life.augmented

AN5480

アプリケーション・ノート

STM32CubeWL を使用した Sigfox™ 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た特定の Sigfox™ アプリケーションの構築方法を説明します。この文書は、非常に重要な情報がまとめられており、対応すべき要素が示されています。

Sigfox™ は、超低ビットレートでの長距離通信ができるように設計され、長寿命バッテリ駆動センサの使用を可能とする無線通信ネットワークです。Sigfox Stack™ ライブラリは、Sigfox™ ネットワークとの相互運用を確保するチャネルアクセスおよびセキュリティプロトコルを管理します。

NUCLEO_WL55JC に基づくアプリケーション、STM32WL Nucleo ボード（高周波帯域用の注文コード NUCLEO-WL55JC1）、および STM32CubeWL MCU パッケージのファームウェアは Sigfox Verified（Sigfox による検証済み）™です。

Sigfox™ アプリケーションの主な機能は次のとおりです。

- アプリケーションと統合可能
- Sigfox Verified（Sigfox による検証済み）™の RC1、RC2、RC3c、RC4、RC5、RC6、および RC7
- Sigfox™ Monarch（STMicroelectronics により、アルゴリズムの特許取得済み）
- 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
- 提供されるユーティリティサービス

STM32CubeWL MCU パッケージのファームウェアは、STM32Cube HAL ドライバをベースにしています。

1 一般情報

STM32CubeWL は、Arm® Cortex®-M プロセッサベースとした 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 および用語

項目（略称）	定義
BSP	ボード・サポート・パッケージ
CS	キャリアセンス
DC/DC	直流-直流コンバータ
FH	周波数ホッピング
IoT	Internet of things（モノのインターネット）
LBT	リッスン・ビフォア・トーク
PAC	ポーティング認証コード
POI	ポイント・オブ・インタレスト
PA	パワーアンプ
PRBS	疑似ランダムビットシーケンス
RC	地域設定
RSA	無線 Sigfox アナライザ
RSSI	受信信号強度インジケータ
Rx	受信
SDR	ソフトウェア定義無線
TCXO	温度補償付き水晶発振器
Tx	送信

2 Sigfox 規格

このセクションでは、特に Sigfox エンドデバイスに焦点を当てて、Sigfox の一般的な概要を説明します。

Sigfox は、長寿命のバッテリ駆動センサを実現する、低ビットレートで長距離通信が可能な無線通信規格です。STM32CubeWL MCU パッケージのファームウェアには Sigfox スタックライブラリが含まれています。

Sigfox では、ネットワークの使用を 1 日、1 デバイスあたり 144 メッセージに制限しています。各メッセージは 1 ビットから最大 12 バイトの長さです。

2.1 エンドデバイスのハードウェアアーキテクチャ

エンドデバイスは、NUCLEO-WL55JC ボードに実装された STM32WL55JC マイクロコントローラです。

このマイクロコントローラは、150 - 960 MHz ISM 帯域で動作する Sub-GHz 無線機能を内蔵し、さまざまなメモリサイズ、パッケージ、ペリフェラルを備えたマイクロコントローラを搭載した STM32WL シリーズに属します。

2.2 地域無線リソース

欧州、北米、およびアジアの市場は、周波数割当てと規制要件が異なっています。Sigfox には、下の表に示すさまざまな RC（地域設定）に分けられた要件があります。

表 2. 地域設定

RC	国
RC1	ヨーロッパ、オマーン、レバノン、南アフリカ、ケニア
RC2	アメリカ、カナダ、メキシコ
RC3c	日本
RC4	ブラジル、コロンビア、ペルー、ニュージーランド、オーストラリア、シンガポール
RC5	韓国
RC6	インド
RC7	ロシア

次の表に、地域設定の規制要件の概要を示します。

表 3. 地域設定の RF パラメータ

RF パラメータ	RC1	RC2	RC3c	RC4	RC5	RC6	RC7
周波数帯域、ダウンリンク（MHz）	869.525	905.2	922.2	922.3	922.3	866.3	869.1
周波数帯域、アップリンク（MHz）	868.130	902.2	923.2	920.8	923.3	865.2	868.8
アップリンク変調	DBPSK						
ダウンリンク変調	GFSK						
アップリンクデータレート	100	600	100	600	100	100	100
ダウンリンクデータレート	600						
最大出力電力（dBm）	14	22	13	22	13	13	14
ミディアムアクセス	デューティサイクル 1%	周波数ホッピング 最大オン時間 400 ms／20 s	キャリアセンス	周波数ホッピング 最大オン時間 400 ms／20 s	キャリアセンス	デューティサイクル 1%	
CS 中心周波数（MHz）	該当なし		923.2	該当なし	923.3	該当なし	
CS 帯域幅（kHz）			200	該当なし	200		
CS 閾値（dBm）			-80	該当なし	-65		

2.3 Rx/Tx 無線タイミング図

エンドデバイスは、ネットワークに非同期でデータを送信します。これは、送信データがデバイスレポートイベントごとにのみ送信されるという事実によるものです。下の図は、ダウンリンクありとなしのタイミングシーケンスを示します。

図 1. アップリンクのみのタイミング図

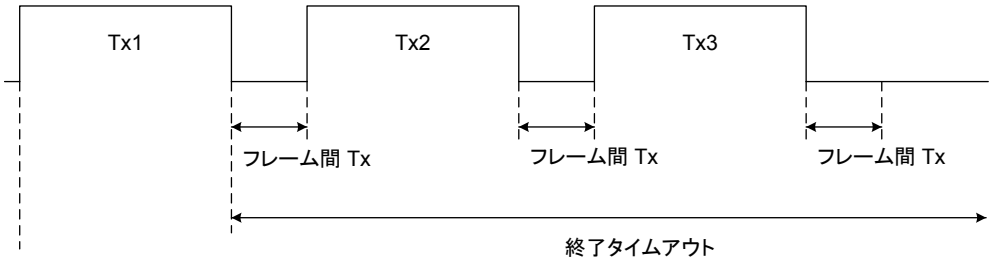
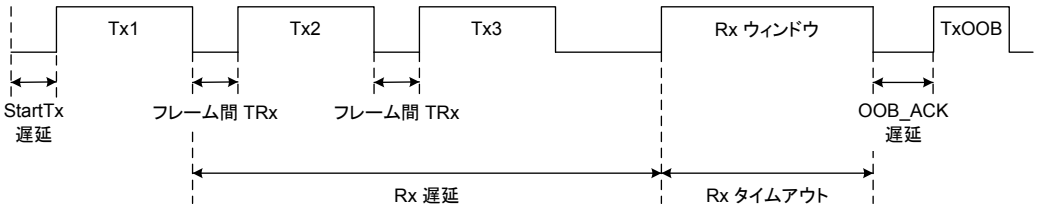


図 2. ダウンリンクありのアップリンクのタイミング図



3 つの送信データ Tx1、Tx2、および Tx3 には同じペイロード情報が含まれます。このように連続して送信することにより、ネットワークが正しく受信する可能性が最大になります。デバイスがネットワークへのリンク品質が良好であると認識すると、消費電力を節約するために Tx1 のみを送信することを決定することがあります（ダウンリンクフレームがリクエストされている場合）。選択する API の推奨方式については、セクション 6.1.2 送信フレーム／ビットを参照してください。

前の図に示したタイミングの詳細を、さまざまな地域設定に対して下の表に示します。

表 4. タイミング

RC	StartTx 遅延	インターフレーム Tx/TRx	Rx 遅延	Rx タイムアウト	OOB_ACK 遅延	終了タイムアウト
RC1	0 s	500 ms	20 s	25 s	1.4 s	該当なし
RC2						10 s
RC3c	最大 100 ms（LBT 開始）	500 ms + LBT	19 s	34 s		該当なし
RC4	10 s	500 ms	20 s	25 s		
RC5	最大 100 ms（LBT）	500 ms + LBT	19 s	34 s		
RC6	0 s	500 ms	20 s	25 s		
RC7						

Tx 周期は、送信されたバイト数と RC ゾーンに依存します。

- RC1 と RC3c では 1 ビットを送信するのに 10 ms かかります。
- RC2 と RC4 では 1 ビットを送信するのに 1.66 ms かかります。

メッセージは、最大 26 バイト長（同期ワード、ヘッダ、ペイロードデータを含む）です。RC1 の場合、Tx 周期は最大 26 x 8 x 10 ms = 2.08 s となります。

2.4 リッスン・ビフォア・トーク（LBT）

RC3c および RC5 では、送信前の LBT が必須です。

RC3c では、デバイスはチャネルが空いているかどうかをリッスンして確認する必要があります。200 kHz の帯域幅内のパワーが 5 ms にわたって -80 dBm（CS しきい値）未満にとどまる場合、チャネルは空きと見なされます。

チャネルが空くと、デバイスは送信を開始します。そうでない場合、送信は開始されません。

2.5 Monarch

Monarch は、ポイント・オブ・インタレスト（POI）に配置される Sigfox ビーコンです。POI が属する地域で許可されている周波数で Sigfox ビーコンの信号を放射します。ビーコンには、Monarch 対応デバイスが復調できる地域設定（RC）情報が含まれています。

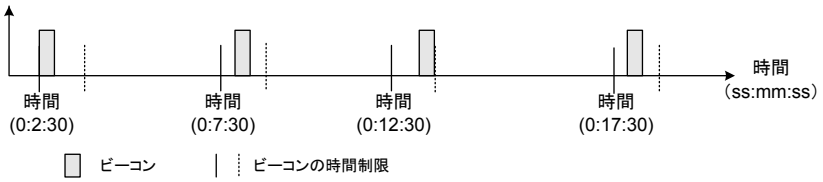
Monarch 対応デバイスは、この情報を受信すると、正しい RC に自動的に切り替えて、ネットワークに情報を送信することが可能になります。

Monarch 機能により、Sigfox IoT デバイスは世界中をシームレスにローミングできます。

2.5.1 Monarch 信号の説明

5 分に 10 秒のランダムなバックオフ周期を加えた周期で Monarch 信号が POI で送信されます。ビーコンの周波数は地域によって異なります。ビーコンは合計 400 ms 続きます。デバイスクロックがセットされている場合、Monarch 信号が存在するときにのみスキャンウィンドウを開いて、エンドデバイスの消費電流を下げることができます。

図 3. Monarch ビーコン



信号は OOK 変調されています。これは、信号はオンまたはオフのいずれかであることを意味します。変調周波数は 16384 Hz（RTC クロックの半分）に規定されています。信号は 1 サンプルの間オンになり、その後オフになります。11、13、または 16（16384 Hz）サンプリングの周期でオンになります。

有効な OOK 周波数 dF は、次のとおりです。

- dF1 = 16384 / 16 = 1024 Hz
- dF2 = 16384 / 13 = 1260.3 Hz
- dF3 = 16384 / 11 = 1489.4 Hz

400 ms の Monarch パターンには 2 つのサブパターンがあります。

- パターン 1（特定の dF で 362 ms）
- パターン 2（別の特定の dF で 38 ms）

表 5. Monarch 信号特性対 RC

RC	Monarch 周波数（Hz）	パターン 1 dF（Hz）	パターン 2 dF（Hz）
RC1	869 505 000	1024	1260.3
RC2	905 180 000		
RC3c	922 250 000		
RC4		1260.3	1489.4
RC5		1024	
RC6	866 250 000		
RC7	869 160 000	1260.3	

2.5.2 Monarch 信号復調

デバイスが Monarch 信号のスキャンを開始すると、デバイスは、表 5 に記載されているすべての Monarch 周波数を 5 分間スイープします。これをスイープ期間と呼びます。

注 時間がわかっている場合、スイープ時間は約 10 秒にいくらかのクロックドリフトを加えた時間に短縮されることがあります。

この間、デバイスはいずれかのパターン 1 との一致を試みます。一致が検出されると、デバイスはスイープ期間を終了して、400 ms のウィンドウ期間と呼ばれる 2 番目の期間に入ります。

デバイスは、パターン 1 との一致が発生した RF 周波数に設定されます。次に、デバイスは、パターン 2 との一致を試みて、Monarch ビーコンが見つかったことを確認します。

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 機能については詳細を省略します）。

SubGHz HAL は、Sub-GHz の無線ペリフェラルに直接実装されています（図 1 を参照）。

SubGHz 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（完全なプロセスではありません）に基づいています。したがって、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、および設定データ構造
- 初期化 API
- 設定およびコントロール API
- MSP とイベントコールバック
- SUBGHZ_SPI（固有のサービス）に基づいたバス I/O 操作

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

無線の初期化時に、次の HAL SubGHz API がコールされます。

- SUBGHZ_HandleTypeDef ハンドル構造を宣言します。
- `HAL_SUBGHZ_Init(&hUserSubghz)` API を呼び出して、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 ローレベルリソースを初期化します。
  - PWR 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 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順位を設定します。

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

- SUBGHZ_Radio_IRQHandler の `HAL_SUBGHZ_IRQHandler`。

3.2 Sub-GHz データ転送

Set コマンド動作は、API `HAL_SUBGHZ_ExecSetCmd();` のポーリングモードで実行します。

Get Status 動作は、API `HAL_SUBGHZ_ExecGetCmd();` のポーリングモードで実行します。

読出し／書込みレジスタへのアクセスは、次の API を使用して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 STM32WL Nucleo 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機能を提供します。

注 無線ミドルウェア（SubGHz_Phy）は、インタフェースファイル `radio_board_if.c/h` を介して無線 BSP をインタフェースします。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。

- 1 番目のオプション
  - `BSP/STM32WLxx_Nucleo/` ディレクトリをコピーします。
  - 次の設定を使用して、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 ファイルで置き換えます。
- 2 番目のオプション
  - `Core/Inc/platform.h` の `USE_BSP_DRIVER` を無効にし、BSP 機能を `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：高周波数帯域、865 MHz から 930 MHz の周波数に調整
- NUCLEO-WL55JC2：低周波数帯域、470 MHz から 520 MHz の周波数に調整

ユーザが 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の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。

ファームウェアは、実行されるボードの帯域はチェックしません。

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 6. BSP 無線スイッチ

機能	説明
`int32_t BSP_RADIO_Init(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)`	ボード設定を返します。高電力、低電力、またはその両方。

スイッチの設定に対する RF 状態を下の表に示します。

表 7. RF 状態とスイッチ設定

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 TCXO

ユーザアプリケーション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 ボードには、温度補償された水晶発振子（TCXO）が使われており、周波数精度が向上しています。

表 8. BSP 無線 TCXO

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

TCXO モードは、`Core/Inc/platform.h` で `USE_BSP_DRIVER` を選択することで、STM32WL Nucleo BSP により規定されます。

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 モードは `radio_board_if.h` で更新することができます。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
```

4.4 電力レギュレーション

ユーザアプリケーションに応じ、電力レギュレーションには LDO または SMPS（DCDC 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 9. BSP 無線 SMPS

機能	説明
`uint32_t BSP_RADIO_IsDCDC (void)`	`IS_DCDC_SUPPORTED` 値を戻します。

DCDC モードは、`Core/Inc/platform.h` の `USE_BSP_DRIVER` を選択することによって、STM32WL Nucleo BSP により規定されます。

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、DCDC モードは `radio_board_if.h` で更新することができます。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 は、`IS_DCDC_SUPPORTED = 0` 設定によって無効にできます。

4.5 最大 Tx RF 出力パワー

パワーアンプ（PA）の最適設定は、要求出力パワーが公称値（HP RFO では 22 dBm、LP RFO では 15 dBm）より低い場合に、その効率を最大化するために使用されます。この設定は RF マッチングネットワークに依存し、ボード・サポート・パッケージ（BSP）で設定が可能なので、ソフトウェアによりパケット送信前に最適な PA 設定を選択することができます。

表 10. BSP ボードの最大電力設定

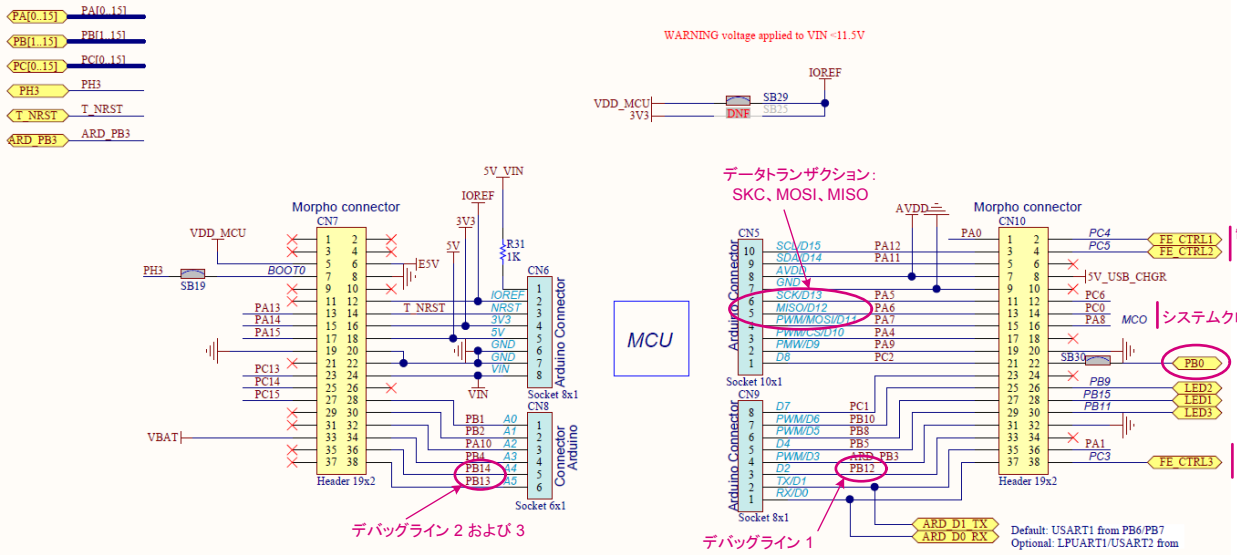
機能	説明
`int32_t BSP_RADIO_GetRFOMaxPowerConfig(BSP_RADIO_RFOMaxPowerConfig_TypeDef Config);`	低-パワーまたは高-パワー RFO の最大出力電力を返します。

4.6 STM32WL Nucleo ボード回路図

下の図に、STM32WL Nucleo ボード（MB1389 リファレンスボード）の回路図の詳細を、いくつかの有用な信号がよく分かるように示します。

- 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 4. NUCLEO-WL55JC 回路図



5 Sigfox スタックの説明

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 次のような STM32WL リソースが含まれています。

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

STM32 マイクロコントローラ用の Sigfox ミドルウェアは、いくつかのモジュールに分割されています。

- Sigfox Core ライブラリ・レイヤ・モジュール
- Sigfox 暗号化モジュール
- Sigfox Monarch（ST アルゴリズム特許）

Sigfox Core ライブラリは Sigfox ミディアム・アクセス・コントローラを実装し、アップリンクペイロードの暗号化およびダウンリンクペイロードの検証を行う Cmac ライブラリとインタフェースします。Cmac ライブラリは、暗号化関数を保持する Credentials ライブラリとインタフェースします。このミディアム・アクセス・コントローラは、ST Monarch ライブラリともインタフェースします。

Sigfox コアライブラリは、ユーザディレクトリ内の `rf_api.c` および `mcu_api.c` などのポーティングファイルともインタフェースします。これらのファイルを変更しないでください。

Sigfox コア、Sigfox テスト、暗号化、および Monarch のライブラリモジュールは、コンパイル済みオブジェクトで提供されます。

ライブラリは wchar32 および 'short enums' を指定してコンパイルされています。これらの設定は、IAR Embedded Workbench® および STM32CubeIDE でデフォルトで使用されます。

µVision Keil® の場合、特に注意が必要です。Tickbox Short enums/wchar のチェックを外し、Misc Controls フィールドに `fshort -enums` を追加しなくてはなりません。

注 デュアルコアアプリケーションの場合、同じ enum フォーマットを保証するために、これらの設定を両方のコアに適用する必要があります。

5.1 Sigfox 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、Sigfox Test Lab によって検証され、Sigfox Verified 認証に合格しています。

しかしながら、STM32WL シリーズマイクロコントローラをベースとしたユーザの最終製品は、ユーザの最終製品の商品化前に再び Sigfox Verified に合格して Sigfox Ready™ 認証を取得しなければなりません。

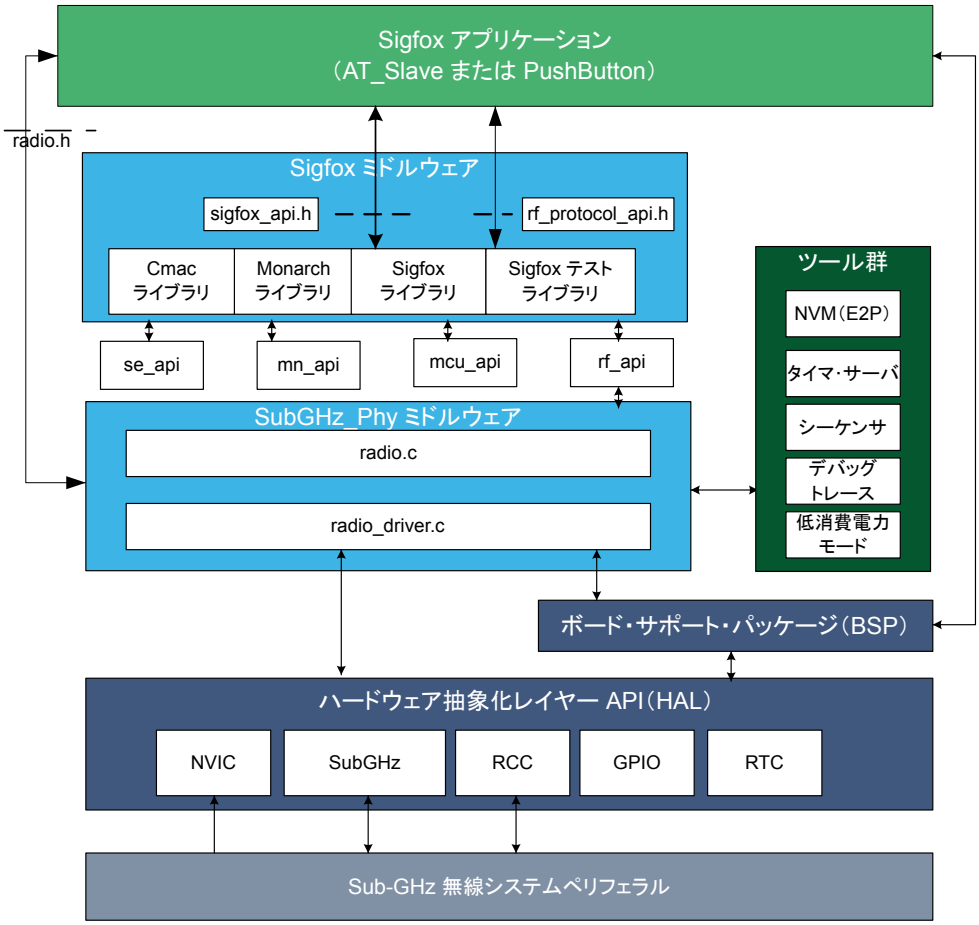
Sigfox 認証では、STM32CubeWL に実装された rollover_counter 4096 の値を提供する必要があります。

5.2 アーキテクチャ

5.2.1 スタティックビュー

下の図に、Sigfox アプリケーション用のファームウェアの主な設計の詳細を示します。

図 5. スタティック Sigfox 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ハードウェアを駆動します。

RTC は、低消費電力 STOP 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時間でシステムをウェイクアップするために使用します。

Sigfox コアライブラリには、ミディアム·アクセス·コントローラ（MAC）と一部のセキュリティ機能が内蔵されています（詳細は セクション 6.1 Sigfox Core ライブラリを参照）。

アプリケーションは、スケジューラを含む無限ループをもとに構築されています。スケジューラは、タスクとイベントを処理します。やるべきことが残っていない場合、スケジューラはアイドル状態に移行し、低消費電力マネージャをコールします。

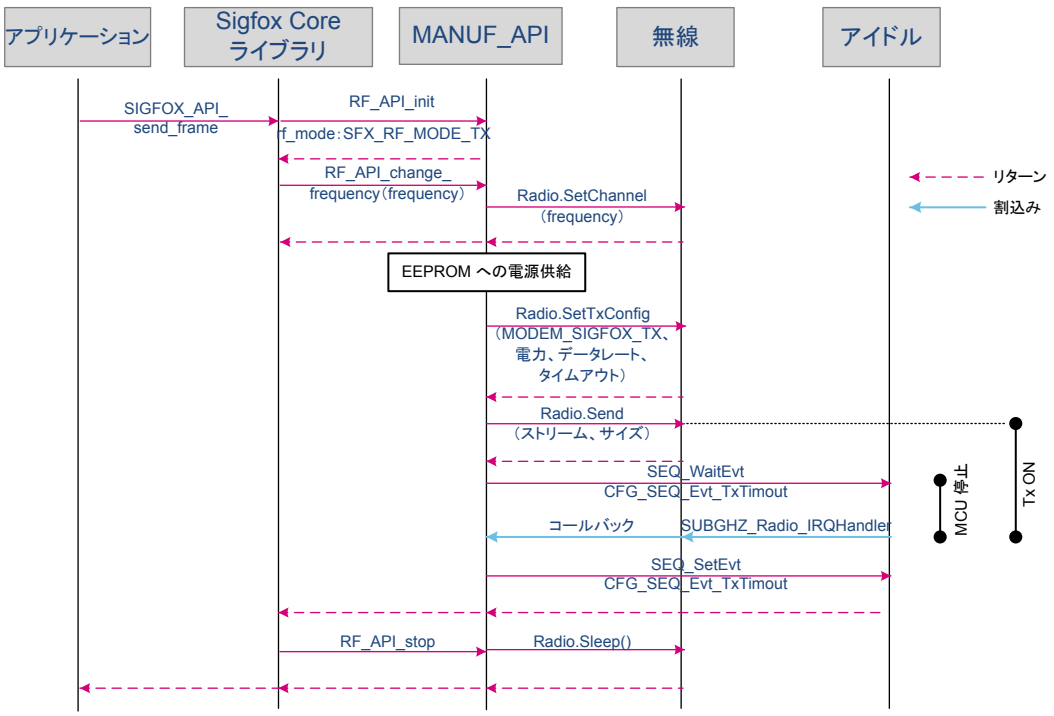
標準的なアプリケーション例：

- 外部ホストとインタフェースする AT レイヤ（ セクション 11.2 AT モデムアプリケーションを参照）
- アプリケーションは、アクション時にセンサデータの読出しと送信を行います（ セクション 11.3 PushButton アプリケーションを参照）。

5.2.2 ダイナミックビュー

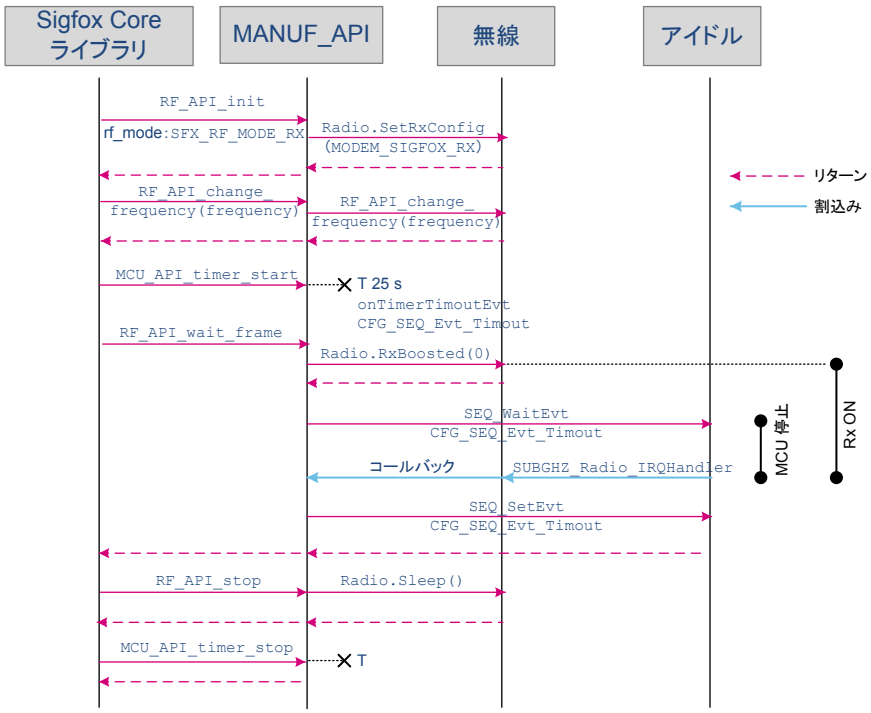
下の図のメッセージシーケンス図（MSC）は、Tx モード（1 回の送信）での API 間のダイナミックコールを示しています。

図 6. 送信 MSC



ダウンリンクウィンドウが要求された場合、Rxdelay の経過後に Rx シーケンスを開始します（図 2 を参照）。
Rxdelay が経過すると、下の図に詳細を示したシーケンスが起動します。

図 7. 受信 MSC



5.3 無線の駆動に必要な STM32 ペリフェラル

Sub-GHz 無線

Sub-GHz 無線ペリフェラルには、`stm32wlxx_hal_subghz` HAL からアクセスします。

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、すべての電力モードで 32 kHz 外部オシレータで動作する 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 はハードウェア初期化中（マイクロコントローラの初回起動時）に 1 回プログラムされます。RTC 出力は約 48 日周期に対応する 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 Sigfox ミドルウェアのプログラミング・ガイドライン

6.1 Sigfox Core ライブラリ

Sigfox Core ライブラリを使用する組込みアプリケーションは、SIGFOX_APIs をコールして通信を管理します。

表 11. アプリケーションレベルの Sigfox API

機能	説明
`sfx_error_t SIGFOX_API_get_device_id (sfx_u8 *dev_id);`	デバイスの ID をパラメータで指定されたポインタにコピーします。ID は 4 バイト長で、16 進フォーマットです。
`sfx_error_t SIGFOX_API_get_initial_pac (sfx_u8 *initial_pac);`	デバイスに格納されている PAC の値を取得します。この値は、デバイスがバックエンドで初めて登録されるときに使用されます。PAC は 8 バイト長です。
`sfx_error_t SIGFOX_API_open (sfx_rc_t *rc);`	ライブラリを初期化し、入力パラメータを 1 回保存します(`SIGFOX_API_close()` がコールされるまで変更できません)。 – `rc` は無線設定ゾーン上のポインタです。既存の定義済み RC を使用する必要があります。
`sfx_error_t SIGFOX_API_close(void);`	ライブラリを閉じて RF を停止します。
`sfx_error_t SIGFOX_API_send_frame (sfx_u8 *customer_data, sfx_u8 customer_data_length, sfx_u8 *customer_response, sfx_u8 tx_repeat, sfx_bool initiate_downlink_flag);`	カスタマーペイロードのある標準 Sigfox フレームを送信します。 • `customer_data` は 12 バイトを超えることはできません。 • `customer_data_length`:データ長(byte) • `customer_response`:受信した応答 • `tx_repeat`: – 0 の場合、1 つの Tx を送信します。 – 1 の場合、3 つの Tx を送信します。 • `init_downlink_flag`:セットされている場合、送信フレームには、受信ダウンリンクフレームと、アウトオブバンド Tx フレーム(電圧、温度、および RSSI)が続きます。
`sfx_error_t SIGFOX_API_send_bit (sfx_bool bit_value, sfx_u8 *customer_response, sfx_u8 tx_repeat, sfx_bool initiate_downlink_flag);`	カスタマペイロードが null の標準 Sigfox™ フレーム(Sigfox ライブラリが生成できる最短フレーム)を送信します。 • `bit_value`:送信されたビット • `customer_response`:受信した応答 • `tx_repeat`: – 0 の場合、1 つの Tx を送信します。 – 1 の場合、3 つの Tx を送信します。 • `init_downlink_flag`:セットされている場合、送信フレームには、受信ダウンリンクフレームと、アウトオブバンド Tx フレーム(電圧、温度、および RSSI)が続きます。
`sfx_error_t SIGFOX_API_set_std_config (sfx_u32 config_words[3], sfx_bool timer_enable);`	標準用の特定の変数を設定します。パラメータは、FH モードと LBT モードで異なる意味を持ちます。 注:この機能は DC では影響しません(詳細は セクション 11.2.21 を参照)。

セカンダリ API については、`sigfox_api.h` で説明されています。ライブラリは次の場所にあります:
`Middlewares\Third_Party\Sigfox\SigfoxLib` ディレクトリ。

6.1.1 Sigfox ライブラリを開く

他の操作を実行する前に、ST_SIGFOX_API_open をコールして、Sigfox ライブラリを初期化する必要があります。

この API は、無線設定ゾーンを表す RC 引数番号を必要とします（セクション 2.2 を参照）。

無線コントロールゾーン 2 および 4 については、FCC（連邦通信委員会）が周波数ホッピングを要求しているため、送信周波数は固定されていません（マクロチャネルのマッピング方法の詳細については セクション 6.1.3 を参照してください）。

6.1.2 送信フレーム／ビット

ST_SIGFOX_API_send_frame が Sigfox のメインライブラリ関数です。このブロック機能は、エンドノードとベースステーション間のメッセージ交換を処理します。

この関数の重要なパラメータは、以下のようにいくつかの送信動作を選択する initiate_downlink_flag です。

- initiate_downlink_flag = 0:ライブラリはアップリンクフレームのみを要求します。送信フレームは tx_repeat = 0 の時には 1 回、tx_repeat = 1 の時には 3 回送信され、ポーズ時間は 500 ms です（図 1 参照）。送信ペイロードは最大で 12 バイトの長さです。
- initiate_downlink_flag = 1:送信されるフレームは 500 ms のポーズで 3 回送信されます。最初の繰り返しの 20 秒後に 25 秒の Rx ウィンドウが開きます（図 2 参照）。受信に成功した場合、受信した 8 バイトのダウンリンクフレームは、customer_response バッファにより表示されるバッファ位置に格納されます。

6.1.3 標準設定

FCC は、トランスミッタが特定のマクロチャネルを選択して、標準で承認された周波数ホッピングパターンを実装することを許可しています。チャネルマップは、SIGFOX_API_set_std_config の第 1 引数で指定します。これは 3 つの 32 ビット設定ワードの配列で構成されます。

マクロチャネルは、その中心周波数を中心とし、25 kHz 間隔の 6 つのマイクロチャネルで構成されています。たとえば、902.2 MHz マクロチャネルの場合、6 つのマイクロチャネルは 902.1375 MHz、902.1625 MHz、902.1875 MHz、902.2125 MHz、902.2375 MHz、902.2625 MHz です。

標準的な Sigfox フレームは、600 bit/s で 200 ms から 350 ms 継続し、FCC は最大 dwell 時間 400 ms を義務付けています。トランスミッタは 20 秒間、与えられたチャネルに戻ることができません。したがって、連続送信には少なくとも 20 / 0.4 = 50 チャネルを使用する必要があります。

実際には、デバイスは 1 日に数フレームしか送信しません（最大 144 メッセージ）。1 つのマクロチャネルのみを有効にし、3 つの繰り返しフレームからなる 2 つのグループ間に 10 秒の遅延を挿入すると（マイクロチャネルごとに 1 フレームは 6 つのマイクロチャネルを意味します）、規制の制限を満足できます。

config_words[0,1,2] 配列の各ビットは、下の表に示すマッピングに従って、マクロチャネルを表します。

表 12. マクロチャネルのマッピング

ビット	config_words[0] 周波数マッピング（MHz）	config_words[1] 周波数マッピング（MHz）	config_words[2] 周波数マッピング（MHz）
0	902.2	911.8	921.4
1	902.5	912.1	921.7
2	902.8	912.4	922
3	903.1	912.7	922.3
4	903.4	913	922.6
5	903.7	913.3	922.9
6	904	913.6	923.2
7	904.3	913.9	923.5
8	904.6	914.2	923.8
9	904.9	914.5	924.1
10	905.2	914.8	924.4
11	905.5	915.1	924.7
12	905.8	915.4	925
13	906.1	915.7	925.3

ビット	config_words[0] 周波数マッピング（MHz）	config_words[1] 周波数マッピング（MHz）	config_words[2] 周波数マッピング（MHz）
14	906.4	916	925.6
15	906.7	916.3	925.9
16	907	916.6	926.2
17	907.3	916.9	926.5
18	907.6	917.2	926.8
19	907.9	917.5	927.1
20	908.2	917.8	927.4
21	908.5	918.1	927.7
22	908.8	918.4	928
23	909.1	918.7	928.3
24	909.4	919	928.6
25	909.7	919.3	928.9
26	910	919.6	929.2
27	910.3	919.9	929.5
28	910.6	920.2	929.8
29	910.9	920.5	930.1
30	911.2	920.8	930.4
31	911.5	921.1	930.7

マクロチャネルは、対応する config_words[x] ビットが 1 にセットされている場合のみ有効です。たとえば、config_words[0] のビット 0 はチャネル 1 に対応し、config_words[1] のビット 30 はチャネル 63 に対応します。FCC 仕様に準拠するには、9 つ以上のマクロチャネルを有効にする必要があります。

次の長いメッセージ設定の例では、チャネル 1 から 9 が 902.2 MHz から 904.6 MHz の周波数範囲で有効になります。

- config_words[0] = [0x0000 01FF]
- config_words[1] = [0x0000 0000]
- config_words[2] = [0x0000 0000]

デフォルトでは、Sigfox のアプリケーションは 1 つのマクロチャネルに `timer_enable = 1` を設定します。RC2 のマクロチャネル 1 の動作周波数は 902.2 MHz で、RC4 のマクロチャネル 63 の動作周波数は 920.8 MHz です。これは Sigfox で動作可能な短いメッセージ設定です（`sigfox_api.h` で定義されている `RCx_SM_CONFIG` 値を参照）。

遅延（`timer_enable`）は、1 つのマイクロチャネルが 20 秒未満の間隔で再利用されるのを避けるために実装されます。1 つのマクロチャネルのみ（6 つのマイクロチャネル）を使用して 3 回の繰り返しを実行する場合、この遅延は 10 秒に相当します。2 つのマクロチャネル（12 マイクロチャネル）を使用する場合、遅延は自動的に 5 秒になります。

認証テストの目的で、`timer_enable` に 0 をセットすることができますが、そうでない場合は 1 にセットする必要があります。ただし、ATS400 コマンド（セクション 11.2.21）を使用してデフォルト設定を修正し、認証プロセスを迅速化することができます。

6.2 Sigfox アドオン RF プロトコルライブラリ

このライブラリは、Sigfox Verified 認証についてデバイスをテストするために使用されます。最終的に、このライブラリは認証後にビルドから削除できます。

表 13. Sigfox アドオン Verified ライブラリ

機能	説明

`sfx_error_t`
`ADDON_SIGFOX_RF_PROTOCOL_API_test_mode`
`(sfx _rc_enum_t rc_enum, sfx_test_mode_t test_mode);`
	Sigfox Verified 認証に必要なテストモードを実行します。
	• `rc_enum`:テストモードが実行される rc
	• `test_mode`:実行するテストモード

`sfx_error_t`
`ADDON_SIGFOX_RF_PROTOCOL_API_monarch_test_ mode`
`(sfx_rc_enum_t rc_enum, sfx_test_mode_t test_mode,`
`sfx_u8 rc_capabilities);`
	この関数は、Sigfox RF およびプロトコルテストに必要な Monarch テストモードを実行します。

このライブラリは `Middlewares\Third_Party\Sigfox\SigfoxLibTest` にあります。

6.3 Cmac ライブラリ

Cmac ライブラリには、キー、PAC、および ID が格納されます。

表 14. Cmac API

機能	説明
`sfx_u8 SE_API_get_device_id` `(sfx_u8 dev_id[ID_LENGTH]);`	この機能では、`dev_id` にデバイス ID をコピーします。
`sfx_u8 SE_API_get_initial_pac (sfx_u8 *initial_pac);`	初期 PAC を取得します。

`sfx_u8 SE_API_secure_uplink_message`
`(sfx_u8 *customer_data,`
`sfx_u8 customer_data_length,`
`sfx_bool initiate_downlink_frame,`
`sfx_se_frame_type_t frame_type,`
`sfx_bool *send_rcsync,`
`sfx_u8 *frame_ptr, sfx_u8 *frame_length);`
	アップリンクフレームビットストリームを生成します。

`sfx_u8 SE_API_verify_downlink_message` `(sfx_u8 *frame_ptr, sfx_bool *valid);`	受信したメッセージを認証し、そのペイロードを復号化します。

Cmac ライブラリは、このディレクトリにあります:`\Middlewares\Third_Party\Sigfox\Crypto`。

注
- このライブラリは、`se_nvm` 関数をインタフェースし、不揮発性メモリから `SFX_SE_NVMEM_BLOCK_SIZE` バイトを保存／取り出します。
- `se_api.h` は Sigfox セキュアエレメントへのインタフェースで、物理的にセキュアなエレメント、または Cmac ライブラリおよび Credentials ライブラリを備えたファームウェアによってエミュレートされたものです。

6.4 Credentials ライブラリ

Credentials ライブラリは、キー、PAC、および ID にアクセスできます。Sigfox キーを使用してデータを暗号化することもできます。

表 15. Credentials API

機能	説明
`void CREDENTIALS_get_dev_id(uint8_t* dev_id);`	デバイス ID を取得します。
`void CREDENTIALS_get_initial_pac (uint8_t* pac);`	デバイスの初期 PAC を取得します。
`sfx_bool CREDENTIALS_get_payload_encryption_fl ag(void);`	暗号化フラグを取得します。デフォルトでは false に設定されます(セクション 11.2.10 ATS411 - ペイロードの暗号化 を参照してください)。
`sfx_error_t CREDENTIALS_aes_128_cbc_encrypt (uint8 _t* encrypted_data, uint8_t* data_to_encrypt, uint8_t block_len);`	秘密鍵でデータを暗号化します。秘密鍵は、`CMAC_KEY_PRIVATE` または `CMAC_KEY_PUBLIC` に設定することが可能です(セクション 11.2.9 ATS410 - 暗号化キー を参照)。
`sfx_error_t CREDENTIALS_wrap_session_key (uint8_t *data, uint8_t blocks)`	Sigfox 秘密鍵をもとにセッションキーを生成します。
`sfx_error_t CREDENTIALS_aes_128_cbc_encrypt_with_session_key (uint8_t *encrypted_data, uint8_t *data_to_encrypt, uint8_t blocks)`	セッションキーでデータを暗号化します。

6.5 Monarch ライブラリ

Monarch API は、`sigfox_monarch_api.h`.で定義されています。

表 16. Monarch API

機能	説明
`sfx_error_t SIGFOX_MONARCH_API_execute_rc_scan (sfx_u8 rc_capabilities_bit_mask, sfx_u16 timer, sfx_timer_unit_enum_t unit, sfx_u8 (* app_callback_handler) (sfx_u8 rc_bit_mask, sfx_s16 rssi));`	Monarch スキャンを開始します。 • `sfx_u8 rc_capabilities_bit_mask` • `sfx_u16` タイマ:スキャン時間の値 • `sfx_timer_unit_enum_t unit`:タイマーの単位 • `app_callback_handler`:スキャンが完了したときに Sigfox ライブラリによってコールされる関数
`sfx_error_t SIGFOX_MONARCH_API_stop_rc_scan(void);`	進行中の Monarch スキャンを停止します。

7 SubGHz_Phy レイヤミドルウェアの説明

無線抽象化レイヤは、次に示す 2 つのレイヤで構成されています。

- 上位レイヤ(radio.c)
  スタックミドルウェアへの高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それらをコールします。
- 低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。このレイヤには、レジスタ名と構造、およびシーケンスの詳細情報があります。しかし、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 レイヤミドルウェアは、BSP によって提供されるハードウェアインタフェースのすぐ上でインタフェースを行う無線抽象化レイヤを含みます(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 機能を呼び出す一連の無線汎用コールバックのすべてが含まれています。この一連の API は、すべての無線で汎用的であり、同じものとなるようになっています。
- radio_driver.c:低レベル無線ドライバ

radio_conf.h は、RF_WAKEUP_TIME、DCDC 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の構造

無線の一般的構造（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 17. Radio_s 構造のコールバック

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	指定されたモデムの無線を設定します。
RadioSetChannel	チャネル周波数を設定します。
RadioIsChannelFree	指定された時間にチャネルが空いているかどうかを確認します。
RadioRandom	RSSI 読み値に基づいて、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。
RadioCheckRfFrequency	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、パケット送信時間（ms）を計算します。
RadioSend	送信するパケットの準備をし、無線での伝送を開始します。
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	無線を指定された時間、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されたアドレスを読み出します。
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	無線が SLEEP モードを終了するため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セットします。
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 モードにセットします。
RadioTxCw	トランスミッタを連続無変調キャリアモードにセットします。

7.2 無線 IRQ 割込み

考えられる Sub-GHz 無線割込み要因の詳細を下の表に示します。

表 18. 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	パケット送信終了	LoRa と GFSK	Tx
1	rxDone	パケット受信終了		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、または長さのエラー	GFSK	
	CrcErr	CRC エラー	LoRa	
7	CadDone	チャネルアクティビティ検出終了		CAD
8	CadDetected	チャネルアクティビティ検出		
9	タイムアウト	Rx または Tx タイムアウト	LoRa と GFSK	Rx および 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 EEPROM ドライバ

EEPROM インタフェース(sgfx_eeprom_if.c)は ee.c の上位として設計されており、EEPROM ドライバを抽象化します。EEPROM は、utilities_conf.h で定義された EE_BASE_ADRESS に物理的に配置されます。

表 19. EEPROM API

機能	説明
`void E2P_Init ( void );`	EEPROM に何もデータが無くなると DEFAULT_FACTORY_SETTINGS が書き込まれます。
`void E2P_RestoreFs ( void );`	DEFAULT_FACTORY_SETTINGS が復元されます。
`Void E2P_Write_XXX`	EEPROM にデータを書き込みます。例: `void E2P_Write_VerboseLevel(uint8_t verboselevel);`
`E2P_Read_XXX`	EEPROM から XXX を読み出します。例: `sfx_rc_enum_t E2P_Read_Rc(void);`

9 ユーティリティの説明

ユーティリティは、\Utilities ディレクトリにあります。

メイン API について次に説明します。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。

9.1 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ると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。シーケンサは、競合状態を防止するメカニズムを実装します。

さらに、シーケンサは、あらゆる機能がイベントを待機できるイベント機能を提供し(特定のイベントが割り込みによってセットされます)、「完了まで実行」コマンドを実装するあらゆるアプリケーションで電力と MIPS を容易に節約できます。

プロジェクトサブフォルダーにある 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。あらゆるタスクは完了するまで実行され、タスクが UTIL_SEQ_WaitEvt をコールして自身を一時保留にしない限り、RTOS が RTOS ティックでできるように、別のタスクに切り替えることはできません。また、シングルメモリスタックが 1 つ使用されます。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
- タスクの優先順位設定
- 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、アプリケーションは以下の手順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が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 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。

シーケンサ ユーティリティは Utilities\sequencer\stm32_seq.c にあります。

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);	特定のイベントがセット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セットします。

次の図で、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ます。

表 21. 標準の while ループとシーケンサによる実装

標準的な方法	シーケンサによる方法

`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*/`
	`}`

9.2 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要求できます。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、タイマを要求できます。

タイマ·サーバは、`Utilities\timer\stm32_timer.c` にあります。

表 22. タイマ·サーバの 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を作成し、タイマ時間経過時に呼び出す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)で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。

9.3 低消費電力関数

低消費電力ユーティリティは、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、DMA クロックが STOP モードでオフになるので、システムは SLEEP モードよりも低い低消費電力モードに移行してはなりません。

コアマイクロコントローラの低消費電力モードを管理するために下の表に示す API を使用します。低消費電力 ユーティリティは `Utilities\lpm\tiny_lpm\stm32_lpm.c` にあります。

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` がリクエストされたプロセスモードを定義します。`UTIL_LPM_ENABLE` または `UTIL_LPM_DISABLE`。(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` がリクエストされたプロセスモードを定義します。`UTIL_LPM_ENABLE` または `UTIL_LPM_DISABLE`。
`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。

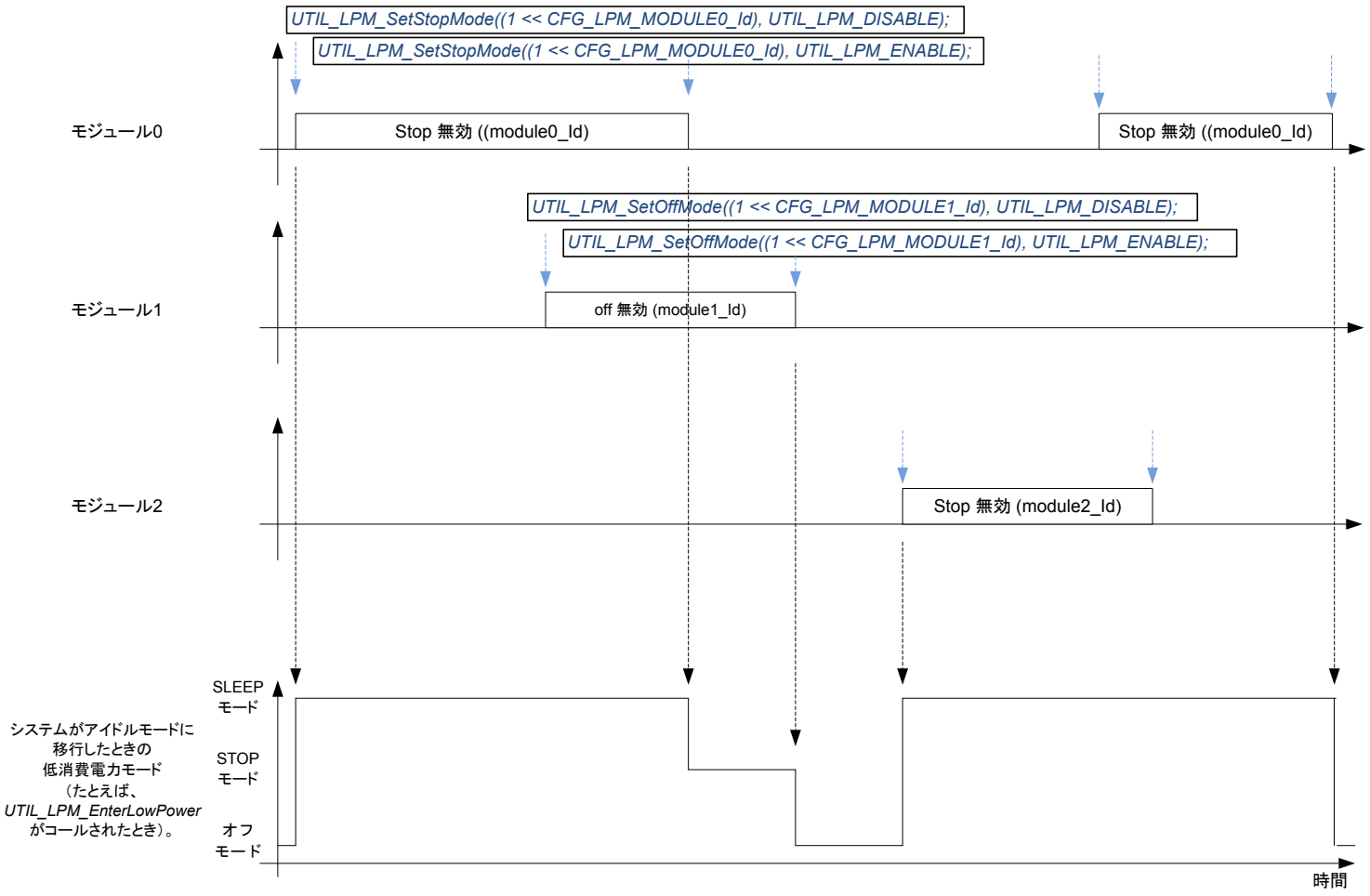
1. シフト値が `utilities_def.h` で定義されているビットマップ。

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 より、`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 モードが無効化されていない場合、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 モードが無効化され、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 モードが無効化されない場合、どのファームウェアモジュールによっても OFF モードが無効化されず、低消費電力に移行します。OFF モードが選択されます。

図 8 は、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（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）に依存して設定されている場合の動作を示しています。

図 8. 低消費電力モードのダイナミックビューの例



低消費電力モードを開始／終了するためにシステムに必要なことを定義するために、低レベル API を実装する必要があります。これらの機能は、プロジェクトのサブフォルダの stm32_lpm_if.c に実装されています。

表 24. 低レベル 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 モードの前にコールされる 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 モードの終了時にコールされる API

SLEEP モードではコアクロックを停止します。各ペリフェラルクロックは、ゲートの有無を設定できます。すべてのペリフェラルで電源が維持されます。

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がオフになり（BOR、バックアップレジスタ、GPIO プル、および RTC を除く）、保持はなく（追加の保持付きの 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。

9.4 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。システム時間は、UNIX® エポック·タイムを記録できます。

コアマイクロコントローラのシステム時間を管理するために下の表に示す API を使用します。systime ユーティリティは Utilities\misc\stm32_systime.c にあります.

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（秒および秒以下）に基づいて、マイクロコントローラ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·タイムをローカル時間に変換します。(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·タイムを time.h インタフェースにより指定された tm 構造に変換するために、SysTimeMkTime と SysTimeLocalTime も提供されています。

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、グリニッジ時と 2 時間の時差があります。時間が設定されている場合、次のコードで端末に現地時間を表示できます。

```
{
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);
}
```

9.5 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 ポートに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。

トレース ユーティリティは `Utilities\trace\adv_trace\stm32_adv_trace.c` にあります。

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呼び出す必要があります。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、それを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されたデータを FIFO (Z-Cpy)に直接書き込みます。

トレース機能のステータス値は、`UTIL_ADV_TRACE_Status_t` 構造体で次のように定義されます。

```
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 (..) 機能は以下の場合に使用できます。

- ポーリングモード:リアルタイム制約が適用されない場合。たとえば、アプリケーションを初期化する際

```
#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い場合、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 は、トレースにタイムスタンプの追加を可能にします(TS_ON または TS_OFF)。

アプリケーションの詳細レベルは、次のコマンドで Core\Inc\sys_conf.h にセットされます。

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

ここで VLEVEL は VLEVEL_OFF、VLEVEL_L、VLEVEL_M、または VLEVEL_H のいずれかが設定可能です。

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

バッファ長が Core\Inc\utilities_conf.h で飽和した場合に備えて、次のユーティリティでバッファ長を増加することができます。

```
#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 モードまたは下位モードに入ることを禁止するために実装されるフックを提供しています。

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

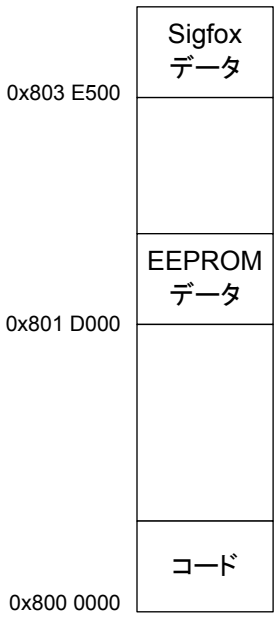
- ```
  void UTIL_ADV_TRACE_PostSendHook (void)
  { UTIL_LPM_SetStopMode((1 << CFG_LPM_UART_TX_Id) , UTIL_LPM_ENABLE );}
  ```

10 メモリ・セクション

コードは 0x0800 0000 に配置されます。`sigfox_data`（認証情報）は 0x0803 E500 に配置されます（スキャッタファイルで変更可能）。

また、EEPROM をアドレス 0x0801 D000（`EE_BASE_ADRESS`）でエミュレートし、電源が失われた場合でも保持する必要がある NVM データを格納します。

図 9. メモリマッピング

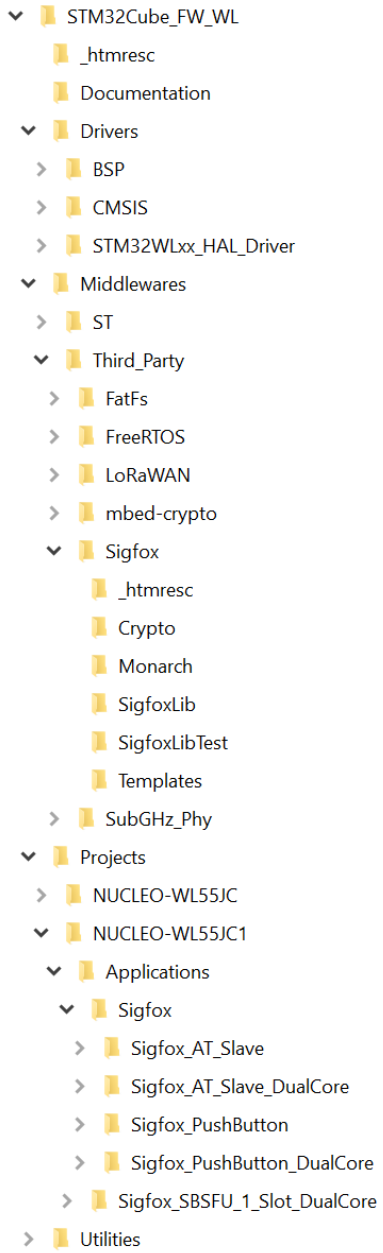


11 アプリケーションの説明

11.1 ファームウェアパッケージ

ユーザが STM32CubeWL マイクロコントローラパッケージのファームウェアを解凍した場合、下図のようなフォルダ構造になります。

図 10. パッケージ概要



STM32CubeWL のファームウェアには、次の 2 つの Sigfox アプリケーションが含まれています：Sigfox_AT_Slave および Sigfox_PushButton。

11.2 AT モデムアプリケーション

このアプリケーションの目的は、外部ホストにより UART 経由の AT コマンドインタフェースを介して制御される Sigfox モデムを実装することです。ここで、この外部ホストはアプリケーションを内蔵しているホストのマイクロコントローラと AT ドライバ、または単にターミナルを実行しているコンピュータであっても構いません。AT_Slave アプリケーションは、UART 上の AT コマンドインタフェースを通じて制御される Sigfox Stack を実装します。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 モードです。

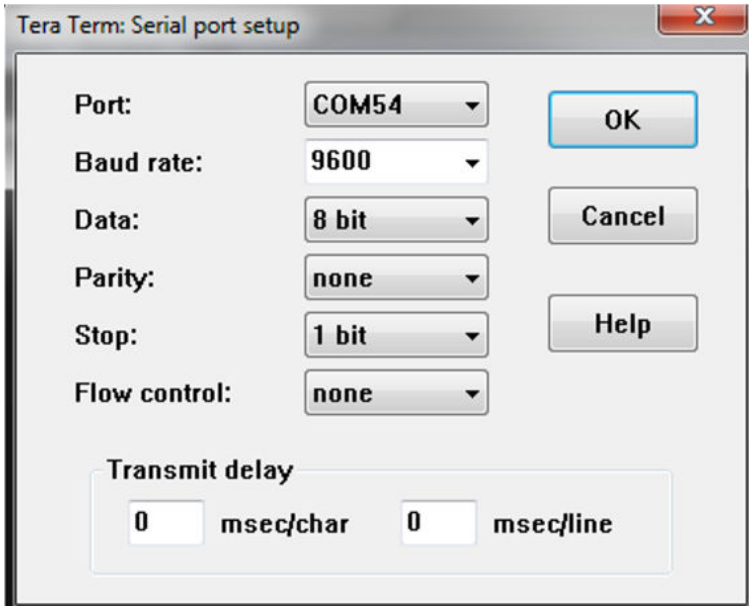
AT_Slave プロジェクトを起動するには、`\Projects\NUCLEO-WL55JC\Applications\Sigfox\Sigfox_AT_Slave` フォルダーに移動し、(IDE 環境で)ツールチェーンフォルダーを 1 つ選択します。

11.2.1 UART インタフェース

この例では、LPUART は 9600 ボーで使用されます。デバイスは STOP 2 モードでもキャラクタを受信できます。

下図の設定では、Sigfox モデムを制御するためのターミナルとして Tera Term を使用します。

図 11. Tera Term シリアルポートの設定



使用可能なコマンドを セクション 11.2.3 から セクション 11.2.24 に次の形式で示します。

- すべてのコマンドの設定パラメータは次の形式です:`ATXX=Y<CR>`。
- すべてのコマンドの取得パラメータは次の形式です:`ATXX=?<CR>`。

11.2.2 デフォルトパラメータ

プログラムを初めて起動したとき(EEPROM が空の状態)のデフォルトパラメータは、次のとおりです。

- 地域設定のデフォルト値:RC1
- 出力電力:13 dBm
- デフォルトの鍵:秘密鍵

これらのデフォルト値は、設定ファイル `sgfx_eeprom_if.c` の `E2P_RestoreFs` を修正することによって変更できます。

デフォルトの秘密鍵と秘密 ID は、Sigfox テスト仕様に記載されているテストキーです。これらは `sigfox_data.h` ファイルに保存されています。

11.2.3 AT? - 使用可能なコマンド

説明	Attention は、リンクが正しく機能しているかどうかを確認するために使用されます。 AT? には、サポートされているすべてのコマンドの簡単なヘルプが用意されています。
構文	AT?<CR>
引数	なし
応答	なし
リザルトコード	<CR><LF>OK<CR><LF>

AT コマンドの一般フォーマットを以下に示します。

- AT+<CMD>:<CMD> \r\n" を実行します。
- AT+<CMD>?:特定のコマンドの簡単なヘルプを表示します。
- AT+<CMD>?=<value>:値を設定するか、パラメータ \r\n" を指定して実行します。
- AT+<CMD>=?:与えられたコマンドの値を取得するために使用されます。

考えられるエラーステータスは次のとおりです。

- OK:コマンドはエラーなく正しく実行されます。
- AT_ERROR:一般エラー
- AT_PARAM_ERROR:コマンドのパラメータが正しくありません。
- AT_BUSY_ERROR:Sigfox モデムはビジーのため、コマンドを完了できません。
- AT_TEST_PARAM_OVERFLOW:パラメータが長すぎます。
- AT_LIB_ERROR:Sigfox ライブラリー般エラー
- AT_TX_TIMEOUT:CS のため Tx はできません(LBT 領域のみ)。
- AT_RX_TIMEOUT:ダウンリンクウィンドウ中に Rx フレームを受信しませんでした。
- AT_RX_ERROR:コマンド受信時のエラー検出
- AT_RECONF_ERROR

11.2.4 ATZ - リセット

説明	(無線、マイクロプロセッサを含む)システム全体に影響を与える NVIC リセットを生成します。
構文	ATZ<CR>
引数	なし
応答	なし
リザルトコード	なし

このコマンドはデバイスをリセットするのみです。EEPROM データは保持されます(セクション 11.2.5 AT$RFS - 出荷時設定 を参照)。

11.2.5 AT$RFS - 出荷時設定

説明	E2P_RestoreFs 機能の sgfx_eeprom_if.c で定義した出荷時設定を復元します。
構文	AT$RFS<CR>
引数	なし
応答	なし
リザルトコード	<CR><LF>OK<CR><LF>

11.2.6 AT+VER - ファームウェアとライブラリのバージョン

説明	ファームウェアおよびライブラリのバージョンを取得します。
構文	AT+VER<CR>
引数	なし
応答	ファームウェアとライブラリのバージョン
リザルトコード	<CR><LF>OK<CR><LF>

11.2.7 AT$ID - デバイス ID

説明	32 ビットのデバイス ID を取得します。
構文	AT$ID<CR> AT$ID=?<CR>
引数	なし
応答	Id<CR><LF:MSB から LSB の 4 バイトの Id(8 ASCII)
リザルトコード	<CR><LF>OK<CR><LF>

11.2.8 AT$PAC - デバイス PAC

説明	8 ビットのデバイス PAC を取得します。
構文	AT$PAC<CR> AT$PAC=?<CR>
引数	なし
応答	PAC <CR><LF:8 バイトの PAC(16 ASCII)
リザルトコード	<CR><LF>OK<CR><LF>

11.2.9 ATS410 - 暗号化キー

説明	デバイス暗号化キーの設定をセットまたは取得します。
構文	ATS410=Arguments<CR> ATS410=?<CR>
引数	0:秘密鍵を使用します。 1:公開鍵を使用します。
応答	Encryption Key Configuration <CR><LF
リザルトコード	<CR><LF>OK<CR><LF>

デフォルトでは、ペイロードの暗号化はオフです。

11.2.10 ATS411 - ペイロードの暗号化

説明	デバイスペイロード暗号化モードを設定または取得します。
構文	ATS411=Arguments<CR> ATS411=?<CR>
引数	0:ペイロード暗号化オフ 1:ペイロード暗号化オン
応答	Payload Encryption Configuration <CR><LF>
リザルトコード	<CR><LF>OK<CR><LF>

11.2.11 AT$SB - ビットステータス

説明	Sigfox ネットワークに 1 ビットを送信します。
構文	AT$SB=<bit value>{,{,<Optional NbTxFlag>}<CR> {,<Optional NbTxFlag>}<CR>
引数	<bit value>:0 または 1 <Optional ResponseWaited>=0:応答待ちなし(デフォルト) <Optional ResponseWaited>=1:応答待ち <Optional NbTxFlag>=0:1 つの Tx フレームを送信 <Optional NbTxFlag>=1:3 つの Tx フレームを送信(デフォルト)
応答	なし
リザルトコード	<CR><LF>OK<CR><LF>

例:

- AT$SB=1:応答を待たずにビット 1 を送信します。
- AT$SB=0,1:応答を待った状態で、ビット 0 を送信します。
- AT$SB=0,1,1:応答待ちで、3 つの Tx フレームを送信した状態でビット 0 を送信します。

11.2.12 AT$SF - ASCII ペイロード(バイト)

説明	Sigfox ネットワークにフレームを送信します。
構文	ペイロードの送信: AT$SF=<payload data>{,<Optional ResponseWaited>} {,<Optional NbTxFlag> }<CR>
引数	<payload data>:ASCII フォーマットで最大 12 バイト(最大 ASCII 24 文字) <Optional ResponseWaited>=0:応答待ちなし(デフォルト) <Optional ResponseWaited>=1:応答待ち <Optional NbTxFlag>=0:1 つの Tx フレームを送信 <Optional NbTxFlag>=1:3 つの Tx フレームを送信(デフォルト)
応答	なし
リザルトコード	<CR><LF>OK<CR><LF>

例:

- AT$SF=313245:応答待ちなしで 0x31 0x32 0x45 ペイロードを送信。
- AT$SF=010205,1:応答待ちありで 0x01 0x02 0x05 ペイロードを送信。

11.2.13 AT$SH - 16 進ペイロード（バイト）

説明	Sigfox ネットワークにフレームを送信します。
構文	ペイロードの送信： AT$SH=<payload length><payload data>{,<Optional ResponseWaited>}{,<Optional NbTxFlag> }<CR>
引数	<payload length>:長さ（バイト） <payload data>:16 進形式で最大 12 バイト <Optional ResponseWaited>=0:応答待ちなし（デフォルト） <Optional ResponseWaited>=1:応答待ち <Optional NbTxFlag>=0:1 つの Tx フレームを送信 <Optional NbTxFlag>=1:3 つの Tx フレームを送信（デフォルト）
応答	なし
リザルトコード	<CR><LF>OK<CR><LF>

例：

- AT$SH=1,A:応答を待たずに 0x41 ペイロードを送信します。
- AT$SH=1,A,1:応答待ちありで、0x41 ペイロードを送信します。

11.2.14 AT$CW - 連続波（CW）

説明	テスト向け連続無変調キャリアを開始／停止します。
構文	AT$CW=<frequency><CR>
引数	<frequency>:周波数（Hz または MHz） <frequency>=0 の場合、テストは停止します。
応答	なし
リザルトコード	<CR><LF>OK<CR><LF>

AT$CW=<input><CR> コマンドは、連続した無変調キャリアを送信します。

注

- デフォルトのパワーは RC1 の 14 dBm で、ATS302 - 無線出力パワーで変更できます。
- このコマンドは、デバイスの CE 認証には必須です。
- 電力は、選択した地域に応じて EEPROM に格納されます。

例：

- AT$CW=868:868 MHz で CW を開始します。
- AT$CW=902000000:902 MHz で CW を開始します。
- AT$CW=0:CW を停止します。

11.2.15 AT$PN - PRBS9 BPBSK テストモード

説明	テスト用の連続変調キャリアを送信します。
構文	AT$PN=<input>,<bitrate><CR>
引数	<frequency>:周波数(Hz または MHz) <frequency>=0 の場合、テストは停止します。 <bitrate>=100 または 600:中心周波数以内で入力した場合
応答	なし
リザルトコード	<CR><LF>OK<CR><LF>

注
- デフォルトのパワーは RC1 の 14 dBm で、ATS302 - 無線出力パワーで変更できます。
- このコマンドは、デバイスの CE 認証には必須です。
- 電力は、選択した地域に応じて EEPROM に格納されます。

例:
- AT$PN=868,100:868 MHz でデータレート 100 CW の BPSK 変調された連続キャリアを 868 MHz で開始します。
- AT$PN=902000000,600:868 MHz でデータレート 600 CW の BPSK 変調された連続キャリアを 902 MHz で開始します。
- AT$PN=0:CW を停止します。

11.2.16 AT$MN - Monarch スキャン

説明	Monarch スキャンを実行します。
構文	AT$MN={<Optional time>}<CR><CR>
引数	<Optional time>:スキャン時間(秒)(デフォルト = 5 秒)
応答	No RC found RC1 found RC2 found RC3c found RC4 found RC5 found RC6 found RC7 found
リザルトコード	<CR><LF>OK<CR><LF>

例:
- AT$MN:Monarch スキャンを 5 秒間実行します。
- AT$MN=10:Monarch スキャンを 10 秒間実行します。

11.2.17 AT$TM - Sigfox テストモード

モデムは、このコマンドを実装する必要があります。このテストモードは、Sigfox RSA(無線シグナルアナライザ)および SDR ドングルと合わせて使用できます(詳細については、https://resources.sigfox.com の Sigfox RSA ユーザガイドを参照)。

このコマンドはテストモード専用であり、Sigfox ネットワークへの接続には使用できません。

Sigfox RSA テスタは、次のように設定する必要があります（RSA バージョン 2.0.1）。

1. Device Configuration を開きます。
2. Radio Configuration を設定します。
3. Payload Encryption Capable を、Payload Encryption Configuration に設定します。
4. Oscillator Aging を 1 に設定します。
5. Oscillator Temperature Accuracy を 1 に設定します。
6. Settings を適用します。
7. Open（テスターを開始）。

説明	Sigfox テストモードを開始します。
構文	AT$TM=<rc>,<mode><CR>
引数 <rc>	テストを実行する必要がある RC の rc = 1、2、3c、4、5、6、または 7。
引数 <mode>	• SFX_TEST_MODE_TX_BPSK=0 表 3 で定義されたアップリンク周波数 Tx_frequency で、同期ビットと PRBS 同期フレームを含む BPSK 26 バイトパケットのみを送信します。アップリンク周波数は RC に依存します。 RSA テスト：UL-RF Analysis を選択後にスタートを押して AT$TM=x,0 コマンドを起動します。
引数 <mode>（続き）	• SFX_TEST_MODE_TX_PROTOCOL=1 使用可能なすべての長さの Sigfox プロトコルフレームをホッピング付きで送信する内部 Sigfox キーを持つフルプロトコル（考えられるすべてのペイロード長 1 から 12 バイトで、ダウンリンクフラグをセットおよびセットしないでビットを送信、帯域外フレームを送信、ダウンリンクフラグをセットおよびセットしないでフレームを送信） config：テストが行われた回数 RSA test： – UL-Protocol を選択後にスタートを押して、AT$TM=x,1 コマンドを起動します。 – UL-Protocol w/Encrypted Payload を選択後にスタートを押し、AT$TM=x,1 コマンド起動前に ATS411=1 をセットします。次のテストの前に忘れずに ATS411=0 を設定してください。 • Mode =SFX_TEST_MODE_RX_PROTOCOL=2 使用可能なすべての長さの Sigfox プロトコルフレームをホッピング付きで送信する内部 Sigfox キーを持つフルプロトコル（考えられるすべてのペイロード長 1 から 12 バイトで、ダウンリンクフラグをセットおよびセットしないでビットを送信、帯域外フレームを送信、ダウンリンクフラグをセットよびセットしないでフレームを送信） 注意　このテストは数分続きます。 RSA テスト – DL-Protocol を選択後にスタートを押して、AT$TM=x,2 コマンドを起動します。 – RSA テスト w/Encrypted Payload を選択後にスタートを押し、AT$TM=x,2 コマンド起動前に ATS411=1 をセットします。次のテストの前に忘れずに ATS411=0 を設定してください。 – Start of listening window を選択後にスタートを押して、AT$TM=x,2 コマンドを起動します。 – End of listening window then を選択後にスタートを押して、AT$TM=x,2 コマンドを起動します。 • SFX_TEST_MODE_RX_GFSK=3 表 3 で定義されたダウンリンク周波数 Rx_frequency で送信される、期待されるパターン = AA AA B2 27 1F 20 41 84 32 68 C5 BA 53 AE 79 E7 F6 DD 9B の GFSK の Rx モード。ダウンリンク周波数は RC に依存します。テストは 30 秒続きます。 RSA test：DL-GFSK Receiver を選択後に start send GSK を押して、AT$TM=x,3 コマンドを起動します。このテストは参考情報であり、必須ではありません。 • SFX_TEST_MODE_RX_SENSI=4 このテストは、デバイスの実際の感度を測定するために使用され、特定のタイミングで、Sigfox キーによりアップリンク 1 フレーム、およびダウンリンク 1 フレームを要求します。 RSA test：DL-Link Budget を選択後に start を押して、AT$TM=x,4 コマンドを起動します。 • SFX_TEST_MODE_TX_SYNTH =5 各 Sigfox チャネル周波数で 1 つのアップリンクフレームを処理します。このテストには数分かかります。 RSA test：UL-Frequency Synthesis を選択後に start を押して、AT$TM=x,5 コマンドを起動します。 • SFX_TEST_MODE_TX_FREQ_DISTRIBUTION=6 このテストは、SIGFOX_API_send_xxx 関数をコールして、0x40 から 0x4B までのアップリンクデータでアップリンクモードのみの完全なプロトコルをテストすることで構成されています。 RSA test：UL-Frequency-Distribution を選択後に start を押して、AT$TM=x,6 コマンドを起動します。 注意　このテストは数分続きます。

	• SFX_TEST_MODE_RX_MONARCH_PATTERN_LISTENING_SWEEP=7 このテストは、LISTENING_SWEEP モードにより、デバイスを 30 秒間パターンスキャンに設定し、期待されるパターンに対して検出されたパターンにより、TRUE または FALSE のステータスを報告することで構成されています。 RSA テスト:RSA では使用できません。 • SFX_TEST_MODE_RX_MONARCH_PATTERN_LISTENING_WINDOW=8 このテストは、LISTENING_WINDOW モードにより、デバイスを 30 秒間パターンスキャンに設定し、期待されるパターンに対して検出されたパターンにより、TRUE または FALSE のステータスを報告することで構成されています。 RSA テスト:RSA では使用できません。 • SFX_TEST_MODE_RX_MONARCH_BEACON=9 RSA テスト:RSA SDR ドングルでは使用できません。Monarch Link Budget を選択後に start を押して、AT$TM=x,10 コマンドを起動します。
引数 <mode>（続き）	• SFX_TEST_MODE_RX_MONARCH_SENSI=10 RSA テスト:RSA SDR ドングルでは使用できません。 – Monarch signal at high power を選択後に start を押して、AT$TM=x,10 コマンドを起動します。 High Power Level interferer for Monarch を選択後に start を押して、AT$TM=x,10 コマンドを起動します。 start Robustness から Low Power Level interferer for Monarch を押して、AT$TM=x,10 コマンドを起動します。 • SFX_TEST_MODE_TX_BIT=11 このテストは、SIGFOX_API_send_bit 関数を 2 回コールして、アップリンクのみのプロトコルと LBT の一部をテストすることで構成されています。 • SFX_TEST_MODE_PUBLIC_KEY=12 公開鍵を有効化して Sigfox フレームを送信します。アップリンク周波数は RC に依存します。 RSA test:UL-Public Key, を選択後に start を押して、AT$TM=x,12 コマンドを起動します。 • SFX_TEST_MODE_PUBLIC_KEY=13 このテストでは、NVM データの PN で関数を 1 回コールし、NVM ストレージを確認することで構成されています。 RSA test:UL-Non-Volatile Memory, を選択後に start を押して、AT$TM=x,13 コマンドを起動し、次に電源を取り外して、再度 AT$TM=x,13 コマンドを送信します。
応答	なし
リザルトコード	<CR><LF>OK<CR><LF>

11.2.18 AT+BAT? - バッテリ・レベル

説明	バッテリ・レベル（mV）を取得します。
構文	AT+BAT?<CR>
引数	なし
応答	バッテリレベル（mV）を返します。
リザルトコード	<CR><LF>OK<CR><LF>

11.2.19 ATS300 - アウトオブバンド・メッセージ

説明	1 つのキープアライブ・アウトオブバンド・メッセージを送信します。
構文	ATS300<CR>
引数	なし
応答	なし
リザルトコード	<CR><LF>OK<CR><LF>

注 アウトオブバンド・メッセージには、Sigfox ネットワークでよく知られているフォーマットがあります。24 時間ごとに送信可能です。

11.2.20 ATS302 - 無線出力パワー

説明	無線出力パワーを設定／取得します。
構文	ATS302=<power><CR> ATS302=?<CR>
引数	<power>(dBm)
応答	なし
リザルトコード	<CR><LF>OK<CR><LF>

注
- RC1 のデフォルトのパワーは 13 dBm です。
- このコマンドは、デバイスの CE 認証には必須です。
- AT$RC で選択した地域の電力が EEPROM に保存されます(地域ごとに 1 つのパワー)。
- ファームウェアでは、推奨される電力より高い電力の入力を防ぐことはできません。

11.2.21 ATS400 - FCC の有効チャネル

説明	FCC の有効チャネルを設定します。
構文	ATS400=<8_digit_word0><8_digit_word1><8_digit_word2>, <timer_enable><CR> 0 で無効、1 で有効
引数	<8_digit_word0> <8_digit_word1> <8_digit_word2> <timer_enable>
応答	なし
リザルトコード	<CR><LF>OK<CR><LF>

注 デフォルト値 = <000003FF><00000000><00000000>,1

例

ATS400=<000001FF><00000000><00000000>,1

連続する Tx フレーム間のタイマが有効になり、次のマクロチャネルが有効になります。902.8 MHz、903.1 MHz、903.4 MHz、903.7 MHz、904.0 MHz、904.3 MHz、904.6 MHz、904.9 MHz、および 905.2 MHz。

注 最低 50 の FCC チャネルを確保するには、9 つ以上のマクロチャネルを有効にする必要があります(9 * 6 = 54)。設定された default_sigfox_channel は、少なくとも設定ワードで有効にする必要があります(セクション 6.1.3 標準設定 を参照)。

11.2.22 AT$RC – 地域設定

説明	地域設定(RC)を設定/取得します。
構文	AT$RC=<rc><CR> AT$RC=?<CR>
引数	<rc>、RC1、RC2、RC3c、RC4、RC5、RC6、RC7
応答	RC1、RC2、RC3c、RC4、RC5、RC6、RC7
リザルトコード	<CR><LF>OK<CR><LF>

AT$RC=<zone><CR> コマンドを使用して、現在のゾーンを設定できます(レスポンス:OK<CR>)

11.2.23 ATE - エコーモード

エコーモードの設定以外には使用しません。

11.2.24 AT+VL - Verbose level

説明	Verbose level を設定／取得します。
構文	AT$VL=<verbose level><CR> AT$VL=?<CR>
引数	<verbose level>:0、1、2 または 3
応答	0、1、2 または 3
リザルトコード	<CR><LF>OK<CR><LF>

Verbose level は EEPROM に格納されます。

11.3 PushButton アプリケーション

PushButton アプリケーションは、スタンドアロンの例です。ユーザがボタンを押すイベントが発生すると、このアプリケーションは温度とバッテリ電圧（mV）を読み出し、それをメッセージで Sigfox ネットワークに送信します。

Sigfox PushButton プロジェクトを起動するには、次のフォルダーに移動します：

`Projects\NUCLEO-WL55JC\Applications\Sigfox\Sigfox_PushButton`。その後、ツールチェーンフォルダを選択します。

注 ユーザボタン 1 が押されない限り、デバイスは常に STOP 2 モードです。

ペイロードの内容は、次のファイルを更新することによって変更できます：

関数 SendSigfox の `Sigfox_PushButton\Sigfox\App\sgfx_app.c`。他のセンサのデータが必要な場合は、ファイル `Sigfox_PushButton\Core\Src\sys_sensors.c` を更新します。

11.4 スタティックスイッチ

スタティックスイッチは、オプション機能（デバッグ、トレースなど）の切り替え、低消費電力の無効化、または一部の RF パラメータの調整に使用されます。

11.4.1 デバッグ･スイッチ

デバッグモードは、`\Projects\<target> \Applications\Sigfox\Sigfox_*app*\Core\Inc\sys_conf.h` において、次のコードにより有効にすることができます。

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

デバッグモードにすると、MCU が低消費電力モードになった場合でも、SWD デバッガピンが有効になります。

注 真の低消費電力を実現するには、`#define DEBUGGER_ENABLED` を 0 に定義する必要があります。

11.4.2 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 モードに移行します。STOP 2 モードへの移行は下のコードにより `\Projects\<target>\Applications\Sigfox\Sigfox_*app*\Core\Inc\sys_conf.h` で無効にすることができます。

```
#define LOW_POWER_DISABLE 0
```

ここで：

- 0：低消費電力 STOP 2 モードは有効です。マイクロコントローラは STOP 2 モードへ移行します。
- 1：低消費電力 STOP 2 モードは無効です。マイクロコントローラは SLEEP モードにのみ移行します。

11.4.3 トレースレベル

トレースモードは、\Projects\<target> \Applications\Sigfox\Sigfox_*app*\Core\Inc\sys_conf.h において、次のコードにより有効にすることができます。

```
#define APP_LOG_ENABLED 1
```

トレースレベルは、\Projects\<target> \Applications\Sigfox\Sigfox_*app*\Core\Inc\sys_conf.h において、次のコードにより選択することができます。

```
#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化）
- VLEVEL_L（機能トレースを有効化）
- VLEVEL_M（デバッグトレースを有効化）
- VLEVEL_H（すべてのトレースを有効化）

11.4.4 プローブピン

\Projects\<target> \Applications\Sigfox\Sigfox_*app*\Core\Inc\sys_conf.h において、次のコードにより 4 つのプローブピンを有効にすることができます。

```
#define PROBE_PINS_ENABLED 0
```

プローブピンを有効にするには、PROBE_PINS_ENABLED を 1 に設定する必要があります。

11.4.5 無線設定

無線設定は Sigfox\Target\radio_conf.h で更新可能です。

Sigfox アプリケーションをドリフトレート仕様に準拠するには、RF_WAKEUP_TIME を 15 ms に設定する必要があります。

12 デュアルコアマネージメント

STM32WL5x デバイスには、次の 2 つの CPU コアを搭載しています。

- Cortex-M4（名称は CPU1）
- Cortext-M0+（名称は 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レイヤは 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次のとおり生成されます。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 つのバイナリの機能アドレスは、別のバイナリからはわかりません。これが、通信モデルを確立する必要がある理由です。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は、セクション 12.1 で説明しているように、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。

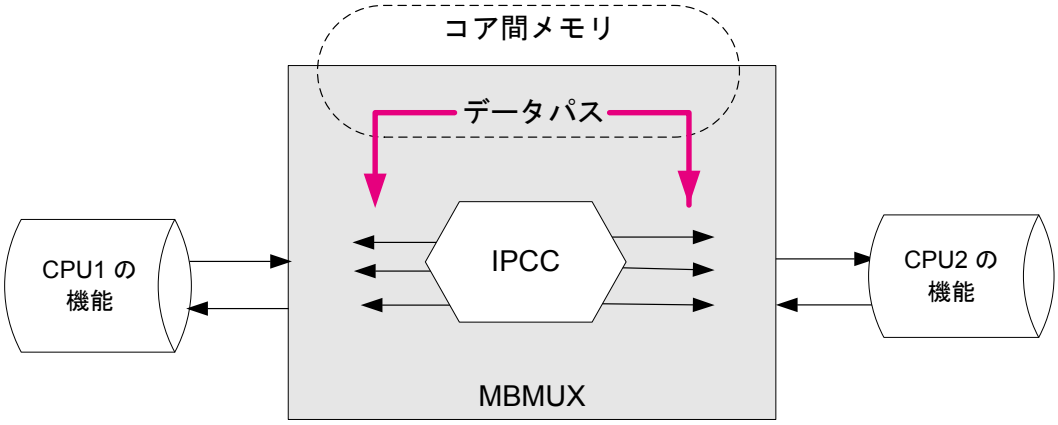
このデュアルコア実装は、メールボックスメカニズムを介したメッセージブロック処理により、シングルコアプログラムの実行と同じように動作するように設計されています。

12.1 メールボックスのメカニズム

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 はさまざまな設定が可能で、各割込み通知を有効／無効にすることが可能です。IPCC には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／書込みが可能です。2 つの CPU 間で交換するデータを保存するすべてのバッファを格納するために使用されます。

図 12. メールボックスの概要



メールボックスは、下位互換性を損なうことなく、バッファ定義をある程度まで変更できるようにするために指定されます。

12.1.1 メールボックスマルチプレクサ（MBMUX）

図 13 で説明されているように、交換するデータは、利用可能な 12 個の IPCC チャネルを介して通信する必要があります。（各方向 6 個）。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 から 6 の数字で識別されます。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。

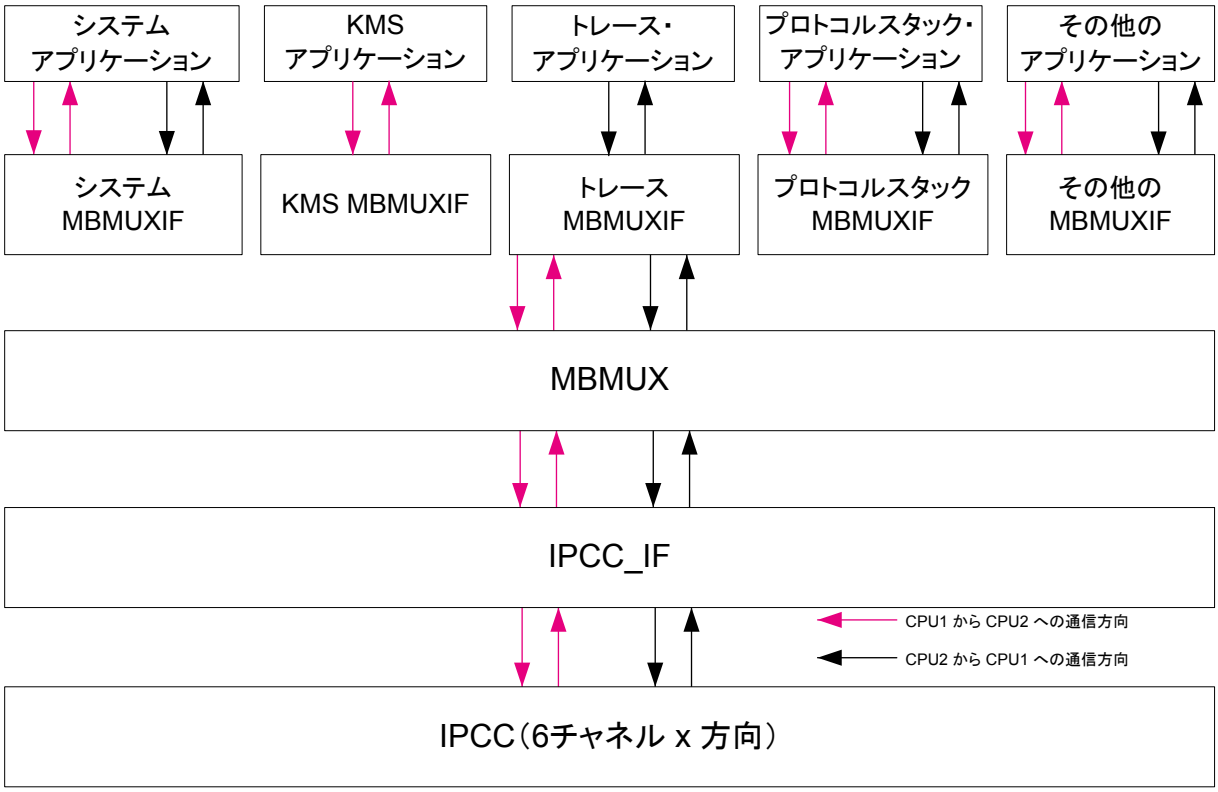
メールボックスは、別のコアによって実行される関数を抽象化するために使用されます。

12.1.2 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。
  これには、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、CPU2 からそのコマンドの応答を受信するために使用されます。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 CPU2 チャネル 0 が使用されます。
  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
  - 機能登録に対する IPCC チャネル
  - 機能の属性および能力について交換される情報
  - 優先度の高い操作（RTC 通知など）のための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 のログに使用される同じチャネル（USART など）で情報を出力することによって、CPU1 が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）およびスタックに実装されている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 13. 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、要求された数の IPCC チャネルが機能に動的に割り当てられます。通常、各方向（CPU1 から CPU2 および CPU2 から CPU1 へ）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 が CPU2 に関数の実行を要求するためにのみ使用します。

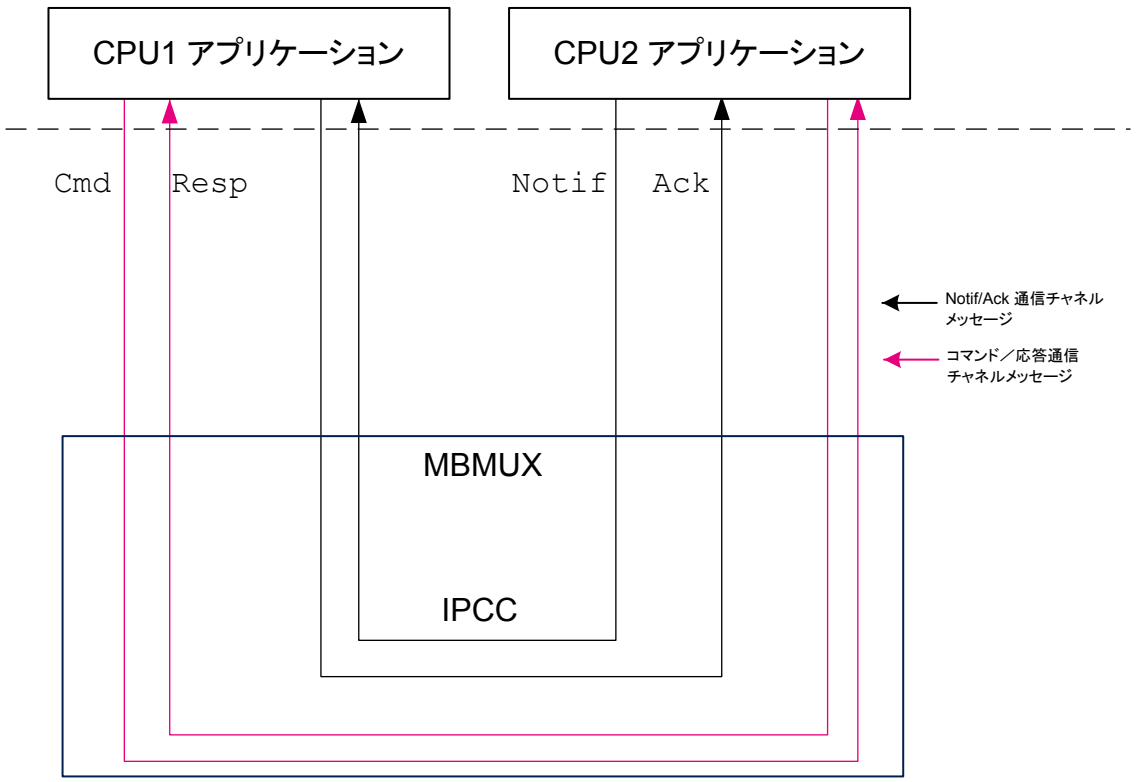
注
- RTC アラーム A は、1 つの IPCC IRQ を使用して割込みを転送するもので、機能とは見なされません。
- KMS ラッパーを機能として使用できるようにするには、ユーザは KMS ラッパーの追加を検討する必要があります。

12.1.3 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。

- CPU1 により CPU2 に送信される `Cmd` コマンドは、次の項目で構成されます。
  - `Msg ID` で、CPU1 によってコールされ CPU2 に実装されている関数を識別します。
  - `Ptr buffer params` は上記の関数のパラメータを含むバッファをポイントします。
  - パラメータの数
- `Resp` は、CPU2 により CPU1 に送信されるレスポンスで、次の項目で構成されます。
  - `Msg ID`（`Cmd Msg ID` と同じ値）
  - `Return value` には上記の関数の戻り値が含まれます。
- `Notif` は、CPU2 により CPU1 に送信される通知で、次の項目で構成されます。
  - `Msg ID` で、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識別します。
  - `Ptr buffer params` は上記の関数のパラメータを含むバッファをポイントします。
  - パラメータの数
- `Ack` は、CPU1 により CPU2 に送信される確認応答であり、次の項目で構成されています。
  - `Msg ID`(`Notif Msg ID` と同じ値)
  - `Return value` には上記のコールバック関数の戻り値が格納されます。

図 14. MBMUX および IPCC チャネルを介したメールボックスメッセージ



12.2 コア間メモリ

コア間メモリは、両方のコアがアクセスできる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。

12.2.1 CPU2 の性能

CPU2 でサポートされている機能のいくつかを(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、サポートされている地域など)CPU1 が詳細に認識している必要があります。

これらの CPU2 の機能は、features_info テーブルに保存されています。セクション 12.2.5 に示すとおり、CPU2 の機能を把握するためにこのテーブルのデータが初期化時に CPU1 により要求されます。

features_info テーブルは次のパラメータで構成されます。

- `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

これらの CPU2 機能を格納するために MB_MEM2 が使用されます。

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

CPU1 が CPU2 の `feature_func_X()` を呼び出す必要がある場合、同じ API を持つ `feature_func_X()` を CPU1 に実装する必要があります。

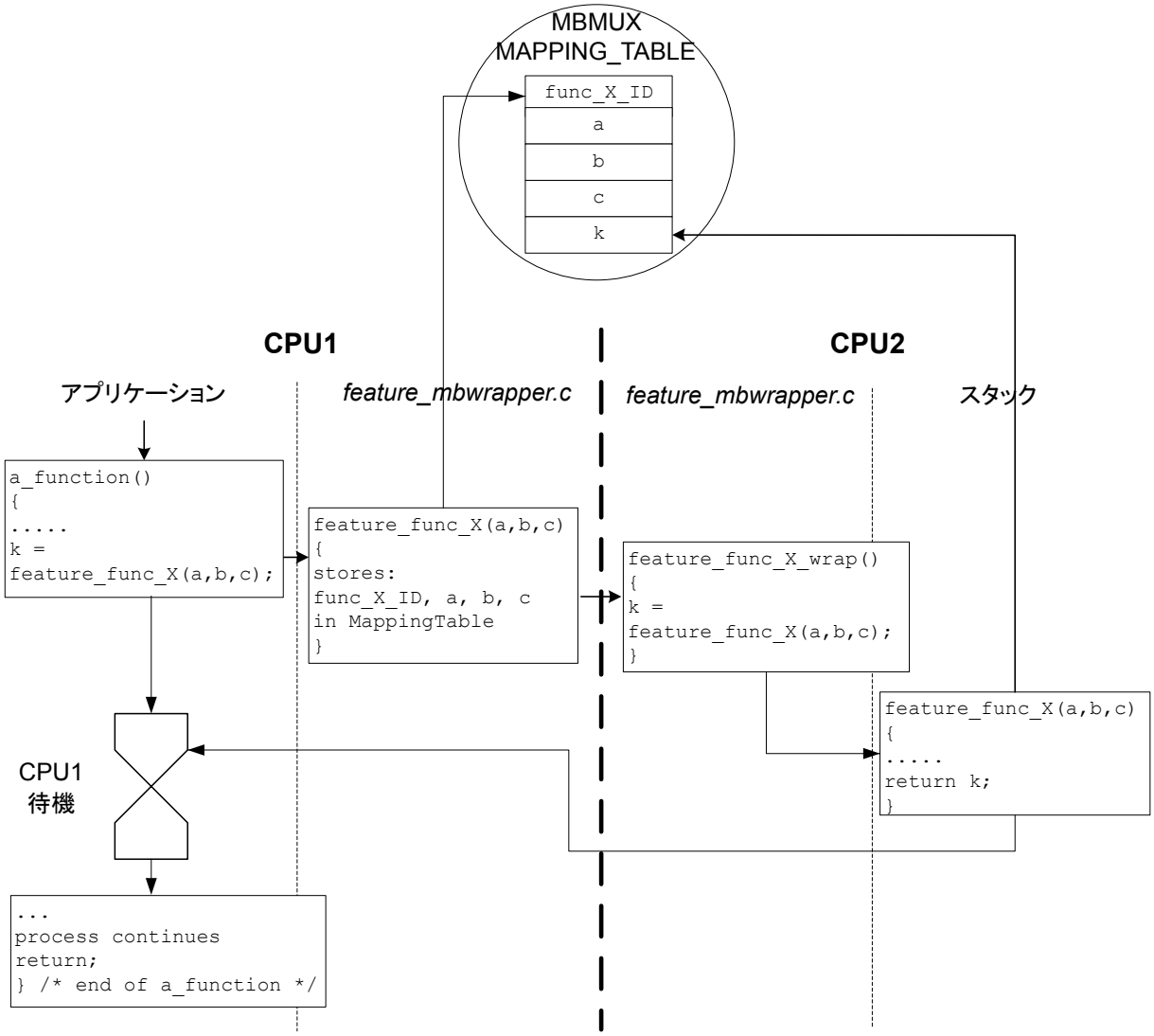
1. CPU1 は、マッピングテーブルに `feature_func_X()` パラメータを含むコマンドを送信します:
   a. 登録時の初期化で `feature_func_X()` に関連付けられた `func_X_ID` がマッピングテーブルに追加されます。`func_X_ID` は両方のコアが認識しなければなりません。これはコンパイル時に固定されます。
   b. CPU1 は、CPU2 が `feature_func_X()` を実行するのを待ってから、低消費電力モードに移行します。
   c. CPU2 が低消費電力モードだった場合はウェイクアップして、`feature_func_X()` を実行します。
2. CPU2 は、レスポンス 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 割込みが CPU1 をウェイクアップします。
   b. CPU1 は、マッピングテーブルから戻り値を取り出します。

反対に、CPU2 が CPU1 の `feature_func_X_2()` を呼び出す必要がある場合は、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()` を含む通知を送信します。
2. CPU1 は、確認を送信し、マッピングテーブルに戻り値を入力します。

下の図にフルシーケンスを示します。

図 15. CPU1 から CPU2 feature_func_X() プロセス

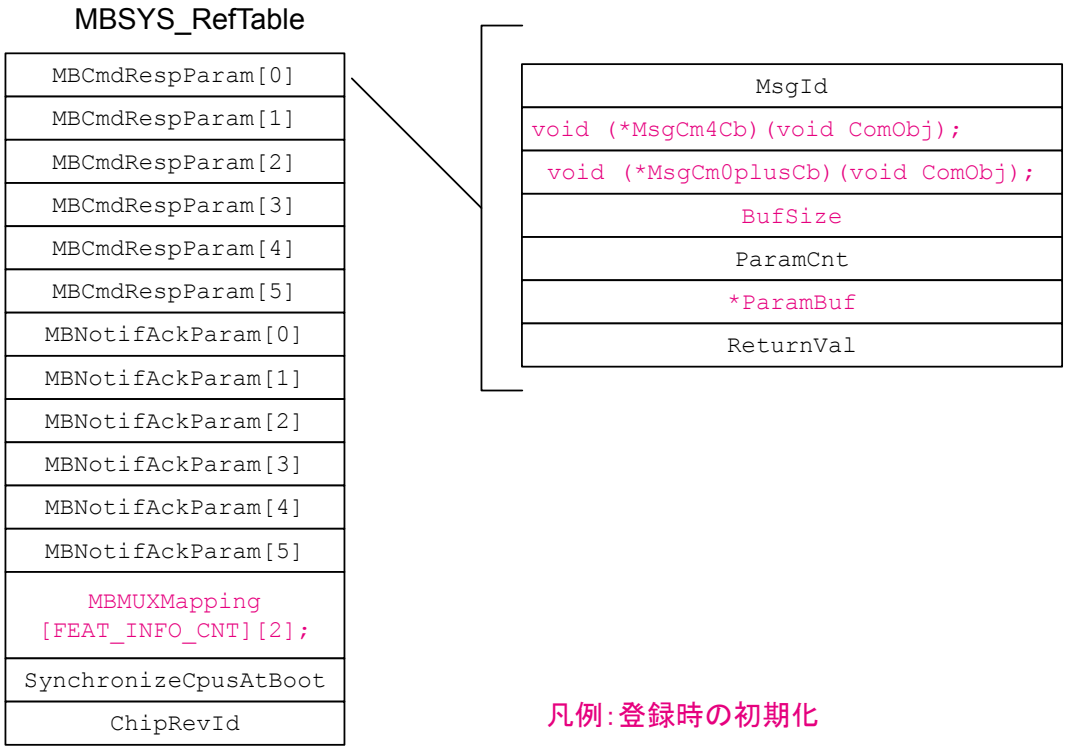


12.2.3 マッピングテーブル

マッピングテーブルは、図 15 の MBMUX 領域に共通の構造です。メモリマッピングは セクション 12.2.5 では、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 16. MBMUX 通信テーブル



この MBSYS_RefTable には次のものが含まれます。

- 6 つの IPCC チャネルのそれぞれの、コマンド／レスポンスおよび通知／確認パラメータの両方に対する 2 つの通信パラメータ構造。
  各通信パラメータは、図 15 の MBMUX マッピングテーブル領域に示すとおり、次の事項で構成されます。
  - `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、登録時の MBMUX の初期化で書き込まれます。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` は `[FEAT_INFO_RADIO_ID][1]` に関連付ける必要があります。
- `SynchronizeCpusAtBoot`:図 17 シーケンスチャートに示すように、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 ()` パラメータ送信に使用され、CPU1 に戻り値を取得します。

12.2.4 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ています（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。

12.2.5 RAM 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 27. STM32WL5x RAM マッピング

ページインデックス	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
アロケーション領域／セクション	CPU1 RAM																CPU1:RAM2_Shared		CPU2:RAM2_Shared		CPU2:RAM2										

1. 各ページ 2 KB。

表 28. STM32WL5x RAM 割当てと共有バッファ

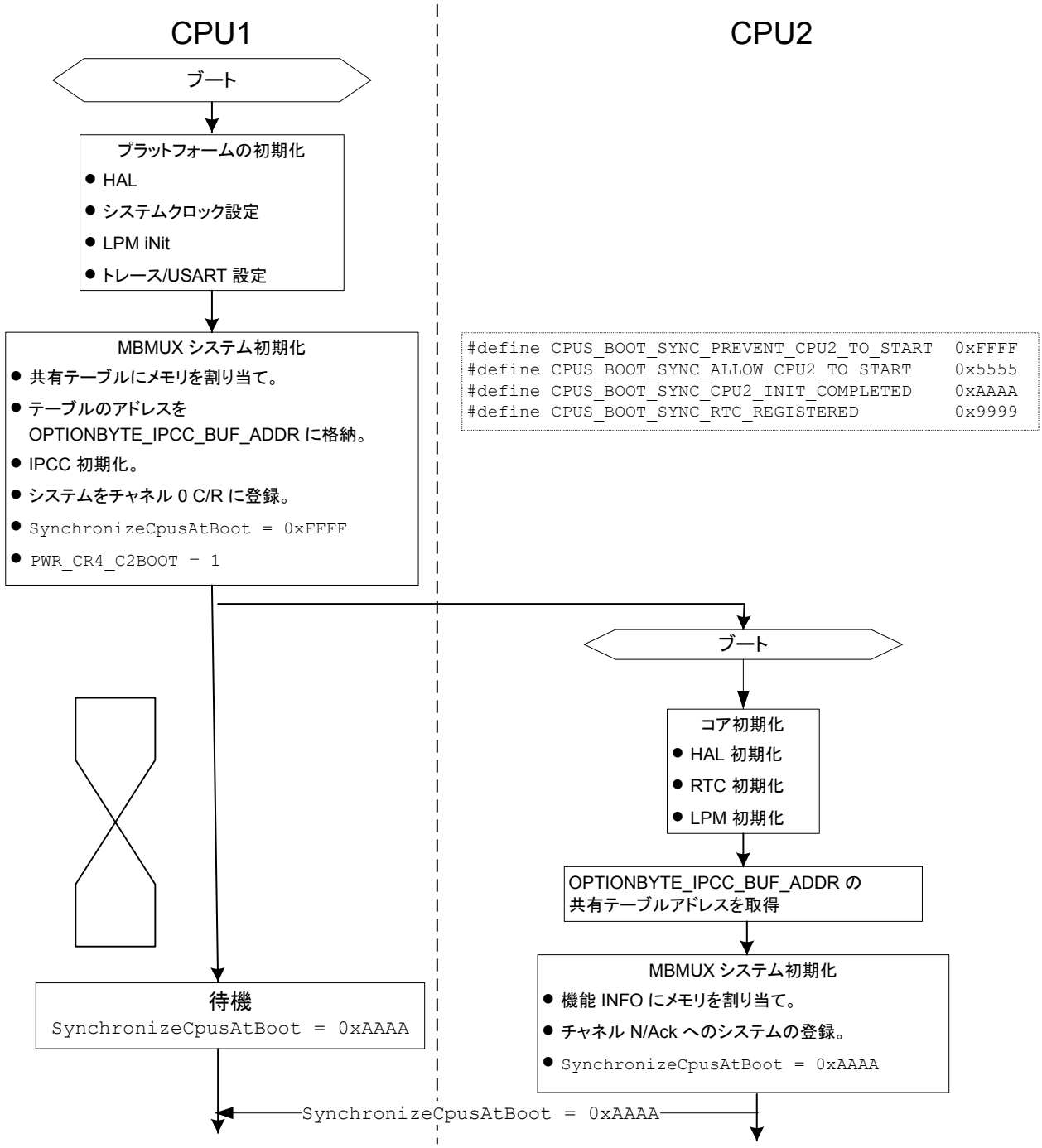
CPU 領域	セクション	モジュール	割当てシンボル	サイズ（バイト）	合計（バイト）
CPU1 RAM	readwrite	-			
	CSTACK				
	HEAP				
CPU1:RAM2_Shared	MAPPING_TABLE	MBMUX_SYSTEM	MBMUX_ComTable_t MBSYS_RefTable	316	316
	MB_MEM1	MBMUX_SIGFOX	uint32_t aSigfoxCmdRespBuff[]	60	292
			uint32_t aSigfoxNotifAckBuff[]	20	
		MBMUX_RADIO	uint32_t aRadioCmdRespBuff[]	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	
		SGFX_MBWRAPPER	uint8_t aSigfoxMbWrapShareBuffer[]	28	
CPU2:RAM2_Shared	MB_MEM2	MBMUX_TRACE	uint8_t ADV_TRACE_Buffer[]	512	860
		MBMUX_SIGFOX	SigfoxInfo_t SigfoxInfoTable	4	
			FEAT_INFO_Param_t Feat_Info_Table	80	

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_MEM2	MBMUX_SIGFOX	FEAT_INFO_List_t Feat_Info_List	8	860
		SGFX_MBWRAPPER	uint8_t aSigfoxMbWrapShare2Buffer[]	0	
		RADIO_MBWRAPPER	uint8_t aRadioMbWrapRxBuffer[]	256	
CPU2：RAM2	readwrite	-			
	CSTACK				
	HEAP				

12.3 起動シーケンス

CPU1 および CPU2 の起動シーケンスの詳細を下の図に示します。

図 17. 起動シーケンス

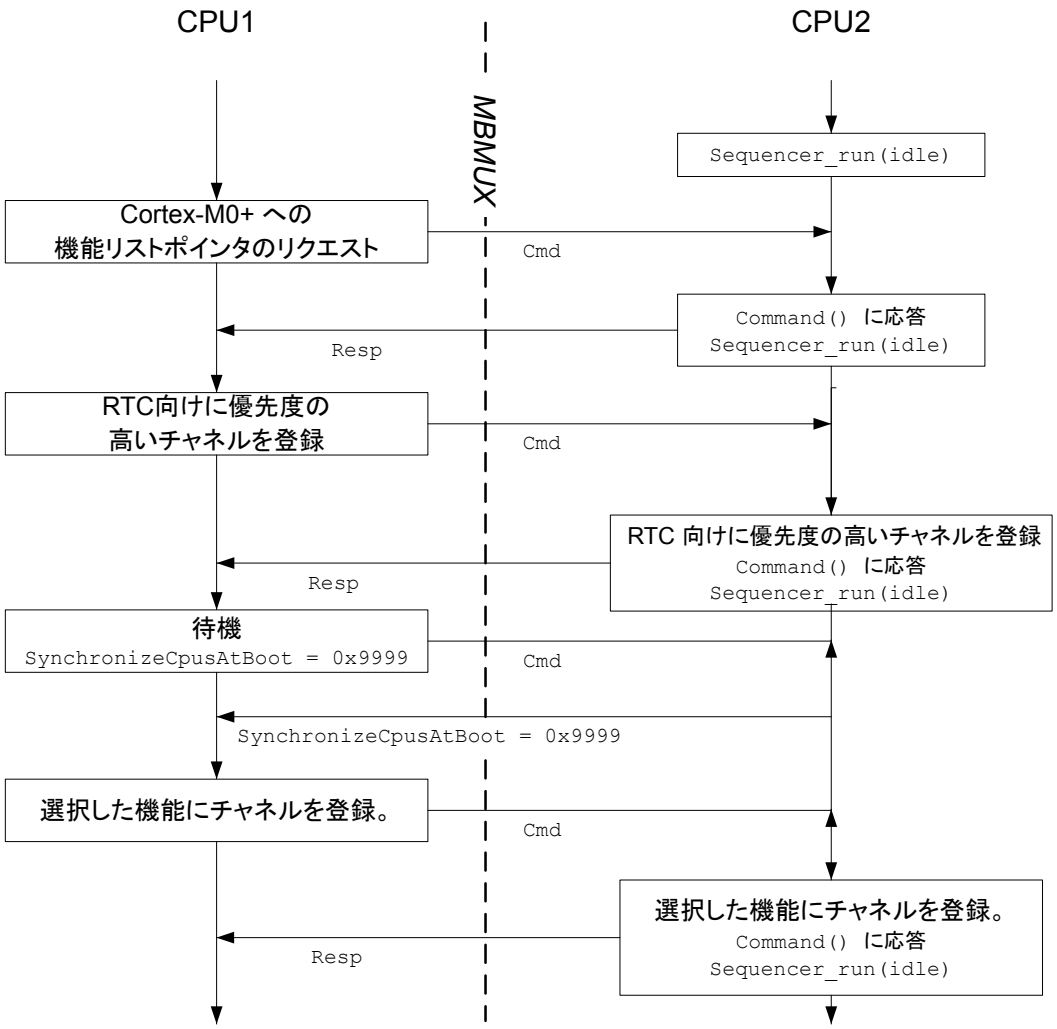


次に示すようにいくつかの手順があります。

1. その初期化シーケンスのマスタプロセッサである CPU1 は次のように動作します。
   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 CPU2 を起動します。
   d. CPU2 が `SynchronizeCpusAtBoot` フラグを 0xAAAA に設定するのを待ちます。
2. CPU2 が起動し、次の動作を行います。
   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、CPU1 に初期化シーケンスが終了してレディになったことを通知します。
3. CPU1 は CPU2 の通知を確認します。

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 18. MBMUX の初期化

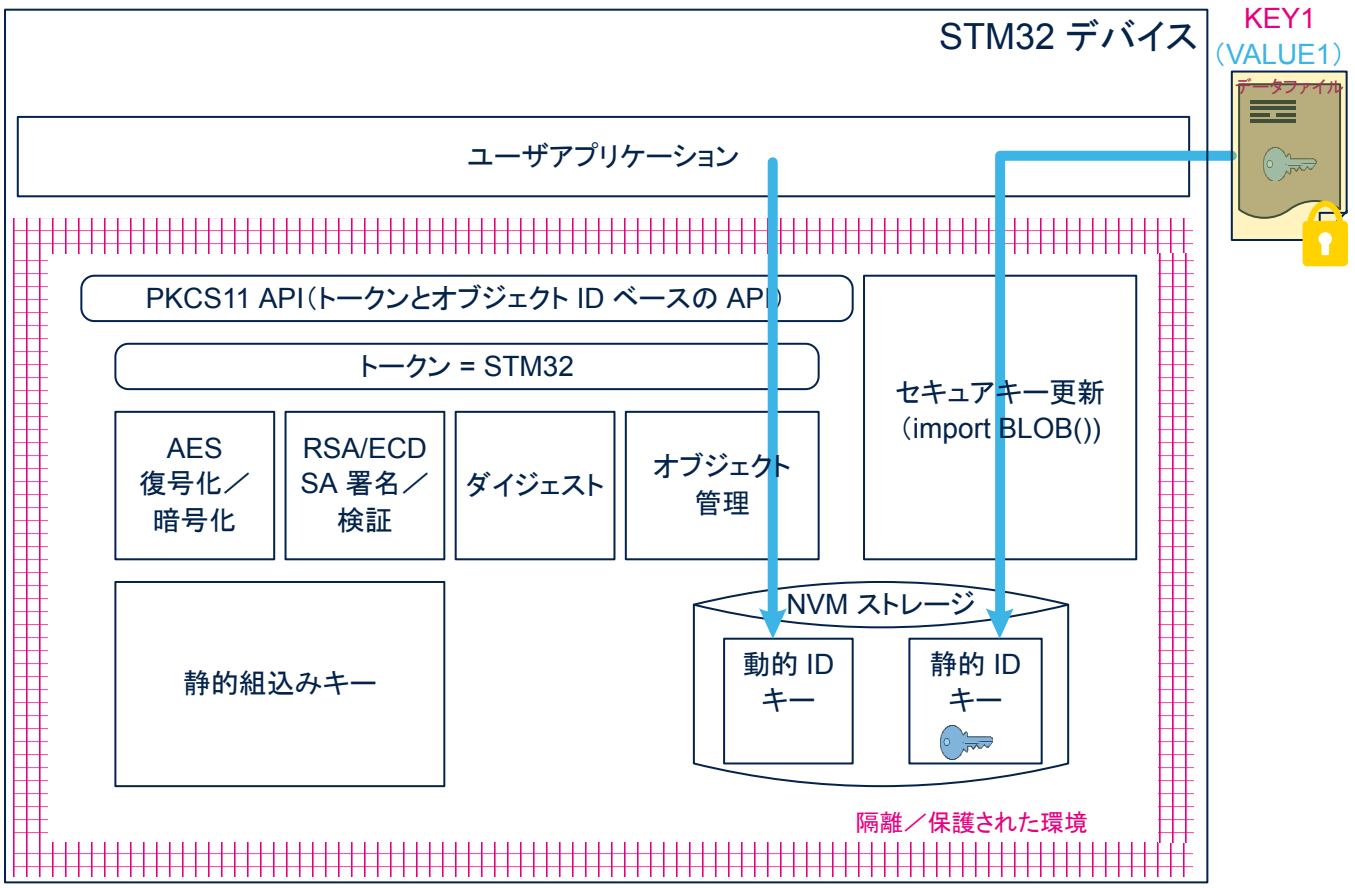


13 キー管理サービス(KMS)

キー管理サービス(KMS)は、標準 PKCS#11 API(OASIS により開発)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)。

下の図に示すように、KMS は保護/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することが可能です。

図 19. KMS の全体的なアーキテクチャ



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

KMS モジュールをアクティベーションするには、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)

13.1 KMS キータイプ

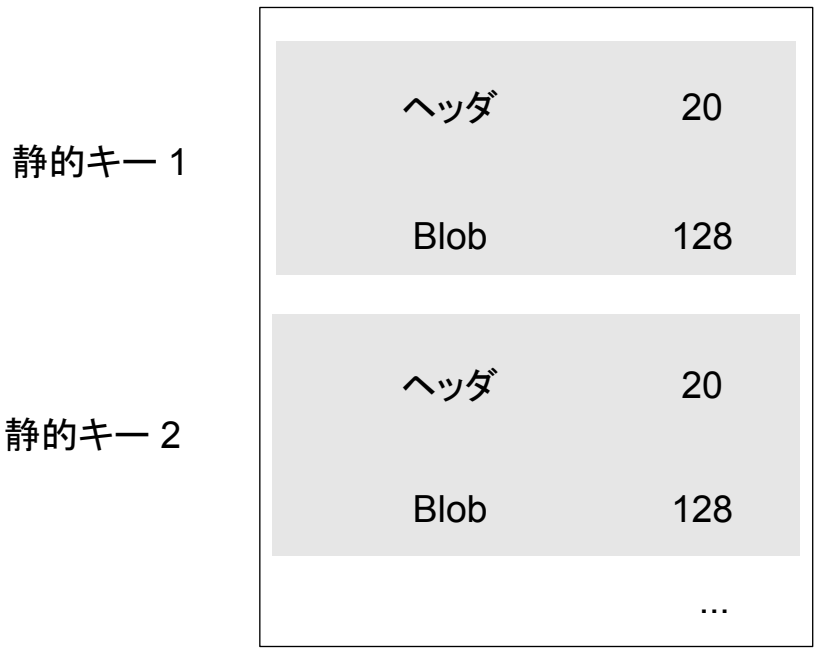
KMS は 3 種類のキーを管理しますが、使用されるのは次の 2 種類のみです。

- 静的組込みキー
  - コードに埋め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(`DeriveKey()` または `CreateObject()`)を使用してキーが作成されたときに定義されるキー ID
  - 削除または変更可能なキーとして定義可能なキー

13.2 KMS キーのサイズ

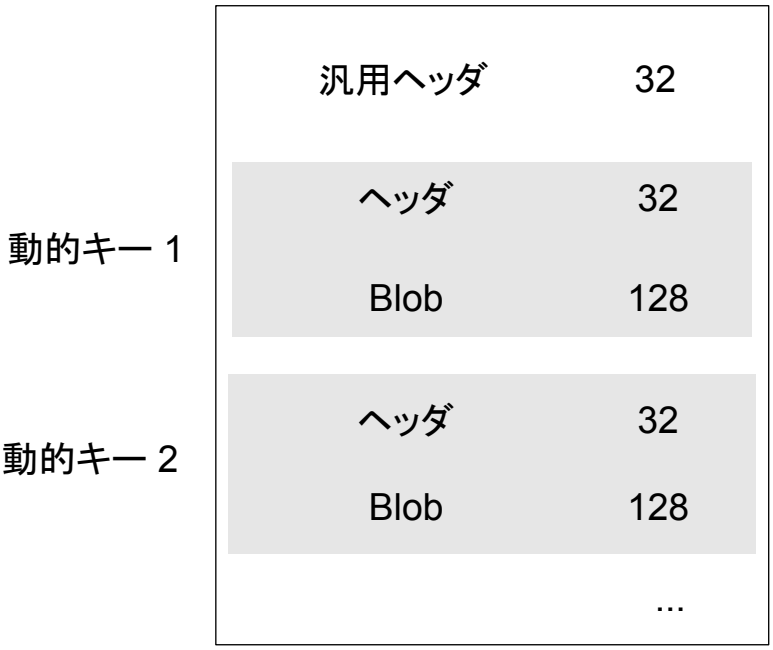
Sigfox のスタックで使用される静的キーと動的キーは、占有するサイズが異なります。下の図に示すように、各静的キーのサイズは 148 バイト = ヘッダー(20) + blob(128)です。

図 20. KMS 静的キーのサイズ



下の図に示すように、KMS キーストレージの先頭に KMS 汎用ヘッダ（32 バイト）があり、各動的キーのサイズは 160 バイト = ヘッダ（32）+ blob（128）です。

図 21. KMS 動的キーサイズ



13.3 Sigfox キー

STM32CubeWL アプリケーションリストでは、KMS はデュアルコアアプリケーションの Cortex-CM0+ でのみ使用されます。ルートキーは静的埋め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。

Sigfox スタックの場合、静的ルートキーは 1 つです：`Sigfox_Key`。

`Sigfox_pac` と `Sigfox_id` は KMS に格納されますが、暗号化キーとしては使用できません。

NVM_DYNAMIC が生成した揮発性キーは 1 つです：`Sigfox_Public_Key`。

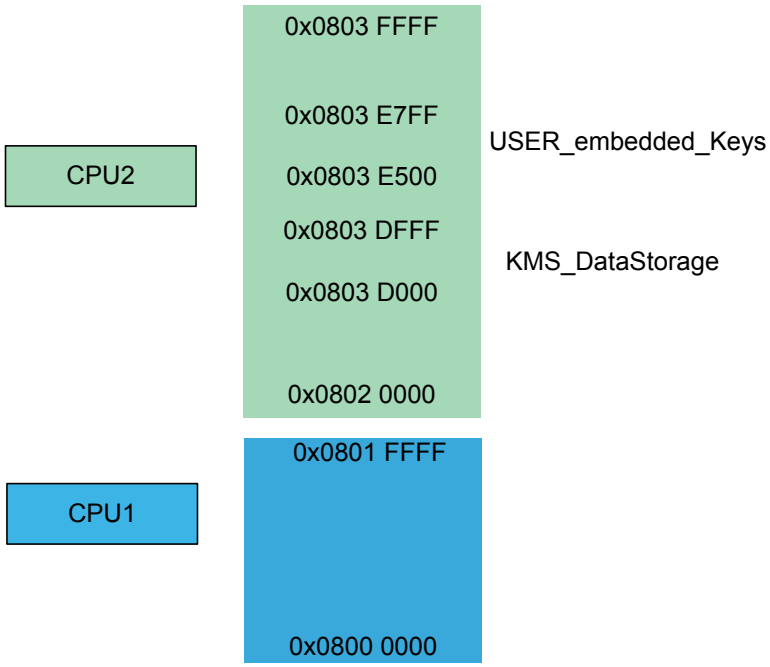
13.4 ユーザアプリケーションの KMS キーメモリマッピング

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 にキーを配置します。

NVM_DYNAMIC キーを KMS キーデータ格納領域 KMS_DataStorage に配置します。

KMS データストレージの NVM のサイズ設定で説明されているように、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。これらは下の図に示すとおり、0x0803 D000 から 0x0803 DFFF に配置されています。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 22. ROM メモリマッピング



13.5 KMS データストレージ用の NVM のサイズ設定

NVM は 2 KB 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サイズは 4 KB です。割当てのサイズは、リンカファイルで定義されます。

ユーザアプリケーションによって提案されたリンカファイルは、最小許容サイズ（2 * 2 KB）を使用します。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。

ユーザアプリケーションは、KMS キーを格納する目的でのみ NVM を使用します。Sigfox キーと、関連する選択された KMS 属性で 128 バイトを占有します。図 21 で説明されているように、KMS ヘッダはキーごとに 32 バイトを要し、すべてのキーに共通するグローバルヘッダには 32 バイトが必要です。この値から、2 KB に格納できるキーの数を計算することができます。

（2048 - 32）/（32 + 128）= 12.6 ==> 12 個の KMS キー（「KMS キー」は、キー値、キー属性、およびヘッダを意味します）。

ユーザアプリケーションは、NVM_DYNAMIC のみが使用されるように構成されます。NVM_STATIC は BLOB 経由で書き込むことができますが、ユーザアプリケーションには対応していません。

NVM_DYNAMIC は C_DeriveKey() 経由で派生キーを、C_CreateObject() キー経由でルートキーをホストできます。

Sigfox アプリケーションは NVM_DYNAMIC を派生キーのみに使用します。Sigfox_PushButton は、ペイロード暗号化が設定されている場合、データが送信（アップリンク）されるたびに 1 つの派生キーを生成します。

NVM のサイズが小さいほど、NVM の書込み／消去が多くなり、期待される寿命は短くなります。

キーを破壊するということは、キーが消去されるということではなく、破壊されたというタグが付けられるということです。このキーは、次のフリップフロップスイッチではコピーされません。破壊フラグも NVM バイトの一部を占有します。

以下の期待寿命の見積もりは、ペイロード暗号化セットの場合に対応しています（アップリンクごとに 1 つのキーが生成され、前のキーは破壊されます）。

- フリップフロップ転送が必要になる前に、最大 12 個の暗号化キーを生成できます。13 番目のアップリンクで、派生キーはページ 2 に格納され、ページ 1 は消去されます。
- 24 回の暗号化データのアップリンクの後、キーの格納場所はページ 1 に戻り、ページ 2 は消去されます。
- 240,000 回のアップリンク後、2 つの NVM ページは 10,000 回消去されています。これはフラッシュセクタの推定寿命です。
- Sigfox アップリンクの最大量は 1 日あたり 144 メッセージなので、期待寿命は約 4.5 年です。NVM のサイズを 2 倍にすると、寿命は 2 倍になります。

注

- ペイロードの暗号化が無効になっている場合、この計算は正しくありません。
- 古いキーは破壊する必要があります。そうしないと、ページ 1 がアクティブキーで完全にいっぱいになった場合、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。

13.6 アプリケーションを構築するための KMS 設定ファイル

Sigfox の例では、`CM0PLUS/Sigfox/App/app_sigfox.h` で `SIGFOX_KMS = 1` を設定して KMS を使用します。

次のファイルには、SubGhz スタックキー情報を書き込む必要があります。

- 組込みキー構造は、`CM0PLUS/Core/Inc/ kms_platf_objects_config.h` で定義されます。
- SubGhz スタックキーに関連付けられた組込みオブジェクトハンドル。KMS モジュールの使用は、次のファイルで定義されています。
  `CM0PLUS/Core/Inc/kms_platf_objects_interface.h`

13.7 組込みキー

SubGHz_Phy プロトコル・スタックの組込みキーは、セキュアな追加ソフトウェア（SBSFU（セキュアブート・セキュア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が保証される ROM 領域に格納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 Integration guide of SBSFU on STM32CubeWL（STM32CubeWL の SBSFU インテグレーションガイド）（AN5544）を参照してください。

これらの組込みキーは、図 22. ROM メモリマッピング に示すように ROM 内に配置されます。

14 パーソナライゼーションとアクティベーション

デフォルトの `sigfox_data.h` を使用してファームウェアをコンパイルしてロードする場合、デフォルトの Sigfox 認証情報がデバイスにロードされます。これにより、ラボの RSA の前でローカルに Sigfox デバイスをテストすることができます。

Sigfox デバイスが Sigfox バックエンドサーバにデータを送信するには、以下の手順が必要です。

1. パーソナライゼーション：すべての Sigfox デバイスには、デバイスをアクティベートして Sigfox データサーバにデータを送信するために必要な ID、PAC、および秘密鍵の認証情報をロードする必要があります。
2. アクティベーション：デバイスがパーソナライズされたら、Sigfox バックエンドサーバによって記録される必要があります。この手順では、Sigfox バックエンドサーバにログオンする必要があります。

注 以下の手順では、STM32CubeProgrammer バージョン 2.6.0 以降が必要です。

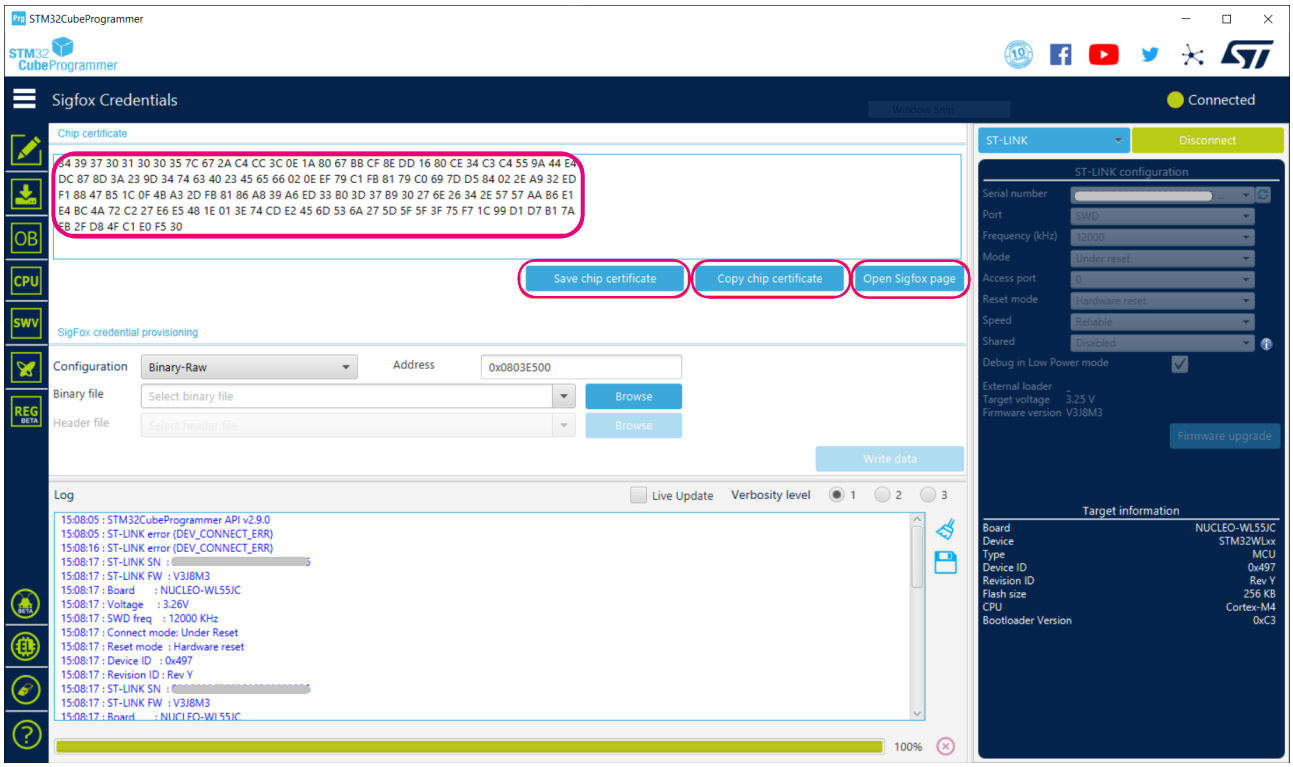
14.1 パーソナライゼーション

STM32WL デバイスを接続すると直ちに Sigfox Credentials（Sigfox 認証情報）ボタンがメインメニューに追加されます。

図 23. STM32CubeProgrammer の Sigfox パネルボタン

Sigfox Credentials（Sigfox 認証情報）ウィンドウを開いた後、チップ証明書が 136 バイトのサイズで自動的に抽出され、chip certificate（チップ証明書）領域に表示されます。この証明書は、バイナリファイルで保存し、クリップボードにコピーして、ST の Web インタフェースで Sigfox の認証情報を取得するために使用できます（セクション 14.1.1 を参照）。ウィンドウに統合された Open Sigfox page（Sigfox ページを開く）ボタンを使って ST ウェブインタフェースにアクセスすることができます。

図 24. STM32CubeProgrammer の Sigfox パネル - 証明書の取得

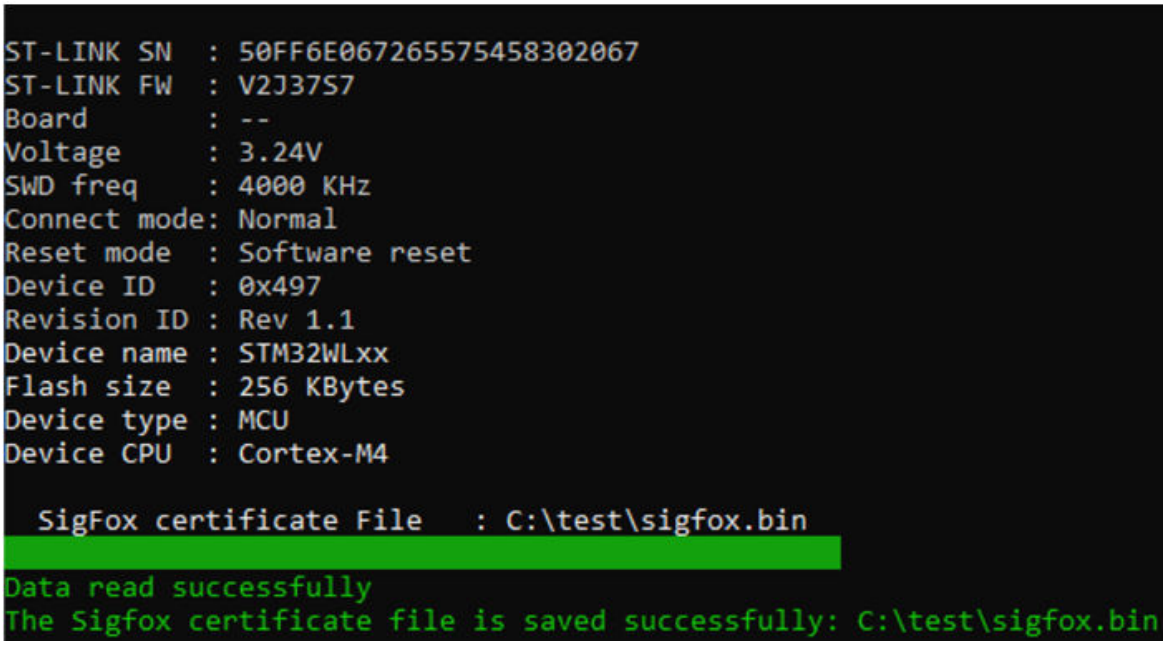


注　STM32CubeMx で Sigfox プロジェクトを生成した場合、Sigfox データが 0x803 E500 に配置されないことがあります。デバイスの Flash メモリが 256 KB より少ない場合、Sigfox データを 0x803 E500 に配置できません。
この 2 つのケースの結果、ユーザは、Sigfox データを別の場所に配置するか、またはマッピングファイルでこの場所を見つけてアドレスをそれに従って更新する必要があります。

チップ証明書をバイナリファイルに保存するコマンドライン:

- コマンド: `-ssigfoxc`
- 説明:このコマンドを使用すると、チップ証明書をバイナリファイルに保存できます。
- 構文: `-ssigfoxc <binary_file_path>`
- 例: `STM32_Programmer_CLI.exe -c port=swd -ssigfoxc /local/user/chip_certif.bin`

図 25. STM32CubeProgrammer の Sigfox CLI - 証明書の取得



14.1.1 認証情報の取得

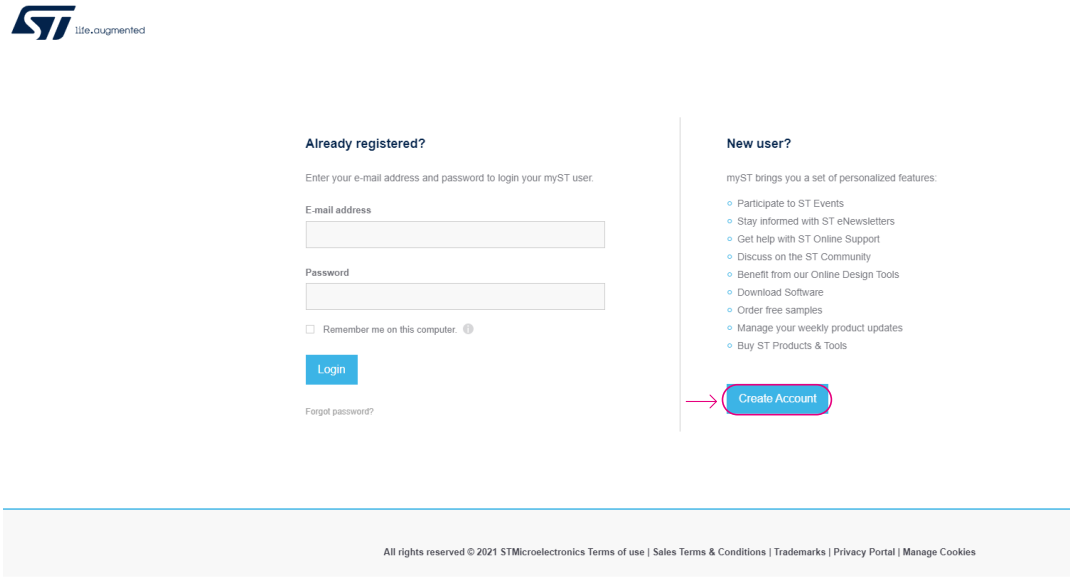
ST では、Sigfox 試用版の認証情報を取得できる Web インタフェースを my.st.com で提供しています。
認証情報は、次のファイルを含む zip ファイルで提供されます。

- `sigfox_data_XXXXXXXX.h`:アプリケーションのソースコードに統合できる認証情報を定義
- `sigfox_data_ XXXXXXXX.bin`:STM32CubeProgrammer によって認証情報をチップにフラッシュします。

次の手順に従って、認証情報を取得します。

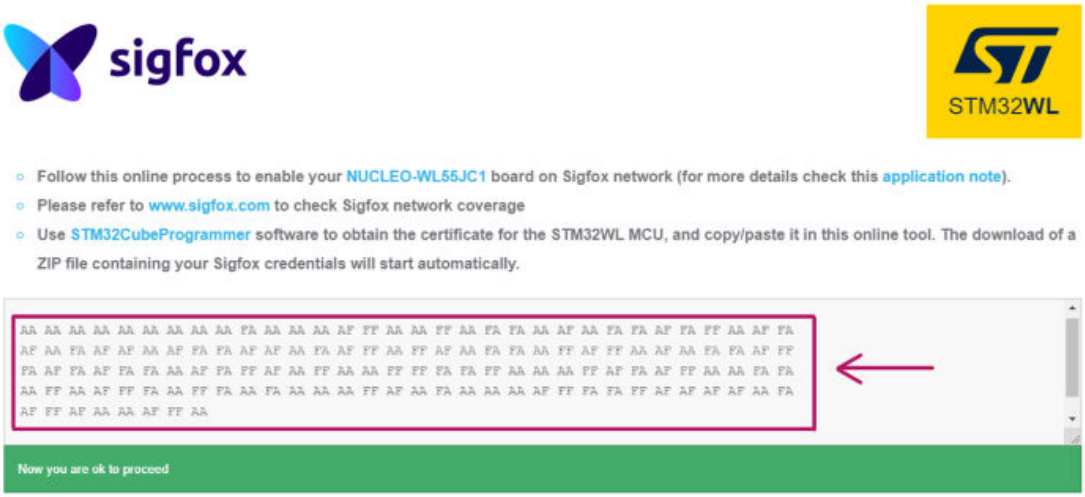
1. https://my.st.com/sfxp に移動し、my.st.com に登録して、特定のユーザアカウントを作成します（まだ無い場合）。

図 26. my.st.com にログイン



2. STM32CubeProgrammer で抽出した証明書をフォームに貼り付けます。

図 27. Sigfox 認証情報ページ

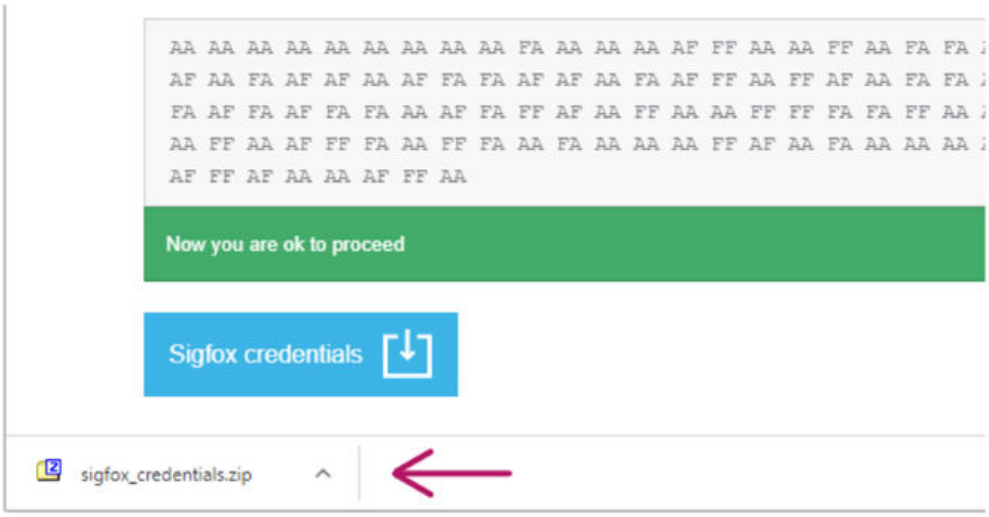


3. download ボタンをクリックします。

図 28. ダウンロードボタン

4. zip ファイルがユーザのコンピュータに自動的にダウンロードされます。

図 29. **Sigfox_credetentials のダウンロード**



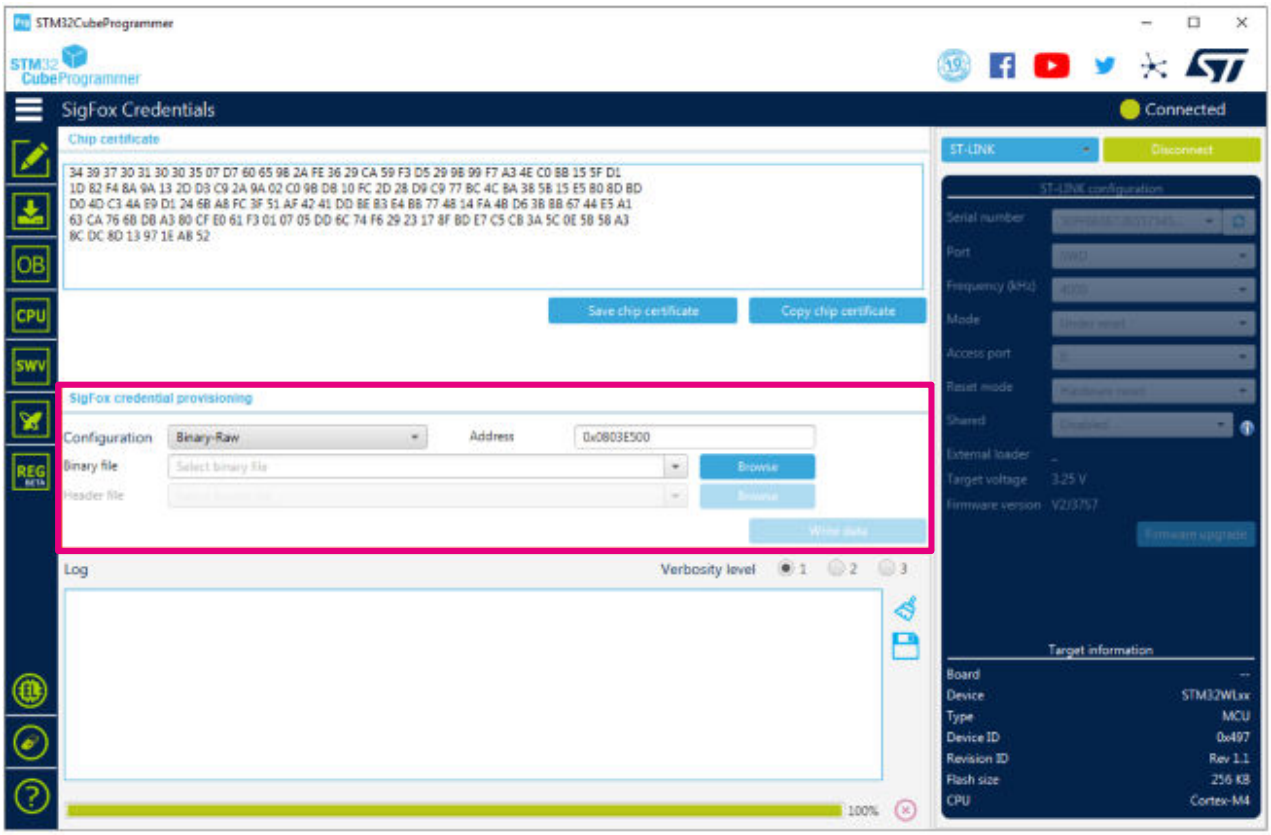
14.1.2 デバイスへの認証情報のロード

ST の Web インタフェースから Sigfox 認証情報を取得すると、それらを STM32CubeProgrammer の Sigfox credential provisioning 機能を使用して、STM32WL デバイスの 0x0803 E500 番地にロードできます。

- ケース 1：Binary-Raw：
  ST Web インタフェースによって返されたバイナリファイルを使用しなければなりません。このファイルは 48 バイトのサイズである必要があり、デフォルトのアドレス 0x0803 E500 に書き込まれます。
- ケース 2：Binary KMS：
  ST Web インタフェースによって返されたヘッダファイルを使用しなければなりません。これは、デフォルトのアドレス 0x0803 E500 に書き込まれます。

注 アドレス 0x0803 E500 は、リンカファイルによって配置されます（STM32CubeIde の `.ld` ファイル、IAR Embedded Workbench の `.icf`、または MDK_ARM の `.sct` を参照）。

図 30. STM32CubeProgrammer Sigfox パネル - 認証情報のフラッシュ



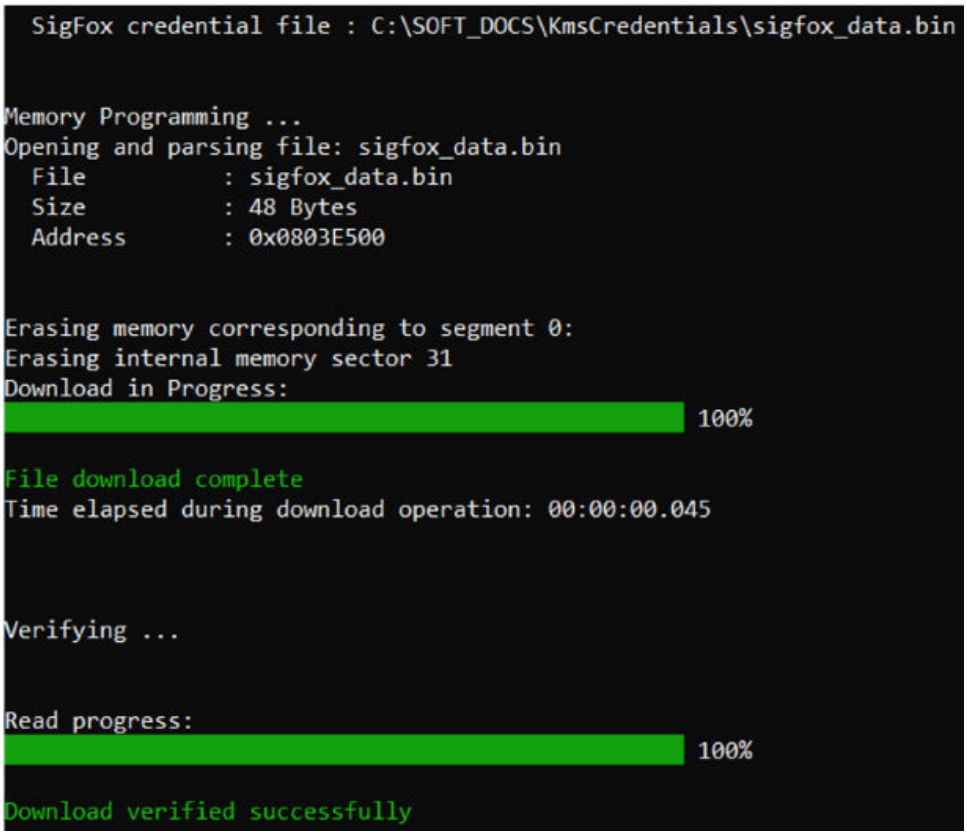
デバイスに認証情報を書き込むために使用するコマンドラインは、次のように定義します。

- コマンド：`-wsigfoxc`
- 説明：このコマンドにより、sigfox 認証情報をデフォルトのアドレス 0x0803 E500 に書き込むことができます。
- 構文：`-wsigfoxc <sigfox_credential_file_path> <address>`
  - `<address>` はオプションです（デフォルトは 0x0803 E500）。
  - `<sigfox_credential_file_path>` はバイナリファイル（例 1 を参照）またはヘッダファイル（下の例 2 を参照）です。

例 1

```
STM32_Programmer_CLI.exe -c port=swd -wsigfoxc " /local/user/sigfox_data.bin " 0x0803E500
```

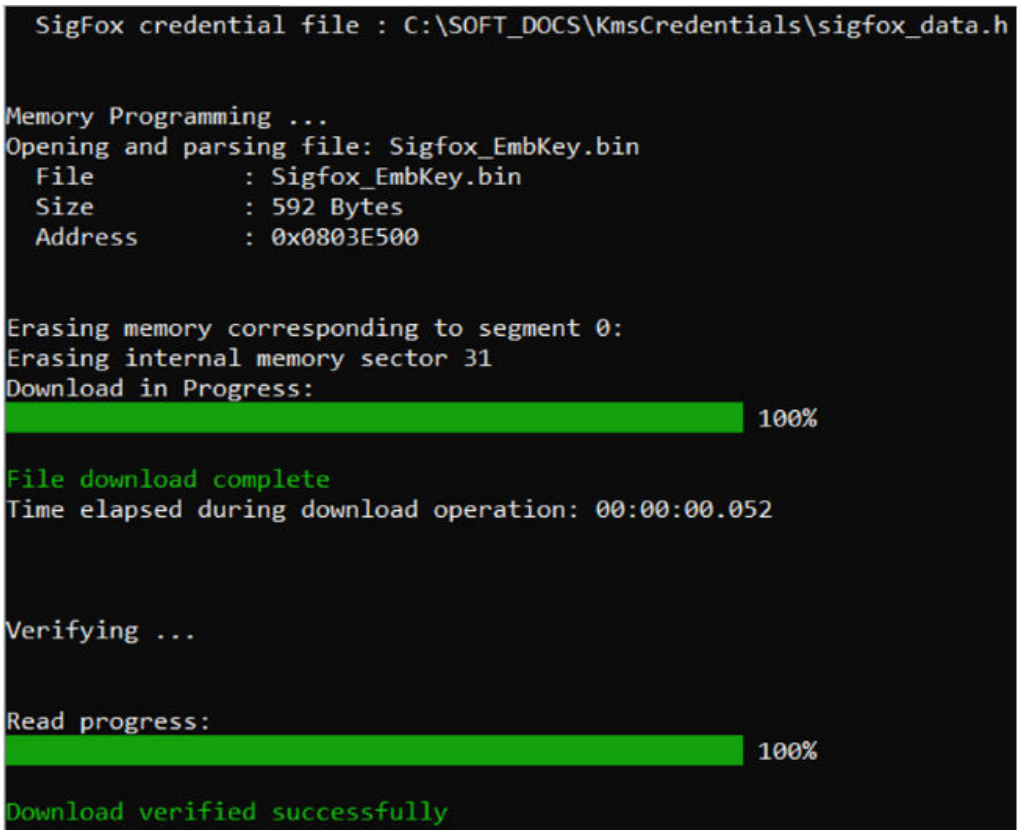
図 31. STM32CubeProgrammer Sigfox CLI - Raw 認証情報の書込み



例 2

```
STM32_Programmer_CLI.exe -c port=swd -wsigfoxc "/local/user/sigfox_data.h"
```

図 32. **STM32CubeProgrammer Sigfox CLI - KMS 認証情報の書込み**

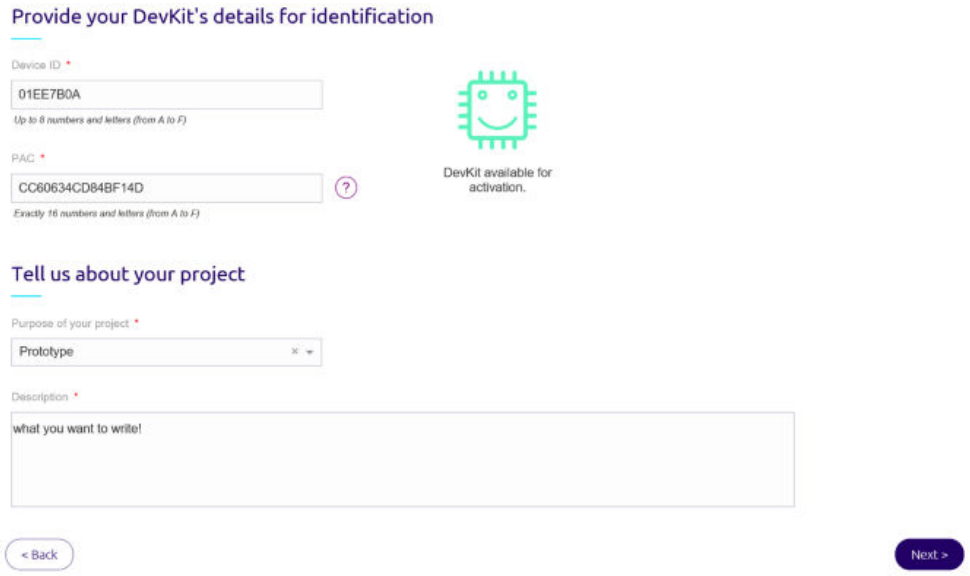


14.2 アクティベーション

次の手順に従ってください。

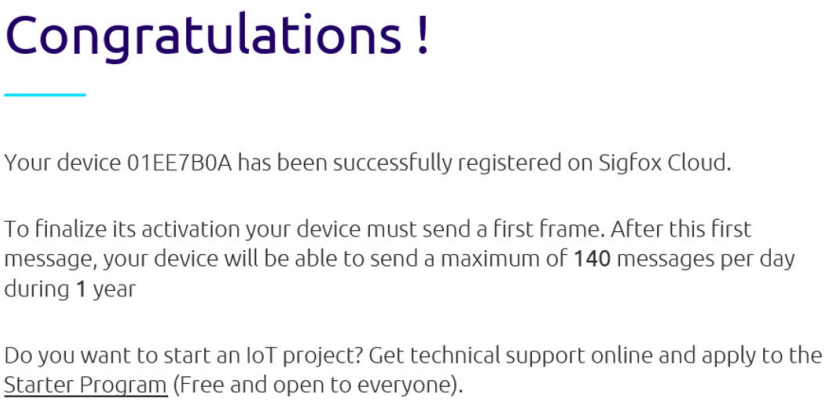
1. `AT$ID?<CR>` と `AT$PAC?<CR>` コマンドを使用して Sigfox ID と PAC を取得します。
2. https://buy.sigfox.com/activate/ に移動してログインします。
3. デバイス ID と PAC を有効化ページ（下の図を参照）にコピーして、Next（次へ）をクリックします。

図 33. デバイスの有効化（1/2）



4. この例では、ブラウザは次に示すページをロードします。

図 34. デバイスの有効化（2/2）



5. これで、デバイスは Sigfox ネットワーク上で 1 年間アクティベートされます（評価アクティベーション）。

14.3 メッセージを参照してください。

https://backend.sigfox.com/device/list に移動して、リストアップされているデバイスを確認します（DEVICE をクリックします）。例えばターミナルで `AT$SF` コマンドを使用して、データを送信できます。デバイスは Sigfox ネットワークにデータを送信し、メッセージはバックエンドで表示されます（デバイスの Id をクリックして MESSAGES タブに移動します）。

注意 Sigfox バックエンドは、デバイスのシーケンス番号と一致するシーケンス番号を記録します。このシーケンス番号は、新しいメッセージが送受信されるたびに両側でインクリメントされます。バックエンドは、デバイスのシーケンス番号がバックエンドのシーケンス番号以上の場合にのみメッセージを受け入れます。デバイスシーケンス番号は、フラッシュメモリ上のデバイスの EEPROM エミュレーションに格納されます。アプリケーションの開発段階では、例えば Cube プログラマなどを使用して EEPROM が消去されることがあります。この場合、デバイスシーケンス番号は 0 にリセットされ、バックエンドのシーケンス番号よりも小さくなります。メッセージは表示されませんが、アップリンクでは EVENTS タブが引き続き表示されます。再びメッセージを表示するには、Disengage sequence number（シーケンス番号の解放）を押します。これにより、バックエンドのシーケンス番号がリセットされ、バックエンドが新しいメッセージを受け入れられるようになります。

15 Sigfox アプリケーションを保護する方法

アプリケーション・ノート How to secure LoRaWAN and Sigfox with STM32CubeWL（STM32CubeWL で LoRaWAN と Sigfox を保護する方法）（AN5682）では、SBSFU フレームワークを使用してデュアルコア Sigfox アプリケーションを保護する方法が説明されています。

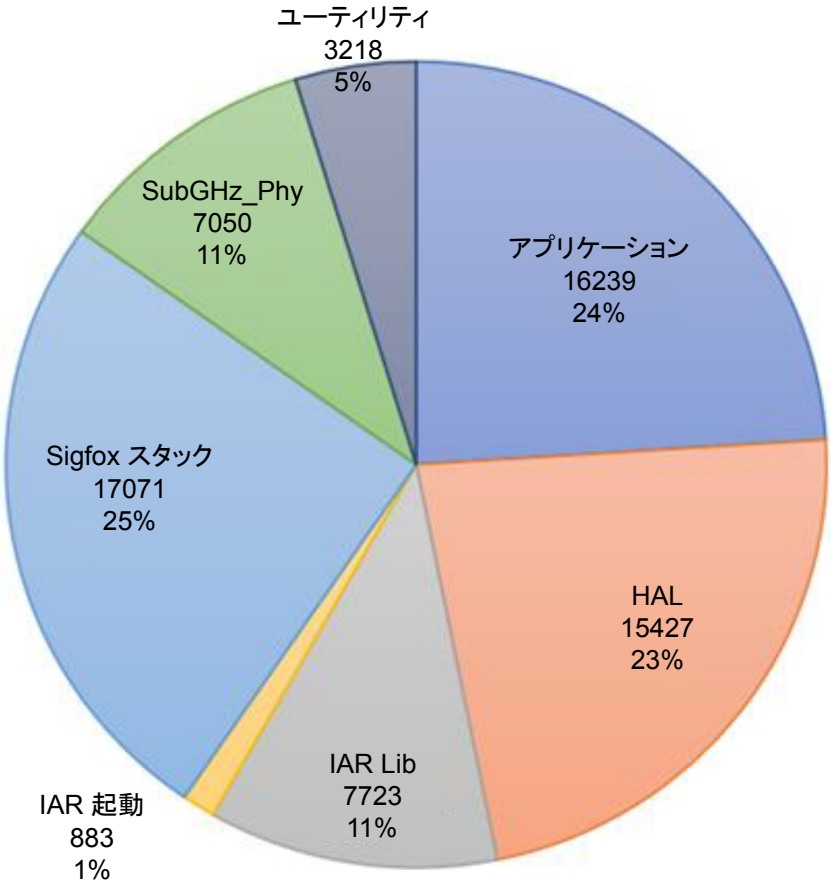
16 システム性能

16.1 メモリ・フットプリント

下の図の値は、IAR Embedded Workbench コンパイラ（EWARM コンパイラ 8.30.1）の次の設定を使用してマップファイルから抽出したものです。

- 最適化：サイズ・レベル 3 で最適化
- デバッグ・オプション：オフ

図 35. メモリフットプリント FLASH Sigfox_AT_Slave



16.2 リアルタイム制約

Monarch アルゴリズムの実行時には、リアルタイムの制約が適用されます。

16.3 消費電力

STM32WL Nucleo-64 ボード（NUCLEO-WL55JC）の消費電力を次の設定で測定しました。

- DEBUG：無効
- TRACE：無効

この条件では、STOP モードでの標準消費電流は 2 µA です。

改版履歴

表 29. 文書改版履歴

日付	版	変更内容
2020 年 5 月 20 日	1	初版発行
2020 年 11 月 17 日	2	更新: • セクション 10 メモリセクション • セクション 11.1 ファームウェアパッケージ • セクション 11.2 AT モデムアプリケーションの概要 • セクション 11.2.3 AT？- 使用可能なコマンド • セクション 11.2.9 ATS410 - 暗号化キー • セクション 11.2.10 ATS411 - ペイロードの暗号化 • セクション 11.2.22 AT$RC - 地域設定 • セクション 11.3 プッシュボタンアプリケーション • セクション 11.4 スタティックスイッチ • セクション 14.1 パーソナライゼーション 追加: • セクション 12 デュアルコア管理 • セクション 13 キー管理サービス（KMS）
2021 年 1 月 18 日	3	更新: • ドキュメント全体で Nucleo-73 を Nucleo-64 に訂正 • 表 3 の RC5 地域設定の RF パラメータ • セクション 5 Sigfox Stack の説明の概要 • セクション 14 パーソナライゼーションとアクティベーションの概要 • セクション 14.2 アクティベーションのステップ 5
2021 年 7 月 7 日	4	更新: • 表 2.頭字語 • セクション 4 BSP STM32WL Nucleo ボードの概要 • セクション 6.1 Sigfox Core ライブラリの概要の最後 • セクション 6.2 Sigfox Addon RF プロトコルライブラリの最後 • セクション 6.3 Cmac ライブラリの最後 • 表 16.Radio_s 構造のコールバック • セクション 9.1 シーケンサ • セクション 9.3 低消費電力機能 • セクション 11.4 スタティックスイッチ • セクション 14.1.2 デバイスへの認証情報のロード • 図 36.メモリフットプリント FLASH Sigfox_AT_Slave セクション 15 Sigfox アプリケーションを保護する方法を追加 削除: • セクション 4 BSP STM32WL Nucleo ボードの「RF ウェイクアップ時間」 • セクション 2.3 Rx/Tx 無線タイムダイアグラムの注
2022 年 2 月 01 日	5	セクション 4.5 最大 Tx RF 出力パワーを追加。 更新: • 表 1. 頭字語 および用語 • セクション 5.1 Sigfox 認証 • 図 7. 受信 MSC • 表 20. シーケンサ API • セクション 14.1 パーソナライゼーション の注記 • セクション 9.1 シーケンサ

目次

表一覧

図一覧

重要なお知らせ（よくお読み下さい）

STMicroelectronics NV およびその子会社（以下、ST）は、ST 製品及び本書の内容をいつでも予告なく変更、修正、改善、改定及び改良する権利を留保します。購入される方は、発注前に ST 製品に関する最新の関連情報を必ず入手してください。ST 製品は、注文請書発行時点で有効な ST の販売条件に従って販売されます。

ST 製品の選択並びに使用については購入される方が全ての責任を負うものとします。購入される方の製品上の操作や設計に関して ST は一切の責任を負いません。

明示又は黙示を問わず、ST は本書においていかなる知的財産権の実施権も許諾致しません。

本書で説明されている情報とは異なる条件で ST 製品が再販された場合、その製品について ST が与えたいかなる保証も無効となります。

ST および ST ロゴは STMicroelectronics の商標です。ST の登録商標については ST ウェブサイトをご覧ください。www.st.com/trademarks その他の製品またはサービスの名称は、それぞれの所有者に帰属します。

本書の情報は本書の以前のバージョンで提供された全ての情報に優先し、これに代わるものです。