

WannaBee™ 多機能開発ボード（DTP-DBWB-MP4(EA)）サンプル / 基板検査プログラム

概要

本プロジェクトは、ESP32-S3を搭載したWannaBee™ 多機能開発ボード上で、各周辺デバイスの動作確認とZ240を使用したZigBee通信を行うためのPlatformIO / Arduinoプロジェクトです。

起動時のモーメンタリスイッチ状態によって、次の動作を行います。

- **SW3を数秒押した状態で起動:** 基板検査モード
- **SW6 OFF で起動:** Coordinatorとして動作
- **SW6 ON で起動:** EndDeviceとして動作

基板検査ではLCD、LED、スイッチ、I2C、SHT30、ADXL345、microSD、Z240の順に確認します。

通常動作では、EndDevice側はセンサ値を取得してZigBee送信およびmicroSDへ記録し、Coordinator側はZigBeeデータを受信後、LCDへのRSSI表示、シリアルモニタに受信データを表示します。

1. 開発環境

- MCU: ESP32-S3
- Board: `esp32-s3-devkitc-1`
- Framework: Arduino
- Build system: PlatformIO
- Serial monitor: 115200 bps
- USB CDC: 有効

`platformio.ini` の主な設定:

```
[env:esp32-s3-devkitc-1]
platform = espressif32
board = esp32-s3-devkitc-1
framework = arduino
upload_port = COM21
monitor_port = COM21
monitor_speed = 115200

build_flags =
    -DARDUINO_USB_MODE=1
    -DARDUINO_USB_CDC_ON_BOOT=1
    -DPRINT_ENABLE=1
```

COMポート番号は使用するPC環境に合わせて変更してください。

2. 主なハードウェア

デバイス	用途	接続
ESP32-S3	メインCPU	-
Z240-MP4(EA)	ZigBee通信	UART2
SHT30	温湿度センサ	I2C

デバイス	用途	接続
AQM0802A	8桁×2行 LCD	I2C
ADXL345	3軸加速度センサ	SPI
microSD	SDカード	SPI
D4/D5/D6	動作確認LED	GPIO

3. プロジェクト構成

```
DevBoard_Z240/  
├─ platformio.ini  
├─ README.md  
└─ src/  
    ├─ main.cpp  
    ├─ modules.cpp  
    ├─ modules.h  
    ├─ testModules.cpp  
    ├─ globals.cpp  
    ├─ globals.h  
    ├─ ZigBee.h  
    │  
    └─ --- Z240高レベルライブラリ ---  
        ├─ HighLayer.c  
        ├─ HighLayer.h  
        ├─ HL_cordinator.cpp  
        ├─ SerialAndPrint.cpp  
        ├─ SerialAndPrint.h  
        ├─ zb_def.c  
        │  
        └─ eeprom_ex.cpp    EEPROMライブラリAPI使用関数群  
            └─ eeprom_ex.h
```

Z240高レベルライブラリを構成するファイルは、README末尾の「付録」で説明します。

アプリケーション側の主なファイル

ファイル	内容
main.cpp	起動処理、動作モード選択、EndDevice / Coordinatorのメイン処理
modules.cpp/.h	GPIO、LCD、SHT30、GPS、ADXL345、SD、Z240動作確認
testModules.cpp	基板検査シーケンス
globals.cpp/.h	共通グローバル変数
ZigBee.h	Z240使用条件、UART、動作モードなどの設定・公開インターフェース
eeprom_ex.cpp/.h	Coordinatorのノード情報保存

Z240高レベルライブラリの詳細は付録およびデータシートを参照してください。

4. 起動時の動作モード

4.1 SW3: 基板検査モード

起動時にSW3がLOWの場合、`testModules()` を実行します。

```
if (digitalRead(PIN_SW3) == LOW) {
  testModules();
  while (1) {
    delay(1);
  }
}
```

検査終了後は通常動作へ移行せず、そのまま待機します。

4.2 SW6: EndDevice / Coordinator切替

通常起動ではSW6の状態でZigBee動作モードを決定します。

SW6	動作
OFF / HIGH	Coordinator
ON / LOW	EndDevice

```
EndDeviceOrNot = false;
DeviceType = COORDINATOR;

if (digitalRead(PIN_SW6) == LOW) {
  EndDeviceOrNot = true;
  DeviceType = ENDDEVICE;
}
```

LCDにはそれぞれ次のように表示します。

EndDevice時

```
RUN as
EndDevic
```

Coordinator時

```
RUN as
Cordintr
```

5. 部品名称 と 主なGPIO割り当て

I2C

用途	GPIO
SDA	8

用途 GPIO

SCL 9

I2C接続デバイス:

- AQM0802A LCD: 0x3E
- SHT30: 0x44

SPI

用途 GPIO

SCLK 14

MISO 21

MOSI 47

ADXL345 CS 13

microSD CS 48

ADXL345とmicroSDは同じSPIバスを共有するため、片方を使用するときはもう片方のCSをHIGHにして非選択にします。

LED (× 3)

LED GPIO

D4 10

D5 11

D6 12

HIGHで点灯します。

モーメンタリスイッチ (× 4)

SW GPIO 用途

SW3 4 ENTER / 起動時の基板検査選択

SW4 5 ボタン入力

SW5 6 UP

SW6 7 DOWN / 起動時のEndDevice選択

各スイッチは押下時LOWです。

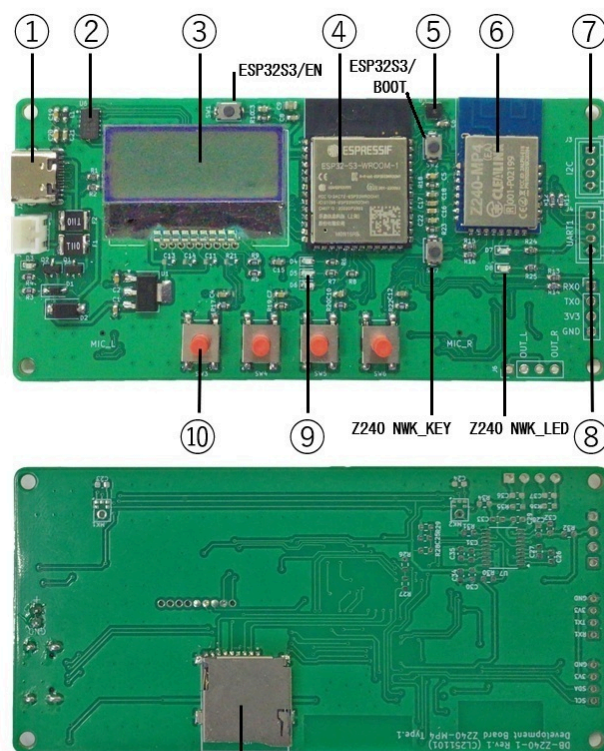
Z240-MP4(EA)

信号 GPIO

UART RX 15

UART TX 16

RESET 3



1. USB type-c
2. 加速度センサー (ADXL345)
3. 小型液晶モニター (AQM0802A)
4. ESP32-S3-WROOM-1
5. 温湿度センサー (SHT30)
6. WannaBee™ 通信モジュール (Z240-MP4 (EA))
7. I2Cコネクタ (Groveコネクタ互換 HY 4P タイプ)
8. UARTコネクタ (Groveコネクタ互換 HY 4P タイプ)
9. LED ×3
10. モーメンタリスイッチ ×4
11. microSDカードスロット

通信速度は115200 bpsです。

6. 基板検査モード

SW3を押した状態で電源投入またはリセットすると、次の検査を順番に実行します。

```
LCD
↓
LED
↓
Buttons
↓
I2C Scan
↓
SHT30
↓
ADXL345
↓
microSD
↓
Z240 Node Status
↓
Finish
```

原則としてSW3（ENTER）で次の項目へ進みます。

6.1 LCD

AQM0802AをI2Cアドレス **0x3E** で確認します。

検出できた場合はコントラストを設定し、**Hello!** を表示します。

LCDはオプション扱いのため、未実装でも検査自体は継続します。

6.2 LED

D4 / D5 / D6を順番に点灯した後、3個同時点灯を行います。

目視で点灯状態を確認します。

6.3 モーメンタリスイッチ

SW3～SW6の状態をSerialとLCDへ表示します。

SW3を押すとボタン検査を終了します。

6.4 I2Cスキャン

I2Cアドレス **0x01** ～ **0x7E** を走査します。

既知のデバイスの場合は次のように識別します。

```
0x3E : AQM0802A
0x44 : SHT30-DIS
```

6.5 SHT30

単発測定で温度・湿度を取得します。

- CRC-8チェックあり
- 温度: degC
- 湿度: %RH
- ALERT: GPIO38

6.6 ADXL345

4線式SPIで通信します。

主な設定:

- SPI MODE3
- 4 MHz
- FULL_RES
- ± 16 g
- 100 Hz
- 3.9 mg/LSB

DEVID = 0xE5を確認した後、X/Y/Zの生値とg値を表示します。

6.7 microSD

microSDをマウントし、カード容量を表示します。

ADXL345とSPIバスを共有しているため、SDアクセス時はADXL345のCSをHIGHにします。

6.8 Z240-MP4(EA)

Z240高レベルライブラリを使用してNode Statusを取得します。

確認内容:

- UART応答
- Device Type
- MAC Address
- Network connection
- PAN ID
- Channel
- Short Address

応答がない場合のみRESETをLOWへ20 msパルスし、再度問い合わせます。

この検査では保存済みのZ240設定を意図的に変更しません。

7. 通常動作: EndDevice

SW6をON（LOW）にして起動するとEndDeviceとして動作します。

EndDevice と Coordinator を正常に ネットワーク形成するためには、先に Coordinator を

```
--- ネットワークの形成に成功しました
--- ネットワークの参加受付を開きます
```

の状態にする必要があります。

Cordinator は、ネットワーク形成可能な状態を 180秒間(デフォルト値) NWK_LED の点滅で示しますので、消灯時には NWK_KEY を押下してください。(LEDが点滅)

この状態でなければ、EndDevice は Cordinator とネットワークの形成ができません。

EndDeviceは、起動時に `checkModules()` を実行し、実装されている周辺デバイスを確認します。

```
SHT30_Exist
ADXL345_Exist
SD_Exist
```

7.1 SHT30データ

SHT30が存在する場合、温湿度を取得して文字列化します。

例:

```
Temp: 25.34 degC
Humd: 48.21 %RH
ALERT(I038): HIGH
```

作成した文字列はZigBeeへ送信し、`MeasuredDataBuf` に追加します。

7.2 ADXL345データ

ADXL345が存在する場合、X/Y/Zの生値およびg値を取得します。

例:

```
raw:    12    -8   257 |  g:  +0.047  -0.031  +1.002
```

この文字列もZigBeeへ送信し、`MeasuredDataBuf` に追加します。

7.3 データ送信周期

現在の `loop()` では、測定・送信後に約8秒待機します。

```
delay(8000);
```

8. microSDログ

SDカードが使用可能な場合、起動時に新しいログファイルを作成します。

ファイル名:

```
/LOG00000.TXT
/LOG00001.TXT
/LOG00002.TXT
...
```

シーケンス番号はESP32のNVSに保存します。

NVS namespace:

```
logger
```

key:

```
sequence
```

ログファイル作成後、EndDeviceで取得した測定データを `appendLogData()` で追記します。

追記データの例：

```
Temp: 34.89 degC
Humd: 45.95 %RH
ALERT(I038): HIGH
raw:   -14   -13   268 | g:  -0.055  -0.051  +1.045
```

9. 通常動作: Coordinator

SW6をOFF（HIGH）にして起動するとCoordinatorとして動作します。

`zb_CordinatorProc()` がZigBee受信を待ち受け、透過送信データや制御コマンドを処理します。

Coordinatorではノード情報を管理し、必要に応じてEEPROMへ保存します。

管理する主な情報:

```
typedef struct {
    uint8_t  valid;
    uint8_t  deviceNo;
    uint8_t  macAddress[8];
    uint16_t shortAddress;
} NodeEntry;
```

最大登録数:

```
MAX_NODE = 20
```

未登録Short Address:

```
0xFFFF
```

10. RSSI表示

Coordinator受信データからRSSIのRAW値を取得し、次の式でdBmへ変換します。


```
uint8_t raw = ZigBeeRecvData[offset + 14];
int16_t rssi_value = (int16_t)raw - 256;
```

例:

RAW	RSSI
0xD5	-43 dBm
0xD4	-44 dBm
0xCB	-53 dBm
0xCA	-54 dBm

現在は次のRAW値を表示対象外としています。

```
bool skip = (raw == 0x00 || raw == 0xFF || raw < 0x88);
```

したがって、次のデータはRSSI表示をスキップします。

- 0x00
- 0xFF
- 0x88未満（-120 dBmより小さい側）

有効な場合、AQM0802Aに次のように表示します。

```
RSSI
-54 dBm
```

11. 受信データ表示

CoordinatorではRSSI表示後、受信データ中の表示可能なASCII文字をSerialへ出力します。

表示対象:

```
0x20 ~ 0x7E
```

CR / LFは改行として出力します。

それ以外の制御コードやバイナリ値はスキップします。

12. Z240-MP4(EA)の初期化

起動時に `zb_proc()` を実行し、Z240の状態を確認します。

主な流れ:

```
zb_init()
↓
ネットワーク状態確認
```

```
↓
未接続なら zb_begin()
↓
Coordinatorの場合は Cluster 0xFC08 のポート生成
↓
zb_ConnectNetwork()
↓
EndDeviceの場合は透過モード設定
```

CoordinatorではZCL Cluster **0xFC08** を使用した透過送信ポートを生成します。

ネットワーク形成時のPAN IDとチャネルは現在モジュール側の自動設定を使用します。

13. グローバル変数

主なグローバル変数は `globals.cpp/.h` にまとめています。

```
bool LCD_Exist;
bool SHT30_Exist;
bool ADXL345_Exist;
bool SD_Exist;
bool EndDeviceOrNot;

char MeasuredDataBuf[200];
uint16_t MeasuredDataLength;

uint32_t logSequence;
char logFileName[32];
```

`SimpleLCD lcd` も共通インスタンスとして定義しています。

14. SimpleLCD

`modules.h` にAQM0802A用の最小LCDクラス `SimpleLCD` を実装しています。

主なAPI:

```
lcd.begin(8, 2);
lcd.setContrast(28);
lcd.clear();
lcd.setCursor(col, row);
lcd.print("text");
lcd.printf("%d", value);
```

LCDが未実装の場合は `_present == false` となり、表示関数は実質何もしません。

そのためLCDなしでも他の処理を継続できます。

15. ビルドと書き込み

PlatformIOから通常どおりBuild / Uploadを実行します。

CLIの場合:

```
pio run
pio run -t upload
pio device monitor
```

Serial monitorは115200 bpsです。

#16.関数一覧

16.1main.cpp

関数	引数	戻り値	役割
setup()	なし	void	GPIO、Serial、各モジュール、Z240などを初期化し、起動時の動作モードを設定する
loop()	なし	void	設定された動作モードに従い、センサーデータ取得、ZigBee通信、ログ処理などを繰り返す
zb_proc()	なし	bool	Z240の初期化、ネットワーク状態確認、ZigBee動作開始処理を行う

16.2modules.cpp

関数	引数	戻り値	役割
initModulesGPIO()	なし	void	LED、スイッチ、CS、割り込み端子など評価ボードのGPIOを初期化する
threetimesBlinkLED()	なし	void	LEDを3回点滅させる
startBlinkLED()	なし	void	起動時のLED点滅表示を行う
sht30Crc(constuint8_t*data,size_tlen)	data:対象データ、len	uint8_t	SHT30受信データのCRCを計算する
sht30Read(float&tempC,float&humidity)	tempC:	bool	SHT30から温度と湿度を読み取る
SHT30_Read(char*buf)	buf:測定データ格納先	void	SHT30の測定値を取得し、送信・ログ用の文字列として格納する

関数	引数	戻り値	役割
<code>adxlWriteReg(uint8_treg,uint8_tvalue)</code>	<code>reg</code> :レジスタ、 <code>value</code>	<code>void</code>	ADXL345の指定レジスタへ値を書き込む
<code>adxlReadReg(uint8_treg)</code>	<code>reg</code> :レジスタ	<code>uint8_t</code>	ADXL345の指定レジスタを読み取る
<code>adxlReadXYZ(int16_t&x,int16_t&y,int16_t&z)</code>	<code>x,y,z</code> :各軸データ格納先	<code>void</code>	ADXL345からX/Y/Z軸のRAW加速度を読み取る
<code>adxlBegin()</code>	なし	<code>bool</code>	ADXL345を確認して初期設定を行う
<code>ADXL345_Read(char*buf)</code>	<code>buf</code> :加速度データ格納先	<code>void</code>	ADXL345のX/Y/Zデータを取得し、送信・ログ用の形式へ格納する
<code>loadLogSequence()</code>	なし	<code>uint32_t</code>	NVSからログファイルのシーケンス番号を読み出す
<code>saveLogSequence(uint32_tseq)</code>	<code>seq</code> :保存するシーケンス番号	<code>void</code>	NVSへログファイルのシーケンス番号を保存する
<code>createLogFile()</code>	なし	<code>bool</code>	SDカード上に新しいログファイルを作成する
<code>appendLogData(constchar*data)</code>	<code>data</code> :追記文字列	<code>bool</code>	現在のSDログファイルヘデータを追記する
<code>checkModules()</code>	なし	<code>void</code>	LCD、SHT30、ADXL345、SDなど各

関数	引数	戻り値	役割
			モジュールの接続状態を確認する
<code>parseZ240NodeStatus(u_Message&message,bool&hasNetworkInfo)</code>	<code>message:</code>	<code>bool</code>	Z240のNodeStatus応答を解析してネットワーク情報の有無を判定する
<code>z240LibraryQuery(u_Message&message,bool&hasNetworkInfo,int&initResult)</code>	<code>message:</code>	<code>bool</code>	Z240高レベル処理を使ってモジュール状態とネットワーク情報を問い合わせる
<code>z240Check(u_Message&message,bool&hasNetworkInfo,int&initResult)</code>	<code>message:</code>	<code>bool</code>	Z240との通信を確認し、NodeStatusとネットワーク情報を取得する

16.3testModules.cpp

関数	引数	戻り値	役割
<code>waitForButtonPressDebounce(uint8_tpin,unsignedlongdebounceMs=50)</code>	<code>pin:</code>	<code>void</code>	チャタリングを除去しながら指定ボタンが押されるまで待つ
<code>buttonTestMode()</code>	なし	<code>void</code>	評価ボード上のスイッチ入力を確認する
<code>testModules()</code>	なし	<code>void</code>	LCD、LED、ボタン、I2C、SHT30、ADXL345、SD、Z240を順番に検査する

17. 使用時の注意

- ADXL345とmicroSDはSPIバスを共有するためCS制御に注意してください。
- Z240のUARTは115200 bpsです。
- Z240 RESETは負論理です。
- 基板検査モードではSW3で項目を進めます。
- SW6の状態は起動時にEndDevice / Coordinatorの選択に使用します。
- Coordinatorのノード情報はEEPROMへ保存される場合があります。

18.付録 Z240高レベルライブラリ

詳細は、製品ページのダウンロードサポートからダウンロードしてご参照ください。

以下はアプリケーション本体から利用するZ240高レベルライブラリです。

```
HighLayer.c
HighLayer.h
HL_cordinator.cpp
SerialAndPrint.cpp
SerialAndPrint.h
zb_def.c
```

通常の基板アプリケーションを理解・変更する場合は、まず `main.cpp`、`modules.cpp`、`testModules.cpp` を参照してください。

Z240の通信プロトコルやCoordinator内部処理を変更する場合に、以下のライブラリ側を確認します。

18.1 HighLayer.c / HighLayer.h

Z240制御の高レベルAPIを提供します。

代表的な構造体:

```
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;
```

EndDevice間通信用:

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

代表的なAPI:

```
zb_init()
zb_begin()
zb_makePort()
zb_read()
zb_write()
zb_write_peer()
```

```
zb_Set_TransparentSending_byFC08()  
zb_sendTPbyFC08()  
zb_getNodeStatus()  
zb_ConnectNetwork()  
zb_CommandTransParent()
```

Cluster定義にはBasic、Identify、Temperature Measurement、Humidity Measurement、IAS Zoneなどがあります。

18.2 HL_cordinator.cpp

Coordinator側の高レベル処理を実装します。

主な処理:

- ZigBee受信待ち
- 透過送信フレーム解析
- DeviceNo / MAC / Short Address管理
- ノード登録
- Short Address問い合わせ応答
- Cluster `0xFC08` 透過送信
- RSSI取得・LCD表示
- ASCII受信データ表示
- ノードテーブル管理

代表関数:

```
zb_CordinatorProc()  
ProcessTransparentPacket()  
GetShortAddressFromDeviceNo()  
SendZdoNodeShortAddress()  
InitNodeTable()  
PrintNodeTable()
```

18.3 SerialAndPrint.cpp / SerialAndPrint.h

Z240用UARTおよびデバッグ表示をまとめた層です。

主な機能:

- Z240 UART初期化
- UART送受信
- 受信バッファ管理
- フィードバック待ち
- HEX表示
- `Printf()` / `PrintLN()` 等のデバッグ出力

`PRINT_ENABLE` / `USE_SerialPrint` により表示処理を有効・無効化できます。

18.4 zb_def.c

Z240コマンド生成・定義側の低レベル処理をまとめています。

高レベルAPIから使用され、Z240へ送信する各コマンドフレームの生成などを担当します。

通常のアプリケーション変更では直接編集する必要はありません。

18.5 ZigBee.hとの関係

ZigBee.h はアプリケーションとZ240高レベルライブラリを接続する設定・公開インターフェースとして使用します。

本、WannaBee™ 多機能開発ボード（DTP-DBWB-MP4(EA)）サンプル・プログラムでは、Z240の動作を Cordinattor と End Device に限定しており、それぞれの動作切り替えを SW6 で行いますので、**ZigBee.h** の変更は不要です。

主な設定:

```
#define ZigBee_BaudRate 115200
#define MY_DEVICE_NO    0x01
#define USE_EEPROM
#define TRANSPARENT
```

Device Type:

```
#define CORDINATOR      0
#define ROUTER          1
#define ENDDEVICE       2
#define SLEEP_ENDDEVICE 3
```

Z240のモード設定やデバッグ出力条件を変更する場合は、このヘッダの定義も確認してください。

18.6 ライブラリ層とアプリケーション層の関係

```
main.cpp
├── modules.cpp / testModules.cpp
└── ZigBee.h
    ├── HighLayer.c / HighLayer.h
    ├── HL_cordinattor.cpp
    ├── SerialAndPrint.cpp / .h
    └── zb_def.c
```

アプリケーション側では、Z240の低レベルフレームを直接組み立てるのではなく、高レベルAPIを経由して使用する構成です。

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
------------	----
