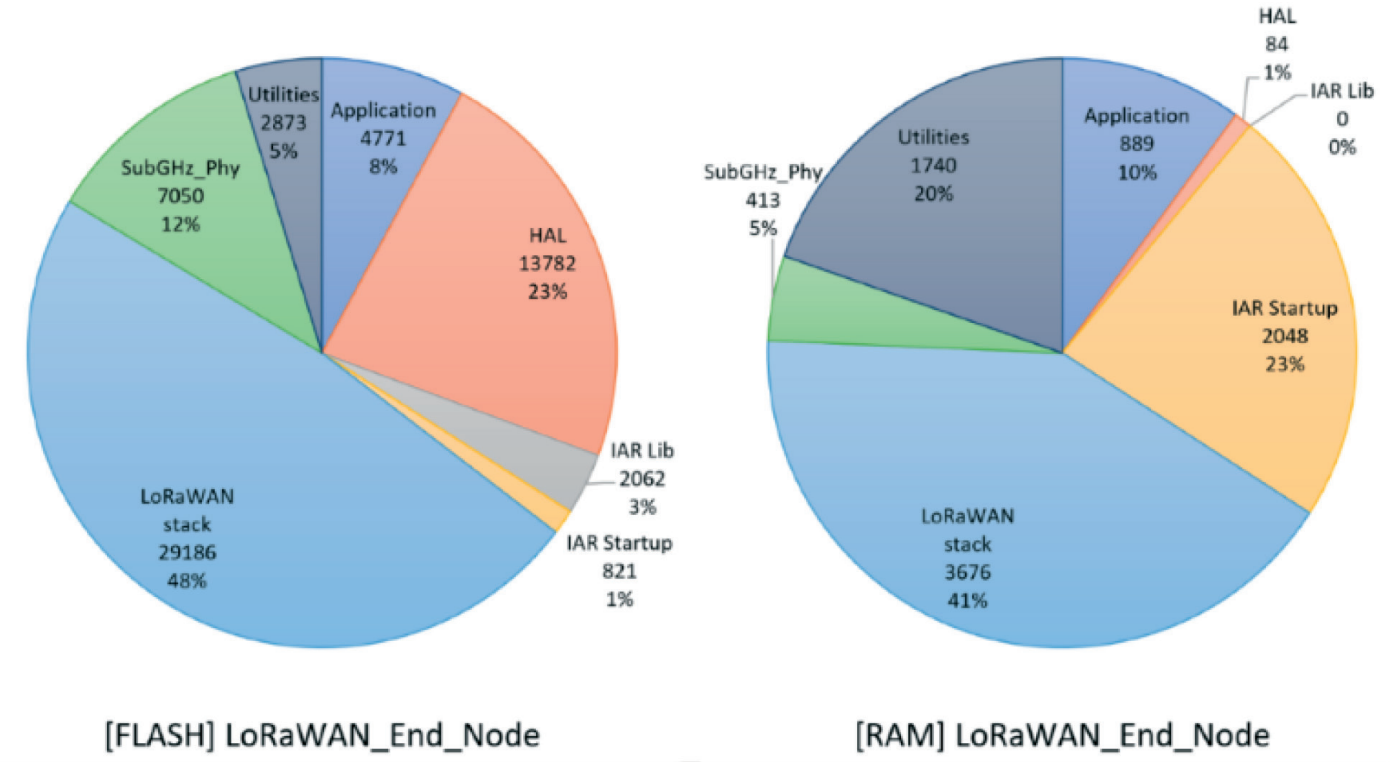
Значения в таблице ниже измерены в следующей конфигурации компилятора IAR Embedded Workbench® (EWARM версии 8.30.1):

- уровень оптимизации 3 по размеру
- опция отладки отключена
- опция трассировки: VLEVEL_M (трассировка отладки включена)
- цель: NUCLEO-WL55JC1
- Приложение LoRaWAN_End_Node
- LoRaMAC класса A
- LoRaMAC, регион EU868 и US915

Таблица 42. Значения объема памяти для приложения LoRaWAN_End_Node

Project module	Flash memory (bytes)	RAM (bytes)	Description
Application	4771	889	Основные, прикладные и целевые компоненты
LoRaWAN stack	29186	3676	Интерфейс промежуточного программного обеспечения Lmhandler, шифрование, MAC и регион
HAL	13782	84	STM32WL HAL и LL драйверы
Utilities	2873	1740	Все службы STM32 (секвенсор, сервер времени, менеджер с низким энергопотреблением, трассировка, память)
SubGHz_Phy	7050	413	Радиоинтерфейс промежуточного программного обеспечения
IAR lib	2062	0	Собственные библиотеки IAR
IAR startup	821	2048	Int_vect, процедуры инициализации, таблица инициализации, CSTACK и HEAP
Total application	60545	8850	Объем памяти для приложения LoRaWAN_End_Node

Figure 19. Flash memory and RAM footprint

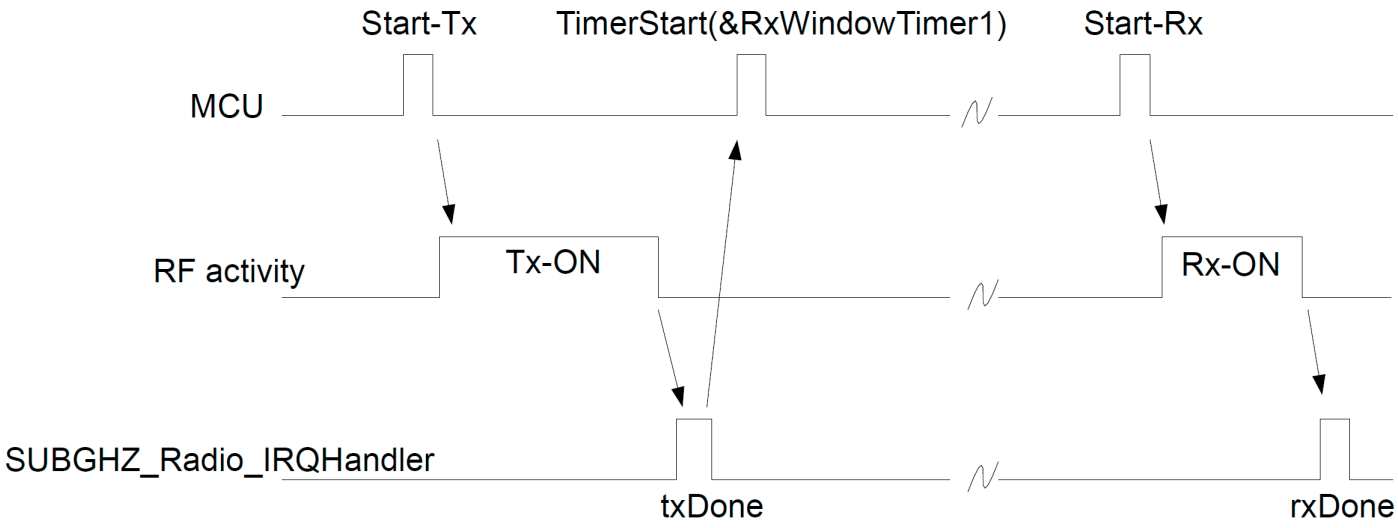


17.2 Ограничения в реальном времени

Асинхронный протокол LoRa RF подразумевает соблюдение строгих рекомендаций по синхронизации Tx / Rx (см. Рисунок ниже).

Плата STM32WL Nucleo (NUCLEO-WL55JC) оптимизирована для прозрачного для пользователя времени малой блокировки и быстрой автоматической калибровки. BSP объединяет время запуска передатчика и ограничения времени запуска приемника (см. Раздел 4 «Платы BSP STM32WL Nucleo»).

Figure 20. Rx/Tx time diagram



Запускается канал окна Rx. Окно Rx1 открывается через 1 секунду (± 20 мкс) после спадающего фронта txDone. Окно Rx2 открывается через 1 секунду (± 20 мкс) после спадающего фронта txDone.

JOIN_ACCEPT использует 5-секундную (± 20 мкс) и 6-секундную задержку после окончания модуляции восходящего канала.

Должен соблюдаться текущий приоритет уровня прерывания планирования. Другими словами, все новые пользовательские прерывания должны иметь приоритет прерывания выше, чем Radio IRQ_interrupt, чтобы избежать задержки полученного времени запуска.

15.3 Потребляемая мощность

Измерение энергопотребления выполняется для платы STM32WL Nucleo NUCLEO-WL55JC1.

Настройка измерений:

- без отладки
- уровень трассировки VLEVEL_OFF (без трассировки)
- нет SENSOR_ENABLED

Результаты измерений:

- Типичное потребление в режиме Stop 2 = 2 мкА (см. Рисунок 22).
- Типичное потребление с TCXO в Tx = 23 мА (см. Рисунок 21).
- Типичное потребление с TCXO в Rx = 7 мА (см. Рисунок 21).

Цифры измерений: мгновенное потребление за 30 секунд.

Figure 21. NUCLEO-WL55JC1 current consumption versus time

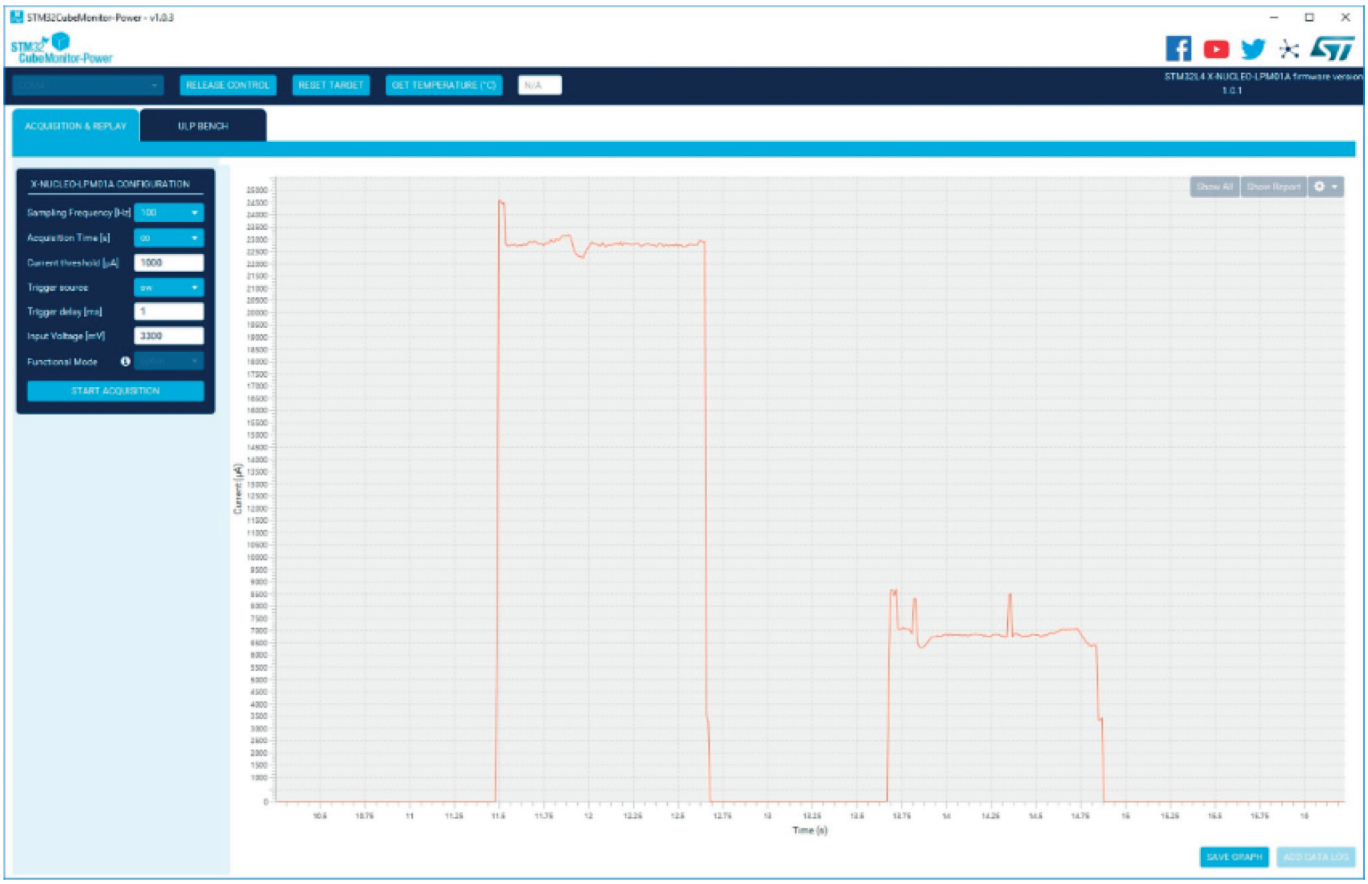


Figure 22. NUCLEO-WL55JC1 current consumption in Stop 2 mode

Table 43. Document revision history

Date	Version	Changes
10-Dec-2019	1	Initial release.
27-Apr-2020	2	Global update of the document structure and content.
17-Nov-2020	3	Updated: • *Figure 1. Project file structure* • Note in *Section 4 BSP STM32WL Nucleo-73 boards* • Intro of *Section 6 LoRaWAN middleware description* • Title and intro of *Section 6.6 Middleware LmHandler application function* • *Table 22* and *Table 23* • *Section 8.1.1 Activation methods and keys* • *Table 38. Switch options for End_Node application configuration* • *Table 39. Switch options for AT_Slave application configuration* • *Table 40. Switch options for PingPong application configuration* • *Section 13.1 Memory footprint* Added: • *Table 26. LmHandler process* • *Section 11 Dual-core management* • *Section 12 Key management services (KMS)* Removed tables "Board unique ID" and "Board random seed" from *Section 6.7*.
18-Feb-2021	4	Updated: • `lora_app.c` renamed `lora_app.h` in Section 8.1.2, Section 8.1.4, Section 8.1.5, Section 8.1.6, Section 8.1.7, Section 8.1.8 and Section 8.1.9 • Table 38. Switch options for End_Node application configuration • Table 39. Switch options for AT_Slave application configuration
9-Jul-2021	5	Updated: • Overview section renamed Section 1 General information • Intro of Section 11 SubGhz_Phy_PingPong application • Figure 7. SubGhz_Phy_PingPong application setup • Intro of Section 14 Dual-core management • Section 14.1.3 MBMUX messages • Section 14.2.5 RAM mapping • Section 15.2 KMS key definition • Section 15.3 LoRaWAN keys • Figure 17. ROM memory mapping • Section 15.5 How to size NVM for KMS data storage • Section 15.6 KMS configuration files to build the application • Section 17.1 Memory footprint • Intro of Section 11 SubGhz_Phy_PingPong application • Figure 7. SubGhz_Phy_PingPong application setup • Intro of Section 14 Dual-core management • Section 14.1.3 MBMUX messages • Section 14.2.5 RAM mapping • Section 15.2 KMS key definition • Section 15.3 LoRaWAN keys • Figure 17. ROM memory mapping • Section 15.5 How to size NVM for KMS data storage • Section 15.6 KMS configuration files to build the application • Section 17.1 Memory footprint

Date	Version	Changes
9-Jul-2021 (cont'd)	5	Added: • Seven tables in Section 6.6 Application callbacks • Section 12 SubGhz_Phy_Per application • Section 16 How to secure a LoRaWAN application
21-Feb-2022	6	Updated: • Table 1. Acronyms and terms • Section 2 STM32CubeWL overview • Section 6 LoRaWAN middleware description Added: • Section 5 LoRaWAN stack description • Section 5.3.3 Required STM32 peripherals to drive the radio • Section 9.1 LoRaWAN user code sections description • Section 9.2.11 Context management • Section 13 LoRaWAN context management description • Section 13.1 NVM context managment data API definition • Section 11.2 Device configuration • Section 11.2.1 Modulation definition • Section 11.2.2 Payload length • Section 11.2.3 Region and frequency • Section 11.2.4 Bandwith, spreading factor and data rate • Section 11.2.5 Preamble length Removed: • STM32CubeWL architecture