# LoRaWAN スタック

LoRaWAN スタック

バージョン 1.0

LoRa スタックのプレゼンテーションにようこそ

# LoRaWAN

- STM32WL の開発で使用する LoRaWAN スタックには LoRaWAN L2 V1.0.3 との互換性がある
- Semtech 社によって GitHub に投稿されている LoRaMac-node に基づき、L2 1.0.4 の実装を伴うプレリリースが 2020 年 11 月 24 日に v4.5.0_rc1 として *https://github.com/Lora-net/LoRaMac-node/releases* に投稿済みである
- サポート対象の機能
  - クラス A（ユニキャスト）
  - クラス C（ユニキャストとマルチキャスト）
  - クラス B（ユニキャストとマルチキャスト）
- 2020 年 10 月より、LoRaWAN L2 V1.0.4 の仕様が *https://lora-alliance.org/resource-hub* から入手可能になっている
- LoRaWAN L2 V1.0.3 と V1.0.4
  - V1.0.4 は LoRaWAN V1.0.x の最終仕様
  - LoRa Alliance Technical Committee は V1.1 から CR の一部を V1.0.4 にバックポートすることを決定
  - V1.0.4 では品質面を大幅に修正
  - 機能面ではこれら 2 つのバージョン間に違いはない

life.augmented

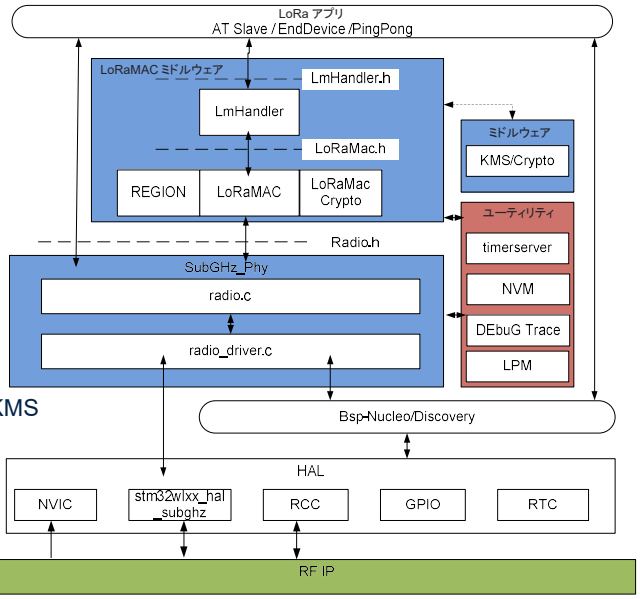
2

STM32WL に組み込まれた LoRaWAN には LoRaWAN L2 V1.0.3 との互換性があります。
これは、Semtech 社が GitHub に投稿した LoRaMac-node（https://github.com/Lora-net/LoRaMac-node/）に基づいています。
サポートされているクラスは、ユニキャスト向けのクラス A、ユニキャストまたはマルチキャストでビーコンを利用した同期送信が可能なクラス B、ユニキャストまたはマルチキャストでの連続送信が可能なクラス C です。
GitHub の LoRaMac-node では、2020 年末までに LoRaWAN L2 V1.0.4 仕様が用意される計画になっていて、2020 年 11 月 24 日に v4.5.0_rc1 としてプレリリースが投稿されています。
LoRaWAN L2 V1.0.3 と V1.0.4 との仕様上の相違点は次のとおりです。

- V1.0.4 は V1.0.x 仕様の最後のバージョンです。
- 品質面の大幅な修正を目的として、V1.1 の CR がいくつか V1.0.4 にバックポートされています。
- これら 2 つのバージョンに機能上の相違点はありません。

# 概要

- STM32WL FW のアプリケーション:
  - End_Node アプリケーション
  - AT スレーブアプリケーション
  - ピンポン･アプリケーション
- ミドルウェア
  - Mac レイヤを含む LoRaWAN
  - 物理レイヤを含む SubGHz_Phy
    - 注:i-cube-lrwan は必要に応じて更新される
  - デュアルコアで LORAWAN_KMS スイッチが ON の場合
    - CM0PLUS¥LoRaWAN¥Target¥lorawan_conf.h の LORAWAN_KMS
  - STM32WB と共通の汎用ユーティリティ
- BSP
  - RF スイッチとボード設定を駆動




life.augmented



STM32WL のファームウェアは、3 種類のアプリケーションとして AT_Slave、End_Node、PingPong を収めています。

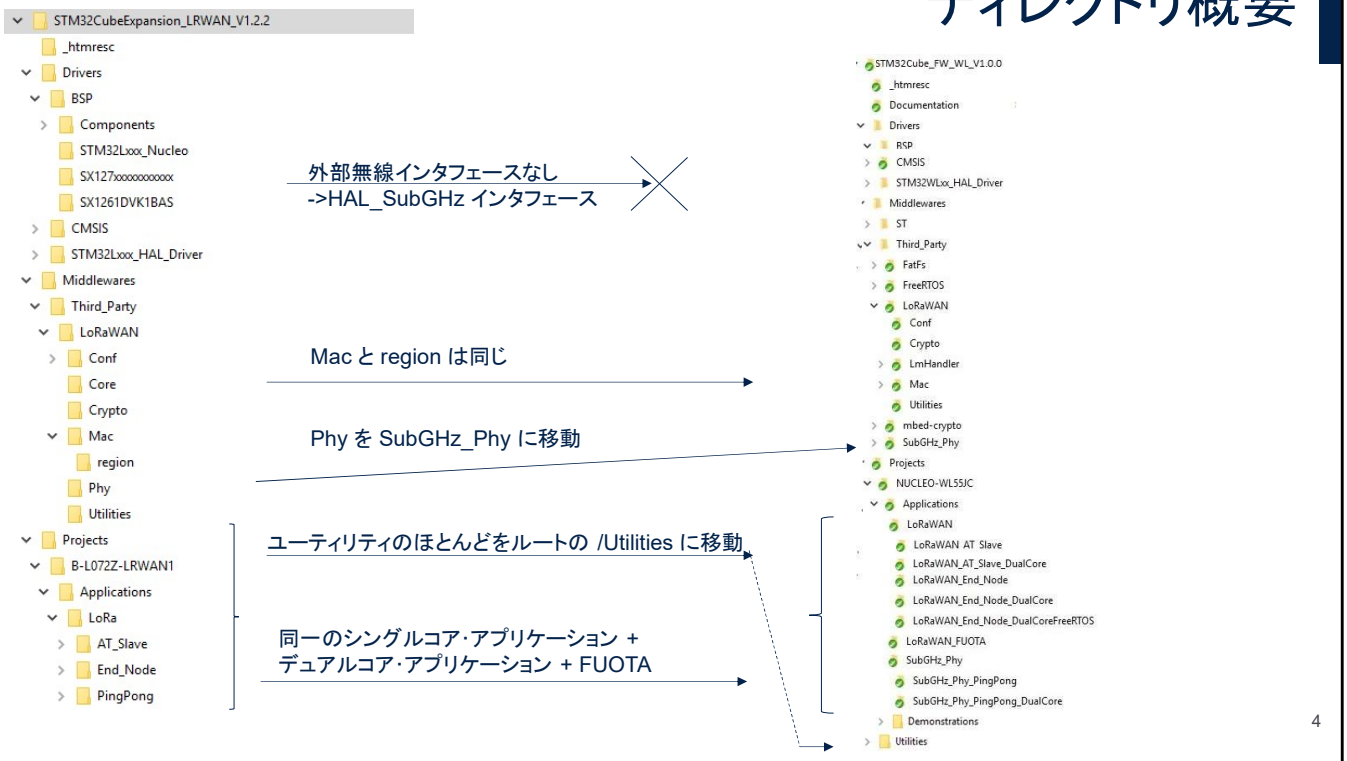
この図には、ファームウェアのアーキテクチャとして、いくつかのアプリケーション層の位置と相互関係を示しています。

各アプリケーションで実行されているミドルウェアは次のとおりです。

- LoRaWAN:メディア･アクセス･コマンド層を備えています。
- SubGHz_Phy:SubGHz 物理層を備えています。
- デュアルコアでの鍵管理ストレージ:lorawan_conf.h ファイルで LORAWAN_KMS スイッチが ON の場合

ボード･サポート･パッケージ(BSP)は、RF スイッチ、TCXO、DCDC の各設定を駆動するように設計されています。STM32CubeMX の制限に起因して、このファームウェアでは BSP の各種ファイルが使用されない点に注意してください。ただし、無線関連項目の radio_board_if.c/.h および LED とプッシュボタンの board_resources.c/.h は例外です。

# ディレクトリ概要




このスライドでは、右側に STM32WL FW パッケージのフォルダ構造を示し、比較のために左側には Cube Expansion のフォルダ構造を示しています。
STM32WL パッケージでは、無線モジュールがチップに内蔵されているので BSP には外付けの無線モジュールがなく、HAL_SUBGHZ インタフェースから無線モジュールにアクセスできることがわかります。
LoRaWAN MAC の地域に変更はありませんが、物理層が SubGHz_Phy に移動しています。
STM32WL パッケージには、CubeExpansion と同じアプリケーションが用意されているほか、そのデュアルコア･バージョン、FreeRTOS を使用した EndNode、FUOTA アプリケーションが追加されています。

# アプリケーション

- 各 LoRaWAN アプリケーションの特徴:
  - マイクロコントローラ固有のファイルはすべてコアに配置
    - CubeMx で生成されたペリフェラル:adc/rtc/dma/usart/
    - すべてのペリフェラル・インタフェース
  - LoRaWAN ディレクトリの内容
    - App:各種アプリケーション・ファイル
    - Target:ミドルウェア設定とボード設定
  - アプリケーションごとに用意された 3 つのツールチェーンとコンパイラ:
    - ARM(EWARM)ツールチェーン向けの IAR Embedded Workbench
    - RealView Microcontroller Development Kit(MDK-ARM)ツールチェーン
    - STM32CubeIDE
- パッケージのその他の特徴
  - LoRaWAN で動作する FUOTA アプリケーション
  - デモンストレーション: LocalNetwork


life.augmented



このファームウェア・パッケージはいくつかの LoRaWAN アプリケーションで構成されていて、たとえば AT_Slave と End_Node があります。
各 LoRaWAN アプリケーションは次のような要素で構成されています。

- CubeMx で生成されたすべてのペリフェラルとそのインタフェースを収めて Core ディレクトリに置かれた、マイクロコントローラ固有の各種ファイル
- LoRaWAN/App ディレクトリと LoRaWAN/Target ディレクトリに置かれた、アプリケーション固有のファイルおよびミドルウェアとボードの設定
- 3 つのツールチェーン:IAR(EWARM)、MDK_ARM、STM32CubeIDE

このファームウェア・パッケージは以下の機能も用意しています。

- LoRaWAN で動作する FUOTA アプリケーション
- LocalNetwork の例などのデモンストレーション

# LoRaWAN ミドルウェア

- インタフェースを LoRaMac.h から LmHandler.h に更新
- LmHandler.c/h は、すべてのアプリケーションに共通するアプリケーション開発を容易にするためのラッパー
  - LmHandler は Semtech 社の同一名ファイルに基づいているが、ATスレーブ·モデムが有効になるように更新済み

- 地域
  - RegionAS923
  - RegionAU915
  - RegionCN470
  - RegionCN779
  - RegionEU868
  - RegionEU433
  - RegionKR920
  - RegionRU864
  - RegionUS915
  - RegionIN865

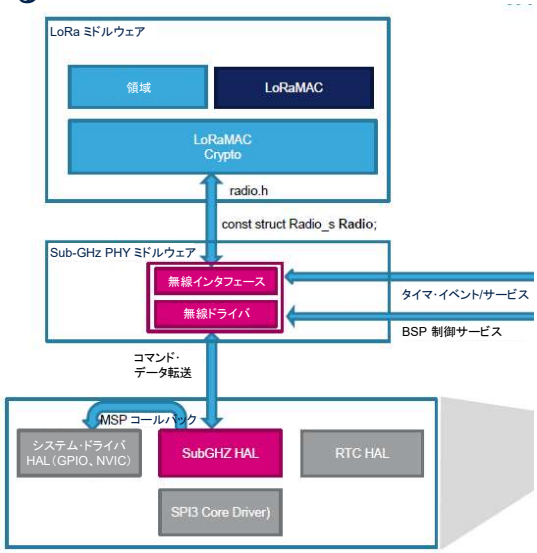
life.augmented



以前は名前を LoRaMac としていた LoRaWAN ミドルウェア·インタフェースが LmHandler に更新されています。
LmHandler は、LoRaMac サービスにアクセスするための API 群を実装する LoRaMac 層インタフェース·ファイルです。
当初は Semtech 社による実装に基づいていましたが、AT スレーブ·モデムを有効にするように ST によって更新されています。
このスライドに挙げたように、いくつかの地域とそれに対応する帯域を各プロジェクトで選択できます。
*注：現在の LoRaWAN スタックには、LoRaWAN 地域パラメータ(RP)v1.0.3 との互換性があります。*

# Sub-GHz_Phy ミドルウェア（1/2）

- この図は、共通の Sub-GHz 物理層と低レベル·ドライバを備えた LoRa ミドルウェアの相互作用モデルを示している




life.augmented

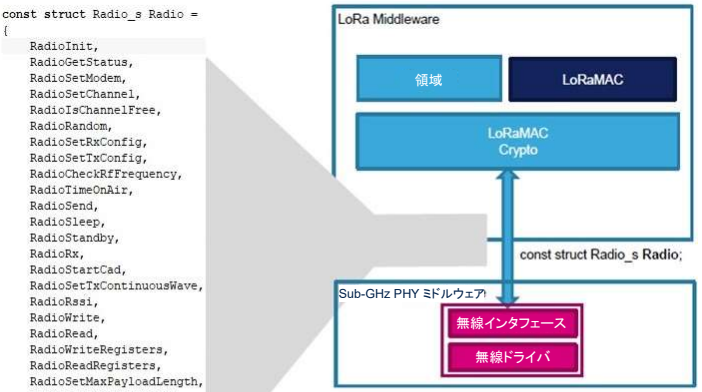
7

ここに挙げた相互作用モデルは、複数のソフトウェア層が、SubGHz_Phy ミドルウェアを通じてペリフェラル HAL と SubGHz HAL から LoRaWAN ミドルウェアと通信する様子を示しています。
SubGHz_Phy ミドルウェアの無線インタフェースは、SubGHz HAL から実行されるコマンドで設定されるほか、タイマによっても設定されます。シーケンサによって無線タスクがトリガされます。

# Sub-GHz_Phy ミドルウェア（2/2）

- LoRaWAN ミドルウェアは、
  Radio_s 構造体を通じて無線と相互作用

```
const struct Radio_s Radio =
{
    RadioInit,
    RadioGetStatus,
    RadioSetModem,
    RadioSetChannel,
    RadioIsChannelFree,
    RadioRandom,
    RadioSetRxConfig,
    RadioSetTxConfig,
    RadioCheckRfFrequency,
    RadioTimeOnAir,
    RadioSend,
    RadioSleep,
    RadioStandby,
    RadioRx,
    RadioStartCad,
    RadioSetTxContinuousWave,
    RadioRssi,
    RadioWrite,
    RadioRead,
    RadioWriteRegisters,
    RadioReadRegisters,
    RadioSetMaxPayloadLength,
    RadioSetPublicNetwork,
    RadioGetWakeupTime,
    RadioIrqProcess,
    RadioRxBoosted,
    RadioSetRxDutyCycle,
    RadioTxPrbs,
    RadioTxCw,
    RadioSetRxGenericConfig,
    RadioSetTxGenericConfig,
};
```




life.augmented

8

ここでは、この相互作用モデルの上位層に注目します。
LoRaWAN ミドルウェアは、Radio_s 構造体を通じて
SubGHz 物理ミドルウェアの無線インタフェースにコマンドを
送信します。

# 無線インタフェース

- radio.h
  - 引き続き、LoRaMac 層またはアプリケーションのインタフェースとして機能
  - i-cube-lrwan と同じ radio.h
- 無線は 2 つのレベルに分割されている
  - 無線上位レベル
    - radio.c（sx126x を駆動する radio.c と同等）
  - 無線下位レベルの機能
    - radio_driver.c は sx126x.c と同等
    - 用意されているインタフェース
      - SubGHz lp サービス向けの stm32wlxx_hal_subg.h
      - RF BSP サービス向けの stm32wlxx_nucleo.h
- 利点
  - radio.c と radio_driver.c はそのレガシーと同等（Semtech 社による変更の反映が容易）



life.augmented



ここでは SubGHz 物理ミドルウェアを詳しく検討します。
LoRaMac 層（アプリケーション）向け無線コマンドのインタフェースは radio.h です。このファイルは、i-cube-lrwan の場合と同様に、前のスライドに示した Radio_s 構造体を記述しています。
STM32WL では、無線が次の 2 種類のレベルに分割されています。

- radio.c：無線の高レベル部分であり、Semtech 社の sx126x ドライバにある radio.c と同等です。
- radio_driver.c：無線の低レベル部分であり、Semtech 社の sx126x.c と同等です。HAL_SubGHz サービスと BSP nucleo サービスをインタフェースとして備えています。

Semtech 社による変更を容易に反映できるように、実装がこのように分離されています。

# 無線インタフェースを使用した送信の例

```
// 無線の初期化
RadioEvents.TxDone = OnTxDone;
RadioEvents.RxDone = OnRxDone;
RadioEvents.TxTimeout = OnTxTimeout;
RadioEvents.RxTimeout = OnRxTimeout;
RadioEvents.RxError = OnRxError;
Radio.Init( &RadioEvents );
// LoRa モードで無線を Tx 設定
Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
                   LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                   LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                   true, 0, 0, LORA_IQ_INVERSION_ON, 3000 );
// 無線周波数の設定
Radio.SetChannel( RF_FREQUENCY );
// 無線によるバッファの送信
Radio.Send( Buffer, BufferSize );
```

life.augmented

10

これは、送信（TX）を設定する無線インタフェースの例です。

- まず、RadioEvents をそれぞれに対応するコールバックで初期化し、無線の初期化を呼び出します。
- つづいて、無線を設定します。この例では、LoRa モードでの送信（TX）に設定します。
- 周波数を設定します。
- 最後に、データを送信します。

# 無線インタフェースを使用した受信の例

```
// 無線の初期化
RadioEvents.TxDone = OnTxDone;
RadioEvents.RxDone = OnRxDone;
RadioEvents.TxTimeout = OnTxTimeout;
RadioEvents.RxTimeout = OnRxTimeout;
RadioEvents.RxError = OnRxError;
Radio.Init( &RadioEvents );
// FSK モードの受信設定
Radio.SetRxConfig( MODEM_FSK, FSK_BANDWIDTH, FSK_DATARATE,
                   0, FSK_AFC_BANDWIDTH, FSK_PREAMBLE_LENGTH,
                   0, FSK_FIX_LENGTH_PAYLOAD_ON, 0, true,
                   0, 0,false, true );
// 無線周波数の設定
Radio.SetChannel( RF_FREQUENCY );
// 無線を受信モードに設定
Radio.Rx( RX_TIMEOUT_VALUE );
```

life.augmented

11

これは、受信（RX）を設定した無線インタフェースの例です。
まず、RadioEvents をそれぞれに対応するコールバックで初期化し、無線の初期化を呼び出します。
つづいて、無線を設定します。この例では、FSK モードでの受信（RX）に設定します。
周波数を設定します。
最後に、無線による受信を有効にします。Rx1 ウィンドウまたは Rx2 ウィンドウが開きます。

# STM32WLxx の HAL_SubGHz

- 無線にアクセスするための HAL API
  - 内部 SPI
  - RCC と NVIC は HAL_SUBGHZ_MspInit() 関数で設定
- SubGHz コマンドは以下の定義済みエニュメレーション構造体を使用
  - SUBGHZ_RadioGetCmd_t：コマンド構造を取得
  - SUBGHZ_RadioSetCmd_t：コマンド構造を設定
- API の一覧
  - HAL_StatusTypeDef HAL_SUBGHZ_Init(SUBGHZ_HandleTypeDef *hsubghz);
  - HAL_StatusTypeDef HAL_SUBGHZ_DeInit(SUBGHZ_HandleTypeDef *hsubghz);
  - void HAL_SUBGHZ_ExecSetCmd(SUBGHZ_HandleTypeDef *hsubghz, SUBGHZ_RadioSetCmd_t command, uint8_t *buffer, uint16_t size);
  - void HAL_SUBGHZ_ExecGetCmd(SUBGHZ_HandleTypeDef *hsubghz, SUBGHZ_RadioGetCmd_t command, uint8_t *buffer, uint16_t size);
  - void HAL_SUBGHZ_WriteRegisters(SUBGHZ_HandleTypeDef *hsubghz, uint16_t address, uint8_t *buffer, uint16_t size);
  - void HAL_SUBGHZ_ReadRegisters(SUBGHZ_HandleTypeDef *hsubghz, uint16_t address, uint8_t *buffer, uint16_t size);
  - void HAL_SUBGHZ_WriteBuffer(SUBGHZ_HandleTypeDef *hsubghz, uint8_t offset, uint8_t *buffer, uint16_t size);
  - void HAL_SUBGHZ_ReadBuffer(SUBGHZ_HandleTypeDef *hsubghz, uint8_t offset, uint8_t *buffer, uint16_t size);

life.augmented

12

HAL SubGHz 層は、内部 SPI リンクを通じて無線にアクセスします。NVIC と RCC は MSP Init で設定されます。
SubGHz コマンドでは、SUBGHZ_RadioGetCmd_t 構造体と SUBGHZ_RadioSetCmd_t 構造体を使用します。
無線にアクセスする API の一覧を登録すると、ここに set コマンドまたは get コマンドが表示されます。

# STM32WLxx の HAL_SubGHz: radioSetCmd

- void HAL_SUBGHZ_ExecSetCmd(SUBGHZ_HandleTypeDef *hsubghz, SUBGHZ_RadioSetCmd_t command, uint8_t *buffer, uint16_t size);
- 例 :

**Set_Sleep() コマンド**

| 0 | 1 |
|---|---|
| OP コード | SleepCfg |
| w | w |

バイト 0　ビット 7:0　**OP コード**:0x84

バイト 1　ビット 7:3　予約済みであり、リセット値に保持する必要があります。

ビット 2　**SleepCfg_Start**:Sub-GHz 無線起動選択
0:SLEEP モードを終了したときにコールド スタート、設定レジスタをリセット
1:SLEEP モードを終了したときにウォーム スタート、設定レジスタを保持

*注:SLEEP モードへ遷移する前にアクティブになっていたモードの設定のみを保持します。他のモードの設定は失われるので、SLEEP モードを終了するときに再設定する必要があります。*

ビット 1　予約済みであり、リセット値に保持する必要があります。

ビット 0　**SleepCfg_RTCEn**:Sub-GHz 無線の RTC ウェイクアップが有効
0:Sub-GHz 無線の RTC ウェイクアップが無効
1:Sub-GHz 無線の RTC ウェイクアップが有効

```
void SUBGRF_SetSleep( SleepParams_t sleepConfig )
{
   BSP_RF_SW_Reset(); /* SW でアンテナをオフに切り替え */

   HAL_SUBGHZ_ExecSetCmd( RADIO_SET_SLEEP, &sleepConfig.Value, 1 );
OperatingMode = MODE_SLEEP;
}
```

```
typedef enum SUBGHZ_RadioSetCmd_e
{
 RADIO_SET_SLEEP                    = 0x84,
 RADIO_SET_STANDBY                  = 0x80,
 RADIO_SET_FS                       = 0xC1,
 RADIO_SET_TX                       = 0x83,
 RADIO_SET_RX                       = 0x82,
 RADIO_SET_RXDUTYCYCLE              = 0x94,
 RADIO_SET_CAD                      = 0xC5,
 RADIO_SET_TXCONTINUOUSWAVE         = 0xD1,
 RADIO_SET_TXCONTINUOUSPREAMBLE     = 0xD2,
 RADIO_SET_PACKETTYPE               = 0x8A,
 RADIO_SET_RFFREQUENCY              = 0x86,
 RADIO_SET_TXPARAMS                 = 0x8E,
 RADIO_SET_PACONFIG                 = 0x95,
 RADIO_SET_CADPARAMS                = 0x88,
 RADIO_SET_BUFFERBASEADDRESS        = 0x8F,
 RADIO_SET_MODULATIONPARAMS         = 0x8B,
 RADIO_SET_PACKETPARAMS             = 0x8C,
 RADIO_RESET_STATS                  = 0x00,
 RADIO_CFG_DIOIRQ                   = 0x08,
 RADIO_CLR_IRQSTATUS                = 0x02,
 RADIO_CALIBRATE                    = 0x89,
 RADIO_CALIBRATEIMAGE               = 0x98,
 RADIO_SET_REGULATORMODE            = 0x96,
 RADIO_SET_TCXOMODE                 = 0x97,
 RADIO_SET_TXFALLBACKMODE           = 0x93,
 RADIO_SET_RFSWITCHMODE             = 0x9D,
 RADIO_SET_STOPRXTIMERONPREAMBLE    = 0x9F,
 RADIO_SET_LORASYMBTIMEOUT          = 0xA0,
} SUBGHZ_RadioSetCmd_t;
```

life.augmented

13

HAL_SubGHz でコマンドを無線に設定する関数は HAL_SUBGHZ_ExecSetCmd です。
この関数で最初のパラメータは、コマンドの OP コードを表します。

- このスライドの右側には、コマンドの OP コードの一覧を示しています。
- 左側には、SUBGRF_SetSleep() 関数の内部で無線をスリープ・モードに設定する HAL_SUBGHZ_ExecSetCmd() の呼び出しの例を示しています。

# STM32WLxx の HAL_SubGHz: radioGetCmd

- void HAL_SUBGHZ_ExecGetCmd(SUBGHZ_HandleTypeDef *hsubghz SUBGHZ_RadioGetCmd_t command, uint8_t *buffer, uint16_t size);
- 例:

```
RadioStatus_t SUBGRF_GetStatus( void )
{
  uint8_t stat = 0;
  RadioStatus_t status;
  HAL_SUBGHZ_ExecGetCmd( RADIO_GET_STATUS,
                         ( uint8_t * )&stat, 1 );
  status.Value = stat;
  return status;
}

int8_t SUBGRF_GetRssiInst( void )
{
  uint8_t buf[1];
  int8_t rssi = 0;

  HAL_SUBGHZ_ExecGetCmd( RADIO_GET_RSSIINST, buf, 1 );
  rssi = -buf[0] >> 1;
  return rssi;
}
```

**Get_Status() コマンド**

| 0 | 1 |
|---|---|
| OP コード | ステータス [7:0] |
| w | r |

バイト 0 ビット 7:0 **OP コード**:0xC0
バイト 1 ビット7 予約済みであり、値を保持する必要があります。
ビット 6:4 **ステータス・モード [2:0]** Sub-GHz 無線動作モード
ビット 3:1 **ステータス CmdStatus [2:0]**:コマンド・ステータス
ビット 0 予約済みであり、リセット値に保持する必要があります。

```
typedef enum SUBGHZ_RadioGetCmd_e
{
  RADIO_GET_STATUS          = 0xC0,
  RADIO_GET_PACKETTYPE      = 0x11,
  RADIO_GET_RXBUFFERSTATUS  = 0x13,
  RADIO_GET_PACKETSTATUS    = 0x14,
  RADIO_GET_RSSIINST        = 0x15,
  RADIO_GET_STATS           = 0x10,
  RADIO_GET_IRQSTATUS       = 0x12,
  RADIO_GET_ERROR           = 0x17,
} SUBGHZ_RadioGetCmd_t;
```

**Get_RssiInst() コマンド**

| 0 | 1 | 2 |
|---|---|---|
| OP コード | ステータス [7:0] | RssiInst [7:0] |
| w | r | r |

バイト 0 ビット 7:0 **OP コード**:0x15
バイト 1 ビット 7:0 **ステータス [7:0]**:*Get_Status() コマンド*を参照
バイト 2 ビット 7:0 **RssiInst [7:0]**:受信時の瞬時 RSSI レベル
信号強度 = -RssiInst/2(dBm)

life.augmented

14

HAL_SubGHz で無線から値を取得する関数は HAL_SUBGHZ_ExecGetCmd です。
この関数で最初のパラメータは、コマンドの OP コードを表します。

- このスライドの右側には、get コマンドの OP コードの一覧を示しています。
- 左側には、無線のステータスと rssi 値を取得する HAL_SUBGHZ_ExecGetCmd の呼び出しの例を示しています。

# 基本レジスタ

- 次の API を使用してレジスタにアクセス可能
  - void HAL_SUBGHZ_WriteRegisters(SUBGHZ_HandleTypeDef *hsubghz,uint16_t address, uint8_t *buffer, uint16_t size);
  - void HAL_SUBGHZ_ReadRegisters(SUBGHZ_HandleTypeDef *hsubghz, uint16_t address, uint8_t *buffer, uint16_t size);
- 無線インタフェースでこれらのレジスタを制御
- RFIP をカスタム利用するための高機能レジスタが存在

| 名前 | 長さ | 説明 |
|---|---|---|
| SUBGHZ_GBSYNCR (REG_BIT_SYNC) | 1 | LoRa 以外のパケットタイプを使用する場合は 0x00 にクリアする必要あり |
| SUBGHZ_GPKTCTL1AR (REG_LR_WHITSEEDBASEADDR_MSB) | 1 | 連続パケット生成モードを設定 |
| SUBGHZ_GWHITEINIRL (REG_LR_WHITSEEDBASEADDR_LSB) | 1 | GFSK パケットのホワイトニング·シード **WHITEINI[7:0]** を設定 |
| SUBGHZ_GCRCINIR (REG_LR_CRCSEEDBASEADDR) | 2 | GFSK パケットの CRC シードを設定 |
| SUBGHZ_GCRCPOLR (REG_LR_CRCPOLYBASEADDR) | 2 | GFSK パケットの CRC 多項式を設定 |
| SUBGHZ_GSYNCR (REG_LR_SYNCWORDBASEADDRESS) | 8 | GFSK の同期ワードを設定 |
| SUBGHZ_LSYNCR (REG_LR_SYNCWORD) | 2 | LoRa 同期ワード（パブリックまたはプライベート） |
| SUBGHZ_RNGR (RANDOM_NUMBER_GENERATORBASEADDR) | 4 | 乱数発生器読込み値 |
| SUBGHZ_RXGAINCR (REG_RX_GAIN) | 1 | レシーバ·ゲイン制御レジスタ。通常ゲインまたはブースト·ゲインで使用。 |
| SUBGHZ_PAOCPR (REG_OCP) | 1 | 過電流保護の最大電流を設定 |
| SUBGHZ_HSEINTRIMR (REG_XTA_TRIM) | 1 | Xtal オシレータの内部コンデンサのトリミング値（xtb も存在） |
| SUBGHZ_HSEOUTTRIMR (REG_XTB_TRIM) | 1 | Xtal オシレータの外部コンデンサのトリミング値 |
| SUBGHZ_SMPSC0R (REG_DCC_BUCK_CTRL) | 1 | SMPS クロック検出 |
| SUBGHZ_PCR (REG_DCC_BUCK_SEL_OUT_DRIVE) | 1 | 電源電流の制限 |

life.augmented

15

HAL Sub-GHz の HAL_SUBGHZ_WriteRegisters () 関数と HAL_SUBGHZ_ReadRegisters() 関数を使用して、Sub-GHz の無線ペリフェラル·レジスタにアクセスできます。
Sub-GHz の無線ペリフェラル·レジスタの一覧を、この図に示しています。
STM32WL のリファレンス·マニュアルに、これらのレジスタすべての詳しい説明があります。

# BSP RF サービス

- Nucleo BSP には以下の無線 API が付属
  - ボード設定
    - BSP_RADIO_Init(void) と BSP_RADIO_DeInit(void)
    - BSP_RADIO_GetWakeUpTime(void)
      - 戻り値はウェイクアップ時間（ms）
    - BSP_RADIO_GetTxConfig(void)
      - ボードのスイッチ設定を返す。Tx Low Power と Tx High Power の一方または両方が配線されている場合のみ
    - BSP_RADIO_IsTCXO
      - TCXO 設定：ボード上に存在する場合と存在しない場合がある
    - BSP_RADIO_IsDCDC
      - DCDC 設定：ボード上に存在する場合と存在しない場合がある
  - RF スイッチ制御
    - BSP_RADIO_ConfigRFSwitch(BSP_RADIO_Switch_TypeDef Config)
      - オフ、Rx モード、Tx High Power モード、または Tx Low Power モードに制御可能

life.augmented

16

STM32WL Nucleo ボードのボード・サポート・パッケージ（BSP）では、以下の各 API を使用して無線ボードに固有なすべての設定を扱っています。

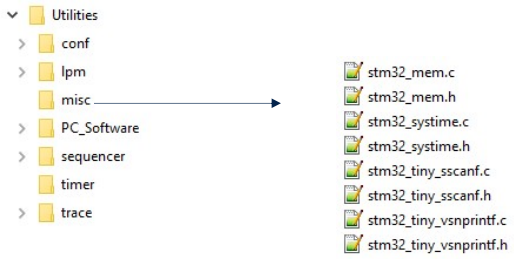
- ウェイクアップ時間
- スイッチ設定
- TCXO
- DCDC
- 無線スイッチ・モード

これらの関数は、Drivers¥BSP¥STM32WLxx_Nucleo¥stm32wlxx_nucleo_radio.c で定義されています。

*注：現時点の実装では、STM32CubeMX の制限に起因して、BSP ファイルの stm32wlxx_nucleo_radio.c と stm32wlxx_nucleo_radio.c がこのファームウェアでは使用されない点に注意してください。ただし、無線関連項目の radio_board_if.c/.h および LED とプッシュボタンの board_resources.c/.h は例外です。USE_BSP_DRIVER または MX_BOARD_PSEUDODRIVER を選択することにより、Core/Inc/platform.h でこれら 2 つの実装のいずれかが選択されます。*

# 組込みユーティリティ

- lpm：低消費電力マネージャ
  - モジュールからの低消費電力要求を集中管理し、適切な低消費電力モードに移行
    - 例：Trace/Dma の出力中はマイクロコントローラは STOP モード 2 に移行しない
- sequencer：旧称 scheduler
  - 低消費電力モードに安全に移行するためのフレームワーク
  - 優先順位の処理によってタスクとイベントを記録および管理
- timer：タイマのリストまたはサーバ
  - ミドルウェアとアプリケーションで使用
- trace：DMA トレース
  - サーキュラバッファと DMA を使用してリアルタイムで出力
- misc
  - systime：時刻を設定／取得
  - tiny_scanf：フットプリントの少ない scanf
  - tiny_printf：フットプリントの少ない printf



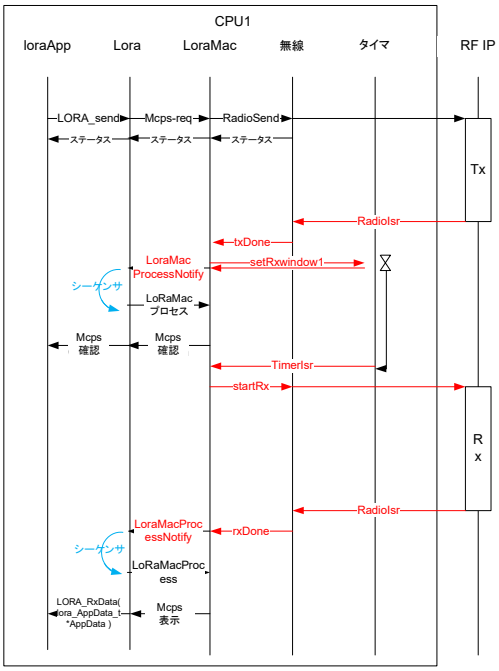

life.augmented

17

STM32WL FW パッケージの組込みユーティリティは他のプロジェクトと共通です。
以下の要素で構成されています。
低消費電力マネージャ：低消費電力モードを管理
シーケンサ（以前の名前はスケジューラ）：すべてのタスクとコールバックを、その適切な優先順位レベルに応じてスケジュール設定
タイマ：共通のタイマ･リスト
トレース：トレース出力を目的として、DMA 転送をリアルタイムで処理
その他：systime や小型の scanf と printf などのその他のモジュール

# 動作概要

- クラス A:フレーム送信シーケンス
- 赤色の矢印は割込みモードでの実行を示す
- シーケンサで FLAG を設定し、LoraMacProcess を起動
  - 例:IT 外部でのフレーム復号




life.augmented



このメッセージ･シーケンス図は、複数のソフトウェア層の間で送信されるメッセージ、フラグ、コールバックを示しています。このようなソフトウェア層として、アプリケーション層、MAC 層、無線層、RF IP 層があります。LoRaWAN のクラス A 送信で、送信（TX）のスケジュール設定と受信ウィンドウ（RX）の開放を目的として、このような送信処理が実行されます。
割込みモードでいくつかの関数呼び出しが実行されますが、それを赤色の矢印で示しています。
青色の矢印は、前のスライドで説明したシーケンサ･ユーティリティの呼び出しを表しています。