KashiwaGeeks ライブラリー

プログラミング・マニュアル

Ver. 0.4

作成日 2017.12.13

Application クラス

Application クラスは Arduino の新たなアプリケーションフレームワークで、各種スリープモードとウォッチドックタイマーによるスリープからの復帰、 INT0、INT1 割り込み処理やタスク実行管理機能を提供する。

1) #include <KashiwaGeeks.h>

Application フレームワークが使用できるようになる。

2) void start(void)

Arduino プログラムで使用される setup()関数に代わる、 フレームワークが提供する初期化関数でプログラム起動時に最初に一回だけ実行される。
この関数内に初期設定用の処理を記述する。

3) void CosoleBegin(uint32_t baudrate)

Serial.begin()関数に代わる関数でコンソールのボーレイトを設定する。

4) void ConsolePrint(format, ...)

Serial.print() 関数に代わる関数で、可変個の変数を指定されるフーマットでシリアルポートに出力する。

5) void DebugPrint(format, ...)

ConsolePrint()と同様。

6) void DisableConsole(void)

ConsolePrint()の出力を停止する。

7) void DisableDebug(void)

DebugPrint()の出力を停止する。

DisableDebug() かつ DisableConsole() のときは消費電力削減のため UART0 のパワーが オフとなる。

8) void LedOn(void)、LedOff(void)

Arduino の LED を点灯または消灯させる。

9) void sleep(void)

アプリケーションがスリープする直前にこの処理が実行される。
スリープする前に実行したい処理をこの関数内に記述する。

10) void wakeup(void)

アプリケーションがスリープから戻ったときにこの処理が実行される。

11) void int0D2(void)

デジタルピン 2 が HIGH になった時にこの処理が実行される。
実行後はスリープ状態に戻る。

12) void int0D2(void)

デジタルピン 3 が HIGH になった時にこの処理が実行される。
実行後はスリープ状態に戻る。

13) void setWDT(uint8_t interval)

ウォッチドックタイマー値を設定する。 デフォルトは 1 秒となっているが、更に電力消費を抑えるために 2, 4, 8 秒のいづれかに設定を変更する。

14) TASK_LIST = { TASK(関数、開始時間、繰り返し時間), ... , END_OF_TASK_LIST };

反復して実行したい処理のリストで、このリストで指定した関数を繰り返し時間（秒）枚に実行される。 開始時間を指定して最初に実行する時間（秒）をずらすことができる。

15) PORT_LIST = { PORT(ポート、関数), ... , END_OF_PORT_LIST };

LoRaWAN からのダウンリンクデータのポートに従って、実行する処理を指定するために使用する。 ADB922S クラスの checkDownLink()メソッドがこれを使用する。

16) ReRun(関数, uint32_t 開始時間)

開始時間後に指定する関数を実行する。

ADB922S クラスとメソッド

ADB922S は TLM922S デバイスを使用する LoRaWA 用の Arduino シールドを意味している。このデバイスは SenseWay、Soracom ともに使用しており、このプログラムはいづれのシールドにも使用できる。

メソッド

1) bool begin(uint32_t baudrate = 9600, uint8_t retryTx = 1, uint8_t retryJoine = 1);

機能： ADB922S の初期化を行う。

引数： uint32_t baudrate シリアル入出力の速度、9600, 19200, 57600, 115200 が有効
uint8_t retryTx 送信リトライ回数を設定する。 0 から 255 回が有効
uint8_t retryJoine Joine のリトライ回数を設定する。

戻り値: 正常完了ならば true、 速度設定失敗ならば false。

2) bool connect(void);

機能： LoRaWA に接続する。 接続に必要なキーが保存されていなければ、joine を試みる。

引数： なし

戻り値: joine していれば true。 キーが保存されていない上、joine を begin()で設定した retryJoine 回数実行しても joine できなければ false。

3) bool reconnect(void);

機能： LoRaWA に接続する。 接続に必要なキーが保存されていても、joine を試みる。

引数： なし

戻り値: joine できれば true。 joine を begin()で設定した retryJoine 回数実行しても joine できなければ false。

4) uint8_t seetDr(LoRaDR);

機能： DR 値を設定する。 DR 値によって最大ペイロード長も定まる。

引数： DR 値、dr0, dr1, dr2, dr3, dr4, dr5

戻り値: ペイロード長、 設定エラーの場合 -1。

5) int sendString(uint8_t port, bool echo, const __FlashStringHelper* format, ...);

機能： 文字列データを送信する。 ダウンリンクデータを受信しているか、
11) の getDownLinkData(void)で確認できる。

引数： uint8_t port 送信したデータは port で指定されるアプリケーションに送られる。
bool echo true ならば送信データをコンソールに表示する。
const __FlashStringHelper* format 送信データフォーマット
… 可変個数の送信データ フォーマットは printf()で使用するものと同じ

戻り値： 送信正常完で LoRa_RC_SUCCESS。
送信データが長すぎる場合は LoRa_RC_DATA_TOO_LONG
joine していない場合は LoRa_RC_NOT_JOINED
その他のエラーの倍は LoRa_RC_ERROR

6) int sendStringConfirm(uint8_t port, bool echo, const __FlashStringHelper* format, ...);

機能： 文字列データを送達確認付きで送信する。 ダウンリンクデータを受信しているか、
11) の getDownLinkData(void)で確認できる。

引数： uint8_t port 送信したデータは port で指定されるアプリケーションに送られる。
bool echo true ならば送信データをコンソールに表示する。
const __FlashStringHelper* format 送信データフォーマット
… 可変個数の送信データ フォーマットは printf()で使用するものと同じ

戻り値： 送信正常完で LoRa_RC_SUCCESS。
送信データが長すぎる場合は LoRa_RC_DATA_TOO_LONG

7) int sendPayload(uint8_t port, bool echo, Payload*);

機能： ペイロードを送信する。 ダウンリンクデータを受信しているか、
11) の getDownLinkData(void)で確認できる。

引数： uint8_t port 送信したデータは port で指定されるアプリケーションに送られる。
bool echo true ならば送信データをコンソールに表示する。
Payload* ペイロードのポインター

戻り値： 送信正常完で LoRa_RC_SUCCESS。
送信データが長すぎる場合は LoRa_RC_DATA_TOO_LONG
joine していない場合は LoRa_RC_NOT_JOINED
その他のエラーの倍は LoRa_RC_ERROR

8) int sendPayloadConfirm(uint8_t port, bool echo, Payload*);

機能： ペイロードを送達確認付きで送信する。 ダウンリンクデータを受信しているか、
11) の getDownLinkData(void)で確認できる。

引数： uint8_t port 送信したデータは port で指定されるアプリケーションに送られる。
bool echo true ならば送信データをコンソールに表示する。
Payload* ペイロードのポインター

戻り値：送信正常完で LoRa_RC_SUCCESS。
送信データが長すぎる場合は LoRa_RC_DATA_TOO_LONG

9) Payload* getDownLinkPayload(void);

機能： 前回送信時のダウンリンクデータを Payload クラスとして取得する。

引数： なし

戻り値： 前回の送信時にダウンリンクデータがなければ、0 が、データがあればペイロードのポインター。

10) uint8_t getDownLinkPort(void);

機能： 前回送信時のダウンリンクデータからポートを取得する。

引数： なし

戻り値： ポート , 0 は受信データなし。

11) String getDownLinkData(void);

機能： 前回送信時のダウンリンクデータからポートを除く文字データを取得する。
データがなければ空文字列が返される。

引数： なし

戻り値： ポートを除く文字列データ、データ無しは空文字列。

12) void sleep(void);

機能： 無期限のディープスリープとなる。 D7 ピンの立ち上がりでスリーブから復帰する。

引数： なし

戻り値: なし

13) void wakeup(void);

機能： 無期限のディープスリープから復帰する。

引数： なし

戻り値: なし

14) void getHwModel(char* model, uint8_t length);

機能： TLM922S のモデル名を取得する。

引数： char*model　モデル名を返すアドレスを指定する。
uint8_t length　取得するモデル名の文字数。

戻り値: 引数　model に指定文字数分のモデル名が返される。

15) void getVersion(char* version, uint8_t length);

機能： TLM922S のバージョンを取得する。

引数： char* version　バージョンを返すアドレスを指定する。
uint8_t length　取得するバージョンの文字数。

戻り値: version に指定文字数分のバージョンがが返される。

16) void getEUI(char* eui, uint8_t length);

機能： TLM922S のデバイス EUI を取得する。

引数： char* eui　デバイス EUI を返すアドレスを指定する。
uint8_t length　取得する EUI の文字数。

戻り値: version に指定文字数分のバージョンがが返される。

17) uint8_t getMaxPayloadSize(void);

機能： プログラミングで設定した送信可能ペイロード長を返す。
KashiwaGeeks ライブラリ ADB922S.h の 45 行目の LoRa_MAX_PAYLOAD_SIZE で指定される。

引数： なし

戻り値: 送信可能ペイロード長　Max 255 バイト

17) bool setTxRetryCount(uint8_t retry);

機能： データ送信リトライ回数を設定する。 0〜255 が有効。

引数： uint8_t retry　　送信リトライ設定回数。

戻り値: 設定正常完で true。設定失敗で false

18) uint8_t getTxRetryCount(void);

機能： 設定されている送信リトライ回数を取得する。

引数： なし

戻り値: 送信リトライ回数

19) void checkDownLink(void);

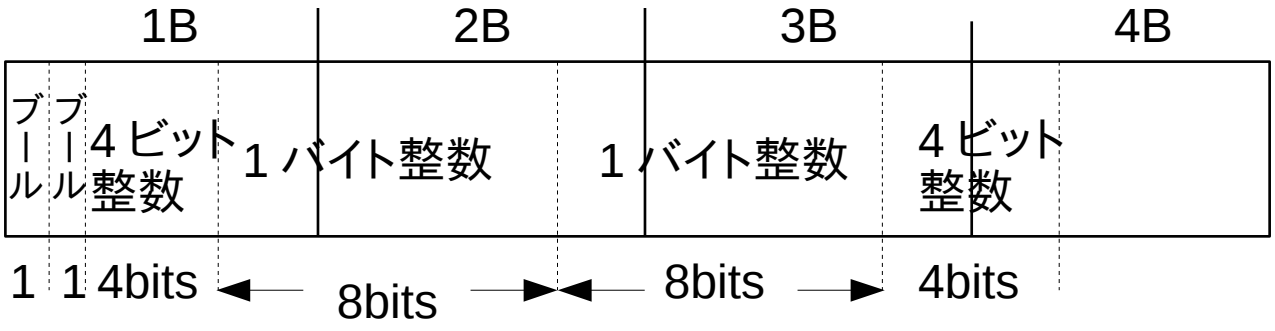
機能： DownLink データがあればそのポートを取出し、PORT_LINK で指定されたポートと紐付けられたコールバック関数を実行する。

引数： なし

戻り値: なし

Payload クラスとメソッド

Payload クラスは LoRaWAN のペイロードを表す。 LoRaWAN のペイロードは最長でも 242 バイトである。 通信距離を伸ばすためにペイロードは 11 バイトあるいは 53 バイトにする必要がある。 このため、Payload クラスでは 1 ビットの bool や 4 ビット、1 バイト、2 バイト、4 バイトの整数をバイトの境界を跨いで表現できるようにしたものである。



コーディング例

```
// 4 バイトのペイロードを生成してデータを格納する

Payload   pl(4);

pl.set_bool(true);
pl.set_bool(false);
pl.set_int4((int8_t) -4 );     // -8 ~ 7
pl.set_int8((int8_t) 120);
pl.set_uint8((uint8_t) 250);
pl.set_uint4((uint8_t)15);

bool  b1 = pl.get_bool();
bool b2 = pl.get_bool();
int8_t  i41 = pl.get_int4();
int8_t i81 = get_int8();
uint8_t u81 = get_uint8();
uint8_t u41 = get_uint4();
```

ペイロードからデータを取り出す場合は、格納した順番で取り出していく。

15 文字までの文字列ならば　set_string(String); String get_string(); が使用できる。
15 文字を超える場合は、Payload は使用できない。

データ格納メソッド

```
void set_bool(bool);
void  set_int4(int8_t);
void set_int8(int8_t);
void set_int16(int16_t);
void set_int32(int32_t);
void set_float(float);
void set_uint4(uint8_t);
void set_uint8(uint8_t);
void set_uint16(uint16_t);
void set_uint32(uint32_t);
void set_string(String);
```

ダータ取出しメソッド

```
bool       get_bool(void);
int8_t     get_int4(void);
int8_t     get_int8(void);
int16_t    get_int16(void);
int32_t    get_int32(void);
float      get_float(void);
uint8_t    get_uint4(void);
uint8_t    get_uint8(void);
uint16_t   get_uint16(void);
uint32_t   get_uint32(void);
String     get_string(void);
```

ADB922S アプリケーション・リファレンスコード

```
#include <KashiwaGeeks.h>

ADB922S LoRa;        // create ADB922S instance.

//================================
//       Initialize Device Function
//================================
#define BPS_9600         9600
#define BPS_19200       19200
#define BPS_57600       57600
#define BPS_115200     115200

void start()
{
    /*  Setup console */
    ConsoleBegin(BPS_57600);
    //DisableConsole();
    //DisableDebug();

    ConsolePrint(F("**** Start*****\n"));

    /*  setup Power save Devices */
    //power_adc_disable();      // ADC converter
    //power_spi_disable();      // SPI
    //power_timer1_disable();   // Timer1
    //power_timer2_disable();   // Timer2, tone()
    //power_twi_disable();      // I2C

    /*  setup ADB922S  */
    if ( LoRa.begin(BPS_19200) == false )
    {
        while(true)
        {
            LedOn();
            delay(300);
            LedOff();
            delay(300);
        }
    }

    /* set DR. therefor, a payload size is fixed. */
    LoRa.setDr(dr3);  // dr0 to dr5

    /*  join LoRaWAN */
    LoRa.reconnect();

    /*  seetup WDT interval to 1, 2, 4 or 8 seconds  */
    //setWDT(8);    // set to 8 seconds
}
```

```
//================================
//        Power save functions
//================================
void sleep(void)
{
    LoRa.sleep();
    DebugPrint(F("LoRa sleep.\n"));
}

void wakeup(void)
{
    LoRa.wakeup();
    DebugPrint(F("LoRa wakeup.\n"));
}

//================================
//        INT0, INT2 callbaks
//================================
void int0D2(void)
{
  ConsolePrint(F("\nINT0 !!!\n"));
}

void int1D3(void)
{
  ConsolePrint(F("\nINT1 !!!\n"));
}


//================================
//   DownLink Data handler
//================================
void port14(void)
{
  ConsolePrint("%s\n", LoRa.getDownLinkData().c_str());
  LedOn();
}

void port15(void)
{
  ConsolePrint("%s\n", LoRa.getDownLinkData().c_str());
  LedOff();
}

PORT_LIST = {
  PORT(14, port14),  // port & callback
  PORT(15, port15),
  END_OF_PORT_LIST
};
```

```
//================================
//   Functions to be executed periodically
//================================

#define LoRa_fPort_TEMP  12
float bme_temp = 10;
float bme_humi = 20;
float bme_press = 50;

short port = LoRa_fPort_TEMP;

int16_t temp = bme_temp * 100;
uint16_t humi = bme_humi * 100;
uint32_t press = bme_press * 100;


/*-------------------------------------------------------------*/
void task1(void)
{
    char s[16];
    ConsolePrint(F("Temperature:  %s degrees C\n"), dtostrf(bme_temp, 6, 2, s));
    ConsolePrint(F("%%RH: %2d%s%%\n"), bme_humi);
    ConsolePrint(F("Pressure: %2d Pa\n"), bme_press);

    disableInterrupt();    //  INT0 & INT1 are disabled

    Payload pl(LoRa.getMaxPayloadSize());
    pl.set_int16(temp);
    pl.set_uint16(humi);
    pl.set_uint32(press);

    LoRa.sendPayload(port, true, &pl);
    LoRa.checkDownLink();

    enableInterrupt();    //  INT0 & INT1 are enabled
}

/*-------------------------------------------------------------*/
void task2(void)
{
    ConsolePrint(F("\n  Task2 invoked\n\n"));
    disableInterrupt();    //  INT0 & INT1 are disabled
    LoRa.sendStringConfirm(port, true, F("%04X%04X%08X"), temp, humi, press);
    LoRa.checkDownLink();
    enableInterrupt();    //  INT0 & INT1 are enabled
}

//================================
//         Execution interval
//   TASK( function, interval by second )
//================================

TASK_LIST = {
        TASK(task1, 0, 15),
        TASK(task2, 8, 15),
        END_OF_TASK_LIST
};


/*  End of Program  */
```