ランニングエレクトロニクス

SBDBT/SBXBT シリーズ用

Bluetooth Low Energy

ペリフェラルサンプルプログラム

ユーザーズマニュアル

2015/05/29 版

RE Running Electronics

目　次

内容

改訂履歴

日付	内容
13/01/30	初版 サンプルプログラム「sbxbt_ble_130126.zip」をリリース。
13/02/01	動作確認済み Bluetooth アダプタを追加
13/02/06	サービスの UUID の記述を間違っていた箇所を修正 サンプルプログラムを実行すると、COM1 に余計なデバッグ出力が出ていたので修正。→サンプルプログラムを「sbxbt_ble_130206.zip」に更新。 「２．本サンプルプログラムの概要」にデバッグ出力について追記。
13/06/13	サンプルプログラムに UART 送受信機能を追加。 また Notification に対応させ「sbxbt_ble_130613.zip」に更新。 それに合わせ本書も全体的に更新。
15/05/29	動作確認済み Bluetooth アダプタを更新 サンプルプログラムを Ver.150529 に更新 ・iOS の更新により接続時に長いパケットが送られくるようになりバッファオーバーフローが発生していた問題を修正。 ・Broadcom Corp. BCM20702A0 (I/O DATA USB-BT40LE)に対応

１．はじめに

本書ではランニングエレクトロニクス SBDBT/SBXBT シリーズ（以下本装置）用、「Bluetooth Low Energy ペリフェラルサンプルプログラム」（以下本サンプルプログラム）の使用方法を説明します。

Bluetooth Low Energy(以下 BLE)は本来低消費電力を目的とした規格ですが、本装置では USB の Bluetooth アダプタを使用している都合上マイコンが常に USB アクセスを行いますので、従来の Bluetooth アダプタを使用するよりは省電力になると思いますが、コイン電池で何年も動作するような事は期待出来ません。

低消費電力性以外にも BLE には使用するメリットがあります。BLE で使用される GATT というベースプロファイルには様々な機器と通信するための仕様が策定されています。ファームウェアでその仕様に合わせた実装を行うことで既存の装置と接続できるようになります。

iPhone や iPad 等の iOS 搭載機器においては従来の Bluetooth によって外部機器と通信を行う場合は MFi ライセンスプログラムに加入する必要がありました。

BLE を使用することによって MFi プログラムに加入することなく Core Bluetooth というフレームワーク、又は techBASIC という開発言語を使用してアプリケーションを作ることによって自由に外部機器と通信することが可能です。

本サンプルプログラムは iOS を使用しての接続テストを行いました。

本サンプルプログラムは様々な事に応用が可能だと思いますので、ユーザー様のご利用の目的に応じて追加・変更してご利用いただきますようお願い致します。

本サンプルプログラムには欠陥が含まれている可能性がありますので、信頼性や正確性を保証することは出来ません。またその欠陥を修正することを保証できません。

本サンプルプログラムご利用の結果についてランニングエレクトロニクスはいかなる責任も負えません。

また、本サンプルプログラムの仕様は予告なく変更する場合がありますので、ランニングエレクトロニクスのサイトを確認して最新の情報を得るようにして下さい。

Bluetooth のプロトコルスタックには Matthias Ringwald 氏の btstack を使用しています。

また、内部にて ChaN 氏の xprintf を利用させていただいています。

すばらしいソフトウェアを公開されている両氏には深く御礼申し上げます。

２．本サンプルプログラムの概要

本サンプルプログラムは、ランニングエレクトロニクス製の SBDBT / SBXBT / SBDBT5V / SBRBT-S / SBRBT-R いずれかのマイコン基板で動作するように開発されました。

また USB 接続の Bluetooth4.0 に対応したアダプタが必要です。現在動作確認できているアダプタは以下のものです。他のアダプタでも動作する可能性があります。もしこの表にないアダプタが動作いたしましたらランニングエレクトロニクスまでお知らせいただけると助かります。

新たに動作確認がとれたアダプタが見つかりましたら本書を更新していきます。

メーカー	型番	参考価格
iBUFFALO	BSHSBD08BK	販売終了
iBUFFALO	BSBT4D09BK※	1,273 円
プラネックス	BT-Micro4※	1,109 円
Logitec	LBT-UAN04C2BK※	1,208 円
サンワサプライ	MM-BTUD40※	1,299 円
Logitec	LBT-UAN04C1BK	1,820 円
Logitec	LBT-UAN04C2BK	1,236 円
GREEN HOUSE	GH-BHDA42	1,280 円
Princeton	PTM-UBT7	1,445 円
I/O DATA	USB-BT40LE	1,530 円

※ユーザー様からのご報告により動作確認済みとさせていただきました。

本サンプルプログラムでは以下の開発環境で開発しました。

開発ツール	Microchip MPLAB IDE v8.88	フリー版
コンパイラ	Microchip MPLAB C30 コンパイラ v3.31	フリー版

テストで使用した相手装置はアップル社製 iPhone5(iOS6.1.2)を使用しました。iPhone4S 以降、iPad3 以降で Bluetooth4.0 対応になったようです。

テスト用アプリとして Punch Through Design LLC 製の LightBlue(Ver1.50)を使用しました。LightBlue は Bluetooth 4.0 Low Energy 用の無料のテストアプリです。AppStore からインストールできます。

UART によるデバッグ出力は COM2 に出力されます。SBDBT/SBDBT5V の場合、10 番ピンに 115200bps で出力されますので USB などに変換して teraterm 等で見ることができます。

３．接続試験

ここでは iOS で LightBlue を使用して接続試験を行う方法を紹介します。LightBlue 以外にもいろいろなテストアプリがありますので、ユーザー様に合った試験方法で接続試験を行なって下さい。

本サンプルプログラムを本装置に書き込んだ後 Bluetooth アダプタを接続し、起動すると赤 LED をと橙 LED が点灯した後、赤 LED のみ消灯した状態になります。

これで接続待機状態です。

この状態で iOS 側で LightBlue のアイコンをタップしてを LightBlue を起動します。

正常に起動しますと右の画面のように「SBDBT BLE TEST」が表示されます。

（初回起動時は名前が表示されていない事があります）

ここで「SBDBT BLE TEST」をタップしますと本装置に接続されます。

本装置の橙 LED が点灯から点滅状態になりますと接続された状態となります。

接続しますと、本サンプルプログラムで定義してあるサービスが表示されます。サービスとは本サンプルプログラムの GATT プロファイル(profile.h)にて定義してある値です。

0x1800 と 0x1801 は名前やアピアランスを定義するためのサービスで、本サンプルプログラムの目的のサービスは 0xFFF0 のサービスとなります。ここで 0xFFF0 をタップしてください。

0xFFF0 のサービスの中には 3 つのキャラクタリスティックがあります。キャラクタリスティックとはサービスの中にある読み込みや書き込みを行う一つの単位です。

まず 0xFFF1 をタップしてください。

この 0xFFF1 のキャラクタリスティックは UART の受信バッファから文字列を取得したり、UART に文字列を送信したりすることができます。

また Notification という機能を使用してポーリングすることなく UART に受信した文字列を取得することもできます。

この Details 画面を開くと LightBlue は一度自動的に Read を行います。受信バッファが空だったときは右のような画面になります。

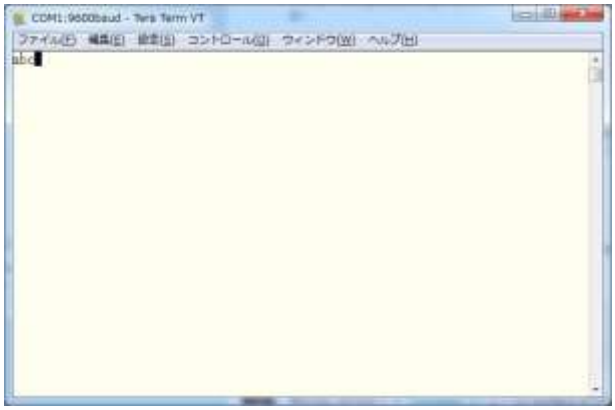
Write をタップするとキャラクタリスティックへ書き込む内容を指定する画面になります。

UART のピンアサインは SPP ファームウェアと同じです。（SBDBT/SBDBT5V では、送信が 7 番ピン、受信が 8 番ピンです。）

もし UART の出力を PC に接続している場合は、teraterm 等のターミナルソフトを使用して 9600bps で開いておいてください。

例えば Write ASCII の枠の中に abc と書き込み、Send をタップしてください。

右のように送信した文字列が受信できると思います。



書き込みが終わりましたら左上の Details をタップし、Details 画面に戻って下さい。

次にteraterm等のターミナルソフトから123等と文字列を送信してください。その後 LightBlue から Read をタップすると、右の画面のように受信した文字列が表示されるはずです。

次に右下の Start Notify を押して下さい。表示は Stop Nofity になるだけで特に変化はありませんが、notification が許可され受信があると受信文字列を知らせてくるようになります。

Teraterm 等のターミナルソフトから 000 等と文字列を送信してください。右のような画面になると思います。これは 0 を 1 バイト受信した瞬間にその 1 バイトを Notification で送ってしまうため 1 バイトだけの表示になってしまっていますが、もっと早いタイミングで送れば 2 バイト、3 バイトと 20 バイトまでが一つのパケットで送られます。

Notification はレスポンスの受信がないため、送信しても受信されないことがあります。特にあまり早く送ってしまうと受信側が取りこぼすことがあります。そのあたりは用途に応じて使用して下さい。

Stop Notify をタップし、左上の Back をタップしてサービスの画面に戻って下さい。

次に、0xFFF2 のキャラクタリスティックをタップしてください。

0xFFF2 は 8 ビット値を読み書きできるキャラクタリスティックになっています。初期状態では 0x00 が入っています。ここで Write をタップしてください。

Write 画面で HEX のほうに 01 を入力してください。

赤 LED が点灯すると思います。

次に 00 を書き込んでみて下さい。

赤 LED が消灯すると思います。

このように値によって制御することができます。

次に 00001234-0000-1000-8000-00805F9B34FB というキャラクタリスティックをタップしてください。このキャラクタリスティックは 16bit 値を読み込むためのキャラクタリスティックで本サンプルプログラム内部のタイマ変数を読み込むことができます。

この変数は 1ms ごとにインクリメントしているので読みこむたびに違った値が読めると思います。

書き込みはできません。

このように LightBlue を使用して簡単に試験を行うことができます。

実際にはアプリを作成した場合はアプリの処理で接続し、アプリの処理でキャラクタリスティックにアクセスするということになると思います。

４．GATT プロファイルについて

本サンプルプログラムで使用した GATT プロファイルは profile.h に定義されていますが、これは profile.gatt ファイルを compile-gatt.py でコンパイルしたものです。

profile.gatt は以下の様な記述がしてあります。

```
0001:  PRIMARY_SERVICE, GAP_SERVICE
0002:  CHARACTERISTIC, GAP_DEVICE_NAME, READ, "SBDBT BLE TEST"
0003:  CHARACTERISTIC, GAP_APPEARANCE, READ, 00 00
0004:
0005:  PRIMARY_SERVICE, GATT_SERVICE
0006:  CHARACTERISTIC, GATT_SERVICE_CHANGED, READ,
0007:
0008:  PRIMARY_SERVICE, FFF0
0009:
0010:  #UART
0011:  CHARACTERISTIC, FFF1, NOTIFY | READ | WRITE | DYNAMIC,
0012:
0013:  #LED
0014:  CHARACTERISTIC, FFF2, READ | WRITE | DYNAMIC,
0015:
0016:  #TIMER
0017:  CHARACTERISTIC, 00001234-0000-1000-8000-00805F9B34FB, READ | DYNAMIC,
```

profile.gatt

左の 4 桁の数字と「:」は行番号を表しているだけですので、ファイルには含まれていません。

6 行目まではサービスの名前やアピアランスの定義で、本来の目的のサービスは 0xFFF0 なので 8 行目からの記述になります。

profile.gatt を profile.h にコンパイルするためには Python が必要です。私は cygwin の python を使用しました。

python compile-gatt.py profile.gatt profile.h

と実行することでコンパイルができます。

キャラクタリスティックにアクセスされると、ble_server.c 内のコールバック関数が呼ばれます。

読み込み時は att_read_callback 関数、書き込み時は att_write_callback 関数です。

関数内で使用しているキャラクタリスティックを判別しているハンドルは、compile-gatt.py 内で 1 から順番に割り振っています。サービスの定義では 1、キャラクタリスティックの定義では UUID で 1 つ、値で 1 つ増加されます。また、NOTIFY の属性が

付くと CHARACTERISTIC CONFIGURATION というハンドルが自動的に負荷されますのでそこでも 1 つ増加します。

さきほどのプロファイルをコンパイルすると、以下の profile.h が生成されます。

```
const uint8_t profile_data[] =
{
    // 0x0001 PRIMARY_SERVICE-GAP_SERVICE
    0x0a, 0x00, 0x02, 0x00, 0x01, 0x00, 0x00, 0x28, 0x00, 0x18,
    // 0x0002 CHARACTERISTIC-GAP_DEVICE_NAME-READ
    0x0d, 0x00, 0x02, 0x00, 0x02, 0x00, 0x03, 0x28, 0x02, 0x03, 0x00, 0x00, 0x2a,
    // 0x0003 VALUE-GAP_DEVICE_NAME-READ-"SBDBT BLE TEST"
    0x16, 0x00, 0x02, 0x00, 0x03, 0x00, 0x00, 0x2a, 0x53, 0x42, 0x44, 0x42, 0x54, 0x20, 0x42,
0x4c, 0x45, 0x20, 0x54, 0x45, 0x53, 0x54,
    // 0x0004 CHARACTERISTIC-GAP_APPEARANCE-READ
    0x0d, 0x00, 0x02, 0x00, 0x04, 0x00, 0x03, 0x28, 0x02, 0x05, 0x00, 0x01, 0x2a,
    // 0x0005 VALUE-GAP_APPEARANCE-READ-00 00
    0x0a, 0x00, 0x02, 0x00, 0x05, 0x00, 0x01, 0x2a, 0x00, 0x00,
    // 0x0006 PRIMARY_SERVICE-GATT_SERVICE
    0x0a, 0x00, 0x02, 0x00, 0x06, 0x00, 0x00, 0x28, 0x01, 0x18,
    // 0x0007 CHARACTERISTIC-GATT_SERVICE_CHANGED-READ
    0x0d, 0x00, 0x02, 0x00, 0x07, 0x00, 0x03, 0x28, 0x02, 0x08, 0x00, 0x05, 0x2a,
    // 0x0008 VALUE-GATT_SERVICE_CHANGED-READ-
    0x08, 0x00, 0x02, 0x00, 0x08, 0x00, 0x05, 0x2a,
    // 0x0009 PRIMARY_SERVICE-FFF0
    0x0a, 0x00, 0x02, 0x00, 0x09, 0x00, 0x00, 0x28, 0xf0, 0xff,
//UART
    // 0x000a CHARACTERISTIC-FFF1-NOTIFY | READ | WRITE | DYNAMIC
    0x0d, 0x00, 0x02, 0x00, 0x0a, 0x00, 0x03, 0x28, 0x1a, 0x0b, 0x00, 0xf1, 0xff,
    // 0x000b VALUE-FFF1-NOTIFY | READ | WRITE | DYNAMIC-
    0x08, 0x00, 0x1a, 0x01, 0x0b, 0x00, 0xf1, 0xff,
    // 0x000c CLIENT CHARACTERISTIC CONFIGURATION
    0x09, 0x00, 0x0a, 0x01, 0x0c, 0x00, 0x02, 0x29, 0x00,
//LED
    // 0x000d CHARACTERISTIC-FFF2-READ | WRITE | DYNAMIC
    0x0d, 0x00, 0x02, 0x00, 0x0d, 0x00, 0x03, 0x28, 0x0a, 0x0e, 0x00, 0xf2, 0xff,
    // 0x000e VALUE-FFF2-READ | WRITE | DYNAMIC-
    0x08, 0x00, 0x0a, 0x01, 0x0e, 0x00, 0xf2, 0xff,
//TIMER
    // 0x000f CHARACTERISTIC-00001234-0000-1000-8000-00805F9B34FB-READ | DYNAMIC
    0x1b, 0x00, 0x02, 0x00, 0x0f, 0x00, 0x03, 0x28, 0x02, 0x10, 0x00, 0xfb, 0x34, 0x9b, 0x5f,
0x80, 0x00, 0x00, 0x80, 0x00, 0x10, 0x00, 0x00, 0x34, 0x12, 0x00, 0x00,
    // 0x0010 VALUE-00001234-0000-1000-8000-00805F9B34FB-READ | DYNAMIC-
    0x16, 0x00, 0x02, 0x03, 0x10, 0x00, 0xfb, 0x34, 0x9b, 0x5f, 0x80, 0x00, 0x00, 0x80, 0x00,
0x10, 0x00, 0x00, 0x34, 0x12, 0x00, 0x00,
    // END
    0x00, 0x00,
}; // total size 211 bytes
```

生成された profile.h

生成された profile.h のコメントを見ると、上記プロファイルでは

・キャラクタリスティック 0xFFF1 の値(VALUE)はハンドル 0x000b
・キャラクタリスティック 0xFFF2 の値(VALUE)はハンドル 0x000e
・キャラクタリスティック 00001234-0000-1000-8000-00805F9B34FB の値(VALUE)はハンドル 0x0010

ということがわかります。

プログラムではこのハンドルの値を使用してコールバックルーチンの処理を行なって下さい。

５．ライセンス

本ファームウェアには、ランニングエレクトロニクスによるプログラムの他に、Matthias Ringwald 氏による Bluetooth プロトコルスタック(btstack)が含まれています。

btstack のウェブサイトで作者 Matthias Ringwald 氏は btstack を商用利用する際は連絡してほしいと記載されています。
ランニングエレクトロニクス製の本サンプルプログラム対応基板は Matthias Ringwald 氏とライセンス契約を締結していますので、本ファームウェアを商用利用する際にもご連絡や追加ロイヤリティは必要ありません。

またランニングエレクトロニクスにて作成した部分、btstack に変更を加えている部分に関しても btstack と同じライセンスとさせていただきます。

本ファームウェアのソースコードを非商用でご利用になる場合、Copyright 表記を消さずに本ファームウェアを使用していることを明記すれば使用できます。ランニングエレクトロニクス製マイコン基板上でご利用になる場合は、非商用・商用にかかわらずご利用いただけます。他の装置に流用して商用利用する場合、ランニングエレクトロニクスにご相談ください。

８．サポート

当店のウェブサイトにサポート掲示板が設置してあります。
ご質問やご要望等ありましたらお気軽に書き込みください。