life.augmented

AN5406

アプリケーション・ノート

STM32CubeWL を使用した LoRa® アプリケーションの構築方法

概要

このアプリケーション・ノートでは、STM32WL シリーズ・マイクロコントローラをベースにして特定の LoRa® アプリケーションを構築するために必要なすべての手順を説明します。

LoRa® は、超低ビットレートの長距離通信を可能とするように設計され、長寿命バッテリ駆動のセンサを実現する無線通信ネットワークです。LoRaWAN® は、LoRa® ネットワークとの相互運用を確保する通信およびセキュリティのプロトコルを定義します。

STM32CubeWL マイクロコントローラ・パッケージに含まれるファームウェアは、LoRaWAN® という名前の LoRa Alliance® 仕様プロトコルに準拠し、次の主な機能を備えています。

- アプリケーションと統合可能
- 低消費電力 LoRa® ソリューションを簡単にアドオン可能
- 極めて低い CPU 負荷
- 遅延要件なし
- 小さな STM32 メモリ・フットプリント
- 低消費電力タイミング・サービス

STM32CubeWL マイクロコントローラ・パッケージのファームウェアは STM32Cube HAL ドライバに基づいています。

このドキュメントでは、STM32WL Nucleo ボード NUCLEO_WL55JC を使用する顧客向けのアプリケーション例を提供します（高周波数帯域の場合は注文コード NUCLEO-WL55JC1、低周波数帯域用の場合は注文コード NUCLEO-WL55JC2）。

このアプリケーション・ノートの情報を最大限に利用してアプリケーションを作成するには、ユーザが STM32 マイクロコントローラと LoRa® テクノロジーに精通し、低消費電力管理やタスク・シーケンスなどのシステム・サービスを理解している必要があります。

1 一般情報

STM32CubeWL は Arm® Cortex®-M プロセッサをベースとした STM32WL シリーズ・マイクロコントローラで動作します。

注 Arm は、米国内およびその他の地域にある Arm Limited（またはその子会社）の登録商標です。

arm

表 1. 頭字語と用語

項目（略称）	定義
ABP	Activation by personalization
ADR	Adaptive data rate（アダプティブ・データ・レート）
BSP	Board support package（ボード・サポート・パッケージ）
DC/DC	DC/DC コンバータ
HAL	Hardware abstraction layer（ハードウェア抽象化レイヤ）
IoT	Internet of things（モノのインターネット）
IPCC	Inter-Processor communication controller（プロセッサ間通信コントローラ）
IRQ	割込みリクエスト
LBT	Lesten before talk
LoRa	Long range radio technology（長距離無線技術）
LoRaWAN	LoRa wide-area network（LoRa 広域ネットワーク）
LPWAN	低消費電力広域ネットワーク
MAC	Media access control（メディア・アクセス制御）
MCPS	MAC common part sublayer
MIB	MAC information base
MLME	MAC sublayer management entity
MSC	Message sequence chart（メッセージ・シーケンス図）
OTAA	Over-this-air activation（無線による有効化）
PA	Power amplifier（パワー・アンプ）
PER	Packet error rate（パケット・エラー・レート）
PRBS	Pseudo-random bit sequence（疑似ランダム・ビット・シーケンス）
RSSI	Receive signal strength indicator（受信信号強度インジケータ）
Rx	受信
SWD	シリアル・ワイヤ・デバッグ
<target>	STM32WL Nucleo ボード（NUCLEO-WL55JC）
Tx	送信

参照文献

[1] LoRa Alliance® 仕様プロトコルによる LoRaWAN 1.0.3 仕様 — 2018 年 1 月

[2] アプリケーションノート「STM32CubeWL 用の LoRaWAN® AT コマンド」（AN5481）

[3] ユーザマニュアル「STM32WL HAL および下位層ドライバの説明」（UM2642）

[4] IEEE Std 802.15.4TM - 2011。低レート・ワイヤレス・パーソナル・エリア・ネットワーク（LR-WPAN）

[5] アプリケーションノート「STM32CubeWL によるロング・パケット 」（AN5687）

[6] アプリケーションノート「STM32CubeWL での SBSFU の統合ガイド （KMS を含む）（AN5544）

[7] アプリケーションノート「STM32CubeWL での LoRaWAN® と Sigfox™ の保護方法（AN5682）

LoRa 規格

LoRa および LoRaWAN の推奨の詳細については、ドキュメント [1] を参照してください。

2 STM32CubeWL の概要

STM32CubeWL マイクロコントローラ・パッケージのファームウェアには、次のリソースが含まれています（図 1 を参照）。

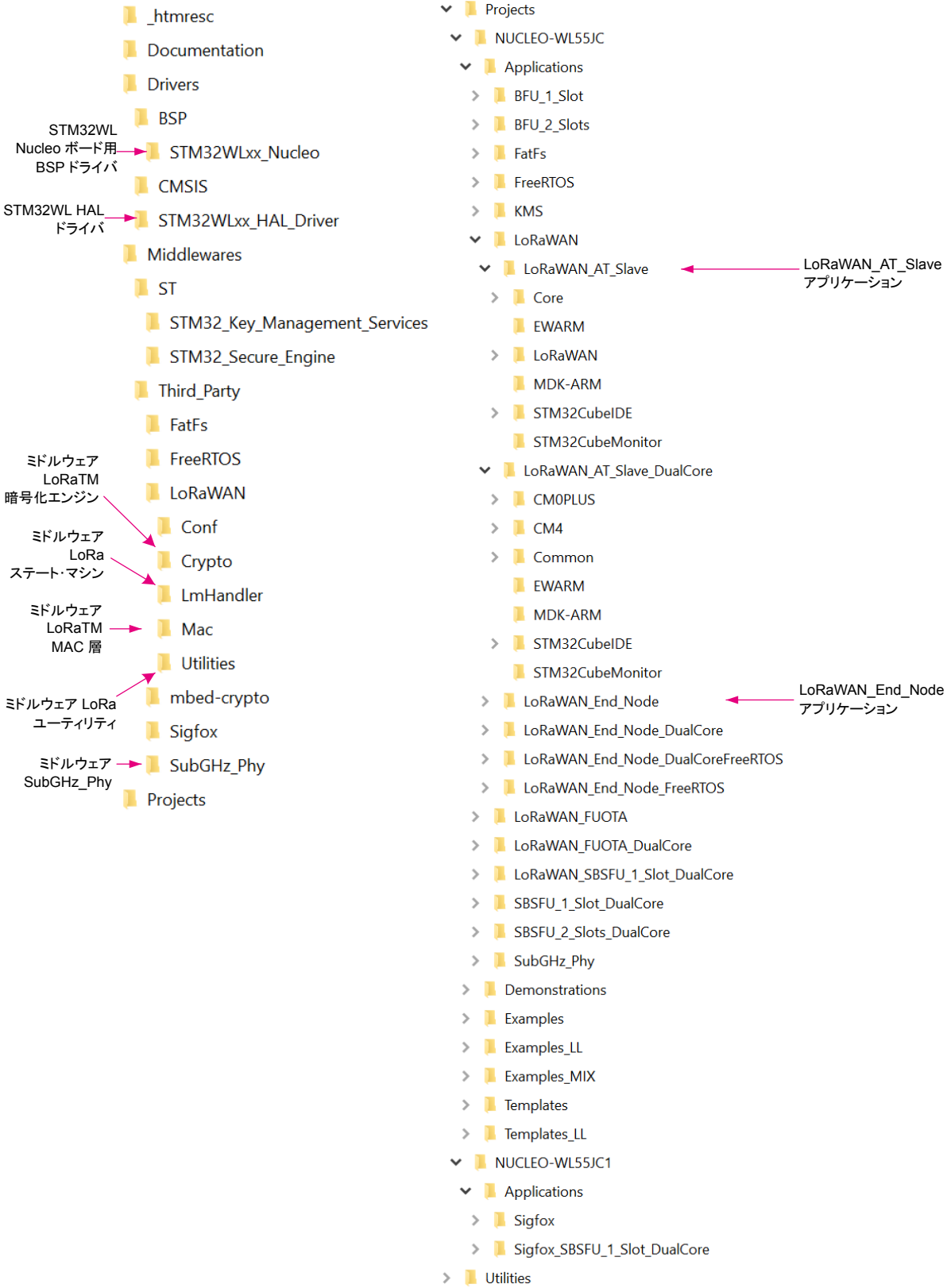
- ボード・サポート・パッケージ：STM32WLxx_Nucleo ドライバ
- STM32WLxx_HAL_Driver
- ミドルウェア：
  - LoRaWAN（以下を含む）：
    - LoRaWAN 層
    - LoRa ユーティリティ
    - LoRa ソフトウェア暗号化エンジン
    - LoRa ステート・マシン
  - 無線および radio_driver インタフェースを含む SubGHz_Phy 層ミドルウェア
- LoRaWAN アプリケーション：
  - LoRaWAN_AT_Slave（SingleCore および DualCore）
  - LoRaWAN_End_Node（SingleCore、DualCore、FreeRTOS 付きの SingleCore、および FreeRTOS 付きの DualCore）
- SubGHz_Phy アプリケーション：
  - SubGHz_Phy_PingPong（SingleCore および DualCore）
  - SubGHz_Phy_Per（SingleCore）

さらに、このアプリケーションは、以下により効率的なシステム統合を実現します。

- タスクをバックグラウンドで実行し、動作を行わない場合には低消費電力モードに移行するシーケンサ
- （STOP モードと STANDBY モードにおいて）RTC 上で動作する仮想タイマをアプリケーションに提供するタイマ・サーバ

詳細については、セクション 8 ユーティリティの説明を参照してください。

図 1. プロジェクト·ファイルの構造



3 SubGHz HAL ドライバ

このセクションでは、SubGHz HAL に焦点を当てます（タイマや GPIO など、その他の HAL 関数については詳しく説明しません）。

SubGHz HAL は、Sub-GHz 無線ペリフェラルのすぐ上に位置します（図 3. 静的 LoRa アーキテクチャ を参照）。

Sub-GHz HAL ドライバは、シンプルなワンショットのコマンド指向アーキテクチャ（完全なプロセスはありません）に基づいています。そのため、LL ドライバは定義されていません。

この SubGHz HAL ドライバは、次の主要部分で構成されています。

- ハンドル、初期化および設定のデータ構造
- 初期化 API
- 設定および制御 API
- MSP とイベント･コールバック
- SUBGHZ_SPI に基づいたバス入出力操作（固有のサービス）

HAL API は、主にバス･サービスに基づいてワンショット動作でコマンドを送信するため、RESET/READY HAL 状態以外の機能ステート･マシンは使用されません。

3.1 SubGHz リソース

次の HAL SubGHz API は、無線の初期化時にコールされます。

- SUBGHZ_HandleTypeDef ハンドル構造体を宣言します。
- `HAL_SUBGHZ_Init(&hUserSubghz)` API をコールして、Sub-GHz 無線ペリフェラルを初期化します。
- `HAL_SUBGHZ_MspInit()` API を実装して、SubGHz 下位レベル･リソースを初期化します。
  - PWR の設定：Sub-GHz 無線ペリフェラルのウェイクアップ信号を有効にします。
  - NVIC の設定：
    - NVIC 無線 IRQ 割込みを有効にします。
    - Sub-GHz 無線の割込みの優先度を設定します。

次の HAL 無線割込みは、`stm32wlxx_it.c` ファイル内でコールされます。

- SUBGHZ_Radio_IRQHandler 内の `HAL_SUBGHZ_IRQHandler`。

3.2 SubGHz データ転送

Set コマンドの操作は、`HAL_SUBGHZ_ExecSetCmd();` API によりポーリング･モードで実行されます。

Get Status 操作は、`HAL_SUBGHZ_ExecGetCmd();` API によりポーリング･モードで実行されます。

読出し/書込みレジスタ･アクセスは、次の API によりポーリング･モードで実行されます。

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

4 BSP STM32WLNucleo ボード

この BSP ドライバは、RF スイッチの設定と制御、TCXO 設定、DC/DC 設定など、無線 RF サービスを管理するための一連の関数を備えています。

注 無線ミドルウェア（SubGHz_Phy）は、`radio_board_if.c/h` インタフェース·ファイルを介して無線 BSP をインタフェース接続します 。カスタム·ユーザ·ボードを使用する場合、次のいずれかを実行することを推奨します。

- 最初のオプション
  - `BSP/STM32WLxx_Nucleo/` ディレクトリをコピーします。
  - 以下を使用して、ユーザ BSP API の名前を変更して更新します。
    - ユーザ RF スイッチの設定と制御（ピン制御、ポート数など）
    - ユーザ TCXO 設定
    - ユーザ DC/DC 設定
  - IDE プロジェクトで、STM32WLxx_Nucleo BSP ファイルをユーザ BSP ファイルに置換
- 2 番目のオプション
  - `Core/Inc/platform.h` で `USE_BSP_DRIVER` を無効化し、BSP 関数を `radio_board_if.c` に直接実装します。

4.1 周波数帯域

STM32WL シリーズでは、次の 2 種類の Nucleo ボードを使用できます。

- NUCLEO-WL55JC1：高周波帯域、865 MHz ~ 930 MHz の周波数
- NUCLEO-WL55JC2：低周波数帯域、470 MHz ~ 520 MHz の周波数

868 MHz でコンパイルされたファームウェアを低周波数帯域ボードで実行しようとした場合、RF 性能が非常に低下することが予想されます。

ファームウェアでは、動作するボードの帯域はチェックされません。

4.2 RF スイッチ

STM32WL Nucleo ボードには RF 3 ポート·スイッチ（SP3T）が内蔵されており、同じボードで次のモードに対応します。

- 高電力送信
- 低電力送信
- 受信

表 2. BSP 無線スイッチ

機能	説明
`int32_t HW_TS_RTC_ReadLeftTicksToCount(void)`	RF スイッチを初期化します。
`BSP_RADIO_ConfigRFSwitch(BSP_RADIO_Switch_TypeDef Config)`	RF スイッチを設定します。
`int32_t BSP_RADIO_DeInit (void)`	RF スイッチの初期化を解除します。
`int32_t BSP_RADIO_GetTxConfig(void)`	ボード設定として、高電力、低電力、またはその両方を返します。

RF の状態とスイッチの設定を下の表に示します。

表 3. RF の状態とスイッチの設定

RF の状態	FE_CTRL1	FE_CTRL2	FE_CTRL3
高電力送信	低	高	高
低電力送信	高	高	高
受信	高	低	高

4.3 RF ウェイクアップ時間

Sub-GHz 無線ウェイクアップ時間は、次の API により回復します。

表 4. BSP 無線ウェイクアップ時間

機能	説明
uint32_t BSP_RADIO_GetWakeUpTime(void)	RF_WAKEUP_TIME 値を返します。

ユーザは、アプリケーションによって異なるタイムアウトを使用して RADIO_SET_TCXOMODE コマンドを設定し、TCXO を起動する必要があります。
タイムアウト値は radio_conf.h で更新できます。デフォルトのテンプレート値は次のとおりです。

```
#define RF_WAKEUP_TIME                          1U
```

4.4 TCXO

ユーザ・アプリケーションにはさまざまなタイプのオシレータを実装することができます。STM32WL Nucleo ボードでは、周波数精度を高めるために温度補償クリスタルオシレータ(TCXO)が使用されます。

表 5. BSP 無線 TCXO

機能	説明
uint32_t BSP_RADIO_IsTCXO (void)	IS_TCXO_SUPPORTED 値を返します。

STM32WL Nucleo BSP で TCXO モードを定義するには、Core/Inc/platform.h 内で USE_BSP_DRIVER を選択します。
ユーザがこの値を更新したい場合(NUCLEO ボードに準拠していない)、または BSP が存在しない場合、TXCO モードを radio_board_if.h 内で更新できます。デフォルトのテンプレート値は次のとおりです。

```
#define IS_TCXO_SUPPORTED                       1U
```

4.5 電力調整

ユーザ・アプリケーションに応じて、電力調整に LDO または SMPS(DC/DC とも呼ばれる)が使用されます。STM32WL Nucleo ボードでは SMPS が使用されます。

表 6. BSP 無線 SMPS

機能	説明
uint32_t BSP_RADIO_IsDCDC (void)	IS_DCDC_SUPPORTED 値を返します。

STM32WL Nucleo BSP で DC/DC モードを定義するには、Core/Inc/platform.h 内で USE_BSP_DRIVER を選択します。
ユーザがこの値を更新したい場合(NUCLEO ボードに準拠していない)、または BSP が存在しない場合、DC/DC モードを radio_board_if.h 内で更新できます。デフォルトのテンプレート値を以下に定義します。

```
#define IS_DCDC_SUPPORTED                       1U
```

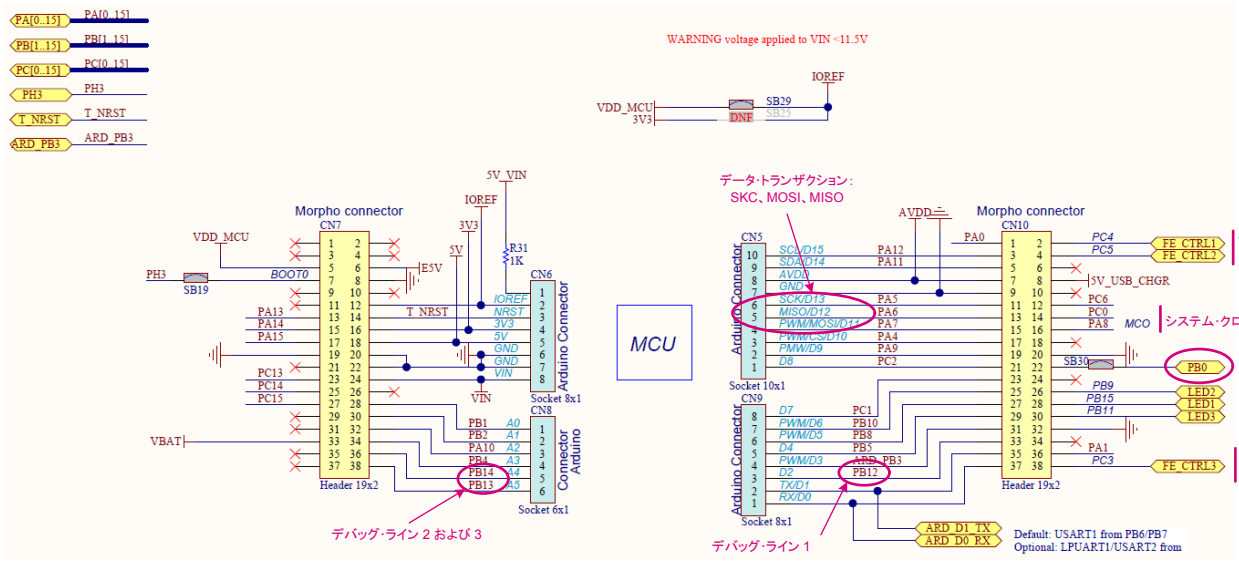
ボード上の SMPS を無効化するには、IS_DCDC_SUPPORTED = 0 を設定します。

4.6 STM32WL Nucleo ボードの回路図

下の図では、STM32WL Nucleo ボード（MB1389 リファレンス・ボード）の回路図の詳細を示し、いくつかの有用な信号を強調しています。

- PC4、PC5、および PC3 の制御スイッチ
- PB0 の TCXO 制御電圧ピン
- PB12、PB13、および PB14 のデバッグ・ライン
- PA8 のシステム・クロック
- SCK → PA5
- MISO → PA6
- MOSI → PA7

図 2. NUCLEO-WL55JC の回路図



5 LoRaWAN スタックの説明

STM32CubeWL マイクロコントローラのパッケージのファームウェアには、次のような STM32WL リソースが含まれます。

- STM32WLxx Nucleo ドライバ
- STM32WLxx HAL ドライバ
- LoRaWAN ミドルウェア
- SubGHz 物理層ミドルウェア
- LoRaWAN アプリケーション例
- ユーティリティ

STM32 マイクロコントローラ用の LoRaWAN スタック·ミドルウェアは、いくつかのモジュールに分割されます。

表 7. LoRaWAN スタックの説明

モジュール	説明	場所
LoRaMAC 層	リンク·レイヤの仕様を実装します。	`Middlewares\Third_Party\LoRaWAN\Mac`
リージョンレイヤ	LoRaMAC 層モジュールの依存インタフェースとしてリージョナルパラメータ仕様を実装します。	`Middlewares\Third_Party\LoRaWAN\Mac\Region`
LoRa 暗号化	AES/CMAC アルゴリズムを実装し、SecureEngine エレメントとインタフェース接続します。	`Middlewares\Third_Party\LoRaWAN\Crypto`
LmHandler	LoRaMac Handler パブリック·インタフェース、認証仕様および FUOTA パッケージを実装します。	`Middlewares\Third_Party\LoRaWAN\LmHandler`
LoRa ユーティリティ	共通ユーティリティ関数を実装します。	`Middlewares\Third_Party\LoRaWAN\Utilities`

LoRaWAN のいくつかの機能が、LoRa Allicance プロトコル仕様に準拠して実装されます。

- リンク·レイヤ仕様:
  - オンボード LoRaWAN クラス A、クラス B、およびクラス C のプロトコル·スタック
  - OTAA または ABP(activation-by-personalization)のいずれかによるエンドデバイス有効化
  - アダプティブ·データ·レート(ADR)をサポート
- リージョナルパラメータの仕様:
  - EU 868MHz ISM バンド ETSI 準拠
  - EU 433MHz ISM バンド ETSI 準拠
  - US 915MHz ISM バンド FCC 準拠
  - KR 920 MHz ISM バンド(韓国政府により規定)
  - RU 864 MHz ISM バンド(ロシア規制により規定)
  - CN 779 MHz および CN470Mhz ISM バンド(中国政府により規定)
  - AS 923 MHz ISM バンド(アジアの政府により規定)
  - AU 915 MHz ISM バンド(オーストラリア政府により規定)
  - IN 865 MHz ISM バンド(インド政府により規定)

さらに、LoRaWAN スタックには以下が統合されています。

- 下記の仕様に準拠した認証ソリューション
- パワーオフ時のコンテキストの喪失を防ぐ NVM コンテキスト管理
- 無線の STANDBY/SLEEP 状態別の低消費電力管理

5.1 LoRaWAN 仕様のバージョン

リンク･レイヤ仕様、およびリージョナルパラメータ仕様は、LoRa-Alliance で定義されています。

LoRaWAN スタックは、Semtech スタックの供給内容に基づいて、2 つの異なるバージョンを実装します。

- LoRaWAN リンク･レイヤ 1.0.3 仕様 + LoRaWAN 1.0.3 リージョナルパラメータ仕様
- LoRaWAN リンク･レイヤ 1.0.4 仕様（TS001-1.0.4）+ LoRaWAN 2-1.0.1 リージョナルパラメータ仕様（RP002-1.0.1）

使用する LoRaWAN Server 設定に適応したバージョンのスタックを選択する必要があります。

5.2 LoRaWAN 認証

NUCLEO-WL55JC ボードと STM32CubeWL ファームウェア･モデム･アプリケーションを含むシステムは、LoRaWAN TestHouse によって検証され、EU868、IN865、KR920、AS923、および US915 帯域の認証に合格しました。

LoRaWAN 認証の実装は、使用する LoRaWAN 仕様のバージョンに依存します。

表 8. LoRaWAN 認証

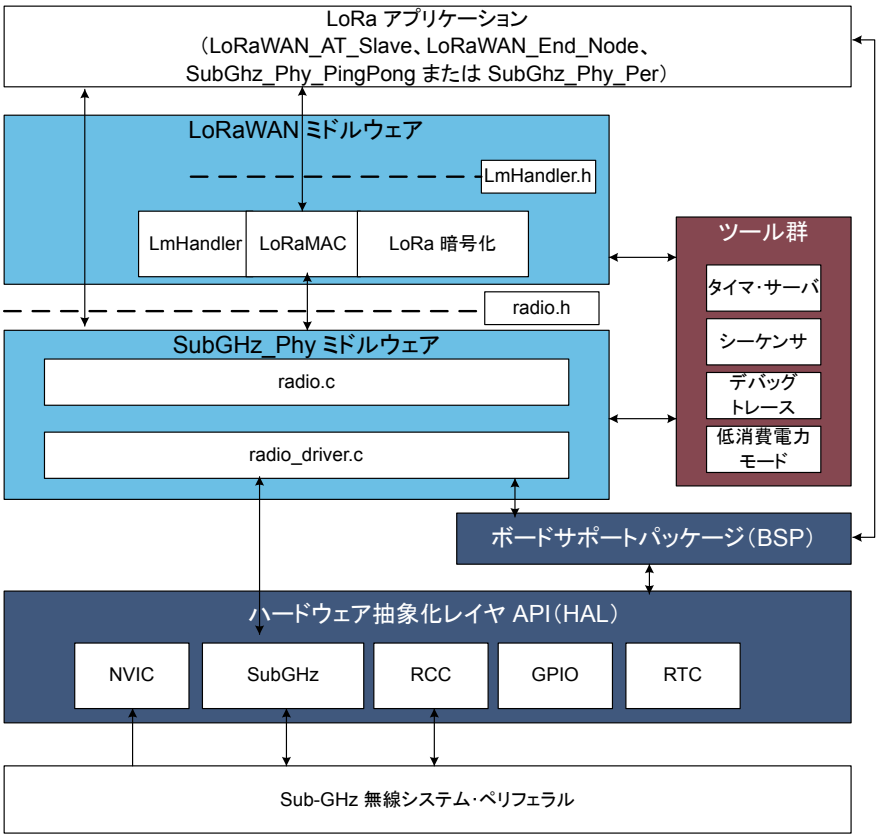
LoRaWAN 仕様のバージョン	LoRaWAN 認証の仕様
LoRaWAN リンク･レイヤ 1.0.3 の仕様	AS923MHz ISM バンド･デバイスの LoRa Alliance エンド･デバイス認証要件 v1.1.0
	EU863-870 MHz ISM バンド･デバイスの LoRa Alliance エンド･デバイス認証要件 v1.6.0
	インドの 865-867 MHz ISM バンド用 LoRa Alliance エンド･デバイスの認証要件：v1.1.0
	LoRa Alliance エンド･デバイス認証要件（韓国 920-923 MHz ISM バンド･デバイス）v1.2.0
	LoRa Alliance エンド･デバイス認証要件（米国およびカナダの 902-928 MHz ISM バンド）v1.5.0
LoRaWAN リンク･レイヤ 1.0.4 仕様（TS001-1.0.4）	すべてのリージョンにおける LoRaWAN 1.0.4 エンド･デバイス認証要件
	LoRaWAN 認証プロトコル 1.0.0 仕様（TS009-1.0.0）

5.3 アーキテクチャ

5.3.1 静的ビュー

下の図に、LoRa アプリケーションのファームウェアの構成を示します。

図 3. 静的 LoRa アーキテクチャ



HAL は、STM32Cube API を使用して、アプリケーションに必要なマイクロコントローラ·ハードウェアを駆動します。LoRa ミドルウェアには、LoRa アプリケーションの実行に必要な特定のハードウェアのみが含まれています。

RTC は、低消費電力モード(STOP 2 モード)でも動作し続ける集中型時間ユニットを提供します。RTC アラームは、タイマ·サーバで管理されている特定のタイミングでシステムをウェイクアップするために使用します。

SubGHz_Phy ミドルウェアは、HAL SubGHz を使用して無線を制御します(上図を参照)。詳細については、セクション 7 を参照してください

MAC は、802.15.4 モデルを使用して SubGHz_Phy を制御します。MAC は SubGHz_Phy ドライバと接続し、タイマ·サーバを使用して時間指定されたタスクを追加または削除します。

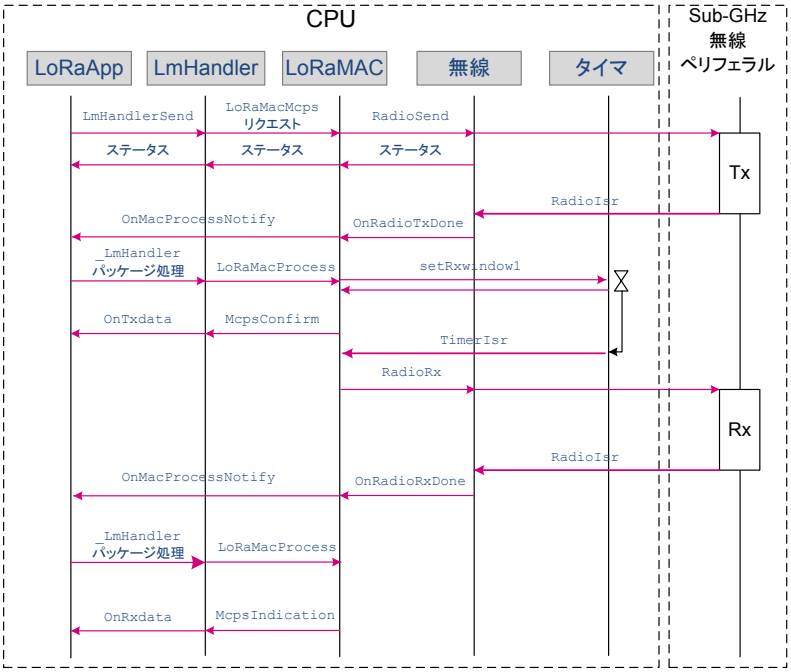
LoRa クラス A を制御するステート·マシンは影響を受けやすいため、MAC とアプリケーション間に中間レベルのソフトウェアが挿入されます(`LmHandler.c`)(上図の LoRaMAC ドライバを参照)。ユーザは限定された数の API を使用して、アプリケーション·レベルで自由にクラス A ステート·マシンを実装できます。詳細については、セクション 6 を参照してください。

アプリケーションは、無限ループを中心として構築され、低消費電力モードの管理、割込みハンドラ(アラームまたは GPIO)の実行、および実行する必要があるタスクが存在する場合は LoRa クラス A の呼出しを実行します。

5.3.2 動的ビュー

以下の MSC（メッセージ·シーケンス図）に、クラス A デバイスがアプリケーション·データを送信し、サーバからアプリケーション·データを受信する様子を示します。

図 4. クラス A Tx および Rx 処理のメッセージ·シーケンス図



無線機でのアプリケーション·データ送信が完了すると、非同期の RadioIRQ によりシステムがウェイクアップされます。ここで RadioIsr によりハンドラ·モードで `txDone` がコールされます。

すべての RadioIsr および MAC タイマにより `LoRaMacProcessNotify` コールバックがコールされ、LoRaMAC 状態の更新と、必要に応じて追加の処理を行うよう、アプリケーション層にリクエストされます。

たとえば、受信の最後に、`rxDone` が ISR（ハンドラ）でコールされますが、復号化を含むすべての Rx パケット処理は、ISR で処理してはなりません。このケースはコール·シーケンスの一例です。Rx1 ウィンドウにデータが受信されなかった場合、別の Rx2 ウィンドウが起動されます。

5.3.3 無線の駆動に必要な STM32 ペリフェラル

Sub-GHz 無線

Sub-GHz 無線ペリフェラルへのアクセスには、`stm32wlxx_hal_subghz` HAL を使用します。

Sub-GHz 無線は、SUBGHZ_Radio_IRQHandler NVIC を介して割込みを発行し、TxDone または RxDone イベントを通知します。その他のイベントは、製品のリファレンス·マニュアルに記載されています。

RTC

RTC（リアルタイム·クロック）カレンダは、32 kHz 外部オシレータからすべての電力モードで動作する 32 ビットカウンタとして使用されます。デフォルトでは、RTC は、毎秒 1024 個のティック（サブセカンド）を生成するようにプログラムされています。RTC のプログラムは、（マイクロコントローラの初回起動時の）ハードウェア初期化中に 1 回行われます。RTC 出力は約 48 日周期に相当する 32bit タイマに制限されています。

注意　ティック時間を変更する場合、1 ms 未満に保つ必要があります。

LPTIM

LPTIM（低消費電力タイマ）は Monarch にのみ使用されます。LPTIM は、Monarch スキャンがリクエストされたときにセットされ、LSE クロックを使用して、16384 Hz で割込みを発行します。

6 LoRaWAN ミドルウェアの説明

6.1 LoRaWAN ミドルウェアの初期化

LoRaMAC 層を初期化するには、LoRaMacInitialization API を使用します。この API では、LoRaMAC 層のプリアンブル·ランタイムと、MCPS および MLME サービスのコールバック·プリミティブの両方が初期化されます（下の表を参照）。

表 9. LoRaWAN ミドルウェアの初期化

機能	説明
LoRaMacStatus_t LoRaMacInitialization (LoRaMacPrimitives_t *primitives, LoRaMacCallback_t *callback, LoRaMacRegion_t region)	LoRaMAC 層モジュールを初期化します。 （セクション 6.3 ミドルウェア MAC 層のコールバックを参照）

6.2 ミドルウェア MAC 層 API

提供されている API は、IEEE802.15.4-2011 で規定されている「プリミティブ」の定義に従っています（ドキュメント [4] を参照）。

LoRaMac とのやりとりは、要求/確認と指示/応答のアーキテクチャにより行われます。アプリケーション層が要求し、LoRaMAC 層が確認プリミティブでこれを確認します。逆に、イベントが発生した場合は、LoRaMAC 層がアプリケーション層に指示プリミティブで通知します。

アプリケーション層は、指示に対して応答プリミティブで応答します。したがって、すべての確認または指示は、コールバックを使用して実装されます。

LoRaMAC 層で提供されるサービスは次のとおりです。

- MCPS サービス
  一般に、LoRaMAC 層では、データ送信とデータ受信に MCPS サービスを使用します。

表 10. MCPS サービス

機能	説明
LoRaMacStatus_t LoRaMacMcpsRequest (McpsReq_t* mcpsRequest, bool allowDelayedTx)	Tx データの送信を要求します。

- MLME サービス
  LoRaMAC 層では、MLME サービスを使用して、LoRaWAN ネットワークを管理します。

表 11. MLME サービス

機能	説明
LoRaMacStatus_t LoRaMacMlmeRequest (MlmeReq_t *mlmeRequest)	リンク·チェックのための JOIN リクエストを生成します。

- MIB サービス
  MIB には、重要なランタイム情報（MIB_NETWORK_ACTIVATION、MIB_NET_ID など）が格納され、LoRaMAC 層の設定（MIB_ADR、MIB_APP_KEY など）が保持されます。

表 12. MIB サービス

機能	説明
`LoRaMacStatus_t LoRaMacMibSetRequestConfirm (MibRequestConfirm_t *mibSet)`	LoRaMac 層の属性を設定します。
`LoRaMacStatus_t LoRaMacMibGetRequestConfirm (MibRequestConfirm_t *mibGet)`	LoRaMac 層の属性を取得します。

6.3 ミドルウェア MAC 層のコールバック

アプリケーションによって実装される LoRaMAC ユーザ・イベント関数プリミティブ（コールバックとも呼ばれる）は、次のとおりです。

表 13. LoRaMacPrimitives_t 構造体の説明

機能	説明
`void (*MacMcpsConfirm) (McpsConfirm_t *McpsConfirm)`	McpsRequest に対する応答。
`Void (*MacMcpsIndication) (McpsIndication_t* McpsIndication, LoRaMacRxStatus_t* RxStatus)`	受信したパケットが使用可能なことをアプリケーションに通知します。
`void (*MacMlmeConfirm) (MlmeConfirm_t *MlmeConfirm)`	LoRaWAN ネットワークを管理します。
`void (*MacMlmeIndication) (MlmeIndication_t* MlmeIndication, LoRaMacRxStatus_t* RxStatus)`	MAC レスポンスが使用可能であることを MAC 層に通知します。

6.4 ミドルウェア MAC 層のタイマ

表 14. MAC タイマ・イベント

機能	説明
`void OnRxWindow1TimerEvent (void* context)`	最初の Rx ウィンドウ・タイマ・イベント時に RxWindow1Delay によって実行されます。 ノーマル・フレーム：RxWindowXDelay = ReceiveDelayX – RADIO_WAKEUP_TIME JOIN フレーム：RxWindowXDelay = JoinAcceptDelayX – RADIO_WAKEUP_TIME
`void OnRxWindow2TimerEvent (void* context)`	最初の Rx ウィンドウ・タイマ・イベント時に RxWindow2Delay によって実行されます。
`void OnTxDelayedTimerEvent (void* context)`	デューティ・サイクル遅延 Tx タイマ・イベント時に実行されます。
`void OnAckTimeoutTimerEvent (void* context)`	確認応答タイムアウト・タイマによって AckTimeout タイマ・イベント時に実行されます。 パケットの再送信に使用されます（LoRaWAN バージョン v1.0.3 のみ）。

機能	説明
`void OnRetransmitTimeoutTimerEvent (void* context)`	確認応答タイムアウト·タイマによって AckTimeout タイマ·イベント時に実行されます。パケットの再送信に使用されます（LoRaWAN バージョン v1.0.4 のみ）。

6.5 ミドルウェア LmHandler アプリケーション関数

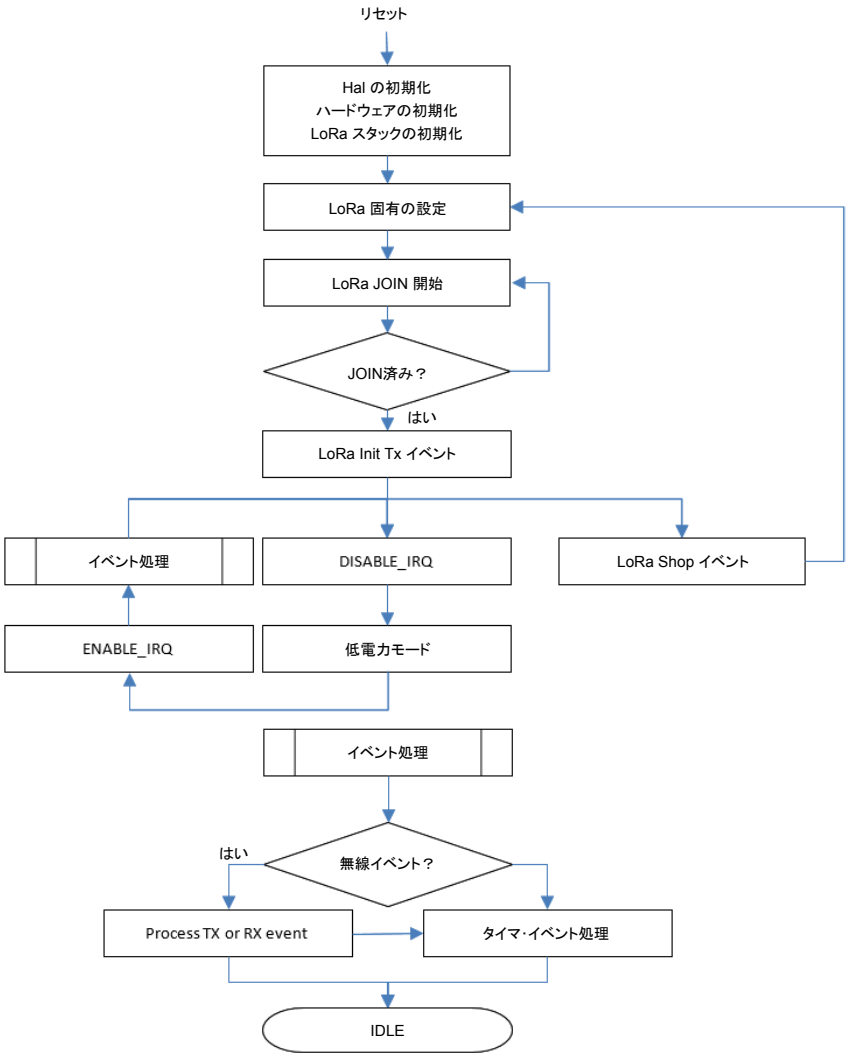
MAC へのインタフェースは、MAC インタフェース `LoRaMac.h` ファイルを介して、次のいずれかのモードで実行されます。

- 標準モード
  インタフェース·ファイル（LoRaMAC ドライバ、図 3 を参照）が提供されるため、ユーザは LoRa ステート·マシンを気にせずに開始できます。このファイルは、次の場所にあります。
  `Middlewares\Third_Party\LoRaWAN\LmHandler\LmHandler.c` また、以下を実装します。
  - LoRaMAC サービスにアクセスするための一連の API
  - LoRa 認定試験ケース（アプリケーション層からは見えません）
- アドバンスト·モード
  MAC 層に直接アクセスするには、ユーザ·ファイルに MAC を記述します。

6.5.1 動作モデル

LoRaWAN_End_Node に対して提案されている動作モデルは、「タイマ駆動」を含む「イベント駆動」パラダイムに基づいています。されます（下の図を参照）。LoRa システム の動作は、タイマ·イベントか、無線イベントとガード·トランジションのいずれかによってトリガされます。

図 5. 動作モデル



次のセクションでは、 LoRaMAC サービスにアクセスするために使用される LoRaWAN_End_Node および LoRaWAN_AT_Slave API について詳しく説明します。対応するインタフェース·ファイルの格納場所は、次のとおりです。

`Middlewares\Third_Party\LoRaWAN\LmHandler\LmHandler.c`

ユーザは、これらの API を使用してアプリケーションを実装する必要があります。

LoRaWAN_End_Node アプリケーションの例は、以下に記述されています。

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

LoRaWAN_AT_Slave アプリケーションの例は、以下に記述されています。

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

6.5.2 アプリケーションの主な関数の定義

表 15. LmHandler の主な関数

機能	説明
`LmHandlerErrorStatus_t LmHandlerInit ( LmHandlerCallbacks_t *handlerCallbacks, uint32_t fwVersion)`	LoRa 有限ステート·マシンを初期化します。
`LmHandlerErrorStatus_t LmHandlerDeInit(void)`	LoRa ステート·マシンの初期化を行わず、すべてのタイマを停止し、MAC パラメータをリセットし、無線をシャットダウンし、既存のすべてのコールバック·リファレンスを削除します。
`LmHandlerErrorStatus_t LmHandlerConfigure (LmHandlerParams_t *handlerParams)`	すべてのアプリケーション·パラメータを設定します。
`void LmHandlerJoin (ActivationType_t mode, bool forceRejoin )`	OTAA または ABP モードでネットワークへの JOIN リクエストを行います。 forceRejoin フラグを指定すると、LoRaWAN コンテキストを復元できる場合でも、強制的に再 JOIN が行われます。
`LmHandlerFlagStatus_t LmHandlerJoinStatus (void)`	本装置がネットワークに JOIN しているかどうかを確認します。
`void LmHandlerStop (void)`	LoRa プロセスを停止し、新しい設定を待ってから再 JOIN アクションを行います。
`LmHandlerErrorStatus_t LmHandlerHalt (void)`	現在のプロセスを中断して LoRa スタックを停止します。
`LmHandlerErrorStatus LmHandlerRequestClass (DeviceClass_t newClass)`	MAC 層に LoRaWAN クラスの変更を要求します。
`LmHandlerErrorStatus_t LmHandlerSend (LmHandlerAppData_t *appData, LmHandlerMsgTypes_t isTxConfirmed, bool allowDelayedTx)`	アップリンク·フレームを送信します。このフレームは unconfirmed の空のフレームまたは unconfirmed/confirmed のペイロード·フレームのいずれかのフレームになります。
`TimerTime_t LmHandlerGetDutyCycleWaitTime (void)`	現在のデューティ·サイクル待ち時間を取得します。
`LmHandlerErrorStatus_t LmHandlerGetVersion (LmHandlerVersionType_t lmhType, uint32_t *featureVersion)`	LoRaWAN の現在の仕様のバージョンを返します。
`LmHandlerErrorStatus_t LmHandlerNvmDataStore (void)`	NVM データ格納プロセスを開始します（詳細については セクション 13 LoRaWAN コンテキスト管理の説明を参照）。

6.6 アプリケーションのコールバック

下の表にあるコールバックは、LoRaWAN_End_Node と LoRaWAN_AT_Slave アプリケーションの両方で使用されます。

表 16. LmHandlerCallbacks_t コールバック構造体の説明

機能	説明
`uint8_t GetBatteryLevel (void)`	バッテリ·レベルを取得します。
`int16_t GetTemperature (void)`	デバイスの現在の温度（°C 単位）を q7.8 フォーマットで取得します。
`void GetUniqueId (uint8_t *id)`	ボードの 64 ビット·ユニーク ID を取得します。

機能	説明
uint32 GetDevAddr (void)	ボードの 32 ビット·ユニーク ID(LSB)を取得します。
void OnRestoreContextRequest(void *nvm, uint32_t nvm_size)	Flash から NVM データ·コンテキストを復元します。
void OnStoreContextRequest(void *nvm, uint32_t nvm_size)	NVM データ·コンテキストを Flash に格納します。
void OnMacProcess (void)	無線 IRQ の受信時に、LmHandler プロセスをコールします。
void OnNvmDataChange (LmHandlerNvmContextStates_t state)	NVM コンテキストが変更されたことを上位層に通知します。
void OnNetworkParametersChange (CommissioningParams_t *params)	ネットワーク·パラメータが設定されたことを上位層に通知します。
void OnJoinRequest (LmHandlerJoinParams_t *params)	ネットワークに JOIN したことを上位層に通知します。
void OnTxData (LmHandlerTxParams_t *params)	フレームが送信されたことを上位層に通知します。
void OnRxData (LmHandlerAppData_t *appData, LmHandlerRxParams_t *params)	アプリケーション·フレームを受信したことを上位層に通知します。
void OnClassChange (DeviceClass_t deviceClass)	LoRaWAN デバイス·クラスの変更を確認します。
void OnBeaconStatusChange (LmHandlerBeaconParams_t *params)	ビーコン·ステータスが変化したことを上位層に通知します。
void OnBeaconStatusChange (LmHandlerBeaconParams_t *params)	ビーコン·ステータスが変化したことを上位層に通知します。
void OnSysTimeUpdate (void)	システム時刻が更新されたことを上位層に通知します。
void OnTxPeriodicityChanged(uint32_t Periodicity)	適用可能な Tx フレーム周期を変更するためにコールされます。 TS001-1.0.4 + TS009 1.0.0 が定義されている場合に使用されるコンプライアンス·テスト·プロトコル·コールバックです。
void OnTxFrameCtrlChanged(LmHandlerMsgTypes_t isTxConfirmed)	適用可能な Tx フレーム制御を変更するためにコールされます。 TS001-1.0.4 + TS009 1.0.0 が定義されている場合に使用されるコンプライアンス·テスト·プロトコル·コールバックです。
void OnPingSlotPeriodicityChanged(uint8_t pingSlotPeriodicity)	ping 周期を変更するためにコールされます。 TS001-1.0.4 + TS009 1.0.0 が定義されている場合に使用されるコンプライアンス·テスト·プロトコル·コールバックです。
void OnSystemReset(void)	システムをリセットするためにコールされます。 TS001-1.0.4 + TS009 1.0.0 が定義されている場合に使用されるコンプライアンス·テスト·プロトコル·コールバックです。

6.7 アプリケーションの拡張関数

これらのコールバックは、LoRaWAN_End-Node と LoRaWAN_AT-Slave の両方のアプリケーションで使用されます。

表 17. 取得/設定関数

関数	説明
`LmHandlerErrorStatus_t LmHandlerGetCurrentClass( DeviceClass_t *deviceClass)`	現在の LoRaWAN クラスを取得します。
`LmHandlerErrorStatus_t LmHandlerGetDevEUI( uint8_t *devEUI)`	LoRaWAN デバイスの EUI を取得します。
`LmHandlerErrorStatus_t LmHandlerSetDevEUI( uint8_t *devEUI)`	LoRaWAN デバイスの EUI を設定します（OTAA の場合）。
`LmHandlerErrorStatus_t LmHandlerGetAppEUI( uint8_t *appEUI)`	LoRaWAN アプリケーションの EUI を取得します。
`LmHandlerErrorStatus_t LmHandlerSetAppEUI( uint8_t *appEUI)`	LoRaWAN アプリケーションの EUI を設定します。
`LmHandlerErrorStatus_t LmHandlerGetNetworkID( uint32_t *networkId)`	LoRaWAN のネットワーク ID を取得します。
`LmHandlerErrorStatus_t LmHandlerSetNetworkID uint32_t networkId)`	LoRaWAN のネットワーク ID を設定します。
`LmHandlerErrorStatus_t LmHandlerGetDevAddr( uint32_t *devAddr)`	LoRaWAN デバイスのアドレスを取得します。
`LmHandlerErrorStatus_t LmHandlerSetDevAddr( uint32_t devAddr)`	LoRaWAN デバイスのアドレスを設定します（ABP の場合）。
`LmHandlerErrorStatus_t LmHandlerGetAppKey( uint8_t *appKey)`	LoRaWAN アプリケーションのルート･キーを取得します。
`LmHandlerErrorStatus_t LmHandlerSetAppKey( uint8_t *appKey)`	LoRaWAN アプリケーションのルート･キーを設定します。
`LmHandlerErrorStatus_t LmHandlerGetNwkKey( uint8_t *nwkKey )`	LoRaWAN ネットワークのルート･キーを取得します。
`LmHandlerErrorStatus_t LmHandlerSetNwkKey( uint8_t *nwkKey )`	LoRaWAN ネットワークのルート･キーを設定します。
`LmHandlerErrorStatus_t LmHandlerGetNwkSKey( uint8_t *nwkSKey )`	LoRaWAN ネットワークのセッション･キーを取得します。
`LmHandlerErrorStatus_t LmHandlerSetNwkSKey( uint8_t *nwkSKey)`	LoRaWAN ネットワークのセッション･キーを設定します。
`LmHandlerErrorStatus_t LmHandlerGetAppSKey( uint8_t *appSKey )`	LoRaWAN アプリケーションのセッション･キーを取得します。
`LmHandlerErrorStatus_t LmHandlerSetAppSKey( uint8_t *appSKey)`	LoRaWAN アプリケーションのセッション･キーを設定します。
`LmHandlerErrorStatus_t LmHandlerGetActiveRegion( LoRaMacRegion_t *region)`	アクティブリージョンを取得します。
`LmHandlerErrorStatus_t LmHandlerSetActiveRegion( LoRaMacRegion_t region)`	アクティブリージョンを設定します。

関数	説明
LmHandlerErrorStatus_t LmHandlerGetAdrEnable(bool *adrEnable)	アダプティブ·データ·レートの状態を取得します。
LmHandlerErrorStatus_t LmHandlerSetAdrEnable(bool adrEnable)	アダプティブ·データ·レートの状態を設定します。
LmHandlerErrorStatus_t LmHandlerGetTxDatarate(int8_t *txDatarate)	現在の Tx データ·レートを取得します。
LmHandlerErrorStatus_t LmHandlerSetTxDatarate(int8_t txDatarate)	Tx データ·レートを設定します(アダプティブ DR が無効の場合)。
LmHandlerErrorStatus_t LmHandlerGetDutyCycleEnable (bool *dutyCycleEnable)	現在の Tx デューティ·サイクルの状態を取得します。
LmHandlerErrorStatus_t LmHandlerSetDutyCycleEnable (bool dutyCycleEnable)	Tx デューティ·サイクルの状態を設定します。
LmHandlerErrorStatus_t LmHandlerGetRX2Params (RxChannelParams_t *rxParams)	現在の Rx2 データ·レートと周波数設定を取得します。
LmHandlerErrorStatus_t LmHandlerSetRX2Params (RxChannelParams_t *rxParams)	Rx2 データ·レートと周波数設定を設定します。
LmHandlerErrorStatus_t LmHandlerGetTxPower(int8_t *txPower)	現在の送信電力値を取得します。
LmHandlerErrorStatus_t LmHandlerSetTxPower(int8_t txPower)	送信電力値を設定します。
LmHandlerErrorStatus_t LmHandlerGetRx1Delay(uint32_t *rxDelay)	現在の Rx1 遅延を取得します(Tx ウィンドウの後)。
LmHandlerErrorStatus_t LmHandlerSetRx1Delay(uint32_t rxDelay)	Rx1 遅延を設定します(Tx ウィンドウの後)。
LmHandlerErrorStatus_t LmHandlerGetRx2Delay(uint32_t *rxDelay)	現在の Rx2 遅延を取得します(Tx ウィンドウの後)。
LmHandlerErrorStatus_t LmHandlerSetRx2Delay(uint32_t rxDelay)	Rx2 遅延を設定します(Tx ウィンドウの後)。
LmHandlerErrorStatus_t LmHandlerGetJoinRx1Delay(uint32_t *rxDelay)	現在の Join Rx1 遅延を取得します(Tx ウィンドウの後)。
LmHandlerErrorStatus_t LmHandlerSetJoinRx1Delay(uint32_t rxDelay)	Join Rx1 遅延を設定します(Tx ウィンドウの後)。
LmHandlerErrorStatus_t LmHandlerGetJoinRx2Delay(uint32_t *rxDelay)	現在の Join Rx2 遅延を取得します(Tx ウィンドウの後)
LmHandlerErrorStatus_t LmHandlerSetJoinRx2Delay(uint32_t rxDelay)	Join Rx2 遅延を設定します(Tx ウィンドウの後)。
LmHandlerErrorStatus_t LmHandlerGetPingPeriodicity	現在の Rx Ping スロット周期を取得します (LORAMAC_CLASSB_ENABLED の場合)。

関数	説明
(uint8_t pingPeriodicity)	
LmHandlerErrorStatus_t LmHandlerSetPingPeriodicity (uint8_t pingPeriodicity)	Rx Ping スロット周期を設定します (LORAMAC_CLASSB_ENABLED の場合)。
LmHandlerErrorStatus_t LmHandlerGetBeaconState (BeaconState_t *beaconState)	ビーコンの状態を取得します(LORAMAC_CLASSB_ENABLED の場合)。
LmHandlerErrorStatus_t LmHandlerDeviceTimeReq(void)	ネットワーク・サーバの時間更新をリクエストします。
LmHandlerErrorStatus_t LmHandlerLinkCheckReq(void)	リンク・コネクティビティ・チェックをリクエストします。
LmHandlerErrorStatus_t LmHandlerPingSlotReq(uint8_t periodicity)	使用する Ping スロット周期でサーバに通知します。

7 SubGHz_Phy 層ミドルウェアの説明

無線抽象化レイヤは、次の 2 つの層で構成されています。

- 上位層(radio.c)
  スタック·ミドルウェアへの上位レベル無線インタフェースを提供します。また、無線状態の維持、割込みの処理、タイムアウトの管理も行います。コールバックを記録し、無線イベントが発生するとコールバックします。
- 下位レベル無線ドライバ
  RF インタフェースへの抽象化レイヤです。この層では、レジスタの名前と構造、および詳細シーケンスを把握しています。ハードウェア・インタフェースについては認識しません。

SubGHz_Phy 層ミドルウェアは、BSP によって提供されるハードウェア·インタフェース上で直接インタフェース接続される無線抽象化レイヤが含まれます(セクション 4 を参照)。

SubGHz_Phy ミドルウェアのディレクトリは、次の 2 つの部分に分かれています。

- radio.c:radio_driver 関数をコールするすべての無線汎用コールバックのセットが含まれています。この API のセットは、すべての無線機で汎用的であり、同じものです。
- radio_driver.c:下位レベル無線ドライバ

radio_conf.h RF_WAKEUP_TIME、DC/DC 動的設定、XTAL_FREQ などの無線アプリケーション設定を含みます。

7.1 ミドルウェア無線ドライバ構造体

無線汎用構造体（struct Radio_s Radio {};）は、すべてのコールバックを登録するように定義されています。次の表にフィールドの詳細を示します。

表 18. Radio_s 構造体のコールバック

コールバック	説明
RadioInit	無線を初期化します。
RadioGetStatus	現在の無線ステータスを返します。
RadioSetModem	指定されたモデムで無線を設定します。
RadioSetChannel	チャネル周波数を設定します。
RadioIsChannelFree	指定された時間、チャネルが空いているかどうかを確認します。
RadioRandom	RSSI 測定値に基づいて、32 ビットのランダム値を生成します。
RadioSetRxConfig	受信パラメータを設定します。
RadioSetTxConfig	送信パラメータを設定します。
RadioCheckRfFrequenc	指定された RF 周波数がハードウェアによってサポートされているかどうかを確認します。
RadioTimeOnAir	与えられたペイロードについて、パケット通信時間（ms 単位）を計算します。
RadioSend	パケットを送信する準備をし、無線での送信を開始します。
RadioSleep	無線を SLEEP モードに設定します。
RadioStandby	無線を STANDBY モードに設定します。
RadioRx	指定された時間、無線を受信モードに設定します。
RadioStartCad	CAD（チャネル·アクティビティ検出）を開始します。
RadioSetTxContinuousWave	無線を連続波送信モードに設定します。
RadioRssi	現在の RSSI の 値を読み取ります。
RadioWrite	無線レジスタの指定アドレスに書き込みます。
RadioRead	無線レジスタの指定アドレスから読み出します。
RadioSetMaxPayloadLength	最大ペイロード長を設定します。
RadioSetPublicNetwork	ネットワークをパブリックまたはプライベートに設定し、同期バイトを更新します。
RadioGetWakeUpTime	無線の SLEEP モード終了に必要な時間を取得します。
RadioIrqProcess	無線 IRQ を処理します。
RadioRxBoosted	指定された時間、無線を最大 LNA ゲインでの受信モードに設定します。
RadioSetRxDutyCycle	Rx デューティ·サイクル管理パラメータを設定します。
RadioTxPrbs	トランスミッタを連続 PRBS モードに設定します。
RadioTxCw	トランスミッタを連続無変調キャリア·モードに設定します。

7.2 無線 IRQ 割込み

考えられる Sub-GHz 無線割込みソースの詳細を下の表に示します。

表 19. 無線 IRQ ビットのマッピングと定義

ビット	転送元	説明	パケットタイプ	動作
0	txDone	パケット送信完了	LoRa と GFSK	Tx
1	rxDone	パケット受信完了		Rx
2	PreambleDetected	プリアンブルを検出		
3	SyncDetected	同期ワードが有効	GFSK	
4	HeaderValid	ヘッダが有効	LoRa	
5	HeaderErr	ヘッダ·エラー		
6	Err	プリアンブル、同期ワード、アドレス、CRC、または長さエラー	GFSK	
	CrcErr	CRC エラー	LoRa	
7	CadDone	チャネル·アクティビティ検出完了		CAD
8	CadDetected	チャネル·アクティビティを検出		
9	タイムアウト	Rx または Tx タイムアウト	LoRa と GFSK	Rx と Tx

詳細については、製品のリファレンス·マニュアルを参照してください。

8 ユーティリティの説明

ユーティリティは、\Utilities ディレクトリに格納されています。

主な API について以下で説明します。セカンダリ API および追加情報は、ドライバに関連するヘッダ·ファイルに記載されています。

8.1 シーケンサ

シーケンサは、バックグラウンドでタスクを実行し、アクティビティがなくなったときに低消費電力モードに移行するための堅牢で容易なフレームワークを提供します。シーケンサには、競合状態を防止するメカニズムが実装されています。

さらに、シーケンサは、イベント(特定のイベントが割込みによってセットされる)を待機するための機能を有効にし、「完了まで実行」コマンドを実装する任意のアプリケーションで MIPS と電力を簡単に節約するためのイベント機能を備えています。

プロジェクトのサブフォルダにある utility_def.h ファイルは、タスク ID とイベント ID を設定するために使用されます。すでにリストされているものは削除しないでください。

シーケンサは OS ではありません。タスクは完了するまで実行され、RTOS が RTOS ティックで別のタスクを実行できるように切り替えるには、UTIL_SEQ_WaitEvt をコールしてタスクをサスペンドする必要があります。また、1 つのシングルメモリ·スタックが使用されます。シーケンサは、タスクとイベントのビットマップ·フラグを一元化する高度な while ループです。

シーケンサには、以下の機能があります。

- 高度なパッケージ化された while ループ·システム
- 最大 32 のタスクと 32 のイベントのサポート
- タスクの登録と実行
- イベントとセット·イベントの待機
- タスクの優先度の設定
- 競合状態での安全な低消費電力への移行

シーケンサを使用するには、アプリケーションで以下を実行する必要があります。

- UTIL_SEQ_CONF_TASK_NBR の値を定義して、サポートする関数の最大数をセットします。
- UTIL_SEQ_RegTask() を用いてシーケンサでサポートする関数を登録します。
- UTIL_SEQ_Run() をコールしてシーケンサを起動し、バックグラウンドの while ループを実行します。
- 関数を実行する必要があるときに UTIL_SEQ_SetTask() をコールします。

sequencer ユーティリティは Utilities\sequencer\stm32_seq.c に格納されています。

表 20. シーケンサ API

機能	説明
void UTIL_SEQ_Idle(void)	実行すべきものが存在しない場合に(クリティカル·セクションの中で - PRIMASK)コールします。
void UTIL_SEQ_Run(UTIL_SEQ_bm_t mask_bm)	保留されていてマスク mask_bm で有効化されている関数の実行をシーケンサにリクエストします。
void UTIL_SEQ_RegTask(UTIL_SEQ_bm_t task_id_bm, uint32_t flags, void (*task) (void))	シグナル(task_id_bm)に関連付けられている関数(task)をシーケンサに登録します。task_id_bm にセットされているビットは 1 つだけである必要があります。
void UTIL_SEQ_SetTask(UTIL_SEQ_bm_t taskId_bm, uint32_t task_Prio)	task_id_bm に関連付けられている関数の実行をリクエストします。task_prio は、関数が終了している場合のみシーケンサによって評価されます。 同時に複数の関数が保留されている場合、優先順位が最も高い(0)ものが実行されます。
void UTIL_SEQ_WaitEvt(UTIL_SEQ_bm_t EvtId_bm);	特定のイベントが設定されるのを待ちます。
void UTIL_SEQ_SetEvt(UTIL_SEQ_bm_t EvtId_bm);	UTIL_SEQ_WaitEvt() で待機するイベントを設定します。

次の図は、標準の while ループの実装とシーケンサの while ループの実装を比較しています。

表 21. 標準の While ループとシーケンサの実装

標準的な方法	シーケンサーによる方法

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

8.2 タイマ·サーバ

タイマ·サーバを使用すると、ユーザは、時間指定されたタスクの実行をリクエストできます。ハードウェア·タイマは RTC に基づいているため、低消費電力モードであっても時間は常にカウントされています。

タイマ·サーバは、ユーザとスタックに信頼できるクロックを提供します。ユーザは、アプリケーションに必要な数だけ、タイマをリクエストできます。

タイマ·サーバは、`Utilities\timer\stm32_timer.c` に格納されています。

表 22. タイマ·サーバ API

機能	説明
`UTIL_TIMER_Status_t UTIL_TIMER_Init(void)`	タイマ·サーバを初期化します。
`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)`	タイマ·オブジェクト作成し、タイマ経過時にコールするコールバック関数を関連付けます。
`UTIL_TIMER_Status_t UTIL_TIMER_SetPeriod(UTIL_TIMER_Object_t *TimerObject, uint32_t NewPeriodValue)`	周期を更新し、タイムアウト値(ミリ秒)を指定してタイマを開始します。
`UTIL_TIMER_Status_t UTIL_TIMER_Start (UTIL_TIMER_Object_t *TimerObject)`	タイマ·オブジェクトを開始し、タイマ·イベントのリストに追加します。
`UTIL_TIMER_Status_t UTIL_TIMER_Stop (UTIL_TIMER_Object_t *TimerObject)`	タイマ·オブジェクトを停止して、タイマ·イベントのリストから削除します。

8.3 低消費電力関数

`low-power` ユーティリティは、ファームウェアによって実装される個別のモジュールの低消費電力要件を一元管理し、システムがアイドル·モードに入るときの低消費電力への移行を管理します。たとえば、DMA を使用してコンソールにデータを出力する場合、STOP モードでは DMA クロックがオフになるため、システムは SLEEP モードより下のレベルの低消費電力モードに移行しないようにする必要があります。

下の表に示す API を使用して、コア·マイクロコントローラの低消費電力モードを管理します。`low-power` ユーティリティは `Utilities\lpm\tiny_lpm\stm32_lpm.c` に格納されています。

表 23. 低消費電力 API

機能	説明
`void UTIL_LPM_EnterLowPower( void )`	選択されている低消費電力モードに移行します。システムのアイドル状態によってコールされます。
`void UTIL_LPM_SetStopMode( UTIL_LPM_bm_t lpm_id_bm, UTIL_LPM_State_t state );`	STOP モードを設定します。`id` ではリクエストされた処理モード、つまり `UTIL_LPM_ENABLE` または `UTIL_LPM_DISABLE` を定義します。(1)
`void UTIL_LPM_SetOffMode( UTIL_LPM_bm_t lpm_id_bm, UTIL_LPM_State_t state );`	STOP モードを設定します。`id` ではリクエストされた処理モード、つまり `UTIL_LPM_ENABLE` または `UTIL_LPM_DISABLE` を定義します。
`UTIL_LPM_Mode_t UTIL_LPM_GetMode( void )`	現在選択されている低消費電力モードを返します。

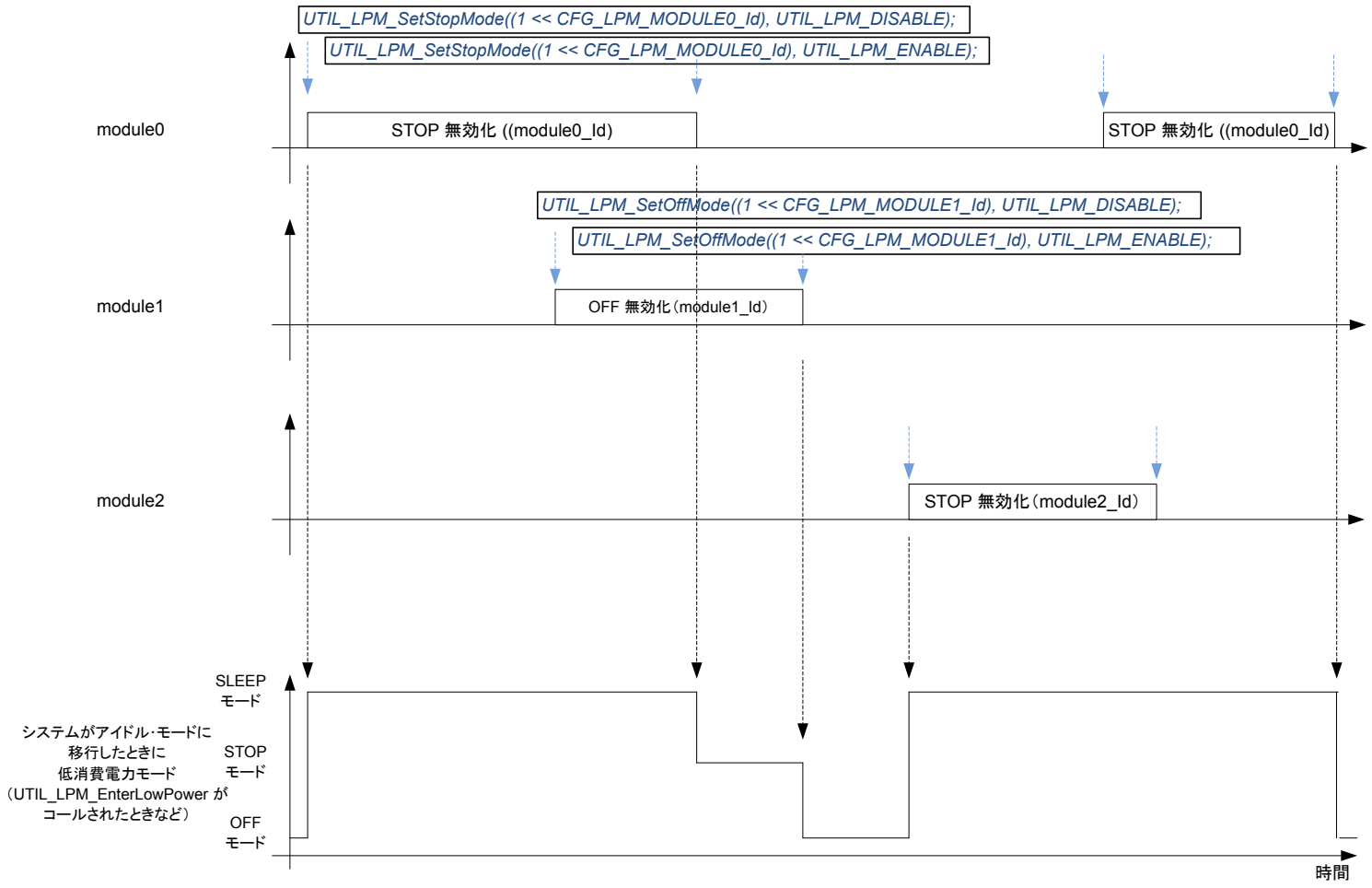
1. `Utilities_def.h` でシフト値が定義されているビットマップ。

デフォルトの低消費電力モードは OFF モードで、STANDBY モードまたは SHUTDOWN モードの場合があります（表 24 の `void PWR_EnterOffMode (void)` で定義 ）。

- 少なくとも 1 つのファームウェア･モジュールによって STOP モードが無効化され、低消費電力に移行した場合、SLEEP モードが選択されます。
- どのファームウェア･モジュールによっても STOP モードが無効にされていない場合、少なくとも 1 つのファームウェア･モジュールによって OFF モードが無効にされ、低消費電力に移行します。STOP モードが選択されます。
- どのファームウェア･モジュールによっても STOP モードが無効にされない場合、どのファームウェア･モジュールによっても OFF モードが無効にされず、低消費電力に移行します。OFF モードが選択されます。

図 6 に、3 つの異なるファームウェア･モジュールが低消費電力要件と低消費電力モードに依存して設定されている場合の動作を示しています（これはシステムが低消費電力モードに移行するときに選択されます）。

図 6. 低消費電力モードの動的ビューの例



下位レベル API を実装して、低消費電力モードを開始/終了するためにシステムで実行する必要があることを定義する必要があります。これらの機能は、stm32_lpm_if.c プロジェクトのサブフォルダで実装されます。

表 24. 下位レベル API

機能	説明
void PWR_EnterSleepMode (void)	SLEEP モードに移行する前にコールされる API
void PWR_ExitSleepMode (void)	SLEEP モードの終了時にコールされる API
void PWR_EnterStopMode (void)	STOP モードに移行する前にコールされる API
void PWR_ExitStopMode (void)	STOP モードの終了時にコールされる API
void PWR_EnterOffMode (void)	OFF モードに移行する前にコールされる API
void PWR_ExitOffMode(void)	OFF モードの終了時に API をコールします

SLEEP モードでは コア·クロックが停止します。各ペリフェラル·クロックは、ゲートすることも、ゲートしないことも可能です。すべてのペリフェラルで電源が保持されます。

STOP 2 モードでは、ほとんどのペリフェラル·クロックが停止します。ほとんどのペリフェラル電源はオフになっています。ペリフェラルの一部のレジスタは保持されず、STOP 2 モード終了時に再初期化する必要があります。メモリとコアのレジスタは保持されます。

STANDBY モードでは、LSI と LSE を除くすべてのクロックがオフになります。すべてのペリフェラル電源は、保持なしで（BOR、バックアップ・レジスタ、GPIO プル、および RTC を除く）オフになり（保持付きの追加の SRAM2 を除く）、STANDBY モード終了時に再初期化する必要があります。コア·レジスタは保持されず、STANDBY モード終了時に再初期化する必要があります。

注 Sub-GHz 無線の電源は、システムのその他の部分から独立しています。詳細については、製品のリファレンス·マニュアルを参照してください。

8.4 システム時間

マイクロコントローラの時間は、マイクロコントローラ·リセットを基準にしています。システム時間には、UNIX® エポック·タイムを記録できます。

下の表に示す API を使用して、コア·マイクロコントローラのシステム時間を管理します。systime ユーティリティは Utilities\misc\stm32_systime.c に格納されています。

表 25. システム時間関数

機能	説明
void SysTimeSet (SysTime_t sysTime)	入力 UNIX エポック(秒およびサブセカンド単位)に基づいて、MCU 時間との差がバックアップ·レジスタに格納されます(STANDBY モードでも保持されます)。(1)
SysTime_t SysTimeGet (void)	現在のシステム時間を取得します。
uint32_t SysTimeMkTime (const struct tm* localtime)	ローカル時間を UNIX エポック·タイムに変換します。(2)
void SysTimeLocalTime (const uint32_t timestamp, struct tm *localtime)	UNIX エポック·タイムを現地時間に変換します。(2)

1. システム時間の参照は、1970 年 1 月 1 日から始まる UNIX エポックです。
2. エポックを time.h インタフェースで規定された tm 構造体に変換するために、SysTimeMkTime および SysTimeLocalTime も用意されています。

UNIX タイムをローカル時間に変換するには、タイムゾーンを加算し、うるう秒を減算する必要があります。2018 年にはうるう秒として 18 秒を減算する必要があります。パリの夏時間には、グリニッジ標準時と 2 時間の時差があります。時間が設定されていると仮定すると、次のコードでローカル時間を端末に表示できます。

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

8.5 トレース

トレース·モジュールを使用すると、DMA を使用して COM ポート·データを出力できます。トレース機能の管理には、次の表に示す API を使用します。

t race ユーティリティは Utilities\trace\adv_trace\stm32_adv_trace.c に格納されています。

表 26. トレース関数

機能	説明
UTIL_ADV_TRACE_Status_t UTIL_ADV_TRACE_Init(void)	TraceInit は、アプリケーションを初期化する際にコールする必要があります。これは、COM または VCOM ハードウェアを DMA モードで初期化し、DMA 送信完了時に処理されるようにコールバックを登録します。
UTIL_ADV_TRACE_Status_t UTIL_ADV_TRACE_COND_FSend(uint32_t VerboseLevel, uint32_t Region, uint32_t TimeStampState, const char *strFormat, ...)	文字列フォーマットをバッファに変換し、出力のために循環キューにポストします。
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)	長さ = len のデータをポストし、出力のために、循環キューにポストします。
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)	ユーザ·フォーマット·データを FIFO に直接書き込みます (Z-Cpy)。

トレース機能のステータス値は、構造体 UTIL_ADV_TRACE_Status_t に次のように定義されています。

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

UTIL_ADV_TRACE_COND_FSend (..) 関数は次のように使用できます。

- リアルタイム制約が適用されない場合はポーリング·モード:アプリケーションの初期化中など。

```
#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 */
```

- リアルタイム·モード:循環キューに十分なスペースが残っていなければ、文字列は追加されず、COM ポートに出力されません。

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

ここで、

– VL はトレースの VerboseLevel です。

– TS を使用して、トレースにタイムスタンプを追加できます(TS_ON または TS_OFF)。

アプリケーションの詳細なレベルは、Core\Inc\sys_conf.h で次のように設定されます。

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

ここで VLEVEL は VLEVEL_OFF、VLEVEL_L、VLEVEL_M、または VLEVEL_H とすることができます。

UTIL_ADV_TRACE_COND_FSend (..) は、VLEVEL ≥ VerboseLevel の場合のみ表示されます。

バッファ長が飽和した場合に備えて、Core\Inc\utilities_conf.h で次のように指定してバッファ長を増やすことができます。

```
#define UTIL_ADV_TRACE_TMP_BUF_SIZE 256U
```

このユーティリティは、DMA がアクティブな間、システムが STOP モードまたはそれ以下のモードに移行することを禁止するために実装されるフックの役割を果たします。

- ```
  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 LoRaWAN_End_Node アプリケーション

このアプリケーションは、マイクロコントローラのバッテリ·レベルと温度を測定します。これらの値は、868 MHz のクラス A の LoRa 無線を使用して定期的に LoRa ネットワークに送信されます。

LoRaWAN_End_Node プロジェクトを起動するには、

`\Projects\<target> \Applications\LoRaWAN\LoRaWAN_End_Node` に移動して、該当するツールチェーン·フォルダを選択します(IDE 環境の場合)。適切なターゲット·ボードから LoRa プロジェクトを選択します。

以下で説明する設定に注目して、アプリケーションをセットアップしてください。

9.1 LoRaWAN のユーザ·コード·セクションの説明

LoRaWAN スタックを実装して使用するための例として、4 つのメイン関数が

`\Projects\<target> \Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.c` で定義されています。

これらの関数のユーザ·コード·セクションにはサンプル·コードが用意されていて、アプリケーション層の特定の機能を処理するように上書きすることができます。

表 27. LoRaWAN のユーザ関数

機能	説明
`void LoRaWAN_Init(void)`	図 5. 動作モデル の説明に従って、LoRaWAN アプリケーションを初期化します。
`static void SendTxData(void)`	汎用コンテンツ·ペイロード生成と、一時ペイロードで生成されたバッファを持つ LmHandlerSend(...) コールを使用した LoRaWAN Tx プロセスの例。
`static void OnTxData(LmHandlerTxParams_t *params)`	LoRaWAN アプリケーションがフレームを正常に送信した場合のコールバックの実装例。 • params パラメータには最後の Tx のステータスが設定されます
`static void OnRxData(LmHandlerAppData_t *appData, LmHandlerRxParams_t *params)`	LoRaWAN アプリケーションがフレームを受信した場合のコールバックの実装例 • appData パラメータには最後の Rx で受信したデータが設定されます • params パラメータには最後の Rx のステータスが設定されます

9.2 デバイスの設定

9.2.1 アクティベーション方法とキー

ネットワーク上でデバイスを有効化するには、OTAA または ABP の 2 つのアクティベーション方法があります。

選択したモードでデバイスを有効化するには、アプリケーションのグローバル変数「ActivationType」を調整する必要があります。

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

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

および

```
#define LORAWAN_DEFAULT_ACTIVATION_TYPE ACTIVATION_TYPE_OTAA
```

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

ここで ActivationType_t enum は次のように定義されます。

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

\Projects\<target> \Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\se-identity.h ファイルには、デバイスの有効化に役立つ設定データが含まれています。

9.2.2 LoRa クラスの有効化

デフォルトでは、クラス A が定義されています。クラスの有効化（クラス A、クラス B、またはクラス C）を変更するには、ユーザは次の手順を実行する必要があります。

- コード

```
#define LORAWAN_DEFAULT_CLASS CLASS_B;
```

入力 \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h で設定します。

- コード

```
#define LORAMAC_CLASSB_ENABLED 1
```

入力 \Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lorawan_conf.h で設定します。

9.2.3 Tx トリガ

アップリンク・アクションを生成するには、2 つの方法があります。

\Projects\<target> \Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.c 内で EventType グローバル変数を次のいずれかで使用します。

- タイマによって使用
- 外部イベントによって使用

次のコードを使用します。

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

ここで TxEventType_t enum は次のように定義されます。

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

TX_ON_EVENT 機能では、LoRaWAN_End_Node アプリケーションのイベントとしてボタン 1 を使用します。

9.2.4 デューティサイクル

アプリケーションで使用されるデューティ・サイクル値（ms）の定義は、

\Projects\<target> \Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 内で、次のコードにより行われます（例）。

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

9.2.5 アプリケーション・ポート

アプリケーションに使用されるアプリケーション・ポートの定義はファイル

\Projects\<target> \Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 内で、次のコードにより行われます（例）。

```
#define LORAWAN_APP_PORT                            2
```

注 LORAWAN_APP_PORT では認証用に予約されているポート 224 を使用することはできません。

9.2.6 確認/確認なしモード

アプリケーションに使用される確認/確認なしモードの定義は、
\Projects\<target> \Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 内で、以下のコードにより行われます。

```
#define LORAWAN_DEFAULT_CONFIRMED_MSG_STATE LORAMAC_HANDLER_UNCONFIRMED_MSG
```

9.2.7 データ・バッファ・サイズ

ネットワークに送信されるバッファのサイズの定義は、
\Projects\<target> \Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 内で、以下のコードにより行われます。

```
#define LORAWAN_APP_DATA_BUFFER_MAX_SIZE                          242
```

9.2.8 アダプティブ・データ・レート（ADR）

ADR の有効化はファイル
\Projects\<target> \Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 内で、以下のコードにより行われます。

```
#define LORAWAN_ADR_STATE                           LORAMAC_HANDLER_ADR_ON
```

ADR が無効の場合、デフォルトのレートの設定がファイル
\Projects\<target> \Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h 内で、以下のコードにより行われます。

```
#define LORAWAN_DEFAULT_DATA_RATE                   DR_0
```

9.2.9 Ping の周期

デバイスをクラス B で切り替えることができる場合、デフォルトの Rx Ping スロット周期の有効化を
\Projects\<target>\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lora_app.h で設定します。
以下のコードを使用します。

```
#define LORAWAN_DEFAULT_PING_SLOT_PERIODICITY 4
```

期待値は 0 ~ 7 の範囲内である必要があります。
結果として生じる周期は、次のように定義されます。

```
period = 2^LORAWAN_DEFAULT_PING_SLOT_PERIODICITY
```

9.2.10 LoRa バンド選択

リージョンと対応するバンド選択の定義は、\Projects\<target>

\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\Target\lorawan_conf.h 内で、以下のコードにより行われます。

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

注　同じアプリケーションで複数のリージョンを定義できます。
リージョンによっては、デフォルトの有効リージョンの定義を、\Projects\<target>
\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h 内で、コードにより行う必要があります(ヨーロッパなど)。

```
#define ACTIVE_REGION LORAMAC_REGION_EU868
```

9.2.11 コンテキスト管理

コンテキスト管理の定義は、
\Projects\<target> \Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN\App\lorawan_conf.h 内で、以下のコードにより行われます。

```
#define CONTEXT_MANAGEMENT_ENABLED 1
```

この機能の詳細については、セクション 13 LoRaWAN コンテキスト管理の説明を参照してください。

9.2.12 デバッグ・スイッチ

デバッグ・モードの有効化は、\Projects\<target>
\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h 内で、以下のコードにより行われます。

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

デバッグ・モードでは、マイクロコントローラが低消費電力モードに移行したときでも、SWD ピンは有効です。

注　真の低消費電力を有効にするには、#define DEBUGGER_ENABLED をリセットする必要があります。

いくつかの追加定義によって、デバッグ用に内部 RF 信号の監視(プローブ)を有効化できます。

```
#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 低消費電力スイッチ

システムがアイドルになると、低消費電力 STOP 2 モードに入ります。

STOP 2 モードへの移行の無効化は、\Projects\<target>
\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h 内で、以下のコードにより行われます。

```
#define LOW_POWER_DISABLE    0 /* Low power enabled = 0, Low power disabled = 1 */
```

ここで、

- 設定値が 0、すなわち低消費電力有効の場合は、マイクロコントローラが STOP 2 モードに移行することを意味します。
  STOP 2 は、低消費電力レギュレータと V_{DD12I} 割込み可能デジタル·コア·ドメインの電源がオフになる STOP モードです。低消費電力 STOP 1 モードよりも有効化されるペリフェラルが少なく、電力消費が低減されます。詳細については、ドキュメント [3] を参照してください。
- 設定値が 1、すなわち低消費電力無効の場合は、マイクロコントローラが SLEEP モードのみに移行することを意味します。

9.2.14 トレース·レベル

トレース·モードの有効化は、\Projects\<target>
\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h 内で、以下のコードにより行われます。

```
#define APP_LOG_ENABLED    1
```

トレース·レベルの選択は、\Projects\<target>
\Applications\LoRaWAN\LoRaWAN_End_Node\Core\Inc\sys_conf.h 内で、以下のコードにより行われます。

```
#define VERBOSE_LEVEL VLEVEL_M
```

次のトレース·レベルが提案されています。

- VLEVEL_OFF: すべてのトレースを無効にします。
- VLEVEL_L: 機能トレースを有効にします。
- VLEVEL_M: デバッグ·トレースを有効にします。
- VLEVEL_H: すべてのトレースを有効にします。

9.3 LoRaWAN_End_Node アプリケーションのデバイス設定の概要

表 28. LoRaWAN_End_Node アプリケーション設定のスイッチ･オプション

プロジェクト･モジュール	詳細	スイッチ･オプション	定義	場所
LoRa スタック	識別	STATIC_DEVICE_EUI	静的または動的エンドデバイスの識別	se-identity.h
	アドレス	STATIC_DEVICE_ADDRESS	静的または動的エンドデバイスのアドレス	
	サポートされているリージョン	REGION_EU868	デバイスでサポートされているリージョン	lorawan_conf.h
		REGION_EU433		
		REGION_US915		
		REGION_AS923		
		REGION_AU915		
		REGION_CN470		
		REGION_CN779		
		REGION_IN865		
		REGION_RU864		
		REGION_KR920		
	チャネルの制限	HYBRID_ENABLED	AU915、CN470、および US915 リージョンに対して、デフォルトで使用可能なチャネル数を制限します。	
	キーの読出し	KEY_EXTRACTABLE	メモリ内のキーの読出しアクセスを定義します。	
	オプション･クラス	LORAMAC_CLASSB_ENABLED	エンドデバイスのクラス B 機能	
	コンテキスト管理	CONTEXT_MANAGEMENT_ENABLED	LoRaWAN スタック･コンテキストを格納/復元します。	
アプリケーション	Tx トリガ	EventType = TX_ON_TIMER	Tx トリガ方式	lora_app.c
	クラス選択	LORAWAN_DEFAULT_CLASS	デバイスのクラスを設定します。	lora_app.h
	デューティサイクル	APP_TX_DUTYCYCLE	送信される 2 つの Tx 間の時間	
	アプリ･ポート	LORAWAN_USER_APP_PORT	Tx データ･フレームによって使用される LoRa ポート	
	確認モード	LORAWAN_DEFAULT_CONFIRMED_MSG_STATE	確認モード選択	
	アダプティブ･データ･レート	LORAWAN_ADR_STATE	ADR 選択	
	デフォルト･データ･レート	LORAWAN_DEFAULT_DATA_RATE	ADR が無効の場合のデータ･レート	
	最大データ･バッファ･サイズ	LORAWAN_APP_DATA_BUFFER_MAX_SIZE	バッファ･サイズの定義	
	Ping 周期	LORAWAN_DEFAULT_PING_SLOT_PERIODICITY	Rx ping スロット周期	
	ネットワーク JOIN の有効化	LORAWAN_DEFAULT_ACTIVATION_TYPE	有効化手順のデフォルト選択	
	初期リージョン	ACTIVE_REGION	デバイス起動時に使用するリージョン	

プロジェクト·モジュール	詳細	スイッチ·オプション	定義	場所
アプリケーション	デバッグ	DEBUGGER_ENABLED	SWD ピンを有効にします。	sys_conf.h
	低消費電力	LOW_POWER_DISABLE	低消費電力モードを無効にします。	
	トレース有効	APP_LOG_ENABLED	トレース·モードを有効にします。	
	トレース·レベル	VERBOSE_LEVEL	トレース·レベルを有効にします。	

10 LoRaWAN_AT_Slave アプリケーション

このサンプル·プログラムの目的は、外部ホストから UART 経由で AT コマンド·インタフェースで制御する LoRa モデムを実装することです。

外部ホストには、アプリケーションと AT ドライバを組み込んだホスト·マイクロコントローラか、単にターミナルを実行するコンピュータを使用できます。

このアプリケーションは、STM32WL Nucleo ボード(NUCLEO-WL55JC)を対象としています。

LoRaWAN_AT_Slave のサンプル·プログラムは、内蔵 LoRa 無線を駆動する LoRaWAN スタックを実装しています。このスタックは、UART 経由で AT コマンド·インタフェースを使用して制御します。モデムは、外部ホストからの AT コマンドを処理しない限り、常に STOP 2 モードです。

LoRaWAN_AT_Slave プロジェクトを起動するには、

`\Projects\<target>Applications\LoRaWAN\AT_Slave` に移動して、LoRa End_Node プロジェクトと同じ手順で該当するツールチェーンを起動します。

文書 [2] には、AT コマンドとその説明のリストが示されています。

次の表に、LoRaWAN_AT_Slave アプリケーション設定の主なオプションを示します。

表 29. LoRaWAN_AT_Slave アプリケーション設定のスイッチ·オプション

プロジェクト·モジュール	詳細	スイッチ·オプション	定義	場所
LoRa スタック	識別	STATIC_DEVICE_EUI	静的または動的エンドデバイスの識別	se-identity.h
	アドレス	STATIC_DEVICE_ADDRESS	静的または動的エンドデバイスのアドレス	
	サポートされているリージョン	REGION_EU868	デバイスでサポートされているリージョン	lorawan_conf.h
		REGION_EU433		
		REGION_US915		
		REGION_AS923		
		REGION_AU915		
		REGION_CN470		
		REGION_CN779		
		REGION_IN865		
		REGION_RU864		
		REGION_KR920		
	チャネルの制限	HYBRID_ENABLED	AU915、CN470、および US915 リージョンに対して、デフォルトで使用可能なチャネル数を制限します。	
	キーの読出し	KEY_EXTRACTABLE	メモリ内のキーの読出しアクセスを定義します。	
	オプション·クラス	LORAMAC_CLASSB_ENABLED	エンドデバイスのクラス B 機能	
	コンテキスト管理	CONTEXT_MANAGEMENT_ENABLED	LoRaWAN スタック·コンテキストの格納と復元	
アプリケーション	アダプティブ·データ·レート	LORAWAN_ADR_STATE	ADR 選択	lora_app.h
	デフォルト·データ·レート	LORAWAN_DEFAULT_DATA_RATE	ADR が無効の場合のデータ·レート	
	Ping 周期	LORAWAN_DEFAULT_PING SLOT_PERIODICITY	Rx ping スロット周期	
	初期リージョン	ACTIVE_REGION	デバイス起動時に使用するリージョン	

プロジェクト·モジュール	詳細	スイッチ·オプション	定義	場所
アプリケーション	デバッグ	DEBUGGER_ENABLED	SWD ピンを有効にします。	sys_conf.h
	低消費電力	LOW_POWER_DISABLE	低消費電力モードを無効にします。	
	トレース有効	APP_LOG_ENABLED	トレース·モードを有効にします。	
	トレース·レベル	VERBOSE_LEVEL	トレース·レベルを有効にします。	

11 SubGhz_Phy_PingPong アプリケーション

このアプリケーションは、2 つの PingPong デバイス（1 つは Ping、もう 1 つは Pong と呼ばれる）間の単純な Rx/Tx RF リンクを示します。

デフォルトでは、各 PingPong デバイスがマスタとして起動され、「Ping」メッセージを送信して応答を待機します。起動時に、各 PingPong デバイスでは 2 つの LED が点滅します。ボードが同期すると（一方のボードの Tx ウィンドウが他方のボードの Rx ウィンドウと調整される）、Ping デバイス（「Ping」メッセージを受信するボード）で緑色の LED が点滅し、Pong デバイス（「Pong」メッセージを受信するボード）で赤色の LED が点滅します。「Ping」メッセージを最初に受信した PingPong デバイスがスレーブになり、「Pong」メッセージでマスタに応答します。

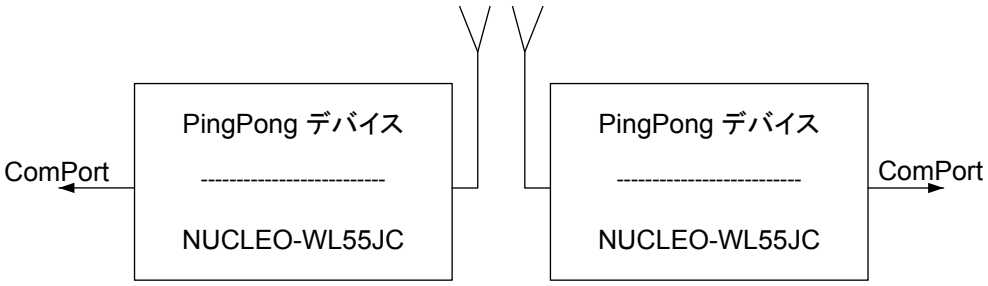
SubGhz_Phy_PingPong プロジェクトを起動するには、

`\Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong` に移動して、LoRa End_Node プロジェクトと同じ手順で該当するツールチェーンを起動させます。

11.1 SubGhz_Phy_PingPong ハードウェア/ソフトウェア環境のセットアップ

STM32WL Nucleo ボード（NUCLEO-WL55JC）をセットアップするには、下図に示すように、このボードとコンピュータを、USB ケーブル（Type-A - Mini-B）と ST-LINK コネクタ（CN1）で接続します。

図 7. SubGhz_Phy_PingPong アプリケーションのセットアップ



11.2 デバイスの設定

11.2.1 変調の定義

このアプリケーションでは、2 つの変調、LoRa と FSK の使用を提案します。1 つの変調を設定するには、次の表に示すように、`\Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h` でこれらの定義を更新する必要があります。

表 30. SubGHz_Phy_PingPong 変調の設定

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 ペイロード長

Tx ペイロードは、PING または PONG という文字列とそれに続く 0 のシーケンスで定義されます。このペイロードの長さの定義は、`\Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h` 内で次のコードを使用して行われます。

```
#define PAYLOAD_LEN    64
```

標準的なペイロード･サイズは、通常 51 ~ 242 です。

\Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.c 内で次のコードを使用して定義されている最大バッファサイズ以下の値に設定する必要があります。

```
#define MAX_APP_BUFFER_SIZE 255
```

11.2.3 リージョンと周波数

このアプリケーションで使用される周波数値の定義は、\Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h 内で次のコードにより行われます。

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

追加の定義では、以下のコードにより、各チャネル計画用に事前定義された周波数値を使用することを提案します（一度に 1 つの定義のみ、コメントアウトを解除することができます）。

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

11.2.4 帯域幅、拡散係数、およびデータ·レート

選択した変調に応じて、帯域幅、拡散係数、およびデータレートの定義を、\Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h 内で次の表に示すように行うことができます。

表 31. SubGHz_Phy_PingPong の帯域幅、SF および DR の設定

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

LORA_BANDWIDTH は、同等の周波数に対応して、0 ~ 2 の範囲内である必要があります。

[0: 125kHz、1：250kHz、2： 500kHz].

LORA_SPREADING_FACTOR は、7 ~ 12 の範囲内である必要があります。拡散係数は、フレームの送信にかかる時間と送信電力に影響します。

FSK_BANDWIDTH は、4800 ~ 500000 の範囲（Hz）内である必要があります。

ビットレート値に相当する FSK_DATARATE は、通常、50 kbps に定義されます。

11.2.5 プリアンブル長

すべての送信/受信パケットには、プリアンブル（通常 8 つのシンボルを持つ）、ヘッダ、ペイロード（サイズはセクション 10.2.2 で定義）、および CRC フィールドが含まれます。

プリアンブル·フィールド·サイズの更新は、\Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_PingPong\SubGHz_Phy\App\subghz_phy_app.h 内で以下の表に示す定義により行われます。

表 32. SubGHz_Phy_PingPong プリアンブルの設定

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

11.3 SubGhz_Phy_PingPong アプリケーションのデバイス設定の概要

表 33. SubGhz_Phy_PingPong アプリケーション設定のスイッチ·オプション

プロジェクト·モジュール = アプリケーション

詳細	スイッチ·オプション	定義	場所
Rx/Tx の設定	RX_TIMEOUT_VALUE	Rx ウィンドウのタイムアウト	subghz_phy_app.c
	TX_TIMEOUT_VALUE	Tx ウィンドウのタイムアウト	
	MAX_APP_BUFFER_SIZE	最大データ·バッファ·サイズ	
	RX_TIME_MARGIN	Rx 終了から Tx 開始までの時間	
	FSK_AFC_BANDWIDTH	AFC バンド幅(Hz)	
	LED_PERIOD_MS	LED 点滅周期	
変調設定	USE_MODEM_LORA	LoRa モデムが選択されます。	subghz_phy_app.h
	USE_MODEM_FSK	FSK モデムが選択されます。	
LoRa/FSK 共通パラメータ	REGION_XXyyy	アクティブ LoRa リージョン:AS923、AU915、CN470、CN779、EU433、EU868、KR920、IN865、US915、または RU864。	
	RF_FREQUENCY	トランシーバの使用周波数	
	TX_OUTPUT_POWER	RF 出力電力:-17 ~ 22 dBm	
	PAYLOAD_LEN	データ·バッファ·サイズ	
LoRa 固有のパラメータ	LORA_BANDWIDTH	バンド幅: • 0:125 kHz • 1:250 kHz • 2:500 kHz • 3:予約済みです。	
	LORA_SPREADING_FACTOR	拡散係数:SF7 から SF12	
	LORA_CODINGRATE	符号化レート: • 1:4/5 • 2:4/6 • 3:4/7 • 4:4/8	
	LORA_PREAMBLE_LENGTH	Tx/Rx プリアンブルの長さ	
	LORA_SYMBOL_TIMEOUT	タイムアウトまでにチェックされるシンボル数	
	LORA_FIX_LENGTH_PAYLOAD_ON	固定/動的ペイロード長オプション	
	LORA_IQ_INVERSION_ON	IQ 反転オプション	
FSK 固有のパラメータ	FSK_FDEV	周波数偏差(Hz)	
	FSK_DATARATE	データ·レート(bit/s)	
	FSK_BANDWIDTH	バンド幅(Hz)	
	FSK_PREAMBLE_LENGTH	Tx/Rx プリアンブルの長さ	

詳細	スイッチ·オプション	定義	場所
FSK 固有のパラメータ	FSK_FIX_LENGTH_PAYLOAD_ON	固定/動的ペイロード長オプション	subghz_phy_app.h
デバッグ	DEBUGGER_ENABLED	SWD ピンを有効にします。	sys_conf.h
低消費電力	LOW_POWER_DISABLE	低消費電力モードを無効にします。	
トレース有効	APP_LOG_ENABLED	トレース·モードを有効にします。	
トレース·レベル	VERBOSE_LEVEL	トレース·レベルを有効にします。	

12 SubGhz_Phy_Per アプリケーション

SubGHz_Phy_Per アプリケーションは、1 つの Tx デバイスと 1 つの Rx デバイスの間で IBM® ホワイトニングを行うパケットエラーレート·テストです。

Tx デバイス

`/SubGHz_Phy/App/subghz_phy_app.c` 内の `#define TEST_MODE to RADIO_TX` を更新します。コンパイルしてロードします。

パケットの内容は `preamble | sync | payload length | payload | crc` です。ここで、

- `crc` は `payload length` と `payload` を使用して計算されます。
- ホワイトニングは、`payload length | payload | crc` を元に計算されます。

送信が開始され、GFSK 50 Kbit/s、64 バイトのペイロードで継続されます。ユーザ·ボタン 1 では `packet length` を 16 バイト単位でインクリメントします。ユーザ·ボタン 2 では `packet length` を 1 バイト単位でインクリメントします。ユーザ·ボタン 3 では、パケット `payload mode` をランプ(0x00、0x01..)から prbs9 まで切り替えます。

Tx での無線送信中、青色の LED がオンになります。

Rx デバイス

`/SubGHz_Phy/App/subghz_phy_app.c` 内の `#define TEST_MODE to RADIO_RX` を更新します。コンパイルしてロードします。

Rx が OK の場合、緑色の LED が点灯します。Rx が KO の場合、赤色の LED が点灯します。

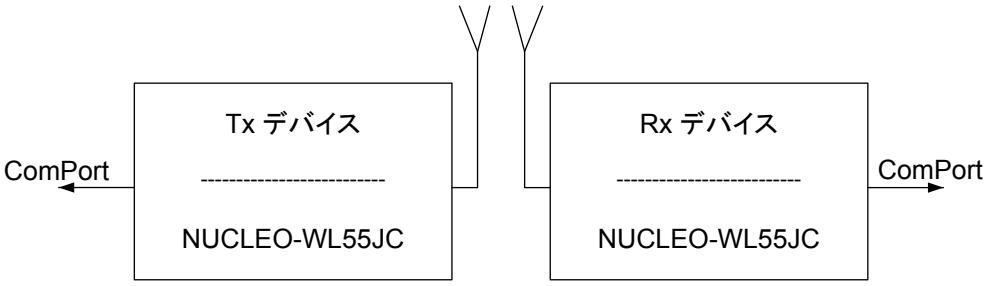
SubGhz_Phy_Per プロジェクトを起動するには、

`\Projects\<target>\Applications\SubGHz_Phy\SubGHz_Phy_Per` に移動して、LoRa End_Node プロジェクトと同じ手順で該当するツールチェーンを起動させます。

12.1 SubGhz_Phy_Per ハードウェア/ソフトウェア環境のセットアップ

STM32WL Nucleo ボード(NUCLEO-WL55JC)をセットアップするには、下図に示すように、このボードとコンピュータを、USB ケーブル(Type-A - Mini-B)と ST-LINK コネクタ(CN1)で接続します。

図 8. SubGhz_Phy_Per アプリケーションのセットアップ



12.2 SubGhz_Phy_Per アプリケーションのデバイス設定の概要

表 34. SubGhz_Phy_Per アプリケーション設定のスイッチ･オプション

プロジェクト･モジュール = アプリケーション

詳細	スイッチ･オプション	定義	場所
Rx/Tx の設定	RX_TIMEOUT_VALUE	Rx ウィンドウのタイムアウト	subghz_phy_app.c
	TX_TIMEOUT_VALUE	Tx ウィンドウのタイムアウト	
	RX_CONTINUOUS_ON	RX モード連続またはタイムアウトあり	
	TEST_MODE	デバイスモード: • RADIO_TX:パケットを無期限に送信 • RADIO_RX:パケットを無期限に受信	
	APP_LONG_PACKET	ロング･パケット･オプション(1)	
	MAX_APP_BUFFER_SIZE	最大データ･バッファ･サイズ	
変調設定	USE_MODEM_FSK	FSK モデムが選択されます。	subghz_phy_app.h
LoRa/FSK 共通パラメータ	REGION_XXyyy	アクティブ LoRa リージョン:AS923、AU915、CN470、CN779、EU433、EU868、KR920、IN865、US915、または RU864。	
	RF_FREQUENCY	トランシーバの使用周波数	
	TX_OUTPUT_POWER	RF 出力電力:-17 ~ 22 dBm	
	PAYLOAD_LEN	データ･バッファ･サイズ	
FSK 固有のパラメータ	FSK_FDEV	周波数偏差(Hz)	
	FSK_DATARATE	データ･レート(bit/s)	
	FSK_BANDWIDTH	バンド幅(Hz)	
	FSK_PREAMBLE_LENGTH	Tx/Rx プリアンブルの長さ	
	FSK_FIX_LENGTH_PAYLOAD_ON	固定/動的ペイロード長オプション	
デバッグ	DEBUGGER_ENABLED	SWD ピンを有効にします。	sys_conf.h
低消費電力	LOW_POWER_DISABLE	低消費電力モードを無効にします。	
トレース有効	APP_LOG_ENABLED	トレース･モードを有効にします。	
トレース･レベル	VERBOSE_LEVEL	トレース･レベルを有効にします。	

1. 詳細については、ドキュメント [5] を参照してください。

13 LoRaWAN コンテキスト管理の説明

NVM コンテキスト管理は、パワーオフまたはボードのリセット前に、現在の LoRaWAN スタック設定を ROM に格納するために使用されます。

推奨の解決策は、サイズが 1420 バイトの `LoRaMacNvmData_t` 構造体を、事前に割り当てられた 2 KB の ROM ページに格納することです。

この構造体は次のように定義されています。

表 35. LoRaWAN NVM コンテキスト構造体

フィールド	タイプ	サイズ（バイト）	説明
Crypto	`LoRaMacCryptoNvmData_t`	40	暗号レイヤに関連するパラメータ。TX/RX ごとに変更します。
MacGroup1	`LoRaMacNvmDataGroup1_t`	24	TX/RX の実行ごとに変更される確率の高い MAC 関連のパラメータ。
MacGroup2	`LoRaMacNvmDataGroup2_t`	228	TX/RX の実行ごとに変更される確率が高くない MAC 関連のパラメータ。
SecureElement	`SecureElementNvmData_t`	192	セキュア-エレメント関連のパラメータ（キー、識別子など）
RegionGroup1	`RegionNvmDataGroup1_t`	20	TX/RX 手順ごとに変更される確率が高いリージョナル実装関連のパラメータ。
RegionGroup2	`RegionNvmDataGroup2_t`	892	TX/RX 手順ごとに変更される確率が高くないリージョナル実装関連のパラメータ。
ClassB	`LoRaMacClassBNvmData_t`	24	クラス B 関連のパラメータ。

13.1 NVM コンテキスト管理データ API の定義

表 36. LoRaWAN コンテキスト管理 API とコールバック

機能	説明
`LmHandlerErrorStatus_t LmHandlerNvmDataStore (void)`	• LoRaMAC を停止します。 • Flash に格納する NVM データを準備します。 • OnStoreContextRequest() コールバックを呼び出して、FLASH_Write 操作を実行します。 • LoRaMAC を再開します。
`void OnRestoreContextRequest (void *nvm, uint32_t nvm_size)`	Flash に格納されている NVM データを RAM のバックアップ·バッファに復元します。
`void OnStoreContextRequest (void *nvm, uint32_t nvm_size)`	NVM データを RAM から Flash に格納します。
`void OnNvmDataChange (LmHandlerNvmContextStates_t state)`	格納または復元アクションが完了したときに、LmHandler によって呼び出されます。

コールバックが定義されていない場合、この機能は無効と見なされます。この機能は、セクション 9.2.11 コンテキスト管理に示すコンテキスト管理の定義によって有効になります。

14 デュアルコアの管理

STM32WL5x デバイスには、次の 2 つの Cortex が内蔵されています。

- Cortex-M4（CPU1）
- Cortext-M0+（CPU2）

デュアルコア・アプリケーションでは、CPU1 にマッピングされるアプリケーション部分はスタックから分離され、ファームウェアの下位層は CPU2 にマッピングされます。

提案されているデュアルコア・モデルでは、2 つの分離されたバイナリが生成されます。CPU1 バイナリは 0x0800 0000 に配置され、CPU2 バイナリは 0x0802 0000 に配置されます。

1 つのバイナリの関数アドレスは、他方のバイナリからはわかりません。そのため、通信モデルを導入する必要があります。このモデルの目的は、ユーザが CPU2 のコアスタックの動作に影響を与えることなく、CPU1 のアプリケーションを変更できるようにすることです。ただし、ST では、2 つの CPU の実装をオープン・ソースで提供しています。

コア間のインタフェース接続は、セクション 14.1 に示すように、IPCC ペリフェラル（プロセッサ間通信コントローラ）とコア間メモリによって行われます。

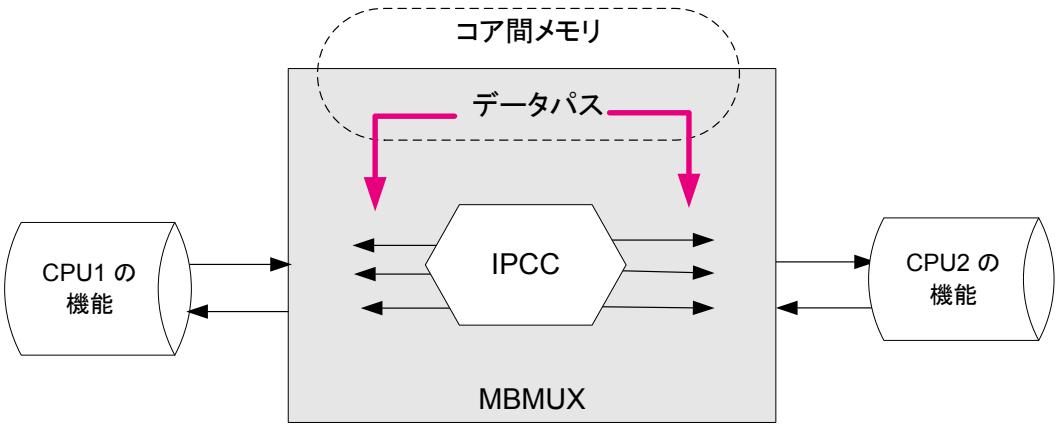
このデュアルコア実装は、メールボックス・メカニズムを介したメッセージ・ブロック処理により、シングルコア・プログラムの実行と同じように動作するように設計されています。

14.1 メールボックス・メカニズム

メールボックスは、2 つのプロセッサ間でデータを交換する方法を実装するサービスです。下の図に示すように、メールボックスは 2 つのリソースで構築されます。

- IPCC：このハードウェア・ペリフェラルは、リモート CPU への割込みをトリガし、通知の完了時に割込みを受信するために使用されます。IPCC は高度な設定が可能で、各割込み通知を無効化/有効化できます。IPCC 内にメモリ管理はありません。
- コア間メモリ：この共有メモリは、両方の CPU からの読出し/書込みができます。2 つの CPU 間で交換されるデータを含むすべてのバッファを格納するために使用されます。

図 9. メールボックスの概要



メールボックスを指定すると、下位互換性を損なうことなく、バッファの定義をある程度まで変更できます。

14.1.1 メールボックス・マルチプレクサ（MBMUX）

図 10 に示すように、交換するデータは、使用可能な 12 個の IPCC チャネル（方向ごとに 6 個ずつ）を介して通信する必要があります。これは、メッセージのルーティングを担当するファームウェア・コンポーネントである MBMUX（メールボックス・マルチプレクサ）を介して行われます。これらのチャネルは 1 ~ 6 で識別されます。追加のチャネル 0 は、システム機能専用です。

データ型は、機能と呼ばれるグループに分けられています。各機能は、独自の MBMUXIF（MBUX インタフェース）を介して MBMUX とインタフェース接続します。

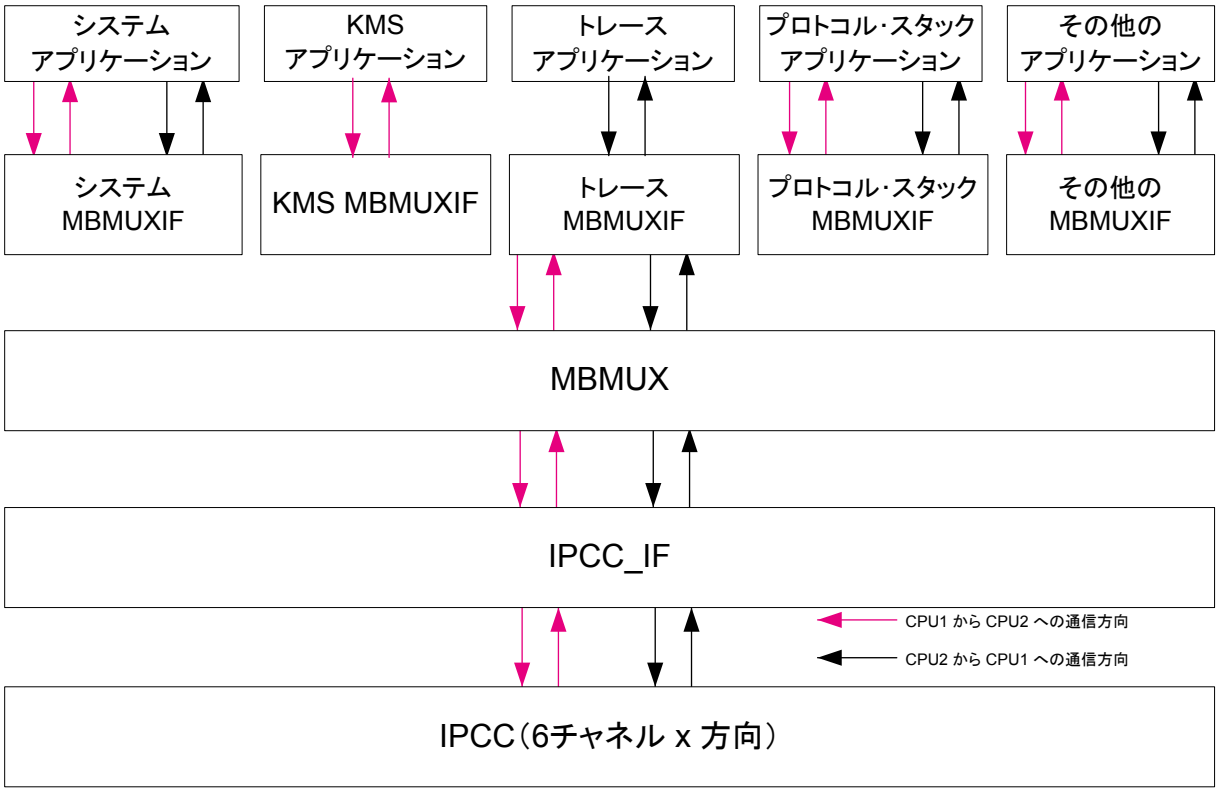
メールボックスは、別のコアによって実行される関数を抽象化するために使用されます。

14.1.2 メールボックスの機能

STM32WL5x デバイスの MBMUX には次の機能があります。

- システム：システムに関連するすべての通信をサポートします。
  ここには、サポートされているスタックの 1 つに関連するメッセージも、どのスタックにも関連しないメッセージも含まれます。CPU1 チャネル 0 は、コマンドがポストされたことを CPU2 に通知し、そのコマンドへの応答を CPU2 から受信するために使用されます。CPU2 チャネル 0 は、非同期イベントがポストされたことを CPU1 に通知するために使用されます。
  次のサービスがシステム・チャネルにマッピングされます。
  - システムの初期化
  - IPCC チャネルと機能の登録
  - 機能の属性および機能について交換される情報
  - 優先度の高い操作（RTC 通知など）のために使用可能な追加のシステム・チャネル
- トレース
  CPU2 は、IPCC を介して CPU1 に送信された情報またはデバッグのために、循環キューに書き込みます。CPU1 は、CPU1 のログに使用されるのと同じチャネル（USART など）にこの情報を出力することによって、この情報の処理を担当します。
- KMS（キー管理サービス）
- 無線
  Sub-GHz 無線は、CPU2 スタックを経由することなく直接インタフェースできます。専用のメールボックス・チャネルが使用されます。
- プロトコル・スタック
  このチャネルは、すべてのプロトコル・スタック・コマンド（Init やリクエストなど）や、スタック実装プロトコルに関連するイベント（応答/表示）のインタフェースに使用されます。

図 10. MBMUX - 機能と IPCC チャネルの間のマルチプレクサ



MBMUX を使用するには、機能を登録する必要があります（デフォルトで登録され、常に IPCC チャネル 0 にマッピングされるシステム機能を除く）。この登録により、リクエストされた数の IPCC チャネルが機能に動的に割り当てられます。通常、方向（CPU1 から CPU2、CPU2 から CPU1 へ）ごとに 1 つです。

次の場合、この機能には一方向のチャネルのみが必要です。

- トレース機能は、CPU2 から CPU1 にデバッグ情報を送信することのみを目的としています。
- KMS は、CPU1 から CPU2 に関数の実行をリクエストするためにのみ使用されます。

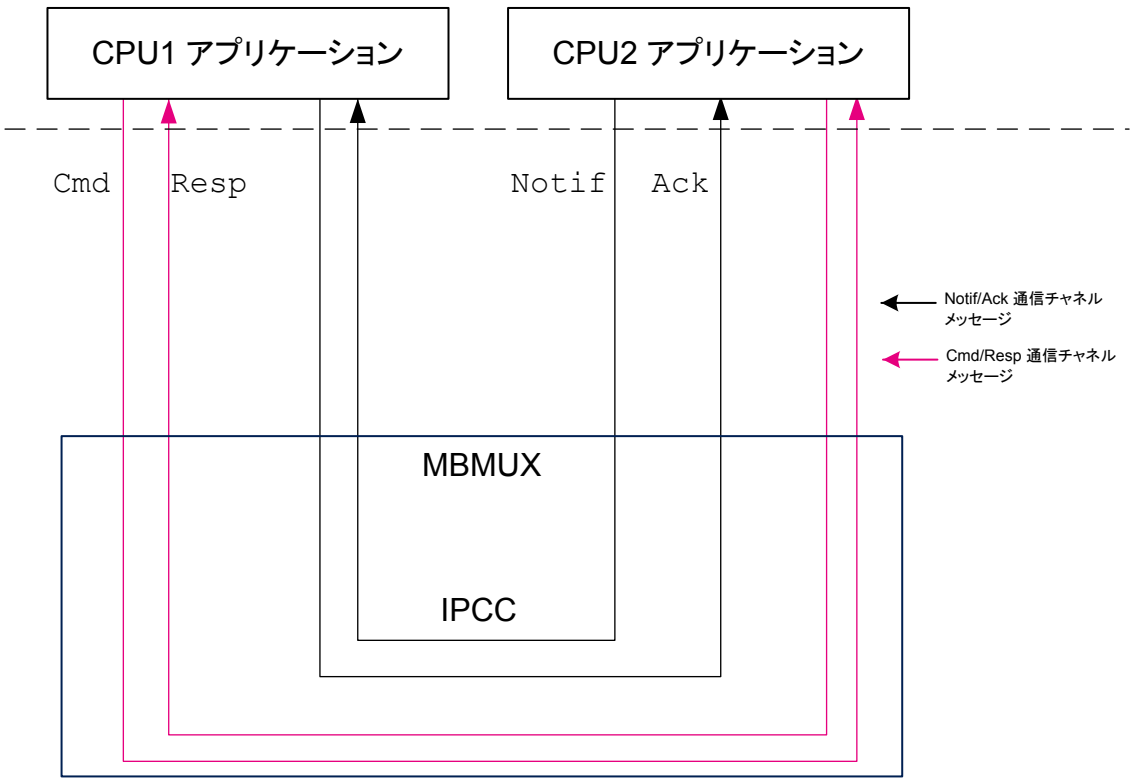
注
- RTC アラーム A は、1 つの IPCC IRQ を使用して割込みを転送するもので、機能とはみなされません。
- KMS ラッパーを機能として使用できるようにするには、ユーザは KMS ラッパーの追加を検討する必要があります。

14.1.3 MBMUX メッセージ

メールボックスでは、次のタイプのメッセージを使用します。

- `Cmd`:CPU1 から CPU2 に送信されるコマンドで、次の内容で構成されます。
  - `Msg ID` は、CPU1 によってコールされ、CPU2 に実装されている関数を示します。
  - `Ptr buffer params` は上記関数のパラメータを含むバッファをポイントします。
  - `Number of params`
- `Resp`:CPU2 から CPU1 に送信される応答で、次の内容で構成されます。
  - `Msg ID`:(`Cmd Msg ID` と同じ値)
  - `Return value`:上記の関数の戻り値が格納されます。
- `Notif`:CPU2 から CPU1 に送信される通知で、次の内容で構成されます。
  - `Msg ID` は、CPU2 によってコールされ、CPU1 に実装されているコールバック関数を示します。
  - `Ptr buffer params` は上記関数のパラメータを含むバッファをポイントします。
  - `Number of params`
- `Ack`:CPU1 から CPU2 に送信される確認応答であり、次の内容で構成されます。
  - `Msg ID`:(`Notif Msg ID` と同じ値)
  - `Return value`:上記のコールバック関数の戻り値が格納されます。

図 11. MBMUX および IPCC チャネルを使用したメールボックス・メッセージ



14.2 コア間メモリ

コア間メモリは、両方のコアからアクセス可能な一元的なメモリであり、データ、関数パラメータ、戻り値を交換するためにコアによって使用されます。

14.2.1 CPU2 の機能

サポートされる機能（CPU2 に実装されているプロトコルスタック、各スタックのバージョン番号、サポートされるリージョンなど）を把握するため、CPU2 の機能の一部が CPU1 で認識されている必要があります。

これらの CPU2 の機能は、features_info テーブルに格納されます。図 RAM のマッピング に示すように、CPU2 の機能を公開するために、初期化時に CPU1 によってこのテーブルのデータがリクエストされます。

features_info テーブルは以下で構成されます。

- `Feat_Info_Feature_Id`:機能名
- `Feat_Info_Feature_Version`:現在の実装で使用されている機能のバージョン番号

MB_MEM2 はこれらの CPU2 の機能を格納するために使用されます。

14.2.2 CPU1 コールから CPU2 関数を実行するメールボックス·シーケンス

CPU1 から CPU2 に対して `feature_func_X()` をコールする必要がある場合、同じ API を持つ `feature_func_X()` を CPU1 に実装する必要があります。

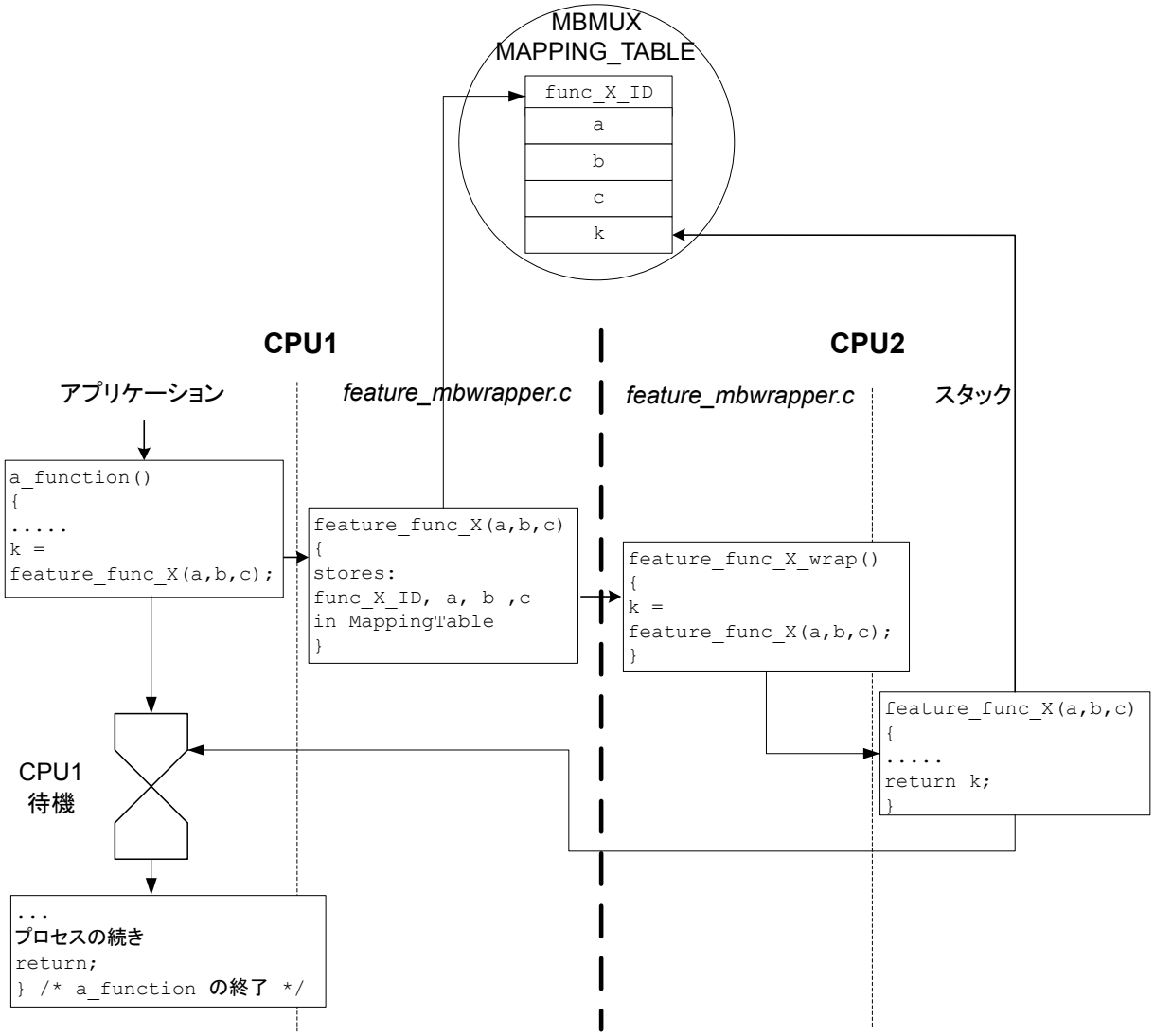
1. CPU1 は、マッピング·テーブルに `feature_func_X()` のパラメータが含まれるコマンド を送信します。
   a. 登録中の初期化時に `feature_func_X()` に関連付けられた `func_X_ID` が、マッピング·テーブルに追加されます。`func_X_ID` は両方のコアで認識される必要があります。これはコンパイル時に修正されます。
   b. CPU1 は、CPU2 で `feature_func_X()` が実行されるのを待って、低消費電力モードに移行します。
   c. CPU2 は、低消費電力モードだった場合はウェイクアップし、`feature_func_X()` を実行します。
2. CPU2 は、応答を送信し、マッピング·テーブルに戻り値を入力します。
   a. IPCC 割込みにより CPU1 がウェイクアップします。
   b. CPU1 がマッピング·テーブルから戻り値を取得します。

反対に、CPU2 から CPU1 に対して `feature_func_X_2()` をコールする必要がある場合は、同じ API を持つ `feature_func_X_2()` を CPU2 に実装する必要があります。

1. CPU2 は、マッピング·テーブルに `feature_func_X_2()` が含まれる通知を送信します。
2. CPU1 は、確認応答を送信し、マッピング·テーブルに戻り値を入力します。

下の図に詳しいシーケンスを示します。

図 12. CPU1 から CPU2 への feature_func_X() のプロセス

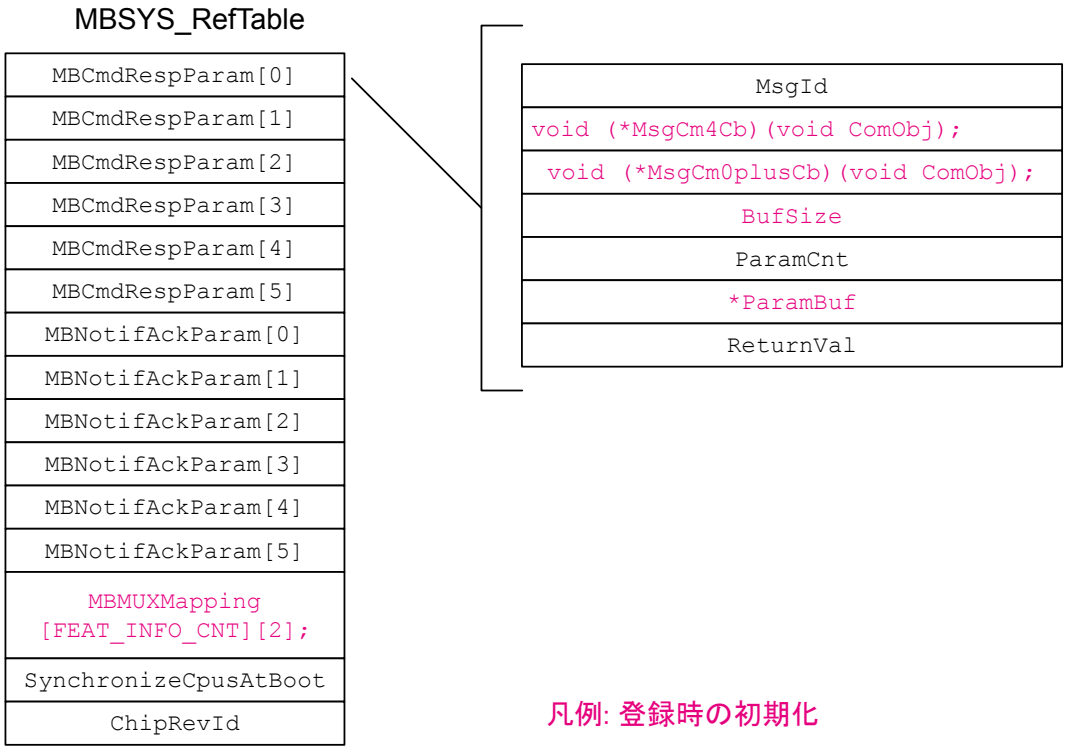


14.2.3 マッピング·テーブル

マッピング·テーブルは、図 12 の MBMUX 領域に共通の構造体です。RAM のマッピングでは、メモリ·マッピングは MAPPING_TABLE として参照されます。

MBMUX 通信テーブル(MBSYS_RefTable)を下図に示します。

図 13. MBMUX 通信テーブル



この MBSYS_RefTable の内容は次のとおりです。

- 6 個の各 IPCC チャネルの、コマンド/応答および通知/確認応答パラメータの両方に対する 2 つの通信パラメータ構造体。
  図 12 の MBMUX マッピング·テーブル領域に示した各通信パラメータは、次の内容で構成されます。
  - `MsgId`:`feature_func_X()` のメッセージ ID
  - `*MsgCm4Cb`:CPU1 コールバック `feature_func_X()` へのポインタ
  - `*MsgCm0plusCb`:CPU2 コールバック `feature_func_X()` へのポインタ
  - `BufSize`:バッファ·サイズ
  - `ParamCnt`:メッセージ·パラメータ番号
  - `ParamBuf`:パラメータへのメッセージ·ポインタ
  - `ReturnVal`:`feature_func_X()` の戻り値
- `MBMUXMapping`:チャネルを機能にマッピングするために使用されるチャート
  このチャートは、登録中の MBMUX の初期化時に入力されます。たとえば、無線機能が `Cmd/Response channel number = 1` に関連付けられている場合、`MBMUXMapping` を `[FEAT_INFO_RADIO_ID][1]` と関連付ける必要があります。
- `SynchronizeCpusAtBoot`:図 14 のシーケンス図に示すように、CPU1 と CPU2 の処理を同期させるために使用されるフラグ。
- `ChipRevId`:ハードウェア·リビジョン ID を格納します。

MB_MEM1 は `command/response set ()` パラメータを送信し、CPU1 のための戻り値を取得するために使用されます。

14.2.4 オプションバイト警告

オプションバイトの誤ったロードを避けるため、コードにトラップが配置されます（製品正誤表の「Option-byte loading failure at high MSI system clock frequency（MSI システムクロック周波数が高いときのオプションバイト・ローディング・エラー）」セクションを参照）。システム・クロックが 16 MHz 以下に設定されている場合、トラップは解除できます。

14.2.5 RAM のマッピング

下の表に、CPU1 および CPU2 の RAM 領域とコア間メモリのマッピングの詳細を示します。

表 37. STM32WL5x RAM のマッピング

ページ インデックス (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
割当て領域/セクション	CPU1 RAM																CPU1:RAM2_Shared		CPU2:RAM2_Shared		CPU2:RAM2										

1. 各ページ 2 KB。

表 38. STM32WL5x RAM の割当てと共有バッファ

CPU 領域	セクション	モジュール	割当てシンボル	サイズ（バイト）	合計（バイト）
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	

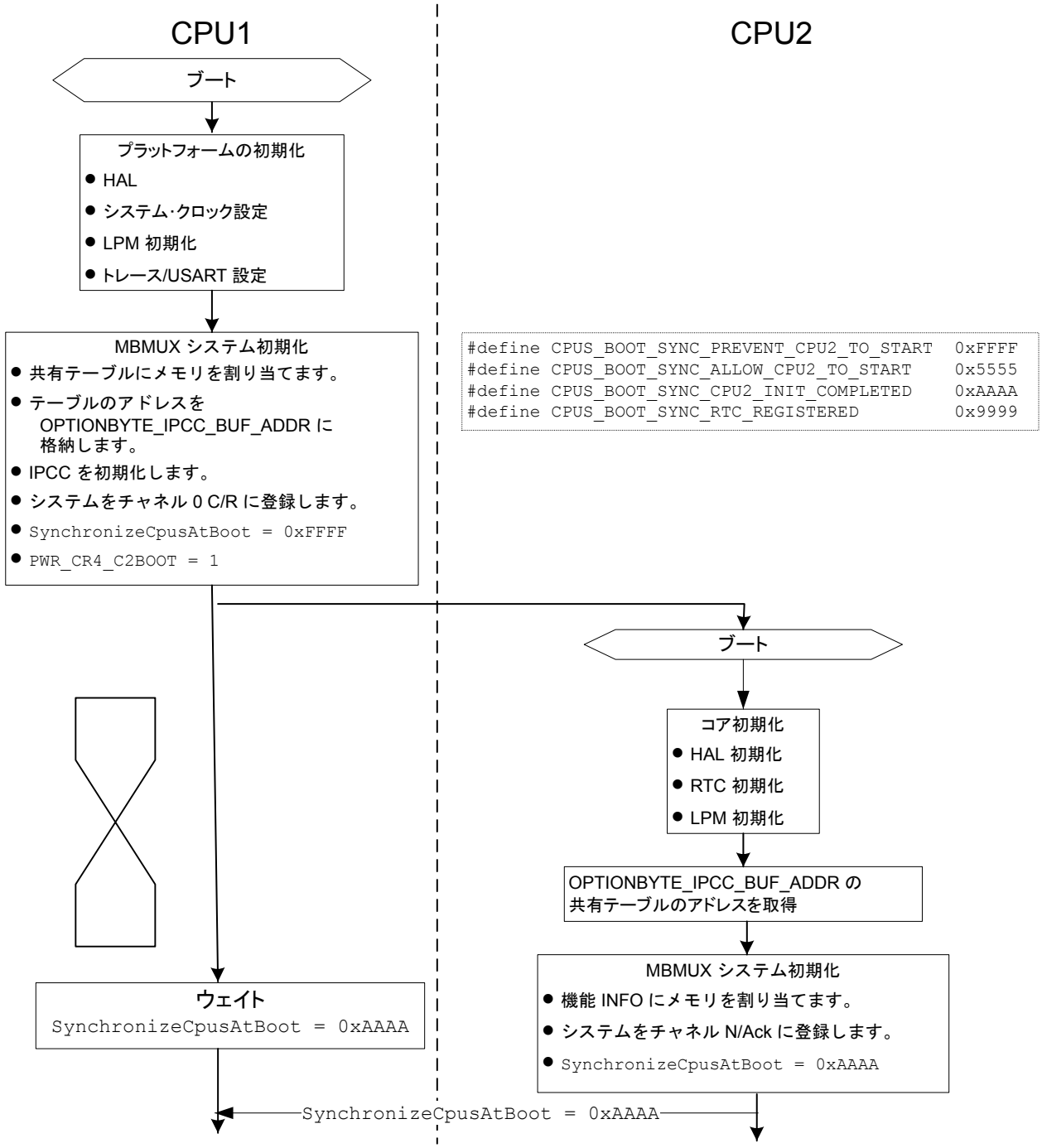
CPU 領域	セクション	モジュール	割当てシンボル	サイズ（バイト）	合計（バイト）
CPU2：RAM2_Shared	MB_MEM3[1]	MBMUX_TRACE	uint8_t ADV_TRACE_Buffer[]	1024	-
		MBMUX_LORAWAN	LoraInfo_t loraInfo	16	
		LMH_MBWRAPPER	uint8_t aLoraMbWrapShare2Buffer[]	280	
		RADIO_MBWRAPPER	uint8_t aRadioMbWrapRxBuffer[]	256	
CPU2：RAM2	readwrite	-			
	CSTACK				
	HEAP				

1. この新しいセクションは、STM32CubeIDE でこれらの BSS RAM 変数を初期化するために Flash の使用が過剰になるのを防ぎます。

14.3 起動シーケンス

CPU1 および CPU2 の起動シーケンスを下の図に示します。

図 14. 起動シーケンス

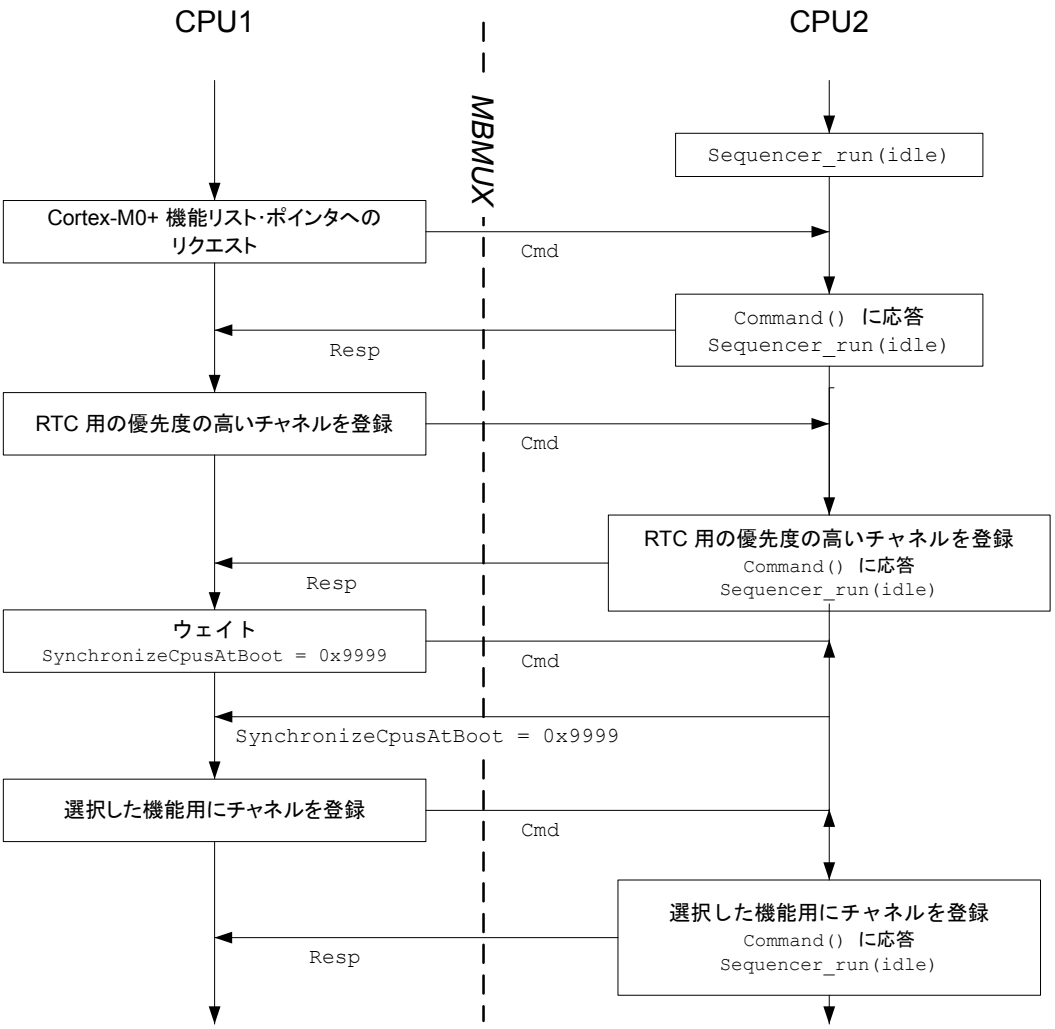


それぞれの手順を以下に示します。

1. CPU1、つまりこの初期化シーケンスのマスタ·プロセッサの手順:
   a. プラットフォームの初期化を実行します。
   b. MBMUX システムを初期化します。
   c. `PWR_CR4_C2BOOT` フラグを 1 に設定して、CPU2 を起動します。
   d. CPU2 により `SynchronizeCpusAtBoot` フラグが 0xAAAA に設定されるのを待ちます。
2. CPU2 がブートし、:
   a. コアの初期化を実行します。
   b. 共有テーブル·アドレスを取得します。
   c. MBMUX システムを初期化します。
   d. `SynchronizeCpusAtBoot` を 0xAAAA に設定し、初期シーケンスが終了してレディ状態であることを CPU1 に通知します。
3. CPU1 が、この CPU2 の通知に対して確認応答します。

その後、両方のコアが初期化され、下の図に示すように、MBMUX を介して初期化が続行されます。

図 15. MBMUX の初期化

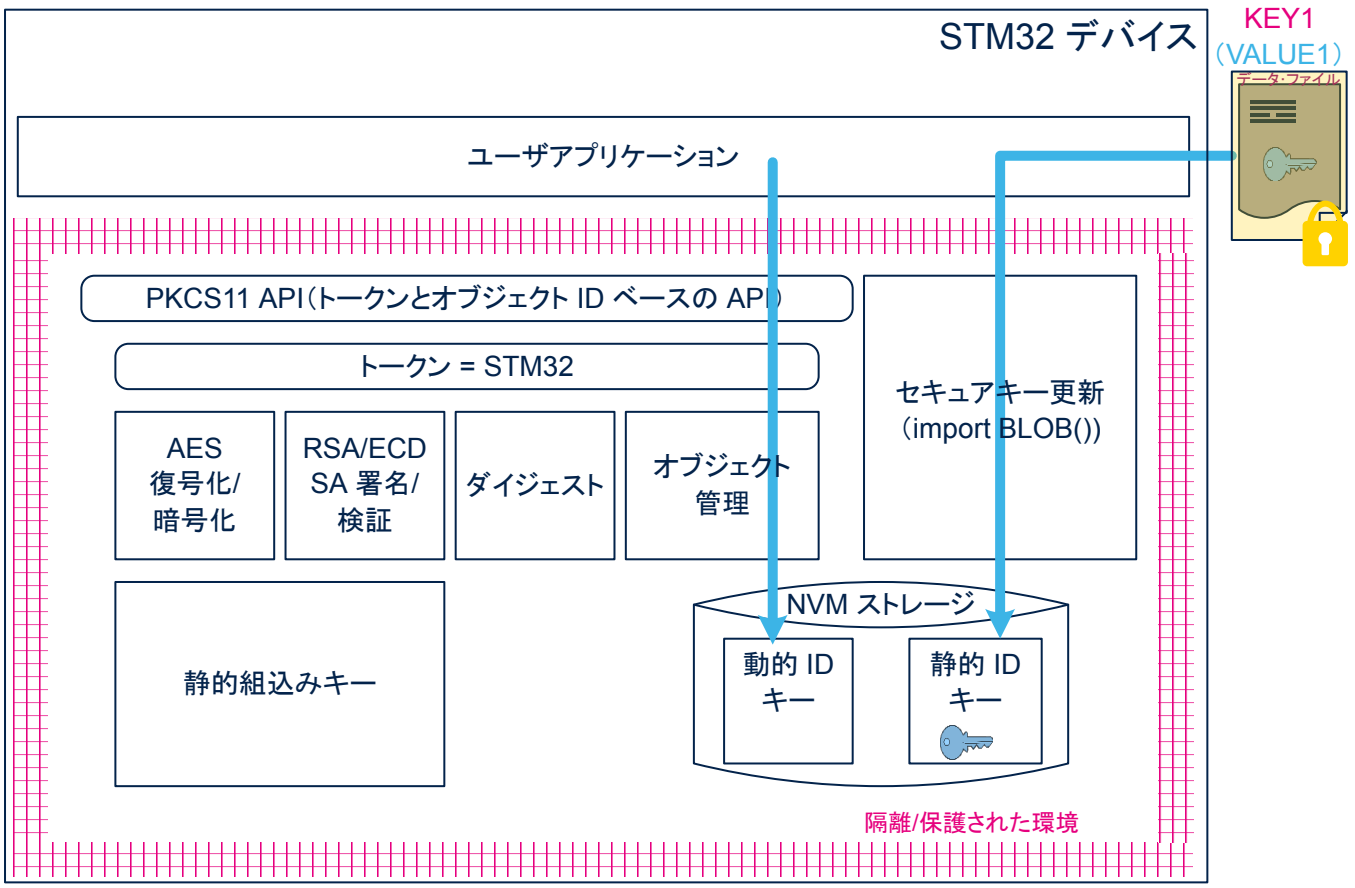


15 キー管理サービス(KMS)

キー管理サービス(KMS)は、標準 PKCS#11API(OASIS で開発)を通じて暗号サービスを提供し、キー値を呼び出し側に対して抽象化するために使用されます(直接キー値を使用せずにオブジェクト ID を使用します)。

KMS は、下の図に示すように、保護/隔離されている環境の外で実行された不正なコードがキー値にアクセスできないように、保護/隔離されている環境内で実行できます。

図 16. KMS の全体的なアーキテクチャ



詳細については、ユーザ・マニュアル「Getting Started with the SBSFU of STM32CubeWL(STM32CubeWL の SBSFU スタートガイド)」(UM2767)の KMS セクションを参照してください。

KMS モジュールを有効化するには、C/C++ コンパイラのプロジェクト・オプションで `KMS_ENABLE` を 1 に設定する必要があります。

KMS は、次に示す PKCS #11 API のみをサポートします。

- オブジェクト管理機能(作成/更新/削除)
- AES 暗号化/復号化機能(CBC、CCM、ECB、GCM、CMAC アルゴリズム)
- ダイジェスト機能
- RSA および ECDSA 署名/検証機能
- キー管理機能(キーの生成/導出)

15.1 KMS のキーの種類

KMS は 3 種類のキーを管理しますが、使用されるのは次の 2 種類のみです。

- 静的組込みキー
  - コードに組み込まれた変更できない事前定義キー
  - 不変キー
- NVM_DYNAMIC キー
  - ランタイム·キー
  - KMS を使用してキーが作成されたときに定義されることがあるキー ID(`DeriveKey()` または `CreateObject()`)
  - 変更可能なキーとして定義または削除できるキー

15.2 KMS のキーの定義

スタックによって使用される静的キーと動的キーは、占有するサイズが異なります。

静的キー

静的キーは、次の 2 つの要素で構成されます。

- blob ヘッダ:5 つの 4 バイトのフィールド(合計 = 20 バイト):`version`、`configuration`、`blob_size`、`blob_count`、および `object_id`。
- blob バッファ:次のように定義される要素のリストに含まれる、必須およびオプションの blob 要素の一部。

表 39. グローバル KMS blob 要素

属性	値	必要	サイズ(バイト)	説明
CKA_CLASS	CKO_SECRET_KEY	○	12	blob 要素のタイプ
CKA_KEY_TYPE	CKK_AES	○	12	キーの種類
CKA_VALUE	「KEY_VALUE」	○	24	キー値(uint32_t フォーマット)
CKA_DERIVE	TRUE/FALSE	×	12	機能をデフォルトで有効/無効にするためのオプションのパラメータ。これらのフィールドは、定義されていない場合、TRUE です。
CKA_ENCRYPT	TRUE/FALSE	×	12	
CKA_DECRYPT	TRUE/FALSE	×	12	
CKA_COPYABLE	TRUE/FALSE	×	12	
CKA_EXTRACTABLE	TRUE/FALSE	×	12	
CKA_LABEL	「UNIQUE LABEL」	○	静的キーの場合は 12 動的キーの場合は 16	固有のラベル

例:

Blob ヘッダと 8 つの blob 要素で構成される静的キー(CKA_CLASS、CKA_KEY_TYPE、CKA_VALUE、CKA_LABEL、CKA_DERIVE、CKA_DECRYPT、CKA_COPYABLE、および CKA_EXTRACTABLE)、は合計で 128 バイトを使用します(blob ヘッダ = 20 バイト、blob バッファ = (12 × 7 + 24) = 108 バイト)。

動的キー

動的キーは、データ·ヘッダ、blob ヘッダ、および blob バッファの 3 つの要素で構成され、次のものが含まれます。

- データ·ヘッダ:8 つの 4 バイトのフィールド(合計 = 32 バイト):`Magic1`、`Magic2`、`slot`、`instance`、`next`、`data_type`、`size`、および `checksum`。

例:

データ·ヘッダ、blob ヘッダ、および 5 つの blob 要素(CKA_CLASS、CKA_KEY_TYPE、CKA_VALUE、CKA_LABEL、および CKA_EXTRACTABLE)で構成される動的キーは、合計で 128 バイトを使用します(データ·ヘッダ = 32 バイト、blob ヘッダ = 20 バイト、および blob バッファ = (12 + 12 + 24 + 12 + 16) = 76 バイト。

注
- NVM 動的メモリは常に初期データ·ヘッダ要素から始まります。
- 動的キーが「削除」されるたび（古いキーが新しいキー値に置き換えられるなど）、前のインスタンスが使用できなくなったことを宣言する追加のデータ·ヘッダがメモリに書き込まれます。

15.3 LoRaWAN のキー

STM32CubeWL アプリケーションでは、KMS はデュアルコア·アプリケーションでのみ、CPU2 上で使用されます。ルート·キーは静的組込みキーとして選択されます。派生キーはすべて NVM_DYNAMIC キーです。

LoRaWAN スタックの不変ルート·キーの詳細を下の表に示します。

表 40. blob 属性を持つ LoRaWAN 静的キー

属性	キー				
	LoRaWAN_Zero_Key	LoRaWAN_APP_Key	LoRaWAN_NWK_Key	LoRaWAN_NWK_S_Key (ABP のみ)	LoRaWAN_APP_S_Key (ABP のみ)
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:#define KEY_EXTRACTABLE によって定義			
CKA_ LABEL	「UNIQUE LABEL」				

その他すべてのキーは NVM_DYNAMIC によって生成された可変のキーです。下の表に詳細を示します。

表 41. blob 属性を持つ LoRaWAN 動的キー

属性	キー							
	LoRaWAN_NWK_S_Key (OTAA のみ)	LoRaWAN_APP_S_Key (OTAA のみ)	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:#define KEY_EXTRACTABLE によって定義							
CKA_ LABEL	「UNIQUE LABEL」							

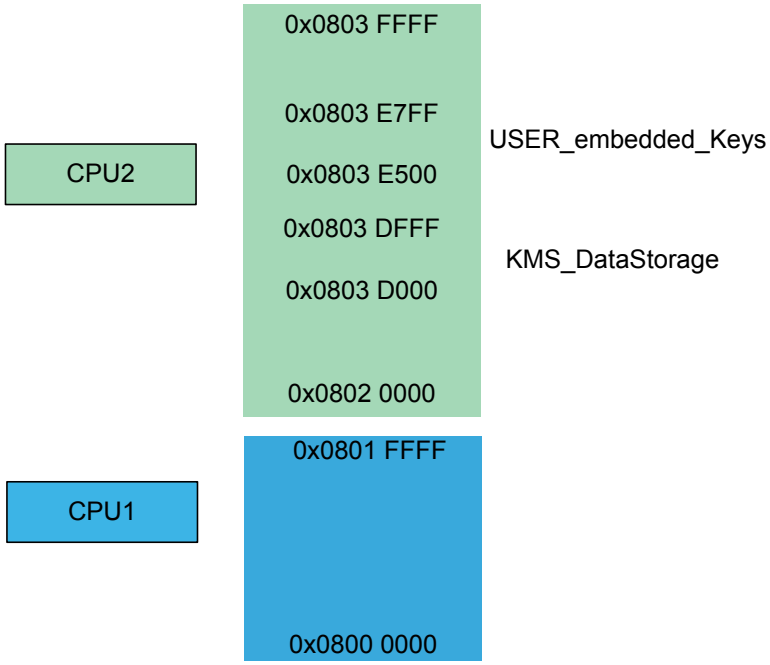
15.4 ユーザ·アプリケーションのための KMS キー·メモリ·マッピング

静的組込みキーは USER_embedded_Keys に対応しています(ルート·キーに使用)。それらは Flash メモリ/ROM の専用のデータ·ストレージに配置されます。ユーザ·アプリケーション用のリンカ·ファイルは、下の図に示すように、0x0803 E500 から 0x0803 E7FF の場所に格納されています。

NVM_DYNAMIC キーは KMS キー·データ格納領域 KMS_DataStorage に配置されます。

データ·ストレージ領域の合計は 4 KB である必要があります(セクション 15.5 を参照)。配置されている場所は、0x0803 D000 から 0x0803 DFFF です(下の図に示す)。さらにキーが必要な場合は、このサイズを増やすことができます。

図 17. ROM メモリマッピング



15.5 KMS データ·ストレージのための NVM のサイズ設定

NVM は 2KB のページで構成されます。ダブル·バッファリング(フリップ/フロップ EEPROM エミュレーション·メカニズム)により、各ページには「ツイン」が必要です。したがって、NVM に割り当てられる最小値は 4KB です。割当てサイズは、リンカ·ファイルで定義されます。

アプリケーションによって提案されるリンカ·ファイルでは、最小許容サイズ(2 × 2KB)が使用されます。関連する制限/欠点を以下で説明します。ユーザは、アプリケーション固有のニーズに応じて NVM のサイズを設定する必要があります。

アプリケーションでは、動的キーを格納するためにだけ、NVM を使用します。データ·ヘッダ、blob ブヘッダ、および 5 つの要素からなる blob バッファを持つ LoRaWAN 動的キーは、108 バイトを使用します。空の NVM は 32 バイトのグローバル·データ·ヘッダで初期化され、旧式のキーの場合は、前のインスタンスを使用不可として宣言するために 32 バイトのデータ·ヘッダが追加で書き込まれます。

上記の値が与えられた場合、2KB にいくつのキーを格納できるかを評価することができます。

(2048 - 32) / (108 + 32) = 14.4 ==> フロップ動作前に 2KB のメモリ·ページに 14 個の動的キーを格納可能

ユーザ·アプリケーション設定で、NVM_DYNAMIC のみが使用されます。NVM_STATIC は blob を介して入力することができますが、ユーザ·アプリケーションでは対応していません。

NVM_DYNAMIC は、派生キー(C_DeriveKey() 経由)およびルート·キー(C_CreateObject() 経由)をホストできます。

LoRaWAN アプリケーションは、以下を生成します。

- 2 つの派生キー(それぞれ ABP モードで JOIN)
- 4 つの派生キー(それぞれ OTAA モードで JOIN)

より複雑なシナリオ(マルチキャストの設定など)では、同時に有効な最大 10 個の派生キーを生成できます。14 個を超えるキーを使用する 1 つのアプリケーションを作成する場合は、追加の NVM ページをリンカ·ファイルに割り当てる必要があります。

NVM のサイズが小さいほど、NVM の書込み/消去の頻度が高くなり、推定寿命は短くなります。

キーの破壊では、このキーが消去されるわけではなく、「破棄済み」としてタグ付けされ、次のフリップ/フロップ切り替えではコピーされません。破棄フラグも NVM バイトの一部を占有します。8 つのキーを破棄した場合、残りの場所は 4 つのキー未満になります。

JOIN のたびに 4 つのキーが生成され、前の JOIN キーが破棄された後のシナリオでは、推定寿命は次のように見積もられます。

- 3 回目の JOIN セッションでは、4 つの新しいキーが生成されますが、前回のキーのためのページ 1 には空きスペースがありません。4 つのキーすべて（有効なまま）はページ 2 に配置されます。NVM ページで可能なのは完全消去のみなので、ページ 1 はただちに消去されます。
- 5 回目の JOIN でも、ページ 2 は消去され、キーは再びページ 1 に格納されます。JOIN 回数が 40,000 回に達すると、2 つの NVM ページでは、Flash セクタの推定寿命である 10,000 回の消去が行われています。
- ユーザ･アプリケーションが非常に頻繁に（2 時間ごとなど）JOIN することが想定されている場合、NVM の推定寿命は 80.000 時間（約 9 年）となります。JOIN プロセスが 1 日 1 回実行された場合、寿命は 10 年よりも大幅に長くなります。

リクエストされた派生キーのうち、同時に有効なもの（破壊されない）が多いほど、フリップ/フロップ･メカニズムの効率が下がります。

結論として、NVM の寿命を保持する必要があるアプリケーションでは、同時に有効な（破棄されない）キーの数よりも NVM のサイズを大きく保つことが推奨されます。

注　古いキーは破棄する必要があります。そうしないと、ページ 1 が有効なキーで完全に満たされた場合、フリップ/フロップ切り替えができず、エラーが生成されます。

15.6 アプリケーションを構築するための KMS 設定ファイル

LoRaWAN アプリケーションで KMS を使用するには、コード

```
#define LORAWAN_KMS  1
```

を `CM0PLUS/LoRaWAN/Target/lorawan_conf.h` 内で設定します。

次のファイルに、SubGHz_Phy スタック･キーに関する情報を書き込む必要があります。

- 組込みキー構造体（`CM0PLUS/Core/Inc/kms_platf_objects_config.h` で定義）
- SubGHz_Phy スタック･キーに関連する組込みオブジェクト･ハンドル、KMS モジュールの使用（`CM0PLUS/Core/Inc/kms_platf_objects_interface.h` で定義）

15.7 組込みキー

SubGHz_Phy プロトコル･スタックの組込みキーは、ROM 領域に格納し、セキュアな追加ソフトウェア（SBSFU、セキュア･ブート、ファームウェア･アップデートなど）によってデータの機密性と完全性を保証する必要があります。SBSFU の詳細については、アプリケーション･ノート「STM32CubeWL での SBSFU の統合ガイド 」（AN5544）を参照してください。

これらの組込みキーは、図 17. ROM メモリマッピング に示すように ROM 内に配置されます。

16 LoRaWAN アプリケーションをセキュアにする方法

ドキュメント [7] では、SBSFU フレームワークを使用してデュアルコア LoRaWAN アプリケーションをセキュアにする方法について説明しています。

17 システム性能

17.1 メモリ·フットプリント

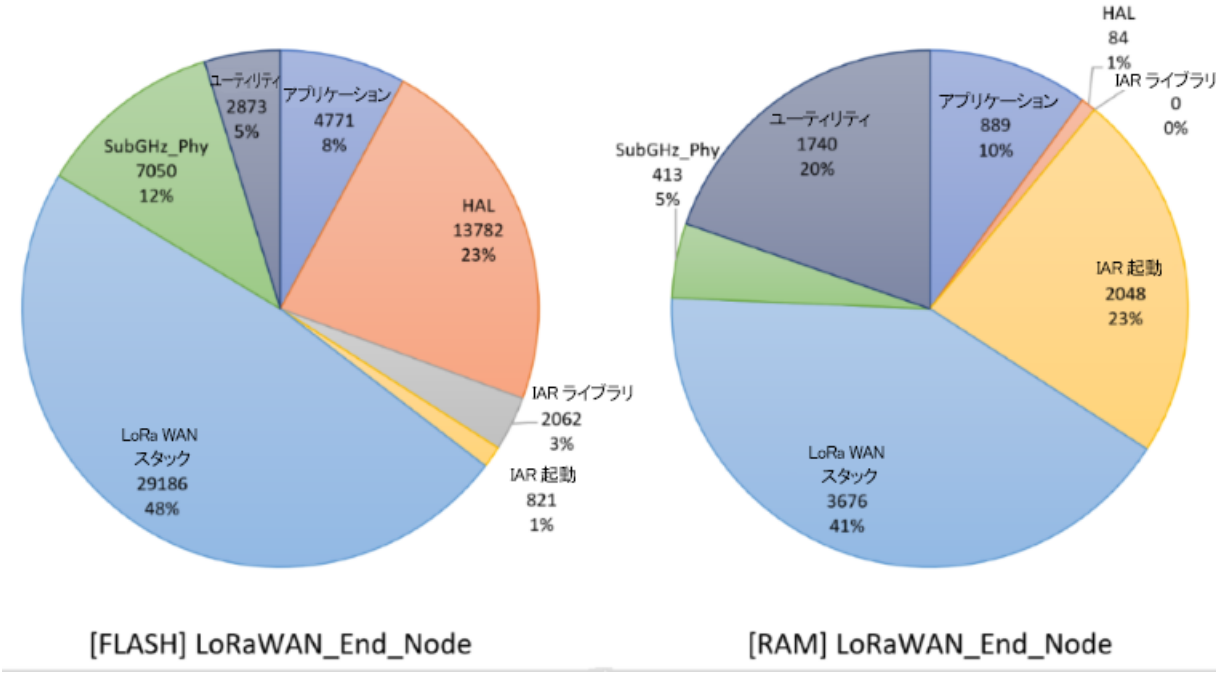
下の表の値は、IAR Embedded Workbench® コンパイラ（EWARM バージョン 8.30.1）の次の設定で測定したものです。:

- サイズの最適化レベル 3
- デバッグ·オプション:オフ
- トレース·オプション:VLEVEL_M（デバッグ·トレース有効）
- ターゲット:NUCLEO-WL55JC1
- LoRaWAN_End_Node アプリケーション
- LoRaMAC クラス A
- LoRaMAC リージョン EU868 および US915

表 42. LoRaWAN_End_Node アプリケーションのためのメモリ·フットプリント値

プロジェクト·モジュール	Flash メモリ(バイト)	RAM（バイト）	説明
アプリケーション	4771	889	コア、アプリケーション、およびターゲットのコンポーネント
LoRaWAN スタック	29186	3676	ミドルウェア Lmhandler インタフェース、暗号、MAC、およびリージョン
HAL	13782	84	STM32WL HAL および LL ドライバ
ユーティリティ	2873	1740	すべての STM32 サービス（シーケンサ、タイム·サーバ、低消費電力マネージャ、トレース、メモリ）
SubGHz_Phy	7050	413	ミドルウェア無線インタフェース
IAR ライブラリ	2062	0	独自の IAR ライブラリ
IAR のスタートアップ	821	2048	Int_vect、初期化ルーチン、初期化テーブル、CSTACK、および HEAP
全アプリケーション	**60545**	**8850**	**LoRaWAN_End_Node アプリケーションのためのメモリ·フットプリント**

図 18. Flash メモリと RAM のフットプリント

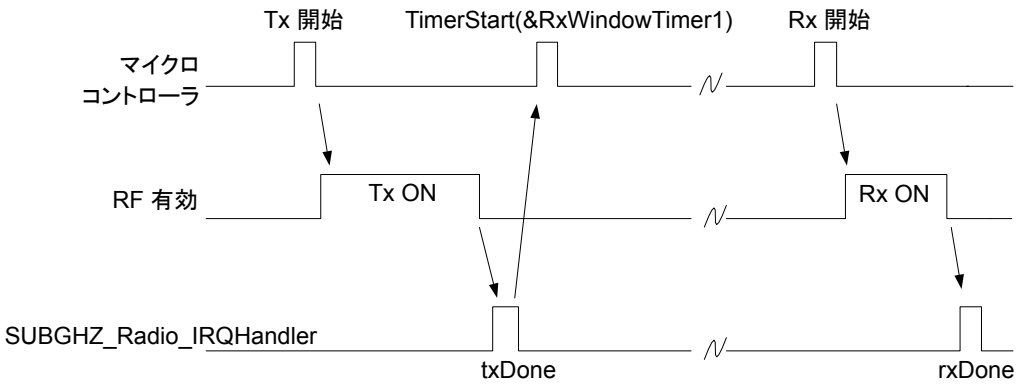


17.2 リアルタイム制約

LoRa RF 非同期プロトコルでは、推奨される厳密な Tx/Rx タイミングに従います（下の図を参照）。

STM32WL Nucleo ボード（NUCLEO-WL55JC）は、ユーザ透過な低ロック時間、高速自動較正に向けて最適化されています。BSP により、トランスミッタの起動時間とレシーバの起動時間の制約が統合されます（セクション 4 BSP STM32WLNucleo ボードを参照）。

図 19. Rx/Tx タイミング図



Rx ウィンドウ·チャネルが始動します。Rx1 ウィンドウは、txDone の立ち下がりエッジから 1 秒（±20µs）後に開きます。Rx2 ウィンドウは、txDone の立ち下がりエッジから 1 秒（±20µs）後に開きます。

JOIN_ACCEPT は、アップリンク変調の終了後に 5 秒（±20µs）と 6 秒の遅延を使用します。

現在スケジュールされている割込みレベル優先度に従う必要があります。つまり、受信した起動時間に遅れるのを避けるために、すべての新しいユーザ割込みの割込み優先度は無線 IRQ_interrupt より高くする必要があります。

17.3 消費電力

消費電力の測定は、STM32WL Nucleo ボード NUCLEO-WL55JC1 について行っています。

測定セットアップ：

- デバッグなし
- トレース·レベル：VLEVEL_OFF（トレースなし）
- SENSOR_ENABLED：無効

測定結果：

- STOP 2 モード = 2 µA（図 21 を参照）。
- Tx = 23 mA （TCXO 有効）（図 20 を参照）。
- Rx = 7 mA （TCXO 有効）（図 20 を参照）。

測定値:30 秒以上の瞬時消費量

図 20. NUCLEO-WL55JC1 の消費電流と時間

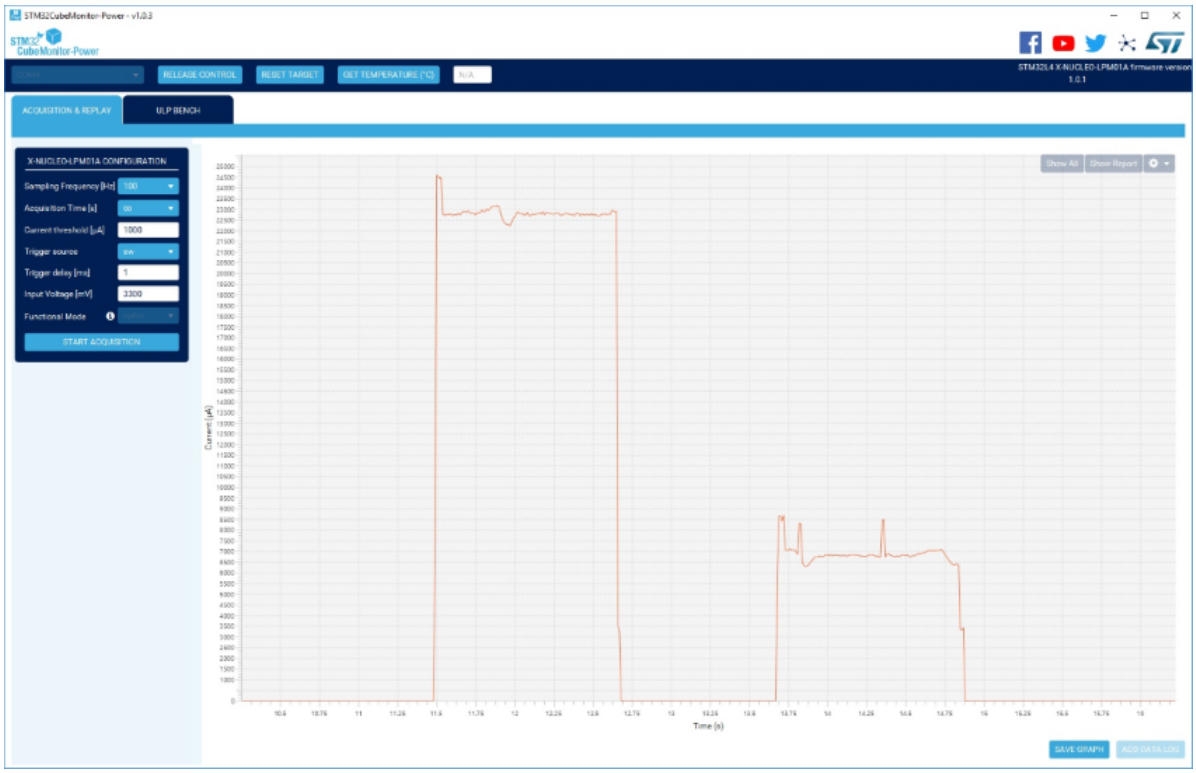
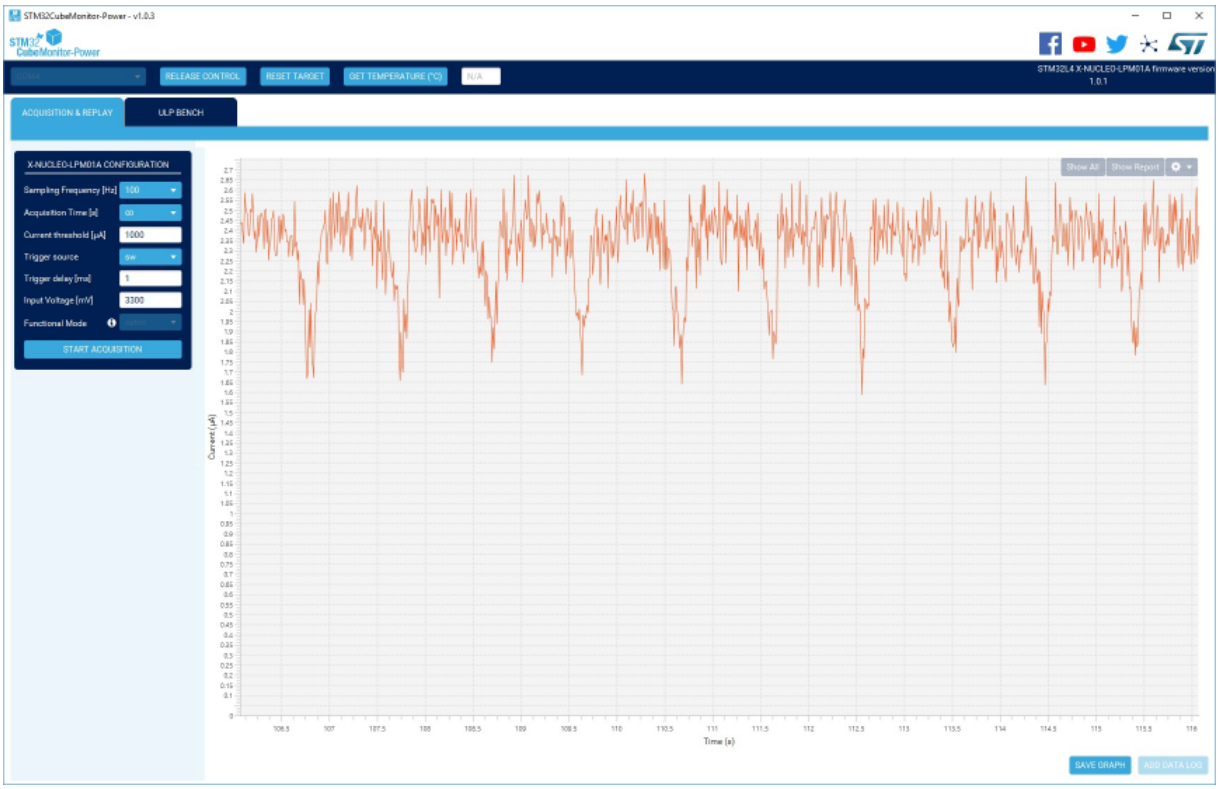


図 21. NUCLEO-WL55JC1 の STOP 2 モード時消費電流



改版履歴

表 43. 文書改版履歴

日付	版	変更内容
2019 年 12 月 10 日	1	初版発行
2020 年 4 月 27 日	2	文書の構造と内容の全体的な更新。
2020 年 11 月 17 日	3	更新: • 図 1.プロジェクト·ファイルの構造 • セクション 4「BSP STM32WL Nucleo-73 ボード」の注 • セクション 6「LoRaWAN ミドルウェアの説明」の概要 • セクション 6.6「ミドルウェア LmHandler アプリケーション関数」のタイトルと概要 • 表 22 および表 23 • セクション 8.1.1「アクティベーション方法とキー」 • 表 38.End_Node アプリケーション設定のスイッチ·オプション • 表 39.AT_Slave アプリケーション設定のスイッチ·オプション • 表 40.PingPong アプリケーション設定のスイッチ·オプション • セクション 13.1「メモリ·フットプリント」 追加: • 表 26LmHandler プロセス • セクション 11「デュアルコア管理」 • セクション 12「キー管理サービス(KMS)」 セクション 6.7 から「ボード固有 ID」と「ボード·ランダム·シード」の表を削除。
2021 年 2 月 18 日	4	更新: • セクション 8.1.2、セクション 8.1.4、セクション 8.1.5、セクション 8.1.6、セクション 8.1.7、セクション 8.1.8、およびセクション 8.1.9 で `lora_app.c` を `lora_app.h` に変更。 • 表 38.End_Node アプリケーション設定のスイッチ·オプション • 表 39.AT_Slave アプリケーション設定のスイッチ·オプション
2021 年 7 月 9 日	5	更新: • 「概要セクション」を「セクション 1 一般情報」に変更 • セクション 11「SubGhz_Phy_PingPong アプリケーション」の概要 • 図 7.SubGhz_Phy_PingPong アプリケーションのセットアップ • セクション 14「デュアルコア管理」の概要 • セクション 14.1.3「MBMUX メッセージ」 • セクション 14.2.5「RAM のマッピング」 • セクション 15.2「KMS のキーの定義」 • セクション 15.3「LoRaWAN のキー」 • 図 17.ROM メモリマッピング • セクション 15.5 「KMS データ·ストレージのための NVM のサイズ設定」 • セクション 15.6「アプリケーションを構築するための KMS 設定ファイル」 • セクション 17.1「メモリ·フットプリント」 • セクション 11「SubGhz_Phy_PingPong アプリケーション」の概要 • 図 7.SubGhz_Phy_PingPong アプリケーションのセットアップ • セクション 14「デュアルコア管理」の概要 • セクション 14.1.3「MBMUX メッセージ」 • セクション 14.2.5「RAM のマッピング」 • セクション 15.2「KMS のキーの定義」 • セクション 15.3「LoRaWAN のキー」 • 図 17.ROM メモリマッピング • セクション 15.5 「KMS データ·ストレージのための NVM のサイズ設定」 • セクション 15.6「アプリケーションを構築するための KMS 設定ファイル」 • セクション 17.1「メモリ·フットプリント」
2021 年 7 月 9 日(続き)	5	追加: • セクション 6.6「アプリケーションのコールバック」の 7 つの表 • セクション 12「SubGhz_Phy_Per アプリケーション」 • セクション 16「LoRaWAN アプリケーションをセキュアにする方法」

日付	版	変更内容
2022 年 2 月 21 日	6	更新: • 表 1. 頭字語と用語 • セクション 2 STM32CubeWL の概要 • セクション 6 LoRaWAN ミドルウェアの説明 追加: • セクション 5 LoRaWAN スタックの説明 • セクション 5.3.3 無線の駆動に必要な STM32 ペリフェラル • セクション 9.1 LoRaWAN のユーザ·コード·セクションの説明 • セクション 9.2.11 コンテキスト管理 • セクション 13 LoRaWAN コンテキスト管理の説明 • セクション 13.1 NVM コンテキスト管理データ API の定義 • セクション 11.2 デバイスの設定 • セクション 11.2.1 変調の定義 • セクション 11.2.2 ペイロード長 • セクション 11.2.3 リージョンと周波数 • セクション 11.2.4 帯域幅、拡散係数、およびデータ·レート • セクション 11.2.5 プリアンブル長 削除: • STM32CubeWL アーキテクチャ

目次

表一覧

図一覧