Z240 ZigBee High-Level Library and Sample Application — Coordinator Guide

The sample application is configured at compile time to operate as a Coordinator.

1. はじめに

本書は、Z240を使用したシステムを容易に構築するための **ZigBee High Level Library** と、そのサンプルプログラムの使用方法について説明します。

本ライブラリは、ZigBeeモジュールとのUART通信、コマンドフレームの生成・解析、ネットワーク接続処理などの低レベル処理をライブラリ内部で行い、アプリケーションプログラムからHigh Level APIを呼び出してZigBee通信を利用できるようにしたものです。

今回のサンプルは **ESP32 DevKit V1 + Z240-MP4 をCoordinatorとして使用するサンプル**です。EndDeviceからのノード登録、ShortAddress問い合わせ、透過通信データの受信と応答、および登録ノード情報のEEPROM保存を行います。

1.1 本書の目的

本書は、ZigBee High Level Libraryを使用してCoordinatorを構築するために必要な、

- プロジェクト構成
- ZigBeeモジュールとのUART接続
- Coordinatorの初期化
- ネットワーク形成
- Cluster `0xFC08` を使用した透過通信
- EndDeviceの登録
- ShortAddressの問い合わせ
- ノードテーブルのEEPROM保存
- High Level API

について、サンプルプログラムに沿って説明することを目的とします。

1.2 ライブラリの特徴

本ライブラリでは、アプリケーション側からZigBeeモジュール固有のコマンドフレームを直接組み立てる処理を減らし、High Level APIによってZigBee通信を扱います。

サンプルでは、特に以下の機能を使用しています。

- Coordinatorとしてのネットワーク形成
- ZigBeeネットワーク状態の取得
- Cluster `0xFC08` のZCLポート生成
- EndDeviceとの透過通信
- EndDeviceの機器番号、ShortAddress、MAC Addressの管理
- ノード登録要求への応答
- 機器番号からShortAddressを問い合わせる通信

- ノードテーブルのEEPROM保存・復元

---

2. ライブラリ概要



---

3. 開発環境

本書では、ESP32-devkit用（Sampleコードでは ESP32S3-devkit）の PlatformIO.iniを記載していますが、実際にご使用になるボードの設定、ハードウェアの設定を行ってください。

項目	内容
開発環境	PlatformIO
Framework	Arduino

項目	内容
Board	ESP32 Dev Module (`esp32dev`)
MCU	ESP32 / ESP32-WROOM-32
ZigBee UART	`Serial2`
UART Baudrate	115200 bps
Monitor Speed	115200 bps

`platformio.ini` では `PRINT_ENABLE=1` を指定し、低レベルライブラリ側のデバッグ表示機能を有効にしています。

```
[env:esp32dev]
platform      = espressif32
board         = esp32dev
framework     = arduino
monitor_speed = 115200

upload_port   = COM41

build_flags =
    -DPRINT_ENABLE=1
```

> `upload_port` は使用するPCのCOM番号に合わせて変更してください。

---

4. ファイル構成

```
project/
├─ platformio.ini
└─ src/
   ├─ main.cpp
   ├─ HighLayer.c
   ├─ HighLayer.h
   ├─ HL_Coordinator.c
   ├─ zb_def.c
   ├─ ZigBee.h
   ├─ SerialAndPrint.cpp
   ├─ SerialAndPrint.h
   ├─ eeprom_ex.cpp
   └─ eeprom_ex.h
```

ファイル	主な役割
`main.cpp`	Coordinatorサンプル本体、初期化、ネットワーク形成、受信開始
`HighLayer.c`	High Level API、Z240制御、ネットワーク処理、透過通信

ファイル	主な役割
`HighLayer.h`	構造体、定数、High Level APIの宣言
`HL_Coordinator.c`	Coordinator固有処理、ノード登録、ShortAddress管理、受信振り分け
`zb_def.c`	グローバル変数、ZCLポート設定データ
`ZigBee.h`	UARTピン、機器番号、動作モードなどの設定
`SerialAndPrint.cpp/.h`	UART通信およびデバッグ表示
`eeprom_ex.cpp/.h`	ノードテーブルのEEPROM保存・復元

---

5. ハードウェア接続

ESP32とZ240のUART通信には `Serial2` を使用します。

ESP32	Z240	用途
GPIO16 (RXD2)	TX	Z240 → ESP32
GPIO17 (TXD2)	RX	ESP32 → Z240
GND	GND	GND

```
#define ZB_SERIAL_PORT Serial2
#define ZigBee_RxPin   16
#define ZigBee_TxPin   17
#define ZigBee_BaudRate 115200
```

---

6. サンプルの設定

主要な設定は `ZigBee.h` にあります。

6.1 自機番号

```
#define MY_DEVICE_NO 0x01
```

Coordinatorの自機番号を `0x01` としています。EndDevice側の機器番号とは重複しない値を設定します。

6.2 デバイスタイプ

```
#define ZIGBEE_DEVICE COORDINATOR
```

このサンプルはCoordinator専用です。

6.3 透過通信方式

```
#define TRANSPARENT_ByFC08
```

Cluster `0xFC08` を使用した透過通信を有効にしています。

EndDeviceを透過送信モード (#define TRANSPARENT)で使用する場合は、Coodinator側も同じ設定にしてください。

6.4 EEPROM

```
#define USE_EEPROM
```

Coordinatorが登録したEndDevice情報をEEPROMへ保存します。

6.5 工場出荷リセット

```
#define EXEC_FACTORY_RESET false
```

通常は `false` とします。起動時に工場出荷リセットを実行する必要がある場合に設定を変更します。

---

7. 基本的な使い方

`setup()` は次の5段階で初期化を行います。

7.1 シリアルポート初期化

ZigBeeモジュールとの通信を一度終了し、GPIO16/GPIO17を使用してUARTを開始します。

```
ZB_UART_END();
delay(800);

ZB_UART_BEGIN();
delay(1000);
```

7.2 ノードテーブル復元

最初にRAM上のノードテーブルを初期化します。

```
InitNodeTable();
```

`USE_EEPROM` が定義されている場合はEEPROMを開始し、保存されているノード情報を復元します。

```
beginEEPROM();
readEEPROM();
```

復元後、`PrintNodeTable()` で登録内容を表示します。

7.3 ZigBeeモジュール確認・設定

```
zb_proc();
```

`zb_proc()` 内で `zb_init()` を実行し、現在のネットワーク状態を確認します。

```
int retcode = zb_init(&zb_ctx, DeviceType, NeedFactoryReset);
```

Coordinatorとして稼働していない場合は、

1. `zb_begin()` でノードタイプをCoordinatorへ設定
2. `zb_Set_TransparentSending_byFC08()` でCluster `0xFC08` のZCLポートを生成
3. `zb_ConnectNetwork()` でネットワークを形成

という順番で処理します。

7.4 ネットワーク情報確認

```
zb_getNodeStatus();
```

取得した情報は `zb_ctx` に保持されます。

```
PANID
Channel
ShortAddr
MAC Address
```

7.5 Coordinator受信処理開始

`loop()` から次の関数を呼び出します。

```
zb_CoordinatorProc();
```

`zb_CoordinatorProc()` は内部で受信待ちを継続するため、通常はこの関数から戻りません。

---

8. Coordinator

8.1 Coordinatorのネットワーク形成

`zb_init()` の戻り値を確認し、Coordinatorネットワークが開始されていなければ設定を行います。

サンプルでは `0x80` をCoordinator稼働中として扱っています。

```
if (retcode != 0x80) {
    ...
}
```

未接続の場合は `zb_begin()` と `zb_ConnectNetwork()` を実行します。

8.2 Coordinator受信処理

Coordinatorの受信処理は `HL_Coordinator.c` の

```
zb_CoordinatorProc();
```

で主に実行します。

受信した透過データは内容を判定し、ノード登録、ShortAddress問い合わせ、定期送信などに振り分けます。

---

9. 透過通信

サンプルでは Cluster `0xFC08` を使用します。

9.1 ZCLポート生成

`zb_def.c` に `Transparent_val` が定義されています。

```
const struct portConfig Transparent_val = {
    &TP_ZCLmakeport_val,
    TP_ZCLportAttrs,
    sizeof(TP_ZCLportAttrs) / sizeof(TP_ZCLportAttrs[0])
};
```

Coordinator初期設定時に次の関数へ渡します。

```
zb_Set_TransparentSending_byFC08(&zb_ctx, &Transparent_val);
```

9.2 透過データ送信

```
zb_sendTPbyFC08(ShortAddr, data, len);
```

送信先ShortAddressを指定して透過データを送信します。

---

10. Coordinator－EndDevice間のサンプル通信

`main.cpp` では、EndDevice側サンプルと対になるパケットを次のように定義しています。

10.1 ノード登録

```
EndDevice → Coordinator
55 AA 55 02 + 機器番号 + ShortAddress + MAC(8)

Coordinator → EndDevice
55 AA 55 12 + 可否 + 機器番号 + ShortAddress
```

CoordinatorはEndDeviceから送信された機器番号、ShortAddress、MAC Addressをノードテーブルへ登録または更新します。

10.2 ShortAddress問い合わせ

```
EndDevice → Coordinator
55 AA 55 04 + 機器番号 + 自ShortAddress

Coordinator → EndDevice
55 AA 55 14 + 機器番号 + 相手ShortAddress
```

EndDeviceは通信したい機器番号をCoordinatorへ問い合わせ、Coordinatorはノードテーブルから該当するShortAddressを返します。

10.3 定期送信

```
EndDevice → Coordinator
55 AA 55 08 + 自ShortAddress + "1234"

Coordinator → EndDevice
55 AA 55 18 00 00 + "67890"
```

Coordinatorは定期送信データを受信すると応答データを返します。

---

11. ノードテーブル

Coordinatorは最大20台のノード情報を管理します。

```
#define MAX_NODE 20
#define MAC_LEN  8
```

ノード情報は `NodeEntry` 構造体で保持します。

```
typedef struct {
    uint8_t  valid;
    uint8_t  deviceNo;
    uint8_t  macAddress[MAC_LEN];
    uint16_t shortAddress;
} NodeEntry;
```

ノードテーブル本体は次の配列です。

```
NodeEntry NodeTable[MAX_NODE];
```

主な管理関数は以下です。

関数	内容
`InitNodeTable()`	ノードテーブルを初期化
`PrintNodeTable()`	登録済みノード一覧を表示
`FindNodeByMacAddress()`	MAC Addressから検索
`FindNodeByDeviceNo()`	機器番号から検索
`FindEmptyNodeEntry()`	空きエントリを検索
`RegisterOrUpdateNode()`	ノードを登録または更新
`GetShortAddrByDeviceNo()`	機器番号からShortAddressを取得

---

12. EEPROM

`USE_EEPROM` を定義すると、ノードテーブルをESP32のEEPROM領域へ保存します。

12.1 EEPROMサイズ

```
#define EEPROM_SIZE 256
```

12.2 保存データ

```
struct EEPROM_configs {
    uint32_t  magic;
    uint16_t  version;
    NodeEntry NodeTable[MAX_NODE];
};
```

マジック値とレイアウトバージョンを保存し、未初期化EEPROMやデータ構造変更後の不正な読み出しを判定します。

```
#define EEPROM_MAGIC          0x5A323430UL
#define EEPROM_LAYOUT_VERSION 0x0001
```

12.3 EEPROM関数

関数	内容
`beginEEPROM()`	EEPROM使用開始
`initializeEEPROM()`	EEPROMとノードテーブルを初期化
`writeEEPROM()`	ノードテーブルを保存
`readEEPROM()`	ノードテーブルを復元

未初期化の場合はノードテーブルを空にして、初期状態をEEPROMへ保存します。

---

13. 主なデータ構造

13.1 `zb_conn_t`

自機のZigBee接続情報を保持します。

```
typedef struct {
    uint8_t  mynumber;
    uint16_t panid;
```

```
    uint8_t  channel;
    uint16_t short_addr;
    uint8_t  mac[8];
    uint16_t cluster_id;
    uint16_t profile_id;
    uint8_t  port_no;
    uint8_t  connected;
} zb_conn_t;
```

13.2 zb_peer_t

EndDevice間通信などで相手機器情報を保持します。

```
typedef struct {
    uint8_t  device_no;
    uint16_t short_addr;
    uint8_t  payload[77];
} zb_peer_t;
```

---

14. APIリファレンス

zb_init()

```
int zb_init(zb_conn_t *c, int zbdev, bool FactoryReset);
```

ZigBeeモジュールの状態を確認し、初期化処理を行います。

zb_begin()

```
bool zb_begin(zb_conn_t *c);
```

指定されたデバイスタイプでZigBeeモジュールを設定します。

zb_makePort()

```
bool zb_makePort(zb_conn_t *c, const struct portConfig *value);
```

ZCLポートを生成します。

zb_bind()

```
bool zb_bind();
```

本サンプルライブラリではNo Operationです。

zb_writeAttr()

```
bool zb_writeAttr(zb_conn_t *c, uint16_t cluster_id, int16_t data);
```

ZCL属性値を書き込みます。

zb_write()

```
int zb_write(zb_conn_t *c, const uint8_t *data, size_t len);
```

透過通信データを送信します。

zb_write_peer()

```
bool zb_write_peer(zb_peer_t *peer, const uint8_t *data, size_t len);
```

EndDevice間通信で相手ノードへデータを送信します。

zb_read()

```
int zb_read(zb_conn_t *c, uint16_t WaitTime);
```

透過通信モードでデータを受信します。

zb_ConnectNetwork()

```
bool zb_ConnectNetwork(zb_conn_t *c);
```

ZigBeeネットワークへの接続またはネットワーク形成を行います。

zb_Set_TransparentSending_byFC08()

```
void zb_Set_TransparentSending_byFC08(
    zb_conn_t *c,
```

```
    const struct portConfig *value
);
```

Cluster `0xFC08` を使用する透過通信用ZCLポートを設定します。

`zb_sendTPbyFC08()`

```
void zb_sendTPbyFC08(
    uint16_t ShortAddr,
    const uint8_t *data,
    size_t len
);
```

Cluster `0xFC08` を使用して指定ShortAddressへ透過データを送信します。

`zb_CoordinatorProc()`

```
void zb_CoordinatorProc();
```

Coordinatorの受信・振り分け処理を行います。

`zb_queryShortAddr()`

```
bool zb_queryShortAddr(zb_peer_t *c, int16_t DeviceNo);
```

機器番号に対応するShortAddressをCoordinatorへ問い合わせます。

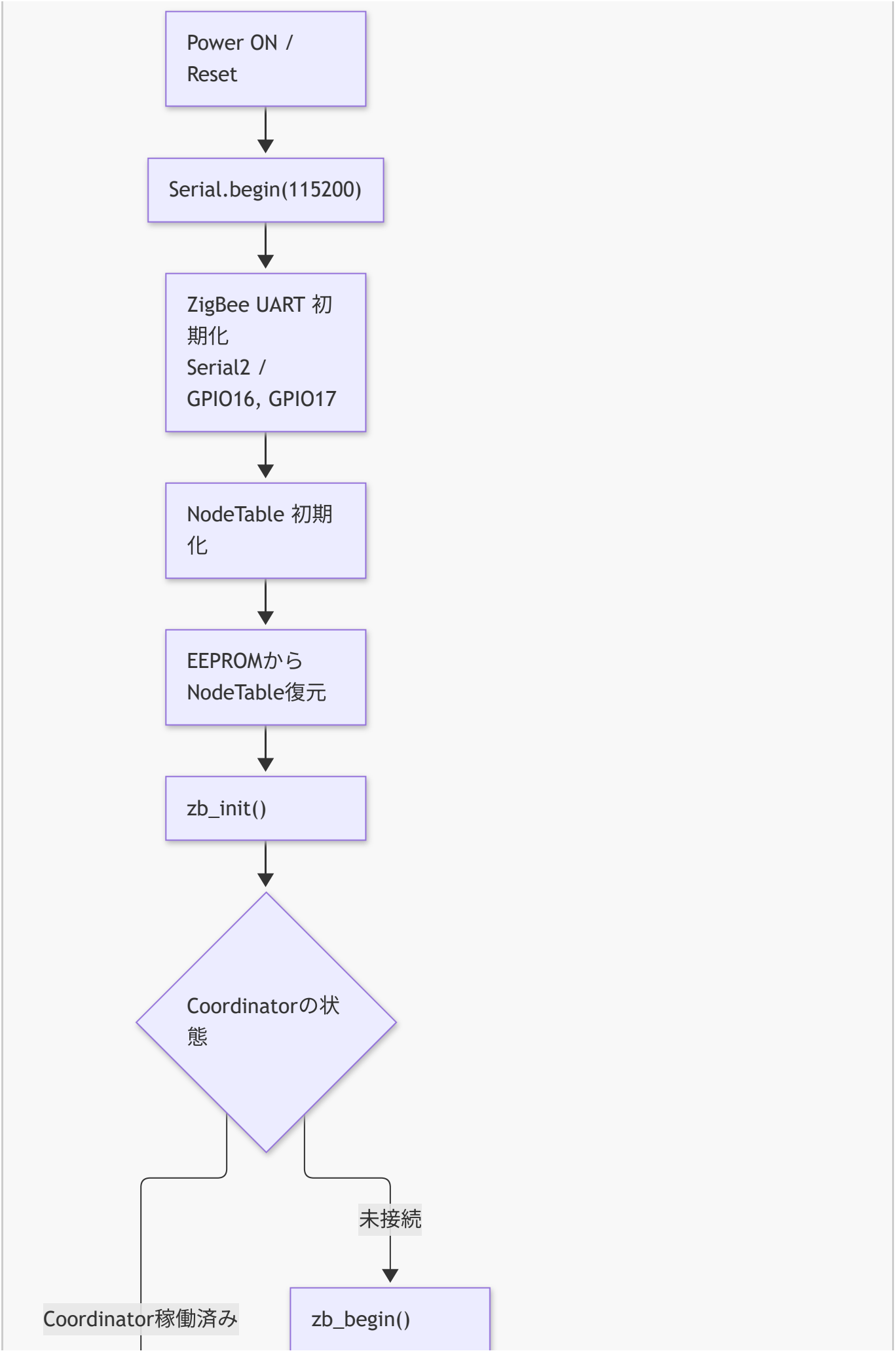
---

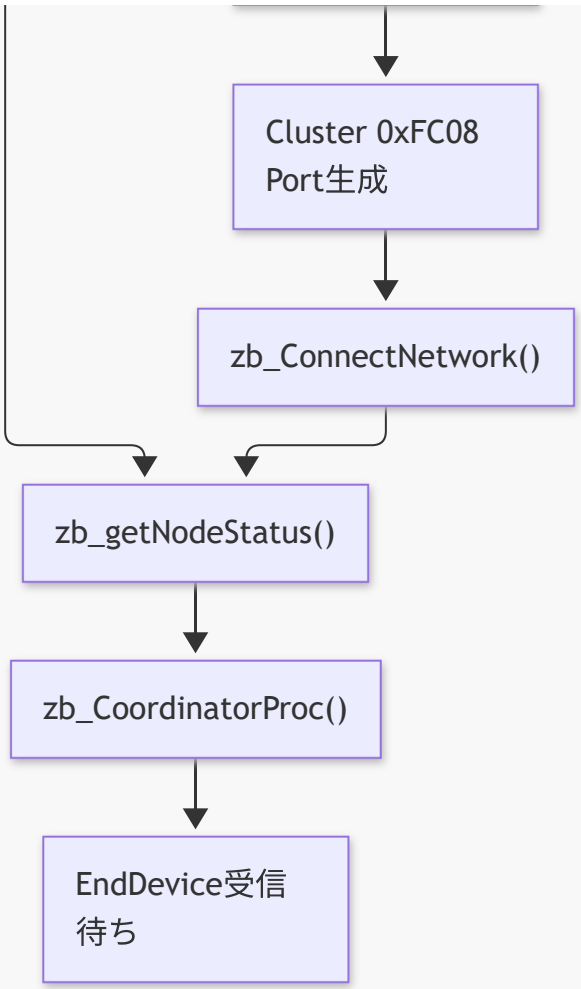
15. ローカル設定関連API

関数	内容
`zb_getNodeStatus()`	ノード状態取得
`zb_ResetModule()`	モジュールリセット
`zb_stopNetwork()`	ネットワーク停止
`zb_setChannel()`	ZigBee Channel設定
`zb_setPanid()`	PAN ID設定
`zb_CommandTransParent()`	透過送信モード設定

---

16. 起動時の処理フロー


Power ON / Reset
Serial.begin(115200)
ZigBee UART 初期化
Serial2 / GPIO16, GPIO17
NodeTable 初期化
EEPROMから NodeTable復元
zb_init()
Coordinatorの状態
未接続
Coordinator稼働済み
zb_begin()



---

17. サンプル実行時の確認項目

シリアルモニタを115200 bpsで開き、起動時に以下を確認します。

1. ZigBee用UARTが正常に開始されること
2. EEPROMからノードテーブルが読み出されること
3. `zb_init()` の戻り値が表示されること
4. Coordinatorネットワークが形成されること
5. PAN ID、Channel、ShortAddress、MAC Addressが表示されること
6. EndDeviceからのパケット待ち状態になること

受信待ち状態では、主に次のパケットを処理します。

```
55 AA 55 02 ... ノード登録要求
55 AA 55 04 ... ShortAddress問い合わせ
55 AA 55 08 ... 定期送信
```

---

18. 補足

18.1 `COORDINATOR` の表記

本プロジェクトのソースコードでは、Coordinatorを示す定数・関数名の一部に `COORDINATOR` / `Coordinator` という表記が使用されています。本書ではソースコード上の識別子については、そのままの名称を記載しています。

---

19. ライセンス

本プロジェクトのソースコードおよびドキュメントは、MIT License のもとで公開しています。

個人利用・商用利用を問わず、MIT License の条件に従って、自由に使用・複製・変更・再配布することができます。

詳細については `LICENSE` ファイルを参照してください。

---

20. サポートについて

本プロジェクトは無償公開のサンプルおよびライブラリであり、技術サポートは提供していません。

本プロジェクトのソースコードおよびドキュメントは現状のまま（AS IS）で提供されます。
使用環境や接続機器などによる動作を保証するものではありません。

本プロジェクトの利用、変更、組み込み等は、利用者自身の責任において行ってください。

なお、本プロジェクトのソースコード、README およびその他のドキュメントは、予告なく変更される場合があります。
ソースコードの変更にあたっては、可能な限り既存バージョンとの互換性を維持する方針としています。

改訂履歴

日付	内容
2026-09-01	初版