Technical Note

TecStar

Silicon Labs 社 BGM1xx
アドバンストガイド（上級編）

2019 年 4 月

株式会社マクニカ
テクスター カンパニー

TecStar
Company

Silicon Labs 社 BGM1xx

アドバンストガイド（上級編）

目次

1 はじめに

この資料は、Silicon Laboratories（以下、Silicon Labs）社製 Bluetooth®モジュール BGM1xx の使用方法について簡易にまとめたものです。内容に誤りがないよう注意は払っておりますが、もし Silicon Labs 社が提供するドキュメント等と差異がございましたら、メーカー提供のものを優先してご参照ください。

本資料は、基本的な使用方法は理解頂いている方を対象とした**アドバンストガイド（上級編）**です。初めて BGM をご使用になる方は、まず**クイックスタートガイド（初級編）**をご参照ください。

2 デバッグする

2-1 printf デバッグ

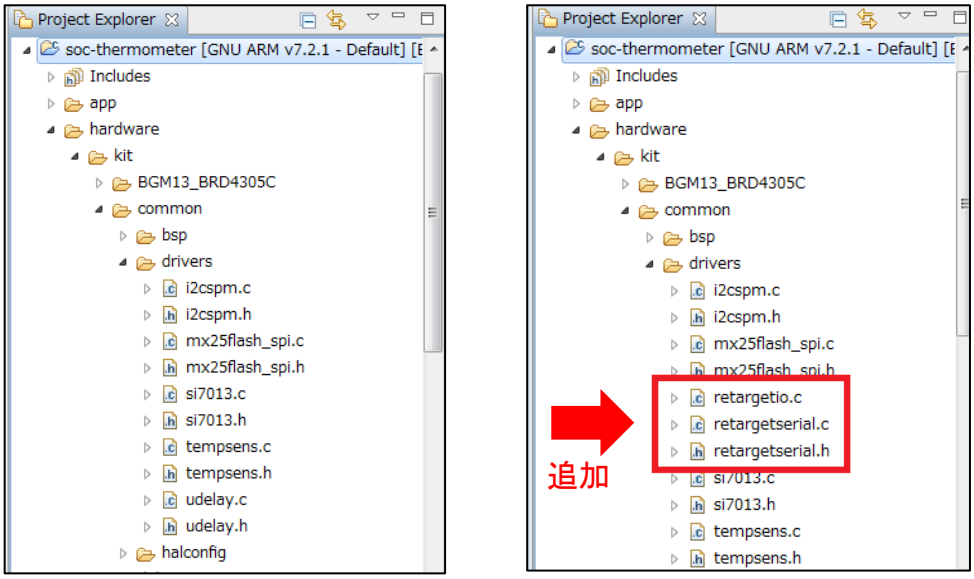
BGM1xx のソフトウェア開発では、break point や step 実行を活用したデバッグを行うこともできますが、Bluetooth 通信中に MCU を停止させてしまうと通信が切れてしまいますので、printf デバッグが非常に効果的です。

ここでは、サンプルコードを例に、printf を実装する手順をご紹介します。

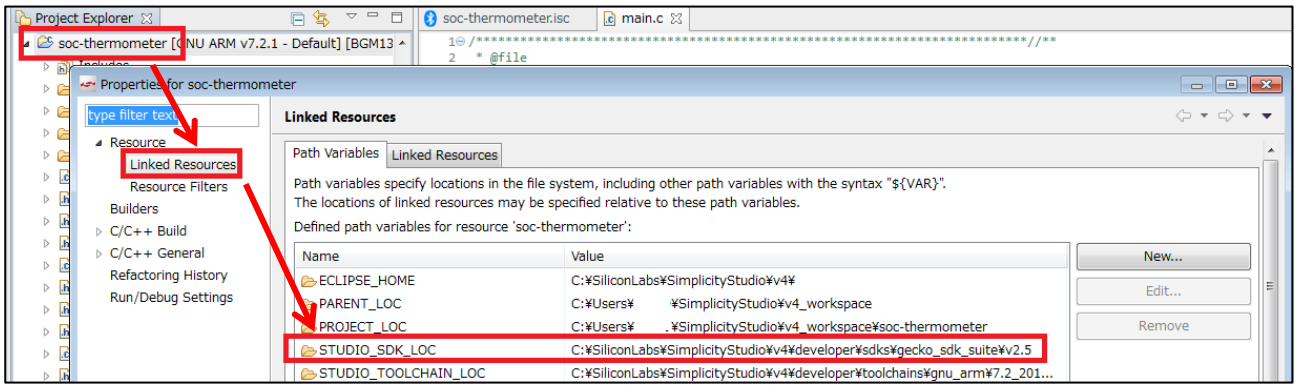
① プロジェクトを作成します。ここでは、“SOC – Thermometer”を使用します。

② プロジェクトの hardware/kit/common/drivers に、retargetio.c, retargetserial.c, retargetserial.h が含まれているかを確認します。

含まれていなければ、、これらのファイルをプロジェクトにコピー(ドラッグアンドドロップ)します。これらのファイルは、“STUDIO_SDK_LOC¥hardware¥kit¥common¥drivers” にあります。



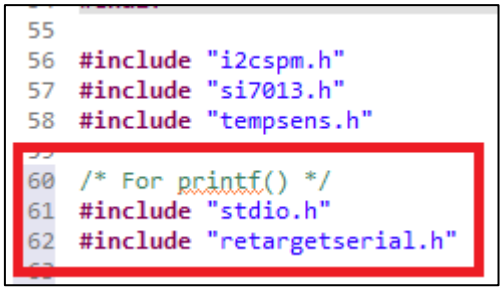
なお、STUDIO_SDK_LOC の位置は、Project Explorer でプロジェクトを選択して右クリック→Property→Resource→Linked Resources の順で確認できます。



③ main.c に、stdio.h と retargetserial.h を include します。

＜記述＞

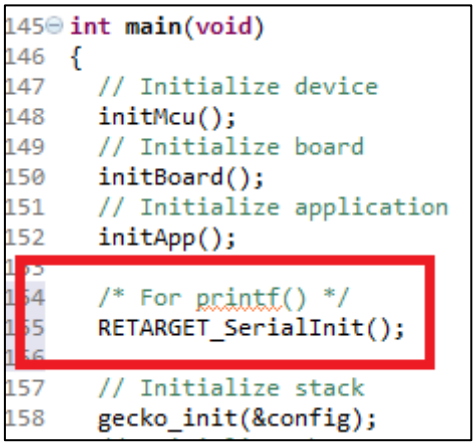
```
#include "stdio.h"
#include "retargetserial.h"
```



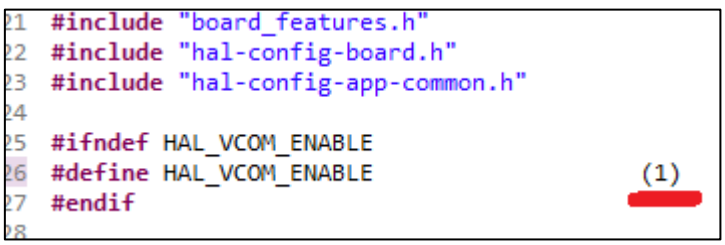
④ RETARGET_SerialInit(); を追加します。初期化の関数ですので、Printf などを使用する前に実施が必要です。ここでは initApp() の直後に入れてみます。

＜記述＞

```
RETARGET_SerialInit();
```



⑤ hal-config.h を開き、HAL_VCOM_ENABLE の値を 1 に変更します。



⑥ printf を使った記述を行います。RETARGET_SerialInit();より後に行ってください。

＜記述＞

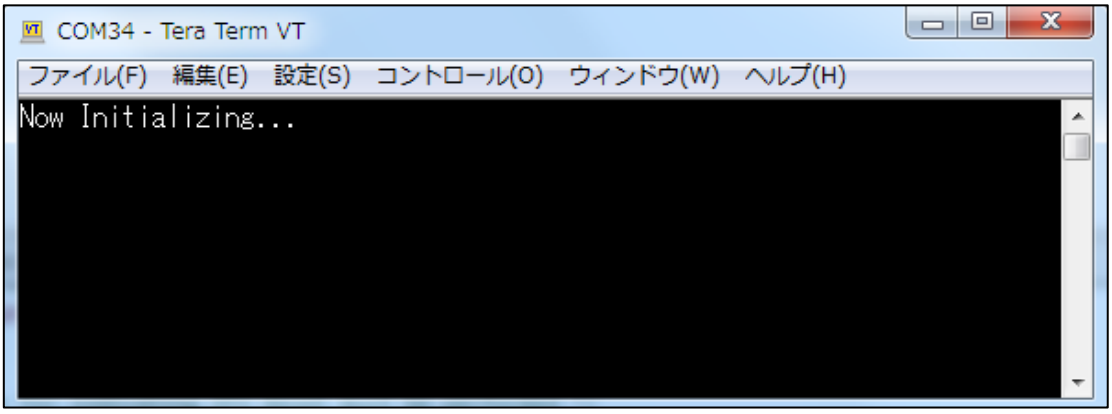
printf("Now Initializing...¥r¥n");

```
54   /* For printf() */
55   RETARGET_SerialInit();
56
57   printf("Now Initializing...\r\n");
58
59   // Initialize stack
60   gecko_init(&config);
```

⑦ ビルドし、BGM にダウンロードします。

⑧ ターミナルソフト（Tera Term など）を使用し、シリアルポート（JLink CDC）をオープンします。UART の設定は、ボーレート 115200, data 8bit, non parity, 1 stop bit としてください。

⑨ メインボードをリセットすると、ターミナルソフト上に文字が表示されます。

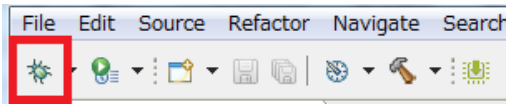


hal-config.h が含まれていない古い SDK をご使用の場合には、幾つかの追加手順が必要です。詳しくはクイックスタートガイド v1.9 をご参照ください。

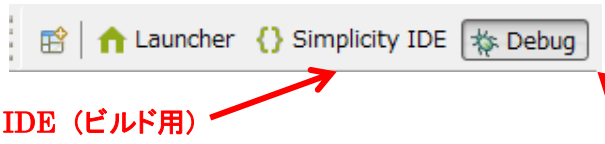
また、シリコンラボ社のナレッジベースにも情報がございますので、こちらも参照ください。（リンク）

2-2 MCU デバッグ機能

プロジェクトをビルドした後、Flash Programmer ではなく Debug アイコンを使ってダウンロードすると、ブレークポイント、ステップ実行などのソフトウェア・デバッグの機能がご使用になれます。

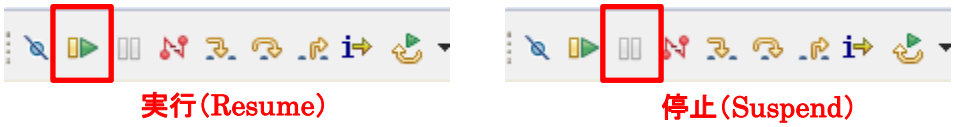


ビルド用の画面（Simplicity IDE）と、デバッグ用の画面（Debug）は右上のアイコンで切り替えます。



◆ コードの実行・停止

コードの実行には Resume ボタン、停止には Suspend ボタンを使用します。



◆ ハードウェア・リセット

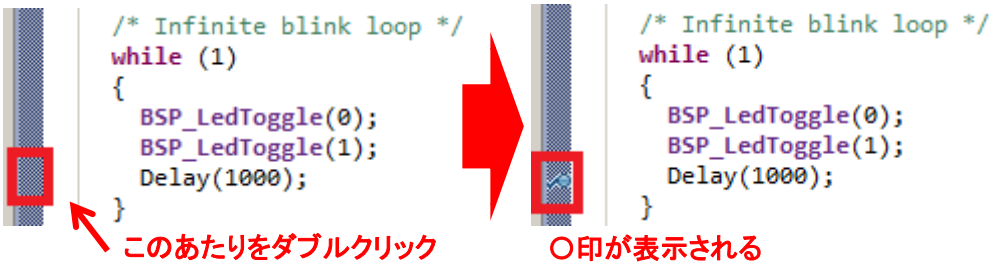
MCU にハードウェア・リセットをかけます。

◆ デバッグ経路の切断

デバッグモード経路を切断して、デバッグ用の画面を終了します。ビルド用の画面に切り替わります。

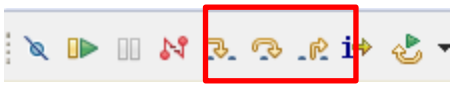
◆ ブレークポイント

ブレークポイントを設定するには、停止させたい行の左横をダブルクリックします。設定されると、水色の小さな○印が表示されます。再度ダブルクリックすれば解除されます。

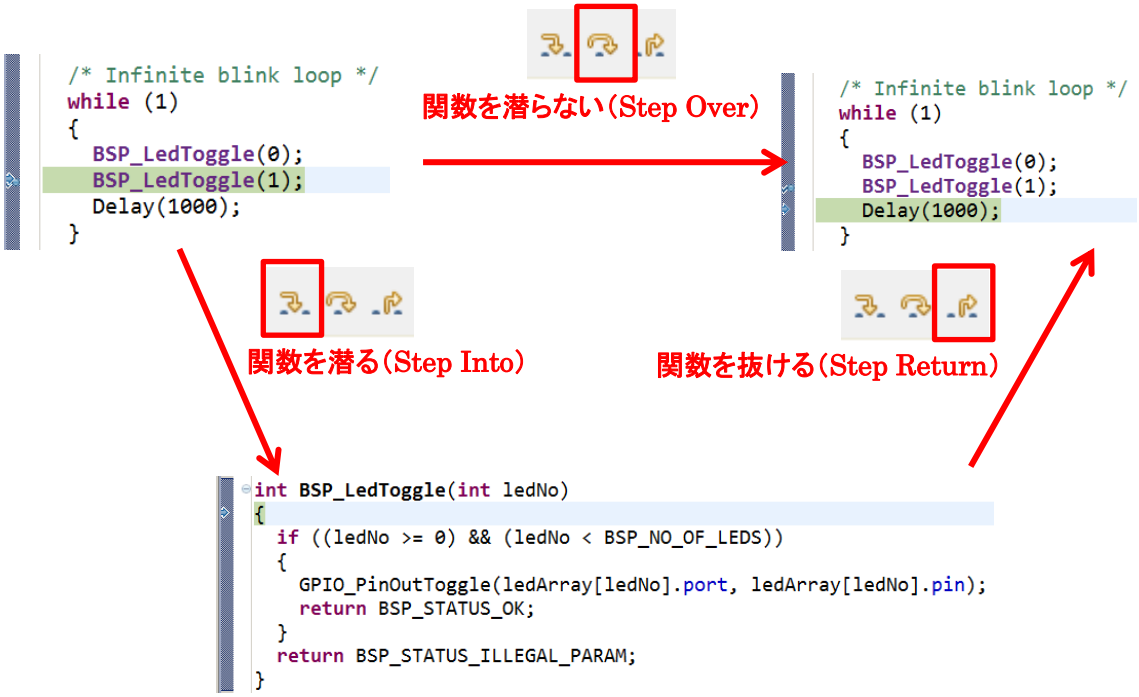


◆　ステップ実行

各種ステップ実行に対応しています。

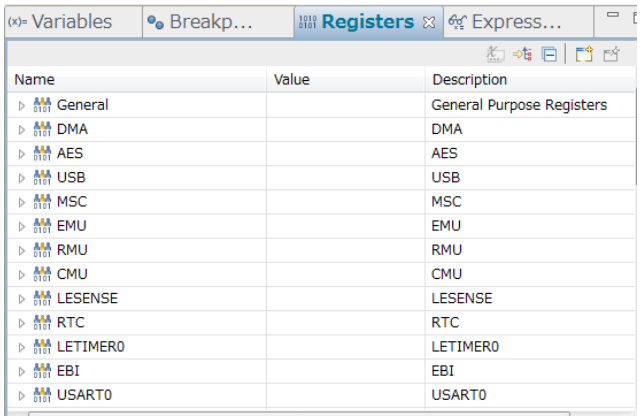


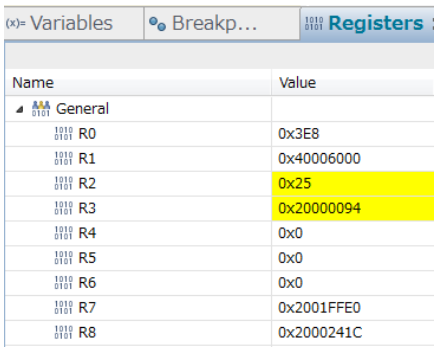
実機で実際に動作を見て頂くのが、判りやすいです。



◆　レジスタ値の閲覧・変更

レジスタ・変数の閲覧や変更は、下のウィンドウ（Register ウィンドウなど）で行うことができます。前回の停止から、値が変化した場合には黄色で表示されます。

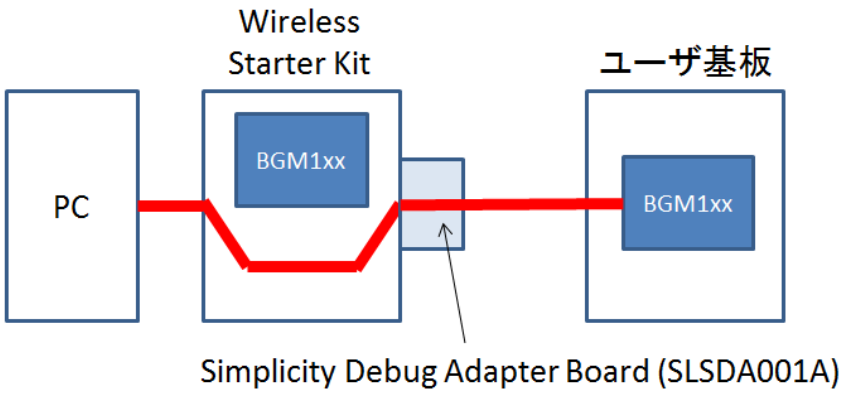




＊上記の説明では、EFM32 向けのコードを使用しています。BGM1xx でも手順は同じです。

2-3 ユーザ基板のデバッグ

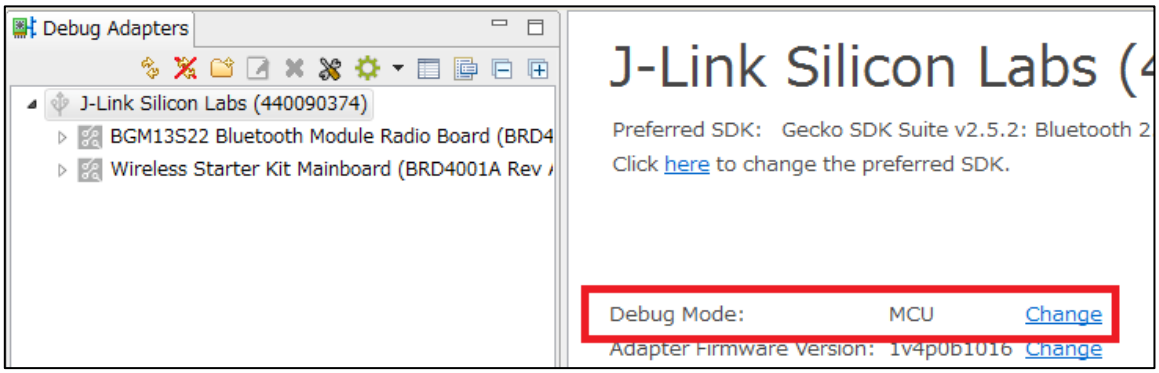
メインボードを使用することで、ラジオボード上の BGM1xx だけでなく、ユーザ基板上の BGM1xx に対してもプログラミングやデバッグを行うことが可能です。メインボードとユーザ基板の接続には、Simplicity Debug Adaptor Board（SLSDA001A）が便利です。



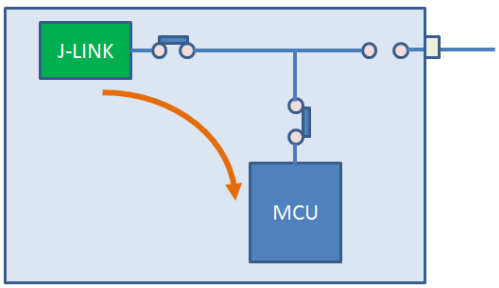
2-3-1 デバッグ対象の切り替え

デバッグ対象を、ラジオボード上の BGM1xx から、ユーザ基板上の BGM1xx に切り替えます。

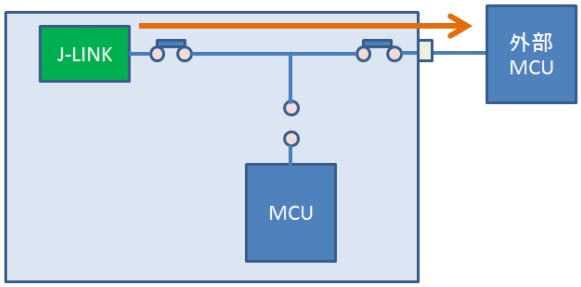
Simplicity Studio の Debug Adapters タブで Wireless Starter Kit を選択すると、画面右に現在の Debug Mode が表示されます。下図では MCU の設定になっています。



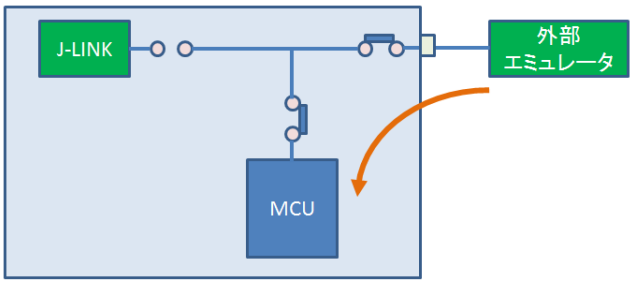
この Debug Mode は、MCU、OUT、IN、OFF の 4 設定があり、ラジオボードをデバッグする際には MCU を、ユーザ基板をデバッグする際には OUT を選択します。



MCU：
Starter Kit 上のラジオボードがデバッグ対象

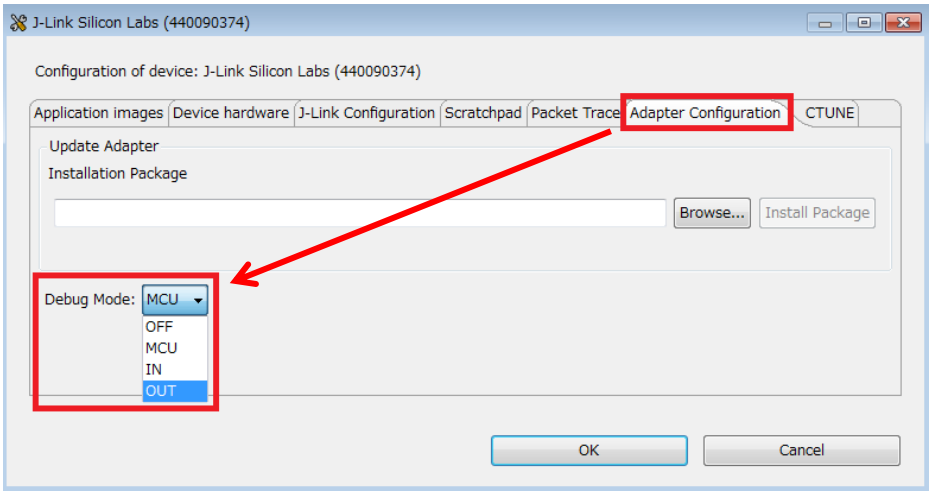


OUT:
外部のユーザボードがデバッグ対象

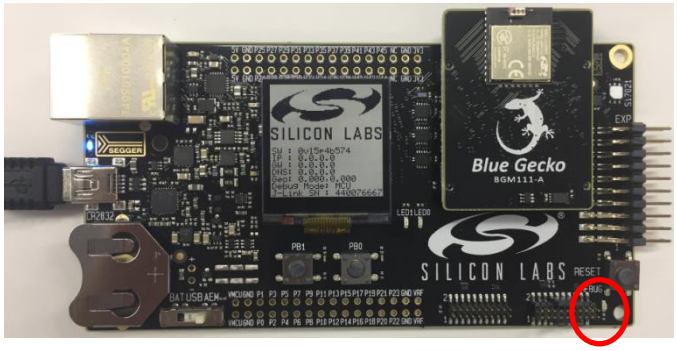


IN:
外部エミュレータを使って、StarterKit 上のラジオボードをデバッグ

Change をクリックし、Adapter Configuration タブの Debug Mode で、OUT を選択します。



デバッグ対象がラジオボードから外部（ユーザ基板）に切り替わると、メインボード右下の DEBUG OUT という LED が点灯します。

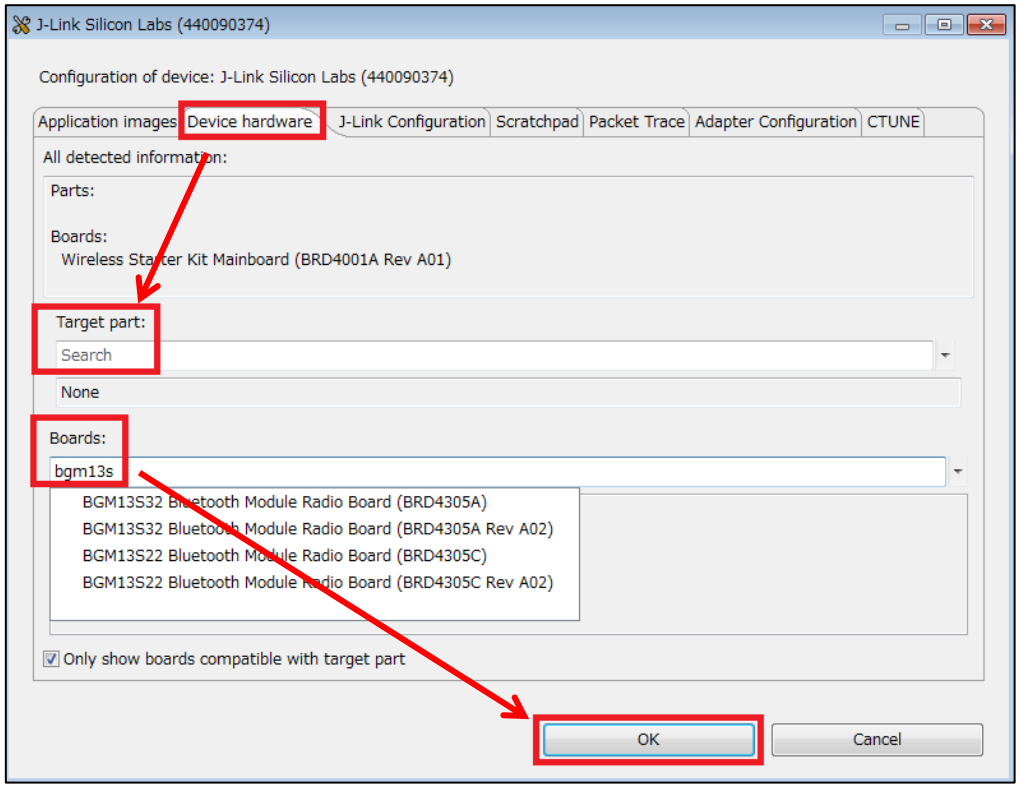


2-3-2 モジュール型番の指定

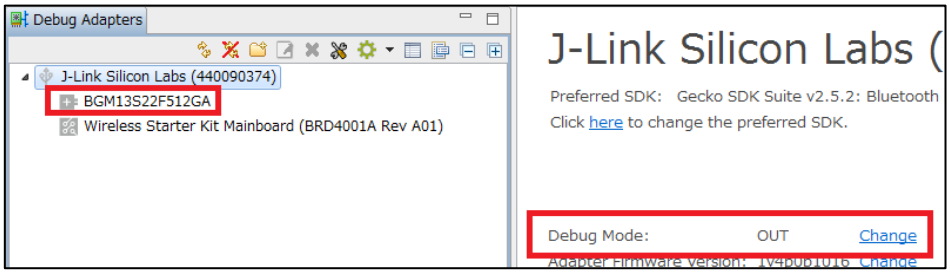
ラジオボードを使用している場合には、使用しているモジュール型番を自動で認識してくれます。しかし、ユーザ基板上のモジュール型番は自動認識されませんので、指定する必要があります。

Device hardware タブの Boards で BGM1xx のラジオボードを選択するか、或いは Target part でモジュール型番（BGM13S など）を選択し、OK をクリックします。以下は注意点です。

- Target part を選んだ場合には、ラジオボード用のデモは表示されません。
- Boards に使用しないボードが登録されている場合には、×をクリックして消去してください。
- Boards でリストアップされる候補が少ない場合には、Target part を None にしてお試しください。



登録が完了すると、Device Adapter タブに指定したモジュール（或いはラジオボード）が追加されます。あとは、Debug Mode = MCU 時のラジオボードと同じ要領で、ご使用頂けます。下図は、Target part で BGM13S を選択した場合です。

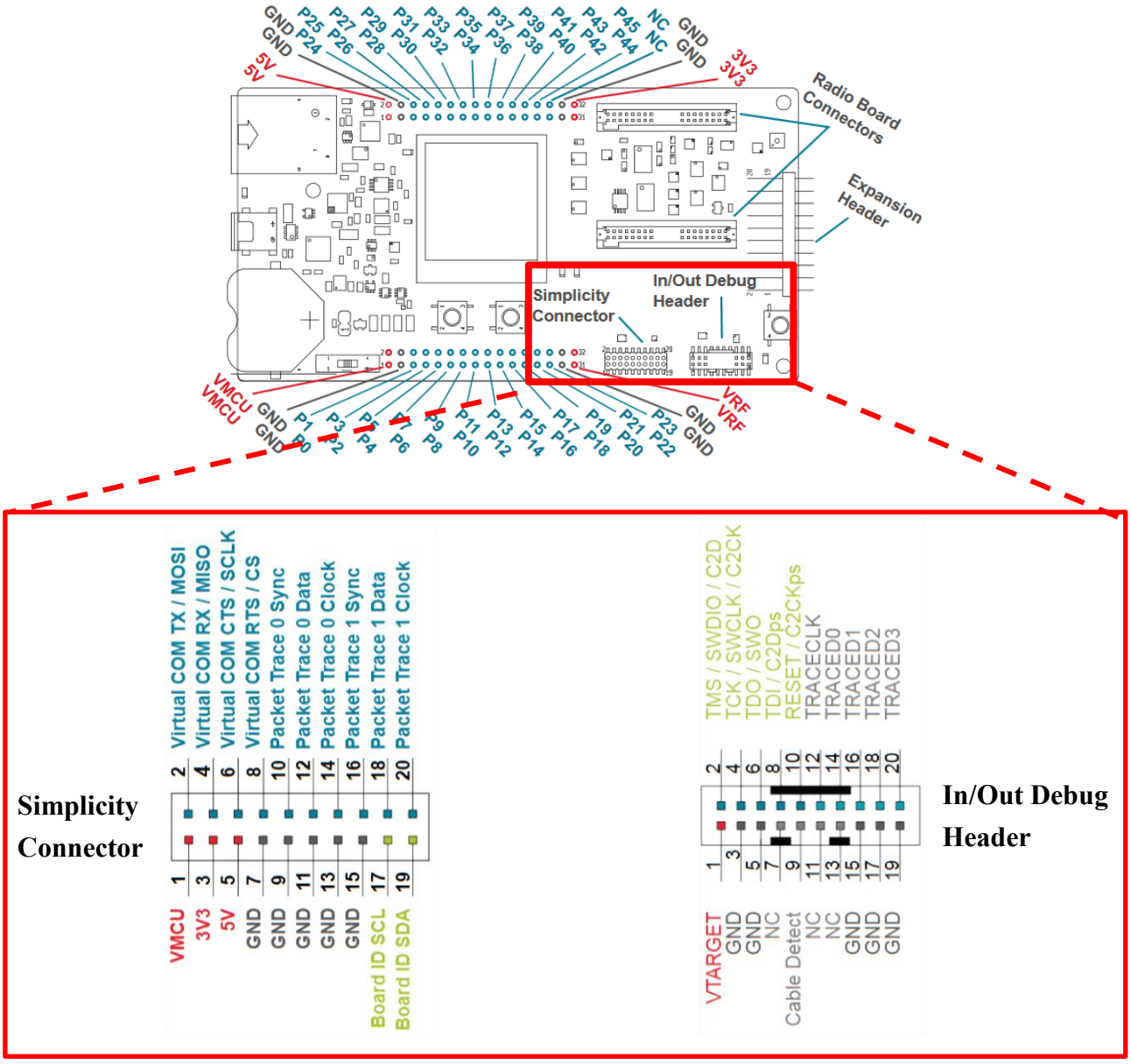


2-4 ユーザ基板との接続

メインボードとユーザ基板の接続方法については、AN958「Debugging and Programming Interfaces for Custom Designs」の中で詳しく説明されています。(リンク)

何通りか方法はありますが、Mini Simplicity Connector を介して接続するのがお勧めです。

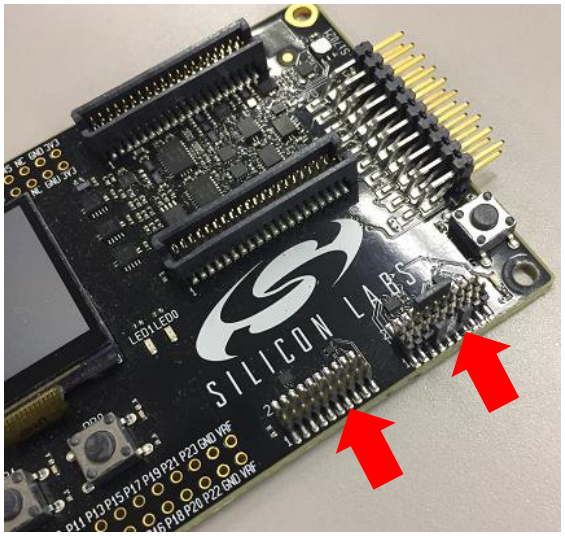
メインボードの右下に、In/Out Debug Header と Simplicity Connector の 2 つのコネクタがあります。

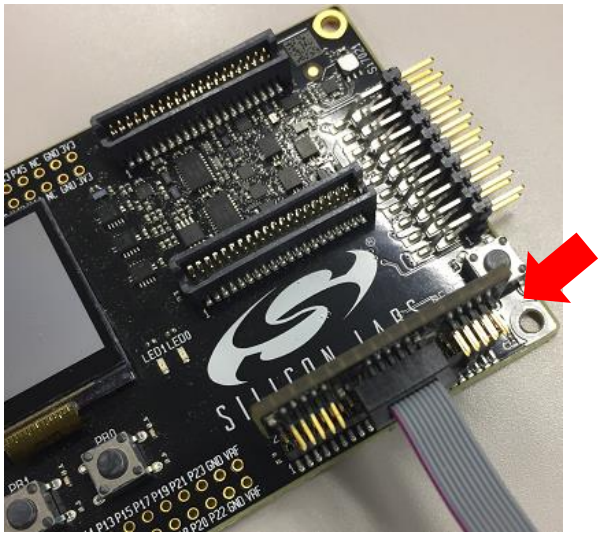


In/Out Debug Header は、arm CPU 用の標準的なデバッグインタフェースです。ダウンロードやデバッグに必要な SWCLK、SWDIO などが割り当てられています。

一方、Simplicity Connector には、printf デバッグなどで使用する仮想 COM ポートや、パケットトレース機能が割り当てられています。

2 つのコネクタに跨るように、Simplicity Debug Adaptor Board（SLSDA001A）を挿入します。





Simplicity Debug Adaptor Board 上にも 10 ピンコネクタが 3 つ並んでいますが、中央の Mini Simplicity Connector に接続します。2 つのコネクタは、ダウンロードに必要な SW インタフェース、printf デバッグに必要な VCOM などを含む 10 ピンに変換されます。

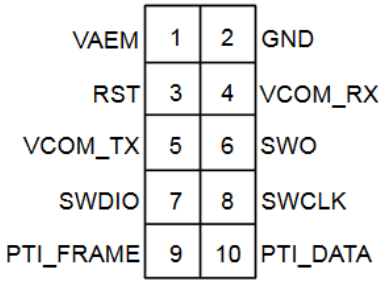


Figure 4.2. Mini Simplicity Connector Pin-Out

Pin #	Pin Name	Pin Function	EFR32 Functionality
1	VAEM	Target Advanced Energy Monitor Voltage Net	VDD
2	GND	Target Ground	VSS
3	RST	Target Reset (Active Low)	RESETn
4	VCOM_RX	Target Pass-through UART/Virtual COM Port Receive	US0_RX
5	VCOM_TX	Target Pass-through UART/Virtual COM Port Transmit	US0_TX
6	SWO	Target Serial Wire Output	SWO
7	SWDIO	Target Serial Wire Data Input/Output	SWDIO
8	SWCLK	Target Serial Wire Clock	SWCLK
9	PTI_FRAME	Target Packet Trace Interface Frame Signal	FRC_DFRAME
10	PTI_DATA	Target Packet Trace Interface Data Signal	FRC_DOUT

信号名称	補足コメント
VAEM	メインボードとユーザ基板とで信号レベルが異なることを想定し、信号線にはレベルシフタが入っています。そのレベルシフタに、ユーザ基板の信号レベル（電源電圧）を伝えるためのピンです。通常はユーザ基板の電源に接続します。
GND	メインボードとユーザ基板の GND を同一レベルにするためのピンです。ユーザ基板の GND に接続してください。
RST	リセットピンです。デバッグ経路が遮断されてしまった場合の復旧手段として使用します。BGM1xx の RESETn に接続してください。
VCOM_RX	メインボードには USB-UART 変換機能が付いており、その UART が VCOM ピンから外部に出ています。printf デバッグや、NCP モードの評価に使用します。BGM1xx の UART ピン（RX）に接続してください。
VCOM_TX	メインボードには USB-UART 変換機能が付いており、その UART が VCOM ピンから外部に出ています。printf デバッグや、NCP モードの評価に使用します。BGM1xx の UART ピン（TX）に接続してください。
SWO	デバッグ用のインタフェースです。Energy Profiler でコード連携機能を使用する場合には、BGM1xx の SWO に接続してください。使用は必須ではありません。
SWDIO	ダウンロード・デバッグ用のインタフェースです。BGM1xx の SWDIO に接続してください。
SWCLK	ダウンロード・デバッグ用のインタフェースです。BGM1xx の SWCLK に接続してください。
PTI_FRAME	パケットトレース機能を使用する際に接続ください。
PTI_DATA	パケットトレース機能を使用する際に接続ください。

3 機能・性能を評価する

3-1 RF PHY の特性を評価する

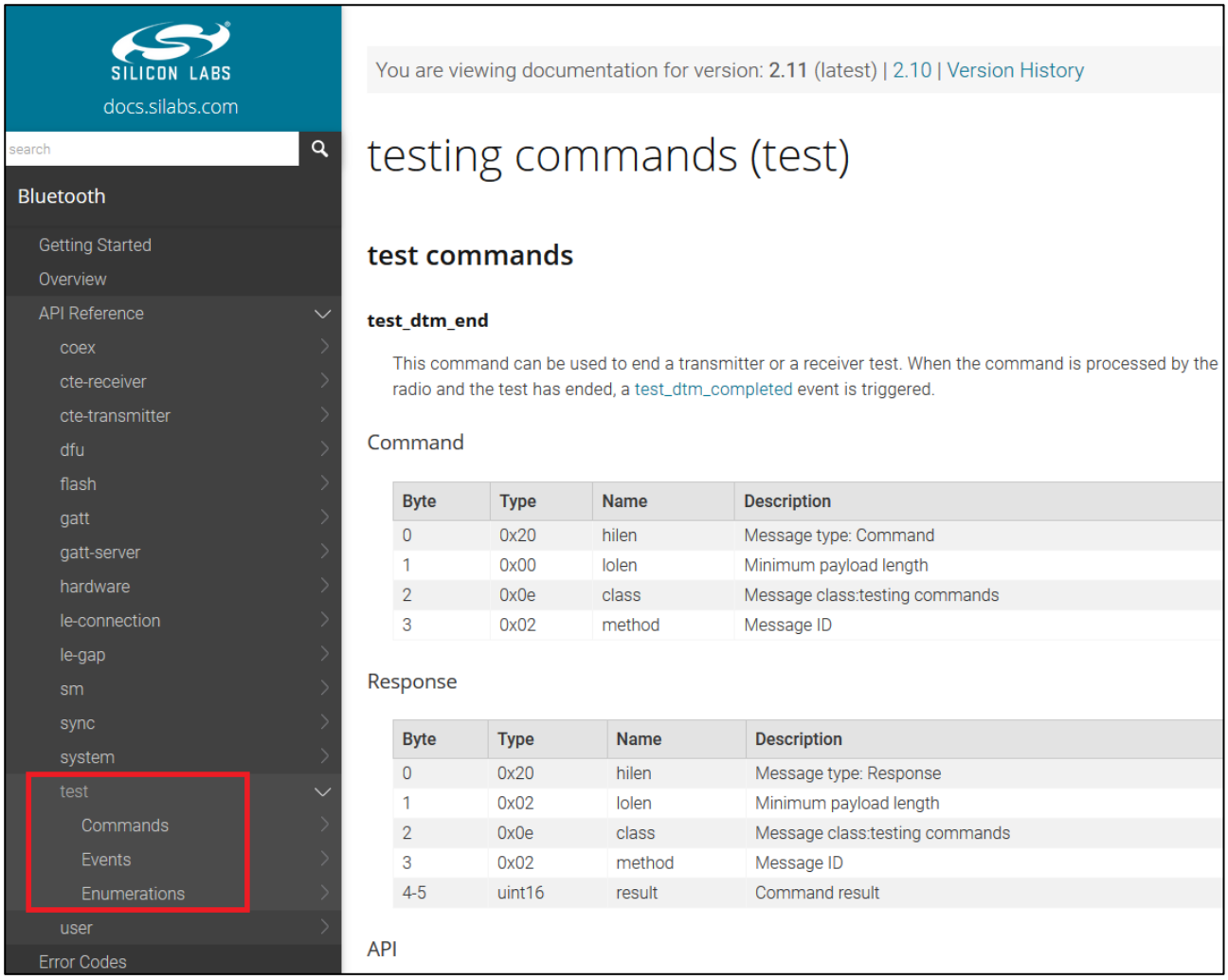
BGM1xx および Bluetooth スタックには、無線試験を想定した機能が実装されています。ここでは手順を簡単にご紹介します。

詳しくはドキュメントが用意されておりますので、AN1046 をご覧ください（リンク）。本章で扱わない DTM（direct test mode）を使用した手順も紹介されています。

3-1-1 テストコマンドを使用する

Bluetooth スタックには、テスト用のコマンドが用意されています。ユーザコードからコマンド実行することで、BGM1xx に特定のテスト用動作をさせることができます。

使用できるコマンドについては、API リファレンス・マニュアルに記載されています。



１例として送信コマンドをご紹介します。各コマンドに対するレスポンスや、その他コマンドについては、API リファレンス・マニュアルをご参照ください。

- 送信コマンド（cmd_test_dtm_tx）

使用するチャネルや PHY タイプ（1M PHY, 2M PHY など）、送信するパケットタイプ（無変調，特定データパターンなど）、パケット長を指定して、送信することができます。

Byte	Type	Name	Description
0	0x20	hilen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0e	class	Message class: testing commands
3	0x00	method	Message ID
4	uint8	packet_type	Packet type to transmit
5	uint8	length	Packet length in bytes **Range:** 0-255
6	uint8	channel	Bluetooth channel **Range:** 0-39 Channel is (F - 2402) / 2, where F is frequency in MHz
7	uint8	phy	PHY to use

packet_type:

Value	Name	Description
0	test_pkt_prbs9	PRBS9 packet payload
1	test_pkt_11110000	11110000 packet payload
2	test_pkt_10101010	10101010 packet payload
3	test_pkt_carrier_deprecated	Unmodulated carrier - deprecated
4	test_pkt_11111111	11111111 packet payload
5	test_pkt_00000000	00000000 packet payload
6	test_pkt_00001111	00001111 packet payload
7	test_pkt_01010101	01010101 packet payload
253	test_pkt_pn9	PN9 continuously modulated output
254	test_pkt_carrier	Unmodulated carrier

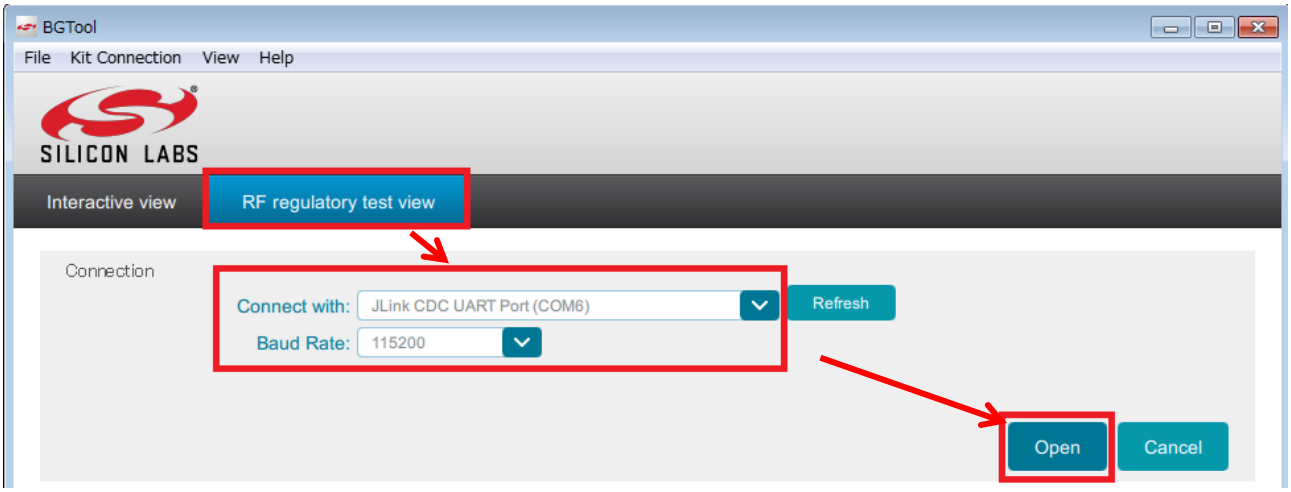
Phy:

Value	Name	Description
1	test_phy_1m	1M PHY
2	test_phy_2m	2M PHY
3	test_phy_125k	125k Coded PHY
4	test_phy_500k	500k Coded PHY

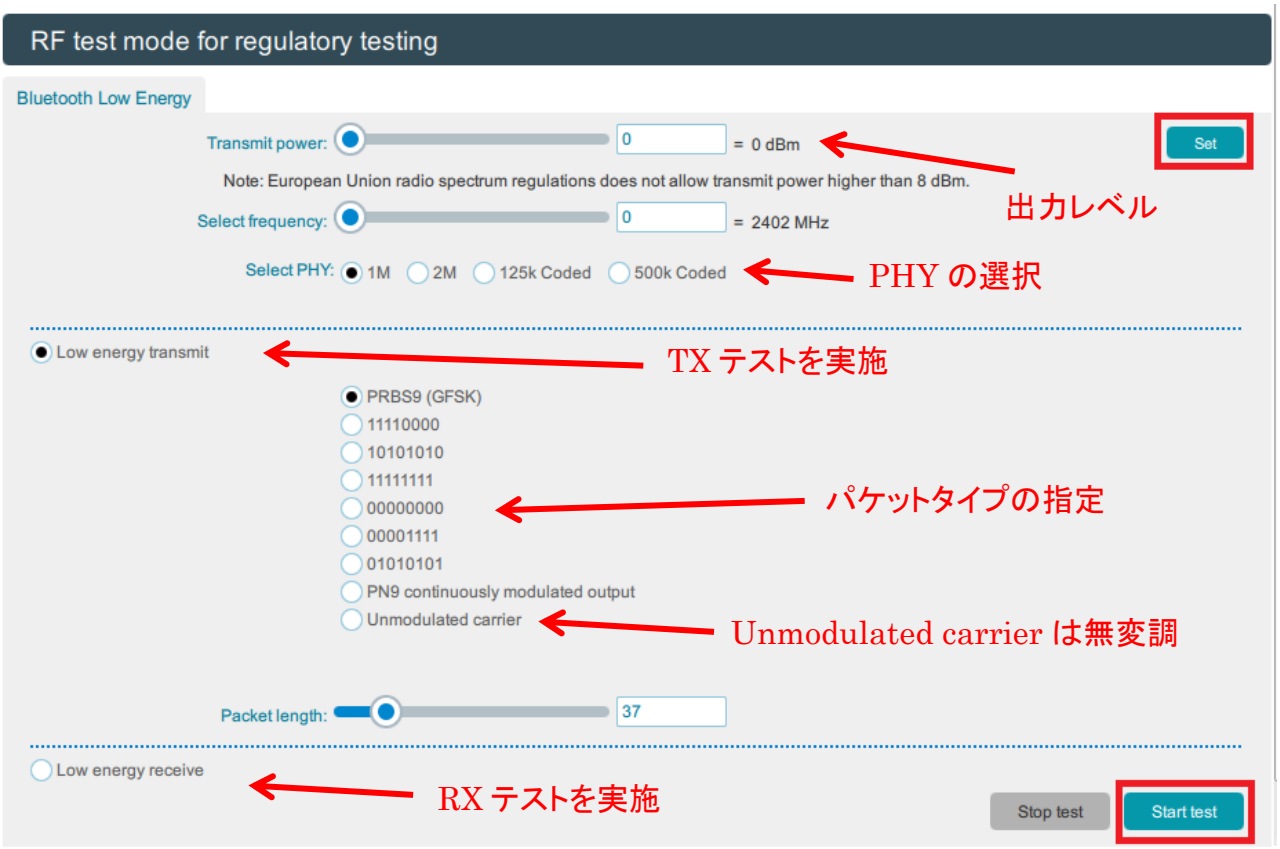
3-1-2 BGTool を使用する

Bluetooth スタックに用意されたテストコマンドは、ユーザコードから実行するだけでなく、BGTool からも使用できます。BGTool を使用するための手順については**クイックスタートガイド**をご参照ください。

BGTool で RF regulatory test view タブを選び、Connect with で JLink CDC UART Port （仮想 COM です）を選択し、ボーレートは 115200 に設定して、Open をクリックします。

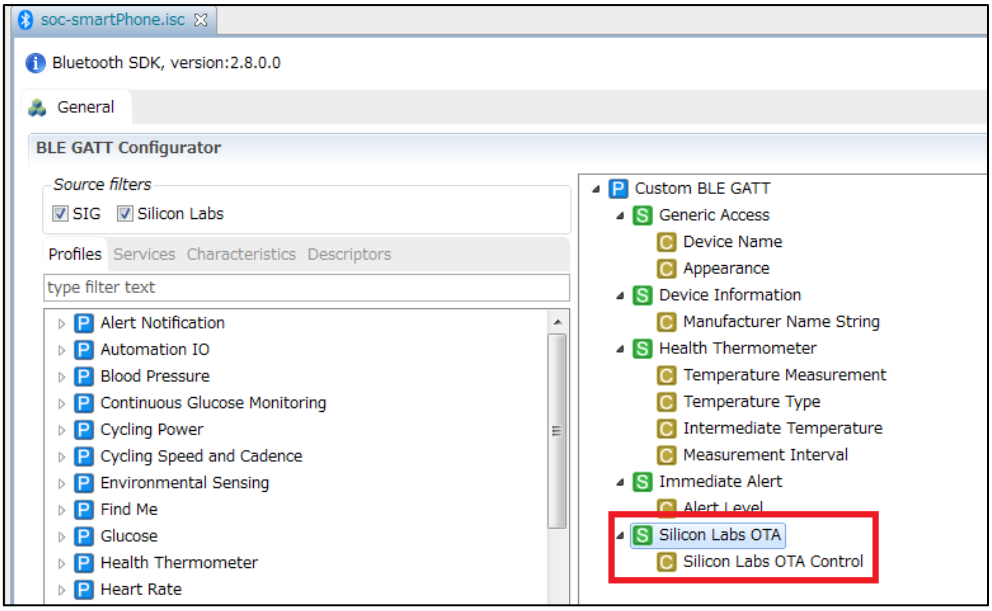


GUI 上で各種設定を行うことができます。必要な設定を行い、Start test で動作開始します。



3-2 OTA update（over-the-air）を使用する

“SOC - Smart Phone App”には、OTA update のサービスが実装されていますので、この機能を使ってアップデートを実践してみましょう。OTA update を使って、SOC - Smart Phone App を SOC – iBeacon に書き換える手順を紹介します。

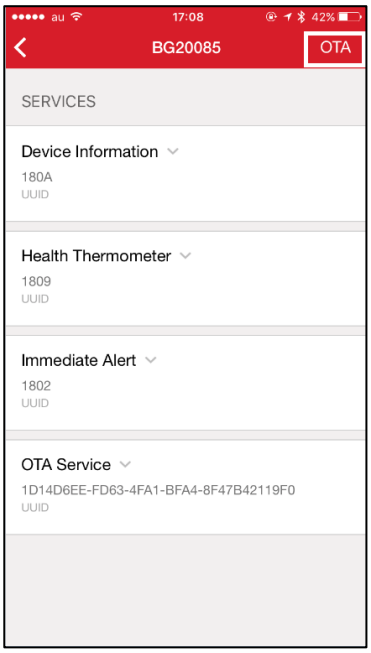


① Simplicity Studio で、SOC - iBeacon のプロジェクトを生成し、Build を実行します。手順は 7-2 を参照ください。

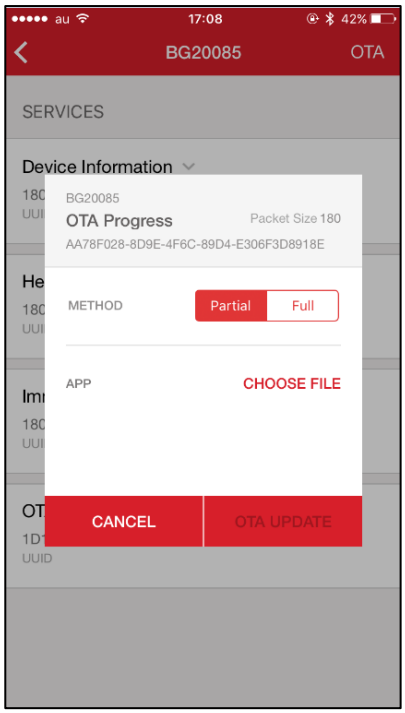
② ¥¥v4_workspace¥soc-ibeacon に create_bl_files.bat が生成されるので、実行して OTA 用のバイナリを作成します。作成したバイナリ（application.gbl）は、¥¥v4_workspace¥soc-ibeacon¥output_gbl に格納されます。

③ スマートフォンからアクセスできるフォルダ（dropbox など）に、application.gbl をコピーします。

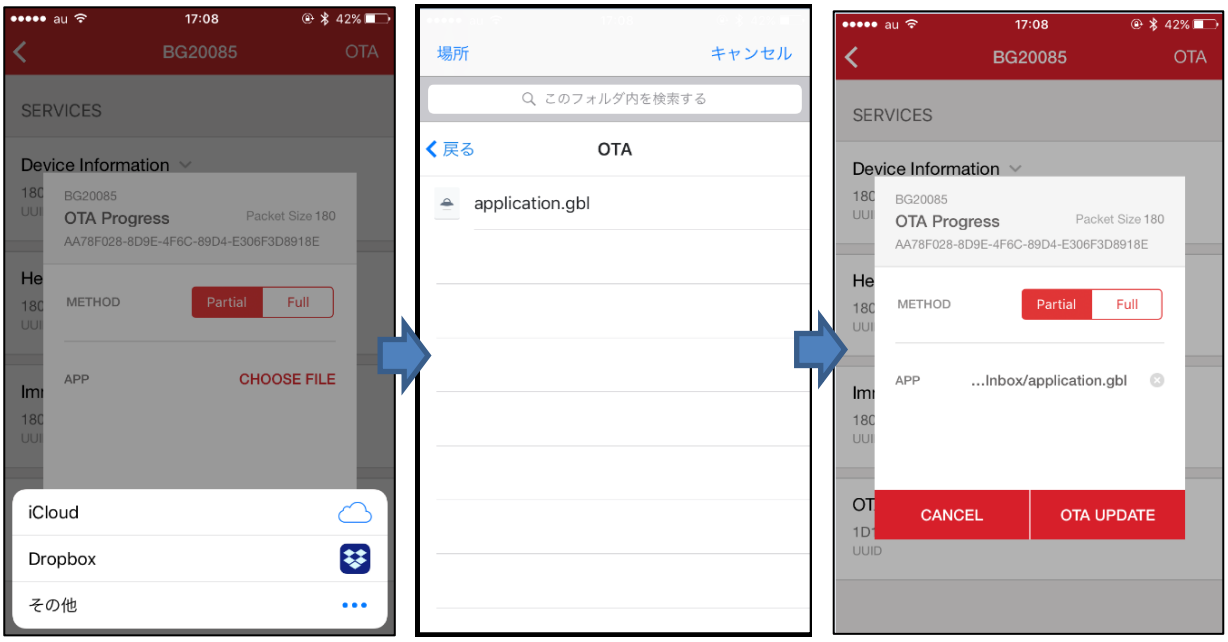
④ スマホアプリを起動して BGM121 に接続し、右上の OTA ボタンを押します。（右図）



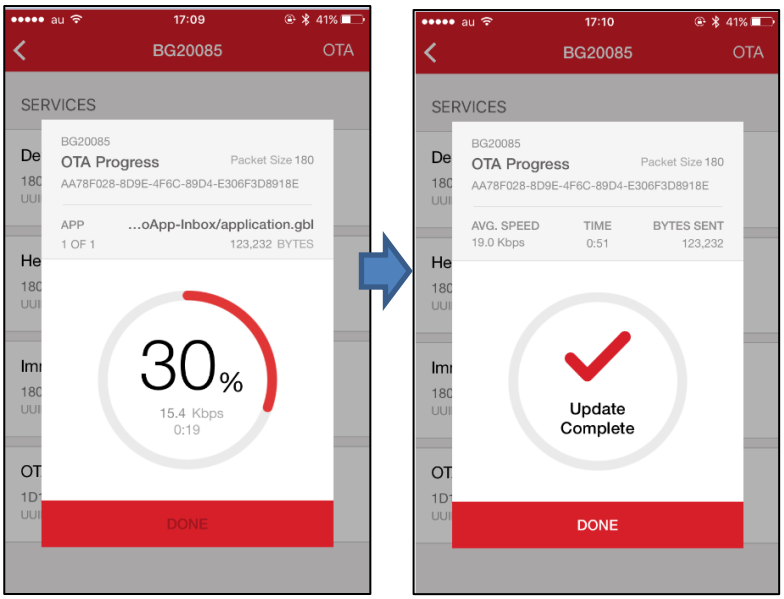
⑤ APP の CHOOSE FILE ボタンを押します。(右図)



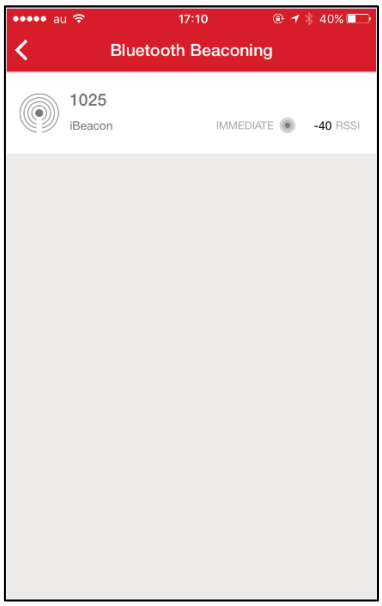
⑥ dropbox から、application.gbl を選択します。(下図)



⑦ OTA update を開始し（左下図）、完了する（右下図）。



⑧ スマホアプリで、iBeacon として動作開始していることが確認できます。（下図）



3-3 スループットを測定する

通信性能を測る指標の一つにスループットがあります。使用する PHY を 2M に設定したからといって、2Mbps のスループットが得られる訳ではありません。様々な要因によって、スループットは低下します。

BLE におけるスループットの考え方については、シリコンラボ社のナレッジベースに参考になる情報があります。(リンク:スループットについて)。コネクション・インターバルや MTU サイズ、ATT 動作種別などがスループットに影響します。

スループットを測定するには、測定したい条件に合せてユーザコードをご設計頂く必要があります。BGM⇔BGM 間のスループット測定については、サンプルコードをご用意していますので、こちらをベースにして、ご使用条件に合せてカスタマイズするのが容易です。

本章では、用意されているサンプルコードについてご紹介します。

3-3-1 シリコンラボ社・サンプルコード (SDK 2.7～2.10)

シリコンラボ社のナレッジベースに、サンプルコードと実装方法が用意されています。(リンク:サンプルコード)

実装の流れは以下の通りです。

- SOC-Empty プロジェクトを作成する
- サンプルコード(gatt.xml)をプロジェクトにインポート／Generate する
- LCD インタフェースを追加する
- main.c をサンプルコード(main_2_7_0.c)に差し替える
- プロジェクトをビルドし、2 つの BGM1xx にダウンロードする

基本的な使用方法は、次章のマクニカ・サンプルコードと同じです。測定したスループットは、スターターキット上の LCD に表示されます。

3-3-2 マクニカ・サンプルコード （SDK 2.11）

SDK 2.7 と 2.11 とで、main.c の実装が少し変更されていますので、シリコンラボ社・サンプルコードに一部手を加えたものをご用意しています。BGM13P および BGM13S 向けです。

主な変更点は以下の通りです。

- SDK 2.11 への対応
- Coded-PHY S2（500kbps）を追加
- 測定したスループットを UART に出力

プロジェクトをビルドし、2 つの BGM13P/S にダウンロードしてください。手っ取り早く評価したい場合には、GNU フォルダの下に hex ファイルが入っていますので、こちらを使用ください。その後の評価手順は以下の通りです。

1. 基板 1 にて、PB0 を押したままリセットボタンを押す（または電源を ON にする）と、Mater モードで起動して Virtual COM に次のメッセージを出力します。

   Booted as Master

2. 基板 2 にて、何も押さずにリセットボタンを押す（または電源を ON にする）と、Slave モードで起動して Virtual COM に次のメッセージを出力します。

   Booted as Slave

3. Master と Slave の両方を起動すると、自動的に接続して Master と Slave の双方の Virtual COM に次のメッセージを出力します。

   Connected

   Phy: 1M

4. Slave の PB0 を押している間 Notification の通信を行い、PB0 を放すと Slave の LCD と Virtual COM に転送レートを出力します。

   Throughput:　710400 bps (1M/Notification)

5. Slave の PB1 を押している間 Indication の通信を行い，PB1 を放すと Slave の LCD と Virtual COM に転送レートを出力します。

   Throughput:　19519 bps (1M/Indication)

6. Master の PB0 を押すと Phy を切り替えることができます。Phy は PB0 を押すごとに，1M → 2M → S8 (125k) → S2 (500k) → 1M の順に切り替わり、Virtual COM に切り替え後の Phy を出力します。以下は Master の Virtual COM の出力例です。

   Phy: 1M

Phy: 2M
Phy: S8
Phy: S2
Phy: 1M

7. Phy を切り替えながら Notification と Indication を測定すると、Slave の Virtual COM に次のように出力されます。

Phy: 1M
Throughput: 710400 bps (1M/Notification)
Throughput: 19519 bps (1M/Indication)
Phy: 2M
Throughput: 1250657 bps (2M/Notification)
Throughput: 38648 bps (2M/Indication)
Phy: S8
Throughput: 94848 bps (S8/Notification)
Throughput: 4684 bps (S8/Indication)
Phy: S2
Throughput: 333696 bps (S2/Notification)
Throughput: 4684 bps (S2/Indication)

8. LCD の表示には時間が掛かるため、LCD の表示が更新されたことを確認してから次の操作をしてください。特に Phy の切り替え（S8 や S2 への切り替え）には時間が掛かります。Master と Slave の両方が切り替わったことを確認してからスループットを測定してください。

4 消費電流を最適化する

4-1 消費電流の簡易測定

Wireless Starter Kit main board（以後、WSTK）と Simplicity Studio を使用することで、簡単に消費電流を測定することができます。

メインボードには電流計が実装されています。USB 給電（5V）から LDO で 3.3V を生成し、その 3.3V（VMCU）に流れる電流を測定しています。ですから、VMCU を BGM1xx への給電に限定するか、あるいは周辺回路への給電にも利用するか、によって測定対象が変わってきます。

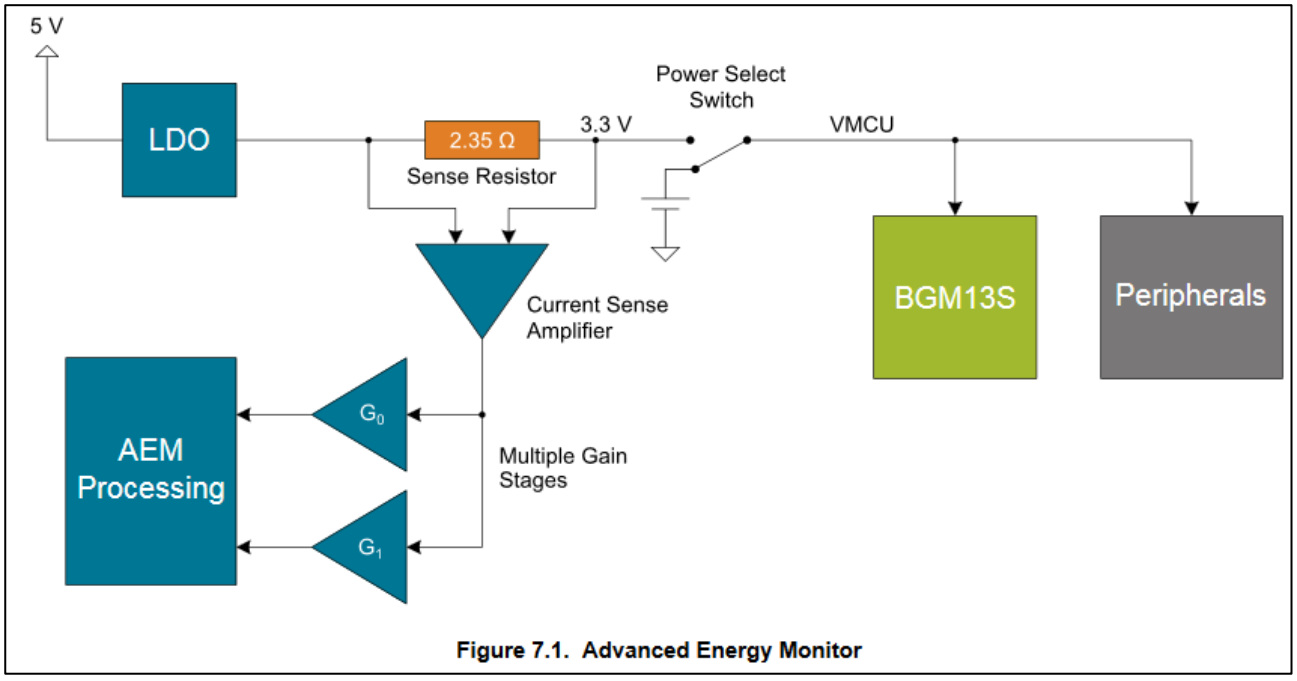


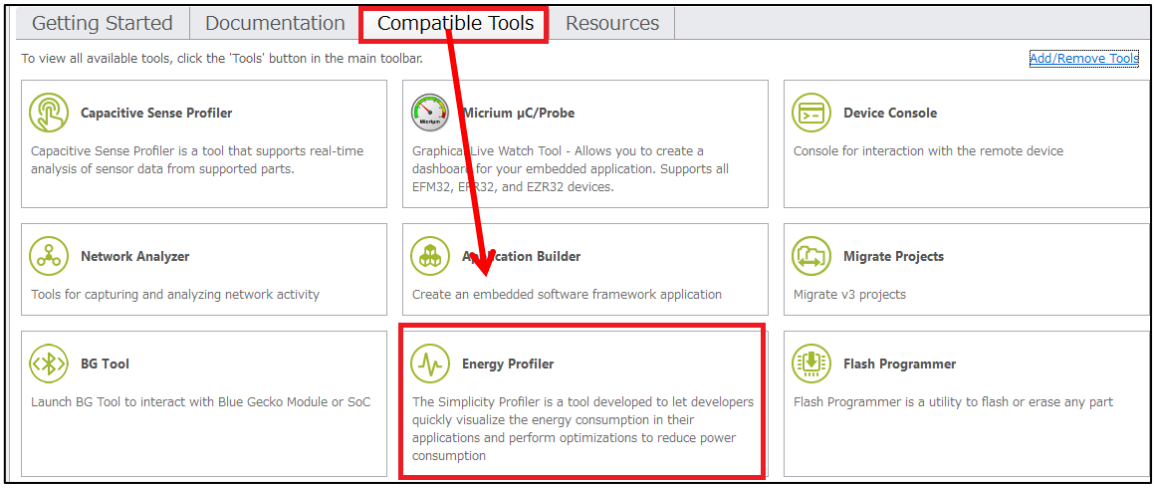
Figure 7.1. Advanced Energy Monitor

ただし、サンプリング周波数はそれほど速くありませんので、厳密な測定にはオシロスコープが必要になります。詳しくは AN969 をご参照ください。（リンク）

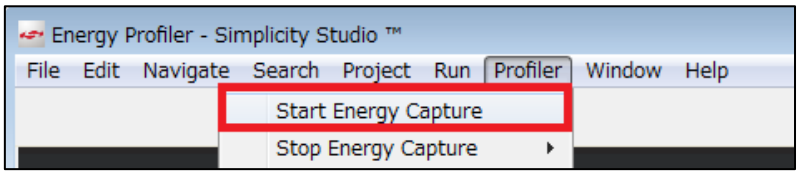
ここでは、ラジオボードを使用して評価する手順を紹介しますが、ユーザ基板への給電に VMCU を使用すれば、ユーザ基板の消費電流を測定することができます。

手順は以下の通りです。

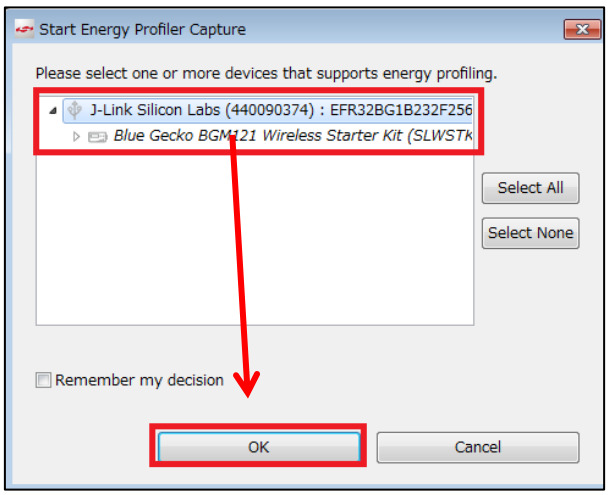
① メインボードにラジオボードを挿入し、PC に接続します。給電スイッチは AEM に設定ください。
② “SOC – Smart Phone App”デモをダウンロードします。
③ Compatible Tools タブから Energy Profiler を選択します。



④ Profiler ⇒ Start Energy Capture を選択します。Quick Access から選択しても良いです。



⑤ 測定対象を聞かれますので、接続している Wireless Starter Kit を選択し、OK をクリック。



⑥ 電流測定がスタートします。

⑦ Paused ボタンを押し、測定をいったん停止します。

スタート／ストップ（Running／Paused）の切り替え、対数／リニア表示の切り替え、X 軸・Y 軸の拡大／縮小などの機能が付いていますので、操作性を体感ください。平均電流も表示されています。

波形上でクリックするとカーソルが出ます。電流のピーク間を測定すると間隔はおよそ 100ms で、その周期でアドバタイズしていることが判ります。

⑧ Running ボタンを押し、測定を再開します。
スマートフォンから接続すると、その挙動が電流波形からも見て取れます。

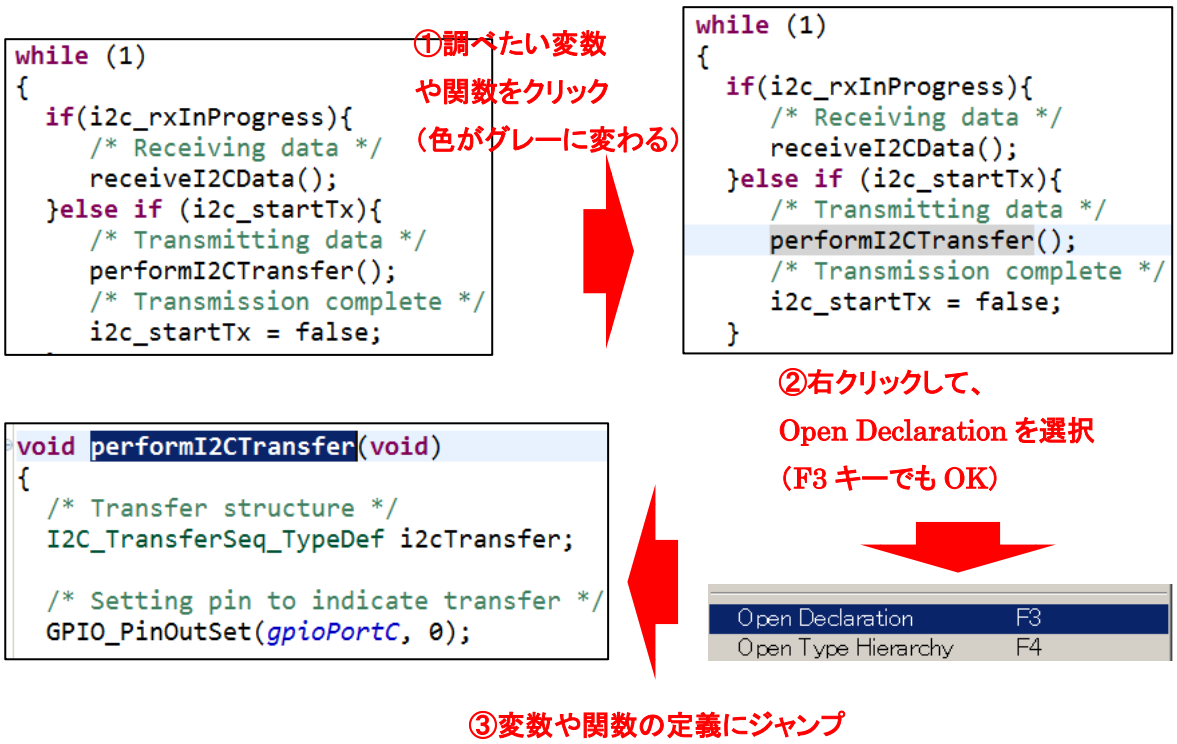
5 ソフトウェア設計する

ソフトウェア設計に役立つ情報をご紹介します。

5-1 ソースコードの追い方

Simplicity IDE でソースコードを追うための方法を紹介します。

◆ 変数や関数を定義している記述を探す



＊上記の説明では、EFM32 向けのコードを使用しています。BGM1xx でも手順は同じです。

5-2 ペリフェラルの実装 （外部割込み）

BGM1xx には、Smart Phone App や iBeacon といった、Bluetooth 制御を学ぶためのサンプルコードが用意されていますが、各ペリフェラルのサンプルコードはありません。

一方で、EFM32 には非常に多くのサンプルコードが用意されています。実際のアプリケーションをイメージしたサンプルコードもありますが、ADC のサンプルコード、UART のサンプルコード、外部割込みのサンプルコード…といった具合に 1 つのペリフェラルにスポットを当てたサンプルコードも多く、ペリフェラルの機能・動作を学ぶのに非常に役立ちます。

実際のアプリケーション設計では、Bluetooth 機能だけでなく、他のペリフェラルも使用することがほとんどかと思いますので、BGM1xx のサンプルコードに別のペリフェラルを追加する手順を学ぶことは非常に有益です。

この章では、2 つのサンプルコードを migration していく手順について紹介します。

題材として、「SOC – iBeacon」と「SLSTK3401A_gpio_int_pg1b」という 2 つのサンプルコードを使用します。「SOC – iBeacon」は BGM1xx 用のサンプルコードで、今回は BGM121 向けに実装されたものを使います。「SLSTK3401A_gpio_int_pg1b」は AN0012「General Purpose Input Output」に付属したサンプルコードで、EFM32PG starter kit（SLSTK3401A）向けです。

「SOC – iBeacon」は、ビーコン送信するだけのサンプルコードです。

「SLSTK3401A_gpio_int_pg1b」は、ボタンを押すと外部割込みが生じ、LED を反転（ON→OFF 或いは OFF→ON）するサンプルコードです。

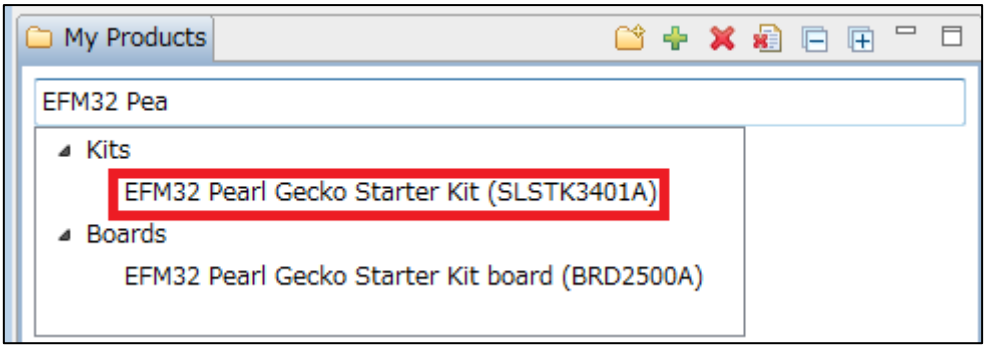
大まかな流れとしては、

- サンプルコードを理解する （5-2-1、5-2-2）
- 「SLSTK3401A_gpio_int_pg1b」のペリフェラル設定を、「SOC – iBeacon」に移植する（5-2-3）

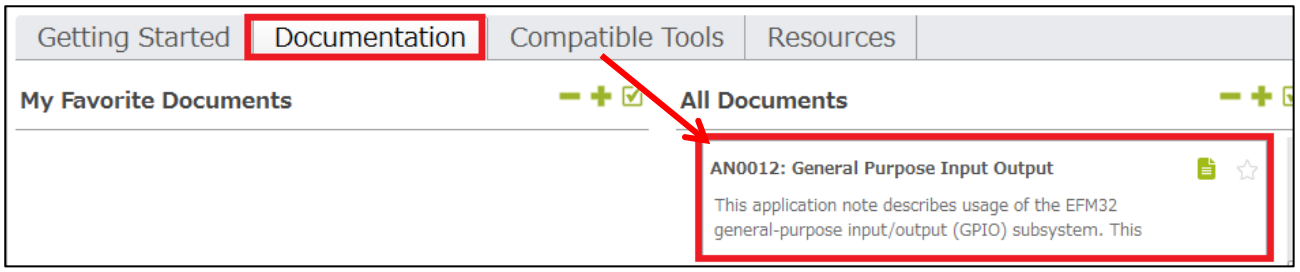
です。

5-2-1 サンプルコードを理解する （SLSTK3401A_gpio_int_pg1b）

まずはサンプルコードをロードします。My Products タブで EFM32 Pearl とタイプし、EFM32 Pearl Gecko Starter Kit を選びます。

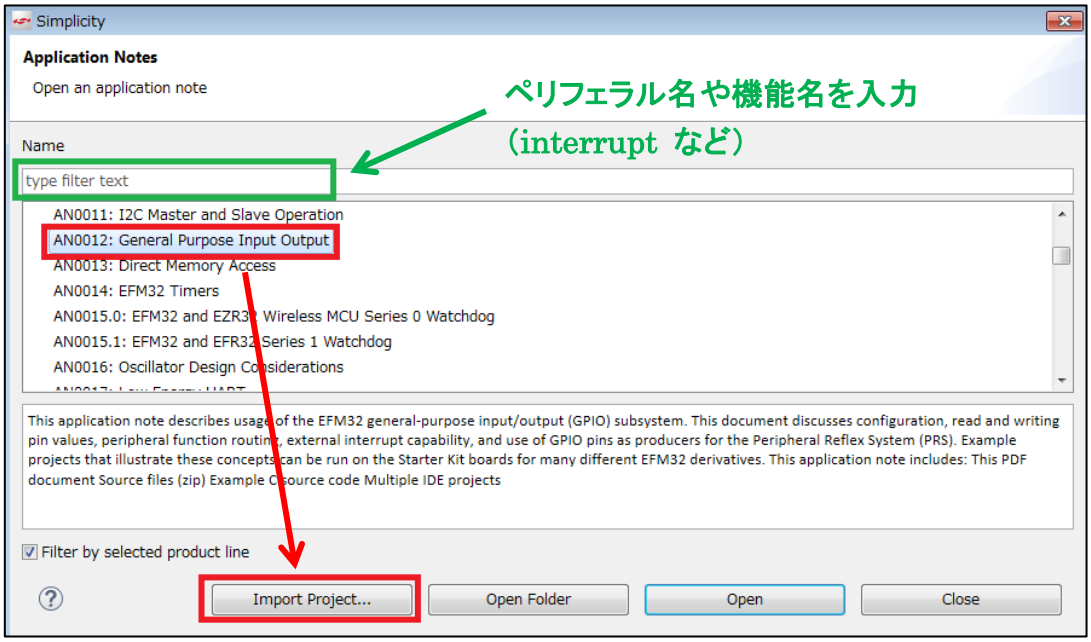


次に、Documentation タブ ⇒ Application Notes ⇒ AN0012 を選択します。

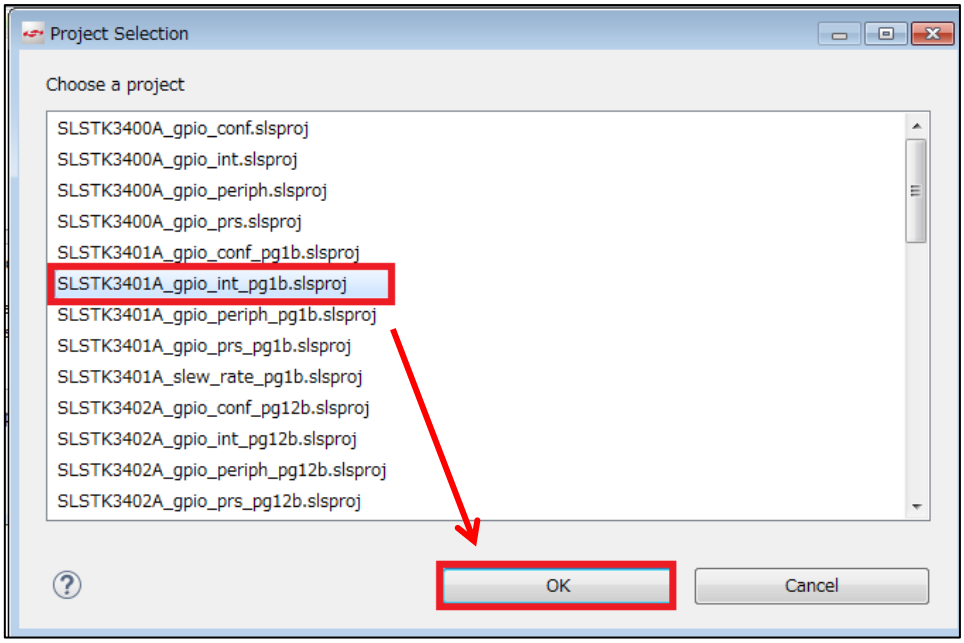


AN0012 が選択されていることを確認して、Import Project をクリックします。

なお、必要なペリフェラルのサンプルコードを見つけられない場合には、下図の緑枠（type filter text）の部分に、ADC や interrupt など のペリフェラル名を入力してください。候補がリストアップされます。



AN0012 に含まれたサンプルコードがリストアップされますので、「SLSTK3401A_gpio_int_pg1b」を選択し、OK をクリックします。



プロジェクトがロードされます。コードが 1 つだけあり、main_gpio_int.c がユーザコードです。
割込みハンドラ、GPIO の初期化関数、main 関数　から構成されています。

main_gpio_int.c

```
41 /**************************************************************************//**
42  * @brief GPIO Even IRQ for pushbuttons on even-numbered pins
43  *****************************************************************************/
44 void GPIO_EVEN_IRQHandler(void)
45 {
46   // Clear all even pin interrupt flags
47   GPIO_IntClear(0x5555);
48
49   // Toggle LED0
50   GPIO_PinOutToggle(BSP_GPIO_LED0_PORT, BSP_GPIO_LED0_PIN);
51 }
52
53 /**************************************************************************//**
54  * @brief GPIO Odd IRQ for pushbuttons on odd-numbered pins
55  *****************************************************************************/
56 void GPIO_ODD_IRQHandler(void)
57 {
58   // Clear all odd pin interrupt flags
59   GPIO_IntClear(0xAAAA);
60
61   // Toggle LED01
62   GPIO_PinOutToggle(BSP_GPIO_LED0_PORT, BSP_GPIO_LED0_PIN);
63 }
```

割込みクリア

偶数番号ピンの割込みハンドラ

LED をトグル（反転）

奇数番号ピンの割込みハンドラ

```
65 /*****************************************************************************//**
66  * @brief GPIO initialization
67  *****************************************************************************/
68 void initGPIO(void)
69 {
70   // Configure GPIO pins
71   CMU_ClockEnable(cmuClock_GPIO, true);
72
73   // Configure PB0 and PB1 as input with glitch filter enabled
74   GPIO_PinModeSet(BSP_GPIO_PB0_PORT, BSP_GPIO_PB0_PIN, gpioModeInputPullFilter, 1);
75   GPIO_PinModeSet(BSP_GPIO_PB1_PORT, BSP_GPIO_PB1_PIN, gpioModeInputPullFilter, 1);
76
77   // Configure LED0 and LED1 as output
78   GPIO_PinModeSet(BSP_GPIO_LED0_PORT, BSP_GPIO_LED0_PIN, gpioModePushPull, 0);
79
80   // Enable IRQ for even numbered GPIO pins
81   NVIC_EnableIRQ(GPIO_EVEN_IRQn);
82
83   // Enable IRQ for odd numbered GPIO pins
84   NVIC_EnableIRQ(GPIO_ODD_IRQn);
85
86   // Enable falling-edge interrupts for PB pins
87   GPIO_IntConfig(BSP_GPIO_PB0_PORT, BSP_GPIO_PB0_PIN, 0, 1, true);
88   GPIO_IntConfig(BSP_GPIO_PB1_PORT, BSP_GPIO_PB1_PIN, 0, 1, true);
89 }
91 /*****************************************************************************//**
92  * @brief  Main function
93  *****************************************************************************/
94 int main(void)
95 {
96   // Chip errata
97   CHIP_Init();
98
99   // Initializations
100  initGPIO();
101
102  while (1){
103    // Enter Low Energy Mode until an interrupt occurs
104    EMU_EnterEM3(false);
105  }
106 }
```

GPIO の初期化関数

ピン設定（ボタン、LED 用）

外部割込み有効

ピンと外部割込みの紐付け

main 関数

while ループに入る前に、
GPIO 初期化関数を呼んでいる

5-2-2 サンプルコードを理解する （SOC – iBeacon）

Device 或いは Solution タブで BGM121　Wireless Starter Kit を選択して、SOC-iBeacon のサンプルコードをロードします。main.c がユーザコードです。

使用する各関数の定義と、main 関数から構成されています。

main.c

```
76   */
77  void bcnSetupAdvBeaconing(void)
78  {
79    /* This function sets up a custom advertisement package according to iBeacon specifications.
80     * The advertisement package is 30 bytes long. See the iBeacon specification for further details.
81     */
82
83    static struct {
84      uint8_t flagsLen;      /* Length of the Flags field. */
85      uint8_t flagsType;     /* Type of the Flags field. */
86      uint8_t flags;         /* Flags field. */
87      uint8_t mandataLen;    /* Length of the Manufacturer Data field. */
88      uint8_t mandataType;   /* Type of the Manufacturer Data field. */
89      uint8_t compId[2];     /* Company ID field. */
90      uint8_t beacType[2];   /* Beacon Type field. */
```

各関数や変数の定義

```
152 /**
153  * @brief  Main function
154  */
155 void main(void)
156 {
157   // Initialize device
158   initMcu();
159   // Initialize board
160   initBoard();
161   // Initialize application
162   initApp();
163
164   // Initialize stack
165   gecko_init(&config);
166
167   while (1) {
168     struct gecko_cmd_packet* evt;
169
170     // Check for stack event.
171     evt = gecko_wait_event();
172
173     // Run application and event handler.
174     switch (BGLIB_MSG_ID(evt->header)) {
175       // This boot event is generated when the system boots up after reset.
176       // Do not call any stack commands before receiving the boot event.
177       case gecko_evt_system_boot_id:
178         // Initialize iBeacon ADV data
179         bcnSetupAdvBeaconing();
180         break;
181
182       default:
183         break;
184     }
185   }
186 }
187
188 /** @} (end addtogroup app) */
189 /** @} (end addtogroup Application) */
```

main 関数

初期化関数

初期化関数として、initMCU(); initBoard(); initApp(); gecko_init(&config); の 4 つが呼ばれています。少し詳しく見ていきます。詳しくは UG136 を参照ください。

- initMCU()

initMCU() は、init_MCU.c の中で定義されています。

init_MCU.c

```
34  void initMcu(void)
35  {
36    // Device errata
37    CHIP_Init();
38
39    // Set up DC-DC converter
40    EMU_DCDCInit_TypeDef dcdcInit = BSP_DCDC_INIT;
41    #if HAL_DCDC_BYPASS
42    dcdcInit.dcdcMode = emuDcdcMode_Bypass;
43    #endif
44    EMU_DCDCInit(&dcdcInit);
45
46    // Set up clocks
47    initMcu_clocks();
48
49    RTCC_Init_TypeDef rtccInit = RTCC_INIT_DEFAULT;
50    rtccInit.enable                = true;
51    rtccInit.debugRun              = false;
52    rtccInit.precntWrapOnCCV0      = false;
53    rtccInit.cntWrapOnCCV1         = false;
54    rtccInit.prescMode             = rtccCntTickPresc;
55    rtccInit.presc                 = rtccCntPresc_1;
56    rtccInit.enaOSCFailDetect      = false;
57    rtccInit.cntMode               = rtccCntModeNormal;
58    RTCC_Init(&rtccInit);
59
60  #if defined(_EMU_CMD_EM01VSCALE0_MASK)
61    // Set up EM0, EM1 energy mode configuration
62    EMU_EM01Init_TypeDef em01Init = EMU_EM01INIT_DEFAULT;
63    EMU_EM01Init(&em01Init);
64  #endif // _EMU_CMD_EM01VSCALE0_MASK
65
66  #if defined(_EMU_CTRL_EM23VSCALE_MASK)
67    // Set up EM2, EM3 energy mode configuration
68    EMU_EM23Init_TypeDef em23init = EMU_EM23INIT_DEFAULT;
69    em23init.vScaleEM23Voltage = emuVScaleEM23_LowPower;
70    EMU_EM23Init(&em23init);
71  #endif //_EMU_CTRL_EM23VSCALE_MASK
72
73    TEMPDRV_Init();
74  }
```

UG136 の中では、init_mcu.c と init_mcu.h の役割は「These files include the device initialization function, which initializes internal settings of the MCU like clocks and power management.」と説明されています。DCDC や RTCC（リアルタイムクロック）の初期設定を行っています。

- initBoard()

initBoard() は、init_board.c の中で定義されています。

init_board.c

```
29 void initBoard(void)
30 {
31   // Enable clock for CRYOTIMER
32   CMU_ClockEnable(cmuClock_CRYOTIMER, true);
33   // Enable clock for PRS
34   CMU_ClockEnable(cmuClock_PRS, true);
35 #ifdef FEATURE_EXP_HEADER_USART3
36   // Enable clock for USART3
37   CMU_ClockEnable(cmuClock_USART3, true);
38 #else
39   // Enable clock for USART0
40   CMU_ClockEnable(cmuClock_USART0, true);
41 #endif
42   // Enable GPIO clock source
43   CMU_ClockEnable(cmuClock_GPIO, true);
44
45   // Put the SPI flash into Deep Power Down mode for those radio boards where it is available
46   MX25_init();
47   MX25_DP();
48   // We must disable SPI communication
49   USART_Reset(MX25_USART);
50 }
```

UG136 の中では、init_board.c と init_board.h の役割は「These files include the board initialization function, which initializes external parts on the board. For example, it enables GPIOs, and initializes external flash on the radio board.」と説明されています。Starter Kit 上に実装された SPI フラッシュメモリなどの外部部品に関する初期化などを行っています。

- initApp()

initApp() は、init_app.c の中で定義されています。

init.app.c

```
25 void initApp(void)
26 {
27   // Enable PTI
28   configEnablePti();
29
30 #if defined(HAL_VCOM_ENABLE)
31   // Enable VCOM if requested
32   GPIO_PinModeSet(BSP_VCOM_ENABLE_PORT, BSP_VCOM_ENABLE_PIN, gpioModePushPull, HAL_VCOM_ENABLE);
33 #endif // HAL_VCOM_ENABLE
34
35 #if (HAL_I2CSENSOR_ENABLE)
36   // Initialize I2C peripheral
37   I2CSPM_Init_TypeDef i2cInit = I2CSPM_INIT_DEFAULT;
38   I2CSPM_Init(&i2cInit);
39 #endif // HAL_I2CSENSOR_ENABLE
40
41 #if defined(HAL_I2CSENSOR_ENABLE)
42   // Enable I2C sensor if requested
43   GPIO_PinModeSet(BSP_I2CSENSOR_ENABLE_PORT, BSP_I2CSENSOR_ENABLE_PIN, gpioModePushPull, HAL_I2CSENSOR_ENABLE);
44 #endif // HAL_I2CSENSOR_ENABLE
45
46 #if defined(HAL_SPIDISPLAY_ENABLE)
47   // Enable SPI display if requested
48   GPIO_PinModeSet(BSP_SPIDISPLAY_ENABLE_PORT, BSP_SPIDISPLAY_ENABLE_PIN, gpioModePushPull, HAL_SPIDISPLAY_ENABLE);
49 #endif // HAL_SPIDISPLAY_ENABLE
50 }
```

UG136 の中では、init_app.c と init_app.h の役割は「These files include the app initialization function, which initializes external parts on the WSTK according to the application. For example, it enables VCOM, sensors, and LCD display on the WSTK.」と説明されています。Starter Kit 上に実装された VCOM（仮想 COM）やセンサ、LCD ディスプレイなどの外部部品に関する初期化などを行っています。

- gecko_init(&config)

gecko_init()は、native_gecko.h の中で定義されています。native_gecko.h は、GATT エディタ（7-2 参照）が自動生成するファイルで、編集不可です。“config“ は main.c の中で定義されており、最大接続数やヒープ領域のサイズなどを指定しています。

main.c

```
51 #ifndef MAX_CONNECTIONS
52 #define MAX_CONNECTIONS 4
53 #endif
54 uint8_t bluetooth_stack_heap[DEFAULT_BLUETOOTH_HEAP(MAX_CONNECTIONS)];
55
56 /* Gecko configuration parameters (see gecko_configuration.h) */
57 static const gecko_configuration_t config = {
58   .config_flags = 0,
59   .sleep.flags = SLEEP_FLAGS_DEEP_SLEEP_ENABLE,
60   .bluetooth.max_connections = MAX_CONNECTIONS,
61   .bluetooth.heap = bluetooth_stack_heap,
62   .bluetooth.sleep_clock_accuracy = 100, // ppm
63   .bluetooth.heap_size = sizeof(bluetooth_stack_heap),
64   .gattdb = &bg_gattdb_data,
65 #if (HAL_PA_ENABLE) && defined(FEATURE_PA_HIGH_POWER)
66   .pa.config_enable = 1, // Enable high power PA
67   .pa.input = GECKO_RADIO_PA_INPUT_VBAT, // Configure PA input to VBAT
68 #endif // (HAL_PA_ENABLE) && defined(FEATURE_PA_HIGH_POWER)
69 };
```

5-2-3 ペリフェラル設定を移植する

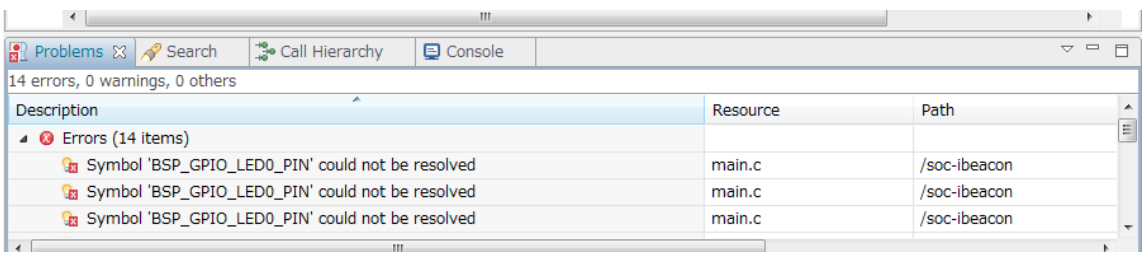
「SLSTK3401A_gpio_int_pg1b」の**割込みハンドラ**と GPIO 初期化関数を、そのまま「SOC-iBeacon」の main.c にコピーします。場所は main()の前です。

```
146    gecko_cmd_le_gap_set_advertise_timing(0, 160, 160, 0, 0);
147
148    /* Start advertising in user mode and enable connections */
149    gecko_cmd_le_gap_start_advertising(0, le_gap_user_data, le_gap_non_connectable);
150  }
151
152 /**
153   * @brief  Main function
154   */
155 void main(void)
156  {
157    // Initialize device
158    initMcu();
159    // Initialize board
160    initBoard();
161    // Initialize application
162    initApp();
163
164    // Initialize stack
165    gecko_init(&config);
166
167    while (1) {
```

このあたりにコピー

```
41 /**************************************************************************//**
42  * @brief GPIO Even IRQ for pushbuttons on even-numbered pins
43  *****************************************************************************/
44 void GPIO_EVEN_IRQHandler(void)
45 {
46   // Clear all even pin interrupt flags
47   GPIO_IntClear(0x5555);
48
49   // Toggle LED0
50   GPIO_PinOutToggle(BSP_GPIO_LED0_PORT, BSP_GPIO_LED0_PIN);
51 }
52
53 /**************************************************************************//**
54  * @brief GPIO Odd IRQ for pushbuttons on odd-numbered pins
55  *****************************************************************************/
56 void GPIO_ODD_IRQHandler(void)
57 {
58   // Clear all odd pin interrupt flags
59   GPIO_IntClear(0xAAAA);
60
61   // Toggle LED01
62   GPIO_PinOutToggle(BSP_GPIO_LED0_PORT, BSP_GPIO_LED0_PIN);
63 }
64
65 /**************************************************************************//**
66  * @brief GPIO initialization
67  *****************************************************************************/
68 void initGPIO(void)
69 {
70   // Configure GPIO pins
71   CMU_ClockEnable(cmuClock_GPIO, true);
72
73   // Configure PB0 and PB1 as input with glitch filter enabled
74   GPIO_PinModeSet(BSP_GPIO_PB0_PORT, BSP_GPIO_PB0_PIN, gpioModeInputPullFilter, 1);
75   GPIO_PinModeSet(BSP_GPIO_PB1_PORT, BSP_GPIO_PB1_PIN, gpioModeInputPullFilter, 1);
76
77   // Configure LED0 and LED1 as output
78   GPIO_PinModeSet(BSP_GPIO_LED0_PORT, BSP_GPIO_LED0_PIN, gpioModePushPull, 0);
79
80   // Enable IRQ for even numbered GPIO pins
81   NVIC_EnableIRQ(GPIO_EVEN_IRQn);
82
83   // Enable IRQ for odd numbered GPIO pins
84   NVIC_EnableIRQ(GPIO_ODD_IRQn);
85
86   // Enable falling-edge interrupts for PB pins
87   GPIO_IntConfig(BSP_GPIO_PB0_PORT, BSP_GPIO_PB0_PIN, 0, 1, true);
88   GPIO_IntConfig(BSP_GPIO_PB1_PORT, BSP_GPIO_PB1_PIN, 0, 1, true);
89 }
90
91 /**************************************************************************//**
```

単にコピーしただけだと、画面下の Problem ウィンドウにエラーが多数出ます。



「SLSTK3401A_gpio_int_pg1b」では、ボタンや LED のピン番号を直接指定せずに、変数を定義して（BSP_GPIO_PB0_PORT やBSP_GPIO_PB0_PIN など）、間接的に指定をしています。その変数と同じ名称の定義が「SOC-iBeacon」にないため、エラーが出ています。

BGM121 wireless starter kit を使用しますので、wireless starter kit のボタンや LED のピン番号を代わりに指定します。

BGM121 wireless starter kit にも、ボタンや LED のピン番号を変数定義したファイルがありますので、ご紹介します。hal-config-board.h というファイル内で定義されています。

hal-config-board.h

```
40  // $[BUTTON]
41  #define BSP_BUTTON_PRESENT                (1)
42
43  #define BSP_BUTTON0_PIN                   (6U)
44  #define BSP_BUTTON0_PORT                  (gpioPortF)
45
46  #define BSP_BUTTON1_PIN                   (7U)
47  #define BSP_BUTTON1_PORT                  (gpioPortF)
48
49  #define BSP_BUTTON_COUNT                  (2U)
50  #define BSP_BUTTON_INIT                   { { BSP_BUTTON0_PORT, BSP_BUTTON0_PIN }, { BSP_BUTTON1_PORT, BSP_BUTTON1_PIN } }
51  #define BSP_BUTTON_GPIO_DOUT              (HAL_GPIO_DOUT_LOW)
52  #define BSP_BUTTON_GPIO_MODE              (HAL_GPIO_MODE_INPUT)
53  // [BUTTON]$
```

ボタンのピン配置を定義

```
157  // $[LED]
158  #define BSP_LED_PRESENT                   (1)
159
160  #define BSP_LED0_PIN                      (4U)
161  #define BSP_LED0_PORT                     (gpioPortF)
162
163  #define BSP_LED1_PIN                      (5U)
164  #define BSP_LED1_PORT                     (gpioPortF)
165
166  #define BSP_LED_COUNT                     (2U)
167  #define BSP_LED_INIT                      { { BSP_LED0_PORT, BSP_LED0_PIN }, { BSP_LED1_PORT, BSP_LED1_PIN } }
168  // [LED]$
```

LED のピン配置を定義

次に、定義した GPIO 初期化関数を、main()の中で呼び出します。

```
209 void main(void)
210 {
211   // Initialize device
212   initMcu();
213   // Initialize board
214   initBoard();
215   // Initialize application
216   initApp();
217
218   // Initialize stack
219   gecko_init(&config);
220
221   // 初期化 GPIO
222   initGPIO();
223
224   while (1) {
225     struct gecko_cmd_packet* evt;
226
227     // Check for stack event.
228     evt = gecko_wait_event();
229
230     // Run application and event handler.
231     switch (BGLIB_MSG_ID(evt->header)) {
232       // This boot event is generated when the system boots up after reset.
233       // Do not call any stack commands before receiving the boot event.
234       case gecko_evt_system_boot_id:
235         // Initialize iBeacon ADV data
236         bcnSetupAdvBeaconing();
237         break;
238
239       default:
240         break;
241     }
242   }
243 }
```

GPIO 初期化関数

最終的に、このようなコードになりました。

```
137   /* Set 0 dBm Transmit Power */
138   gecko_cmd_system_set_tx_power(0);
139
140   /* Set custom advertising data */
141   gecko_cmd_le_gap_bt5_set_adv_data(0, 0, len, pData);
142
143   /* Set advertising parameters. 100ms advertisement interval.
144    * The first two parameters are minimum and maximum advertising interval,
145    * both in units of (milliseconds * 1.6). */
146   gecko_cmd_le_gap_set_advertise_timing(0, 160, 160, 0, 0);
147
148   /* Start advertising in user mode and enable connections */
149   gecko_cmd_le_gap_start_advertising(0, le_gap_user_data, le_gap_non_connectable);
150 }
151
152 // 追加（SLSTK3401A_gpio_int_pg1b から）
153
154 /**************************************************************************//**
155  * @brief GPIO Even IRQ for pushbuttons on even-numbered pins
156  *****************************************************************************/
157 void GPIO_EVEN_IRQHandler(void)
158 {
```

移植　ここから

```
159   // Clear all even pin interrupt flags
160   GPIO_IntClear(0x5555);
161
162   // Toggle LED0
163   GPIO_PinOutToggle(BSP_LED0_PORT, BSP_LED0_PIN);
164 }
165
166 /**************************************************************************//**
167  * @brief GPIO Odd IRQ for pushbuttons on odd-numbered pins
168  *****************************************************************************/
169 void GPIO_ODD_IRQHandler(void)
170 {
171   // Clear all odd pin interrupt flags
172   GPIO_IntClear(0xAAAA);
173
174   // Toggle LED01
175   GPIO_PinOutToggle(BSP_LED1_PORT, BSP_LED1_PIN);
176 }
177
178 /**************************************************************************//**
179  * @brief GPIO initialization
180  *****************************************************************************/
181 void initGPIO(void)
182 {
183   // Configure GPIO pins
184   CMU_ClockEnable(cmuClock_GPIO, true);
185
186   // Configure PB0 and PB1 as input with glitch filter enabled
187   GPIO_PinModeSet(BSP_BUTTON0_PORT, BSP_BUTTON0_PIN, gpioModeInputPullFilter, 1);
188   GPIO_PinModeSet(BSP_BUTTON1_PORT, BSP_BUTTON1_PIN, gpioModeInputPullFilter, 1);
189
190   // Configure LED0 and LED1 as output
191   GPIO_PinModeSet(BSP_LED0_PORT, BSP_LED0_PIN, gpioModePushPull, 0);
192   GPIO_PinModeSet(BSP_LED1_PORT, BSP_LED1_PIN, gpioModePushPull, 0);
193
194   // Enable IRQ for even numbered GPIO pins
195   NVIC_EnableIRQ(GPIO_EVEN_IRQn);
196
197   // Enable IRQ for odd numbered GPIO pins
198   NVIC_EnableIRQ(GPIO_ODD_IRQn);
199
200   // Enable falling-edge interrupts for PB pins
201   GPIO_IntConfig(BSP_BUTTON0_PORT, BSP_BUTTON0_PIN, 0, 1, true);
202   GPIO_IntConfig(BSP_BUTTON1_PORT, BSP_BUTTON1_PIN, 0, 1, true);
203 }
204
```

変数名を変更

```
207   * @brief  Main function
208   */
209  void main(void)
210  {
211    // Initialize device
212    initMcu();
213    // Initialize board
214    initBoard();
215    // Initialize application
216    initApp();
217
218    // Initialize stack
219    gecko_init(&config);
220
221    // 初期化 GPIO
222    initGPIO();
223
224    while (1) {
225      struct gecko_cmd_packet* evt;
226
227      // Check for stack event.
228      evt = gecko_wait_event();
229
230      // Run application and event handler.
231      switch (BGLIB_MSG_ID(evt->header)) {
232        // This boot event is generated when the system boots up after reset.
233        // Do not call any stack commands before receiving the boot event.
234        case gecko_evt_system_boot_id:
235          // Initialize iBeacon ADV data
236          bcnSetupAdvBeaconing();
237          break;
238
239        default:
240          break;
241      }
242    }
243  }
244
245  /** @} (end addtogroup app) */
246  /** @} (end addtogroup Application) */
```

GPIO 初期化関数を実行

こうして実装したコードを、ビルドして実行すると、外部割込みを実装できたことが確認できます。

今回は例として外部割込みの実装手順を紹介しましたが、UG136 に割り込み実装についての注意点が書かれていますので、必ず参照ください。（リンク）

割込み処理中は、他の処理が行えなくなります。通常のアプリ設計であれば、それでも良いですが、Bluetooth スタック動作も並行して行う必要がありますので、割り込みハンドラ内で重い処理を行うのは好ましくありません。割り込みハンドラ内では単にフラグを立てておき、通常ループ内で処理させるといったコーディングが求められます。

5-3 ブートローダーの作成

ブートローダーは、BGM 起動後に最初に動作するアプリケーションです。ブートローダーで必要な処理を行い、それからユーザアプリケーションが起動する、という仕組みになっています。

Silicon Labs 社が提供するブートローダーには Gecko Bootloader（新しい）と legacy bootloader（古い）の 2 種があり、SDK 2.7 以降のアプリケーションは Gecko Bootloader 上でのみ動作するようになっています。

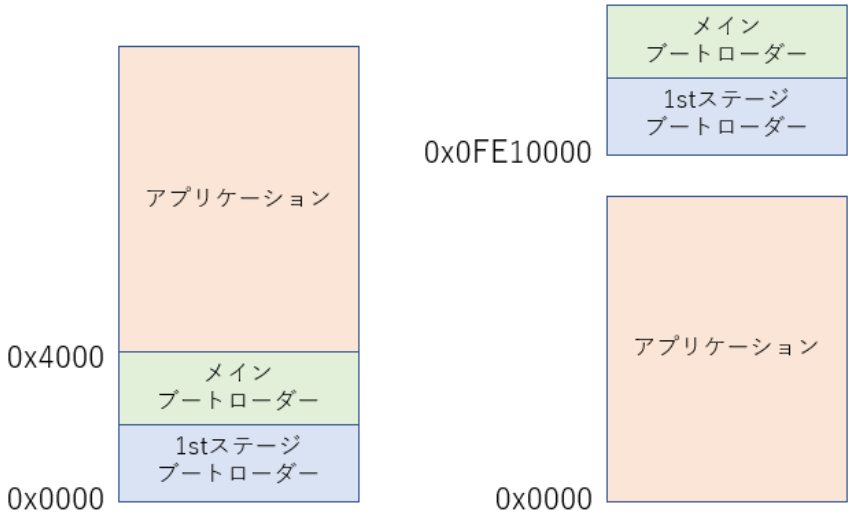
詳細は、UG266「Silicon Labs Gecko Bootloader User's Guide」（リンク）をご覧ください。

ここでは Gecko Bootloader に限定して説明します。

5-3-1 ブートローダーの配置アドレス

Gecko Bootloader は 1st ステージブートローダーとメインブートローダーの 2 段構成になっていて、まずは 1st ステージブートローダーが起動し、処理終了後にメインブートローダーが起動します。その後、ユーザアプリケーションが起動する、という流れです。

下図は、BGM1xx のメモリマップ上のどこにブートローダーおよびアプリケーションが配置されるかを示したものです。



BGM111, 113, 12x, 11S
（EFR32BG1-based）

BGM13P, 13S
（EFR32BG13-based）

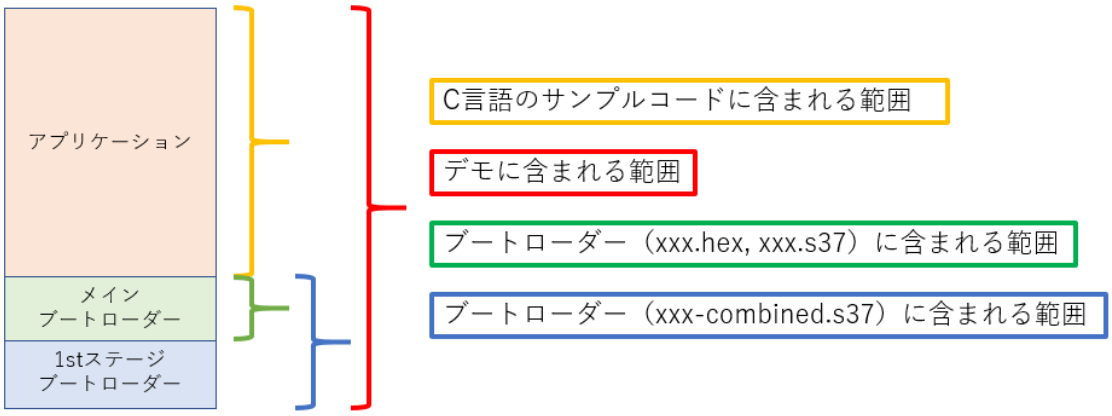
5-3-2 生成した HEX に含まれる範囲

ブートローダーをビルドすると、

- xxx.hex
- xxx.s37
- xxx-combined.s37

という HEX ファイルが生成されます。(.hex はインテル HEX、.s37 はモトローラ HEX です)
.bin も生成されますが、基本的にはアドレス情報が入った HEX ファイルを使用します。

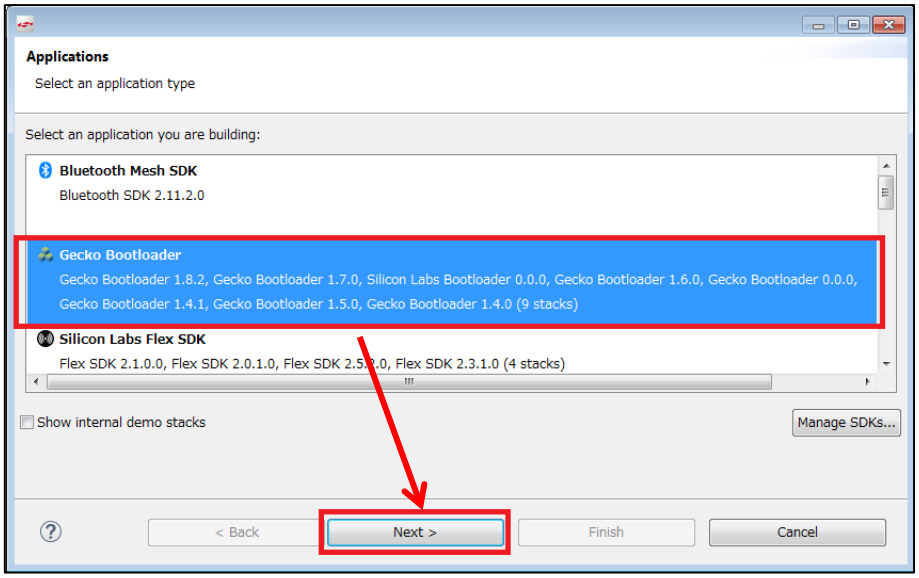
末尾に-combined が付いたファイルには、1st ステージブートローダーとメインブートローダーが含まれています。-combined が付いていないファイルには、メインブートローダーのみが含まれています。
また、Bluetooth SDK に含まれる C サンプルコードにはブートローダーが含まれず、デモには含まれています。図示すると以下の通りです。



5-3-3 ブートローダーの作成手順

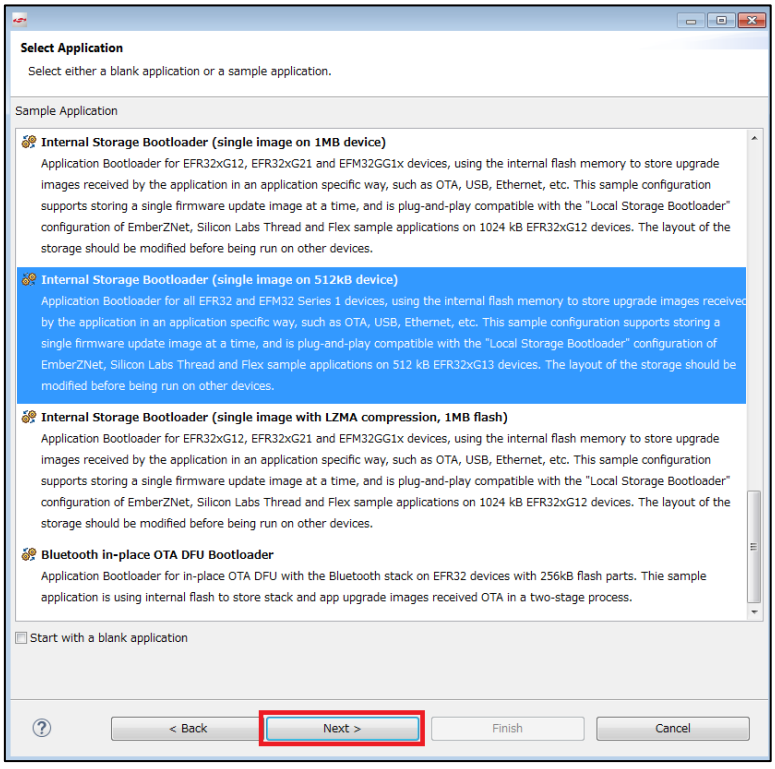
ブートローダーの作成手順は以下の通りです。

① New Project から Gecko Bootloader を選択します。



② 複数の Gecko Bootloader がインストールされている場合には、使用するバージョンを指定します。

③ 多数のサンプルプロジェクトが表示されるので、ベースにするプロジェクトを選択します。使用するデバイスやプロトコルによって、適切なプロジェクトが違ってきます。

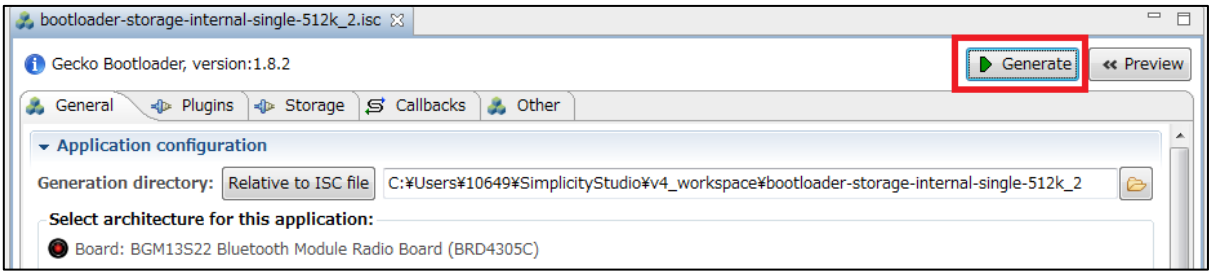


標準的なブートローダーは以下の通りです。

モジュール	動作モード	プロジェクト名
BGM111, 113, 12x, 11S （EFR32BG1-based）	SOC	Bluetooth in-place OTA DFU Bootloader
BGM13P, 13S （EFR32BG13-based）	SOC	Internal Storage Bootloader (single image on 512kB device)
BGM1xx	NCP	BGAPI UART DFU Bootloader

④ プロジェクト名、作業フォルダ、使用コンパイラなどを指定して、プロジェクトを生成します。

⑤ 必要に応じて、各種設定を行ってください。詳しくは UG266 を参照ください。設定が終わったら Generate ボタンを押し、ソースコードに反映します。



⑥ C 言語のサンプルコードと同様に、ビルドして、ダウンロードします。xxx.hex/.s37 と xxx-combined.s37 とで含まれる範囲が異なることにご注意ください。

5-4 新しい SDK への移行

SDK 2.3～2.6 を次のバージョンに移行する場合には、下記を参照ください。

＜マクニカオンラインサービス FAQ＞

- Bluetooth Smart SDK 2.3.x から 2.4.x へ移行する手順を教えてください
- Bluetooth Smart SDK 2.4.2 から 2.6.x へ移行する手順を教えてください
- Bluetooth Smart SDK 2.6.x から 2.7.x へ移行する手順を教えてください

SDK 2.7～2.11 については、単に使用する SDK を選び直してください。

6 FAQ

BGM について、よく頂く質問をまとめました。

6-1 開発環境・ツール

■ オフラインインストーラがアップデートされたことを知る方法はありますか？

➢ シリコンラボ社のナレッジベースに最新バージョンの情報が掲載されていますので、定期的にこちらをチェックください。(リンク)

■ コードをダウンロードするための簡易ツールはありますか？

➢ スタンドアロンで使用できるツールとして Simplicity Commander が提供されています。(リンク)
Simplicity Studio に搭載されている Simplicity Commander と同じものです。

■ 量産工程で使用できるようなライターやプログラマはありますか？

➢ パッケージが特殊なので、オンボード書き込みが現実的です。J-Link や、Computex 社 FP-10 (リンク)などがご使用頂けます。

6-2 トラブルシューティング

■ コードをダウンロードしましたが、main に飛びません(動作しません)

➢ ブートローダーが書かれていない可能性があります。ブートローダーをビルドしてダウンロードするか、或いはデモ(ブートローダーを含んでいる)をダウンロードしてください。

➢ SDK 2.62 以前から SDK 2.7 移行に変更した場合には、ブートローダーの書き換えが必要です。ブートローダーの仕様が変更になっています。
https://service.macnica.co.jp/support/faq/126929

■ NCP モード時の消費電流が高い

➢ NCP モード時は EM2 が無効になっています。Wakeup ピンを用意することで低消費電力化が可能です。
https://service.macnica.co.jp/support/faq/128533

■ SOC モード時の消費電流が高い

➢ Simplicity Studio の Debug モードで使用していないでしょうか？Debug モードで使用する際には、クロックの都合上 EM2 に移行しないようになっています。Simplicity Studio との通信を切断してからお試しください。

■ ラジオボードが認識されません (Simplicity Studio)

➢ 給電スイッチが AEM になっていないか、デバッグ経路が OUT になっていないか、お確かめください。

■ デバイスにアクセスできなくなりました （Simplicity Studio）

➢ デバッグ経路が遮断（Lock）されると、デバイスへのアクセスが行えなくなります。Unlock することでアクセスできるようになります。
https://service.macnica.co.jp/support/faq/108137

改版履歴

Version	改定日	改定内容
1.0	2017 年 01 月	・新規作成。マクニカオンラインで公開
2.0	2019 年 03 月	・クイックスタートガイドとアドバンストガイドに分割
2.1	2019 年 04 月	・スループット測定を追記。FAQ 更新

参考文献

- Silicon Labs 社　各種ドキュメント
- Silicon Labs 社　ナレッジベース、コミュニティフォーラム

本社
〒222-8561 横浜市港北区新横浜 1-6-3　TEL 045-470-9841　FAX 045-470-9844