UG136 : Silicon Labs *Bluetooth* ® C アプリケーション開発者ガイド

本書は、Silicon Labs Bluetooth スタックを使用して Silicon Labs Wireless Gecko 製品向け C ベース･アプリケーションを開発する際の重要な資料です。このガイドでは、Bluetooth スタックのアーキテクチャ、アプリケーション開発フロー、MCU コアおよびペリフェラルの使用と制限、スタック構成オプション、およびスタック・リソースの使用について説明します。

本書は､Wireless Gecko 向け Bluetooth アプリケーションの開発において､Bluetooth スタック API リファレンス、Gecko SDK API リファレンス、および Wireless Gecko リファレンス・マニュアルに記載されていなかった部分を補うことを目的としています。本書では、開発者が利用可能なハードウェア・リソースを最大限活用できるように、詳しく説明します。

要点

- プロジェクト構造と開発フロー
- Bluetooth スタックと Wireless Gecko の構成
- 割り込み処理
- イベントおよびスリープ管理
- リソースの使用方法と利用可能なリソース

目次

第 1 章　はじめに 4
1.1　前提条件 4
第 2 章　アプリケーション開発フロー 5
2.1　アプリケーション構築フロー 6
第 3 章　プロジェクト構造 7
3.1　Bluetooth スタック・ライブラリとブートローダ・ライブラリ 7
3.2　GATT データベース 7
3.2.1　gatt_db.h 7
3.3　EMLIB および EMDRV ペリフェラル・ソース・コード 7
3.4　aat.h 7
3.5　application_properties.c 8
3.6　native_gecko.h 9
3.7　InitDevice.h 10
3.8　無線ボード固有の機能 10
第 4 章　Bluetooth スタックと Wireless Gecko デバイスの構成 11
4.1　Wireless Gecko MCU とペリフェラルの構成 11
4.1.1　enter_DefaultMode_from_RESET() 11
4.1.2　Bluetooth クロック 11
4.1.3　DC-DC 構成 12
4.2　gecko_stack_init() を使用した Bluetooth スタック構成 13
4.2.1　CONFIG_FLAGS 14
4.2.2　Mbedtls 14
4.2.3　スリープ 14
4.2.4　Bluetooth 接続 14
4.2.5　OTA 構成 15
4.2.6　PTI 15
4.2.7　Bluetooth 5 アドバタイズ・セット 15
第 5 章　Bluetooth スタックのイベント処理 16
5.1　ブロッキング・イベント・リスナ 16
5.2　ノンブロッキング・イベント・リスナ 16
5.2.1　スリープとノンブロッキング・イベント・リスナ 17
第 6 章　割り込み 18
6.1　外部イベント 18
6.2　優先度 19
第 7 章　Wireless Gecko のリソース 20
7.1　フラッシュ 21
7.2　リンク方法 22

7.3 RAM 22
7.3.1 Bluetooth スタック 22
7.3.2 Bluetooth 接続プール 22
7.3.3 Bluetooth GATT データベース. 23
7.3.4 C STACK 23
7.3.5 C HEAP 23
7.4 RTCC 23
第 8 章 付録 : フラッシュおよび RAM 消費量の計算 24
第 9 章 資料 28

第 1 章　はじめに

本書は、Silicon Labs Bluetooth スタック用の C 開発者ガイドです。

本書は、開発についてさまざまな角度から説明するもので、Bluetooth スタックを使用する Wireless Gecko 製品向けに C 言語で開発を行う際の重要な資料になります。

本書では以下のトピックについて説明します。

- セクション 第 2 章 アプリケーション開発フロー では、アプリケーション開発フローとプロジェクト構造について説明します。
- セクション 第 4 章 Bluetooth スタックと Wireless Gecko デバイスの構成 では、プロジェクトに含まれるライブラリと、アプリケーション・コード内の実際の Wireless Gecko の構成について説明します。
- セクション 第 5 章 Bluetooth スタックのイベント処理 は、Silicon Labs Bluetooth スタックを使用する開発者にとって重要なセクションです。ここではイベント・ベースのアーキテクチャ内でアプリケーションがスタックとどのように同期して動作するかを説明します。
- セクション 第 6 章 割り込み とセクション 第 7 章 Wireless Gecko のリソース では、ペリフェラルとチップセット・リソースについて簡単に説明し、スタックの使用向けに予約されるリソース、割り込みの処理、スタックのメモリ・フットプリントとアプリケーションに使用可能なメモリについて取り上げます。

1.1　前提条件

本書では、Silicon Labs の Bluetooth SDK 2.3.0 以降が開発マシン（Windows、MAC OSX、または Linux）に適切にインストールされ、読者がクイック・スタート・ガイドおよび SDK の事例に精通していることを前提としています。また、読者は Bluetooth 技術の基礎を理解している必要があります。詳細については、『*UG104.13：アプリケーション開発の基礎：Bluetooth 技術*』を参照してください。

インストールとツールの概要については、『*QSG108：Silicon Labs の Bluetooth ソフトウェアのご使用について*』を参照してください。Silicon Labs Simplicity Studio 開発環境でアプリケーション例を使用して開始する手順については、『*QSG139：Simplicity Studio を使用した Bluetooth 開発*』を参照してください。

最後に、Silicon Labs ではクイック・コード GATT データベースを生成するための、Bluetooth Developer Studio 用のプラグインを提供しています。プラグインのクイック・スタート・ガイドについては、『*QSG126：Bluetooth® Developer Studio クイック・スタート・ガイド*』を参照してください。

現在サポートされているコンパイラと IDE のバージョンを次の表に示します。

IDE	SDK	Compiler
Simplicity Studio 4.1.0 or newer	v2.3.x	IAR v7.80.2 and GCC 4.9.3
Simplicity Studio 4.0.6 or newer	v2.1.x	IAR v7.80.2
Simplicity Studio 4.0.2 or newer	v2.0.x	IAR v7.6
IAR Embedded Workbench	v2.x.x	IAR v7.6 or newer recommended IAR v7.4 (minimum)

第 2 章　アプリケーション開発フロー

次の図に、ファームウェア構造の概要を示します。開発者はアプリケーションをスタックの上に構築しますが、Silicon Labs はこのスタックをプリコンパイル済みのオブジェクト・ファイルとして提供し、これによってエンド・デバイスの Bluetooth 接続が可能になります。

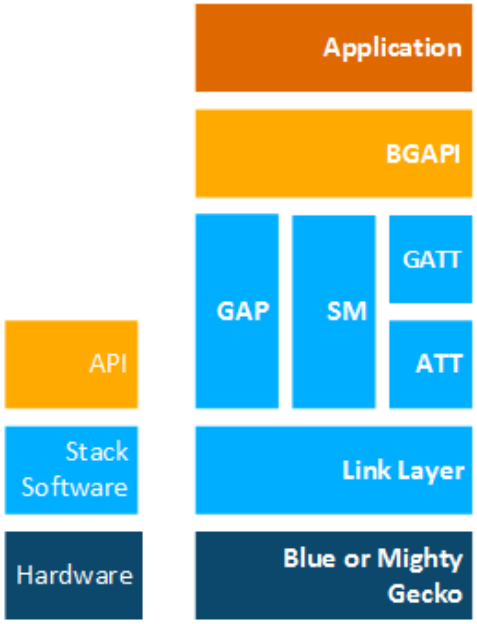


図 2.1.　Bluetooth スタック・アーキテクチャのブロック図

Bluetooth スタックには以下のブロックが含まれています。

- ブートローダ – 現在、以下の 3 つのブートローダが提供されています。詳細については、『*UG103.06：アプリケーション開発の基礎：ブートローディング*』を参照してください。
  - レガシ OTA（無線）ファームウェア・アップデート
  - レガシ UART DFU（デバイス・ファームウェア・アップデート）
  - Gecko ブートローダ。詳細については、『*UG266：Gecko ブートローダ・ユーザ・ガイド*』および『*AN1086：Silicon Labs Bluetooth アプリケーションでの Gecko ブートローダの使用*』を参照してください。
- Bluetooth スタック – リンク・レイヤ、汎用アクセス・プロファイル、セキュリティ・マネージャ、属性プロトコル、および汎用属性プロファイルで構成される Bluetooth の機能。
- OTA スーパーバイザ –ブートローダの後に起動するアプリケーション。これは、ユーザ・アプリケーションが有効かどうかをチェックし、有効な場合は、スーパーバイザがアプリケーションを起動します。アプリケーション・イメージが有効でない場合、スーパーバイザは構成済みのブートローダを起動し、有効なアプリケーション・イメージの受け取りを試みます。これには、レガシ OTA ブートローダまたは Gecko ブートローダを使用する必要があります。

2.1 アプリケーション構築フロー

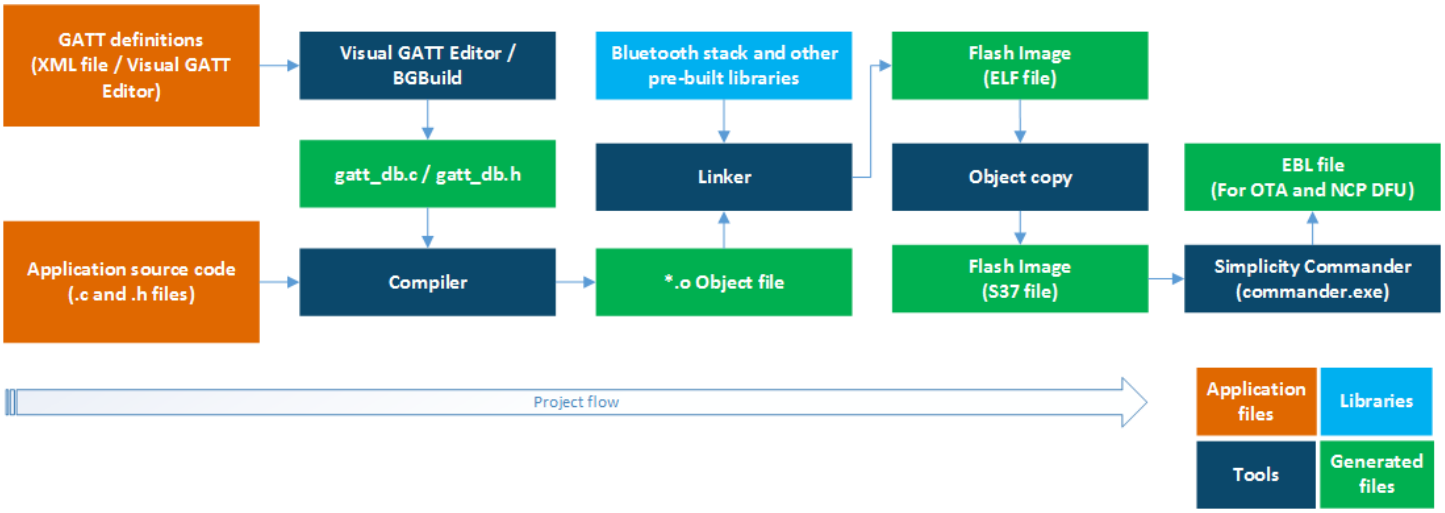


図 2.2. Bluetooth プロジェクトの構築フロー

プロジェクトを構築するには、まず、Bluetooth サービスと特性（GATT 定義）を定義し、Silicon Labs が提供する例、または空のプロジェクト・テンプレートからアプリケーション・ソース・コードを記述します。この手順については、『*QSG139：Simplicity Studio を使用した Bluetooth アプリケーション開発*』で説明しています。

SDK v2.1.0 以降では、Bluetooth サービスと特性を定義する 2 つの方法が用意されています。最初の方法は、Simplicity Studio の Visual GATT Editor GUI を使用します。これは、GATT を設計し、*gatt_db.c* および *gatt_db.h* を生成するためのグラフィカル・ツールです。また、このツールを使用して *.xml* ファイルと *.bgproj* GATT 定義ファイルをインポートできます。Visual GATT Editor は、Simplicity Studio プロジェクトで GATT を定義し生成するためのデフォルト・ツールです。

2 番目の方法は、*.xml* または *.bgproj* を『*UG118：Blue Gecko Bluetooth® Profile Toolkit 開発者ガイド*』に従って作成し、次にプリコンパイル・ステップで BGBuild 実行可能ファイルを使用して GATT 定義ファイルを .c および .h に変換する方法です。この方法は IAR Embedded Workbench プロジェクトで使用されます。

プロジェクトをコンパイルすると、オブジェクト・ファイルが生成されます。このファイルはその後、SDK で提供されるプリコンパイル・ライブラリとリンクされます。リンクにより、サポートされている Wireless Gecko デバイスに対してプログラム可能なフラッシュ・イメージが生成されます。

第 3 章　プロジェクト構造

このセクションでは、アプリケーション・プロジェクト構造と、プロジェクトに含める必要のある必須のリソースおよびオプションのリソースについて説明します。

3.1　Bluetooth スタック・ライブラリとブートローダ・ライブラリ

Bluetooth スタック・ライブラリは以下で構成されます。

- binbootloader.o：ブートローダ・イメージ（プリコンパイル済みのレガシ OTA ブートローダ）。このファイルは、EFR32xG1 デバイスでのみサポートされているレガシ・ブートローダに必要です。v2.3.0 SDK 以降では統合 Gecko ブートローダが利用可能で、これはすべての EFR32 デバイスをサポートしています。詳細については、『*UG266：Silicon Labs Gecko ブートローダ・ユーザ・ガイド*』を参照してください。
- binstack.o：Bluetooth スタック、スーパーバイザ、および Bluetooth スタックで使用されるライブラリのバイナリ・イメージ。
- stack.a：Bluetooth スタック、EMLIB および EMDRV のエクスポートされたシンボル。アプリケーションと共有されます。
- bgapi.a：コア以外の機能の BGAPI コマンドを実装します。SDK v2.3.0 以降では、*bgapi.a* をすべてのプロジェクトに含める必要があります。これ以前の SDK では、NCP アプリケーションにのみ含める必要があります。

3.2　GATT データベース

GATT（汎用属性プロファイル）データベースは、Bluetooth デバイスの Bluetooth プロファイル、サービスおよび特性を記述するための標準的な方法です。Silicon Labs Bluetooth スタックを使用する場合、GATT 定義は、Simplicity Studio の Visual GATT Editor GUI で直接編集するか、または XML で記述して、プレビルド・タスクとして BGBuild 実行可能ファイルに渡します。GATT データベースの作成方法と構文の詳細については、『*UG118：Blue Gecko Bluetooth® Smart Profile Toolkit 開発者ガイド*』を参照してください。

3.2.1　gatt_db.h

gatt_db.h ヘッダ・ファイルは、BGBuild.exe または Visual GATT Editor により自動生成された GATT データベース構造を表記したものです。GATT のタイプ定義は、gatt_db_def.h から gatt_db.h に自動的に含められます。

3.3　EMLIB および EMDRV ペリフェラル・ソース・コード

EMLIB は低レベルのペリフェラル・サポート・ライブラリです。Silicon Laboratories の EFM32、EZR32、EFR32 MCU および SoC のすべてに対応した統合 API を提供します。EMDRV は、EFR32 オンチップ・ペリフェラル用の機能に特化した高性能ドライバのセットです。ドライバは通常、DMA ベースで、利用可能なすべての低エネルギー機能を使用します。ほとんどのドライバについて、API は、同期機能と非同期機能の両方を提供します。

開発者は、プロジェクトに含めるペリフェラル・ドライバを選択できます。EMDRV ソース・コードの必要なファイルをプロジェクトに含めることができますが、スタック用に予約されているリソースを使用しないことと、無線割り込みの優先度を最も高くすることに注意してください。セクション 第 6 章 割り込み とセクション 第 7 章 Wireless Gecko のリソース では、割り込みとリソースの制限についてさらに詳しく説明します。

EMLIB と EMDRV の詳細については、Gecko SDK API のドキュメントとさまざまなアプリケーション・ノートを参照してください。

3.4　aat.h

AAT（アプリケーション・アドレス・テーブル）は、ファームウェア・アップデート・ファイル形式 EBL のサポート構造（メタ・データ）です。アップグレード・イメージ内のアプリケーションのサイズと場所を定義します。これはアプリケーション・イメージ内の最初に配置され、アプリケーション・ベクタ・テーブルへのポインタを含んでいます。スタック・スーパーバイザは、アプリケーション・ベクタ・テーブルを初期化し、アプリケーションを呼び出す前にベクタ・テーブルからスタックの場所を呼び出します。

ATT は GBL アップグレード・イメージ内のアプリケーションのサイズと場所を定義しませんが、それでも必要です。このため、AAT はすべてのプロジェクトに必須の要素であり、アプリケーションには aat.h ヘッダ・ファイルを含める必要があります。

3.5 application_properties.c

このファイルには、タイプ、バージョン、セキュリティなどのアプリケーション・イメージに関する情報をなど、アプリケーション・プロパティ構造体が含まれます。この構造体は Gecko ブートローダ API の application_properties.h で定義されます（<Simplicity Studio Gecko SDK>¥platform¥bootloader¥documentation¥Gecko_Bootloader_API_Reference¥index.html の Gecko ブートローダ API リファレンスを参照）。事前に生成されたファイルは Simplicity Studio プロジェクトに含められ、アプリケーション固有のプロパティを含めるように変更できます。アプリケーション・プロパティには Gecko ブートローダ API を使用してアクセスできます。以下のメンバーは定義を変更して更新することができます。

```
// Version number for this application (uint32_t)
#define BG_APP_PROPERTIES_VERSION

// Capabilities of this application (uint32_t)
#define BG_APP_PROPERTIES_CAPABILITIES

// Unique ID (e.g. UUID or GUID) for the product this application is built for (uint8_t[16])
#define BG_APP_PROPERTIES_ID
```

OTA スーパーバイザを使用する場合、アプリケーション・プロパティの app.capabilities 構造体メンバーに、Bluetooth スタックのバージョン番号を含める必要があります。

```
#define BG_APP_PROPERTIES_CAPABILITIES (BG_VERSION_MAJOR << 24) | (BG_VERSION_MINOR << 16) | (BG_VERSION_PATCH << 8)
```

アプリケーション・プロパティ構造体は、Gecko ブートローダを使用する場合は必須で、フラッシュ内のどこにでも配置できます。

3.6 native_gecko.h

このファイルには 2 つの目的があります。最初の目的は、実際の Bluetooth スタック API と、そのスタックのコマンドとイベントを含めること、2 番目の目的は、構成、イベント、およびスリープ管理 API を Bluetooth スタックに提供することです。

構成、イベント、およびスリープ管理 API を以下で説明します。

```
void gecko_init(const gecko_configuration_t*config)
```

この関数は、単一の引数（gecko_configuration_t 構造体へのポインタ）を取ります。この関数の目的は、構造体に提供されたパラメータで Bluetooth スタックを構成し初期化することです。構成オプションと gecko_init() の使用方法については次のセクションで詳しく説明します。gecko_init() は、Bluetooth スタックを初期化するためにアプリケーションによって呼び出される必要があります。

SDK v2.3.0 以降でこの関数は廃止されますが、下位互換性は維持されます。

```
void gecko_stack_init(const gecko_configuration_t*config)
```

この関数は、単一の引数（gecko_configuration_t 構造体へのポインタ）を取ります。この関数の目的は、構造体に提供されたパラメータで Bluetooth スタックを構成し初期化することです。関数 gecko_stack_init() が呼び出された後、各スタックが使用するコンポーネントを個別に初期化する必要があります。個別に行うことで、不要なスタック・コンポーネントを含めずにメモリを最適化することができます。

以下の API は、スタック・コンポーネントを個別に初期化するために使用できます。

- gecko_bgapi_class_dfu_init();
- gecko_bgapi_class_system_init();
- gecko_bgapi_class_le_gap_init();
- gecko_bgapi_class_le_connection_init();
- gecko_bgapi_class_gatt_init();
- gecko_bgapi_class_gatt_server_init();
- gecko_bgapi_class_endpoint_init();
- gecko_bgapi_class_hardware_init();
- gecko_bgapi_class_flash_init();
- gecko_bgapi_class_test_init();
- gecko_bgapi_class_sm_init();

```
struct gecko_cmd_packet* gecko_wait_event(void)
```

これは、イベントが Bluetooth スタックから送信されるのを待ち、イベントを受信するまでブロックする、ブロッキング関数です。イベントが受信されると、gecko_cmd_packet 構造体へのポインタが返されます。

Bluetooth スタック構成で EM スリープ・モードが有効になっている場合、Bluetooth スタックからイベントを受信しないと、デバイスは自動的に EM1 または EM2 モードになります。できるだけデバイスを消費電力の最も低いスリープ・モードにするには、gecko_wait_event() を使用するのが一番簡単な方法です。

Bluetooth スタックのイベント処理は、セクション 第 5 章 Bluetooth スタックのイベント処理 で詳しく説明します。

```
struct gecko_cmd_packet* gecko_peek_event(void)
```

これは、Bluetooth スタックから Bluetooth イベントを要求する、ノンブロッキング関数です。イベントが要求され、イベント・キューが空でない場合、gecko_cmd_packet 構造体へのポインタが返されます。イベント・キューにイベントがない場合は NULL が返されます。

このノンブロッキング・イベント・リスナを使用する場合、EM スリープ・モードは Bluetooth スタックで自動的に管理されないためアプリケーション・コードで管理する必要があります。このスリープ・モード管理は gecko_can_sleep_ms() 関数と gecko_sleep_for_ms() 関数で行いますが、これらは後で説明します。

スタックのイベント処理は、セクション 第 5 章 Bluetooth スタックのイベント処理 で詳しく説明します。

```
int gecko_event_pending(void)
```

この関数は、イベント・キューに保留中の Bluetooth スタック・イベントがあるかどうかを確認します。保留中の Bluetooth イベントが検出された場合、関数は 0 以外の値を返し、gecko_peek_event() または gecko_wait_event() によってイベントを処理する必要があることを示します。イベントが検出されない場合は、0 が返されます。

```
uint32 gecko_can_sleep_ms(void)
```

この関数を使用して、Bluetooth スタックをスリープ状態にできる時間を決定します。戻り値は、次の Bluetooth 動作が発生するまでスタックをスリープ状態にできるミリ秒数です。スリープ状態にできない場合は、0 が返されます。この関数は、ノンブロッキング gecko_peek_event() イベント処理でのみ使用されます。

```
uint32 gecko_sleep_for_ms(uint32 max)
```

この関数は、最大ミリ秒数（このミリ秒数は関数の単一パラメータで設定されます）スタックを EM スリープ状態にする場合に使用されます。戻り値は、実際にスリープ状態になったミリ秒数です。外部イベントによって、スタックのスリープ状態が解除される可能性があります。この関数は、ノンブロッキング gecko_peek_event() イベント処理でのみ使用されます。

3.7 InitDevice.h

このヘッダ・ファイルには、スタック初期化を除く、デバイスの初期化関数が含まれています。この初期化関数は、たとえば、クロックと電源管理を初期化しますが、USART、I2C、ADC、GPIO などのペリフェラル・ハードウェア・インターフェイスも初期化します。

Bluetooth スタックとペリフェラルの初期化については、セクション 第 4 章 Bluetooth スタックと Wireless Gecko デバイスの構成 でさらに詳しく説明します。

3.8 無線ボード固有の機能

Bluetooth SDK に用意されている例の一部では、LCD ディスプレイや相対湿度/温度センサーなど、WSTK 開発キット固有の機能を使用しています。スターター・キットを使用するクイック・プロトタイプ開発用に、これらの機能を使用するための関数が簡略化されています。以下のヘッダ・ファイルを使用すると、特定の無線ボードの機能を簡単に構成できます。

bspconfig.h

BSP（ボード・サポート・パッケージ）ヘッダには、無線ボード固有の構成が含まれています。これらの構成は、WSTK の IO のトグル、スターター・キットの LCD ディスプレイの駆動など、WSTK 固有の関数のパラメータとして使用されます。

flashpwr.h

このヘッダ・ファイルには、一部の無線ボード（BRD4100A など）で SPI フラッシュ・チップを低消費電力モードに構成するための関数が含まれています。これは、たとえば、SPI フラッシュが低消費電力モードでない場合は、最小の EM2、EM3、または EM4 電流を達成できないため、スリープ電流を測定する際に役立ちます。

pti.h / boards_pti.h

PTI（パケット・トレース・インターフェイス）は Wireless Gecko SoC の組み込みブロックで、上下双方向の無線パケットを生データとしてデバッグ・インターフェイスにルーティングします。これらのパケットを Simplicity Studio のネットワーク・アナライザで取り込んで表示することができます。ネットワーク・アナライザには、Bluetooth パケット用のデコーダがあり、Bluetooth ネットワークのデバッグ、分析、および測定に使用できます。

PTI は HW ブロックを有効にするためにコードで初期化される必要があり、PTI ピンは対応する無線ボード・ピンに配線されている必要があります。

第 4 章　Bluetooth スタックと Wireless Gecko デバイスの構成

Wireless Gecko で Bluetooth スタックとアプリケーションを実行するには、MCU とそのペリフェラルが適切に構成されている必要があります。ハードウェアを初期化した後、gecko_init() 関数を使用してスタックも初期化する必要があります。

4.1　Wireless Gecko MCU とペリフェラルの構成

4.1.1　enter_DefaultMode_from_RESET()

enter_DefaultMode_from_RESET 関数は、ボード設計に基づいて MCU コアとペリフェラルの設定を初期化するために使用されます。この関数は、main() の最初に呼び出される必要があります。

アプリケーションを Silicon Labs の無線ボード用に開発する場合、これらの設定は SDK で提供される例では正しく設定されていますが、カスタム・ハードウェア設計用のアプリケーションを作成する開発者は、必要に応じて設定を構成する必要があります。Simplicity Studio の Hardware Configurator を使用すると、グラフィカル・ユーザ・インターフェイスでペリフェラルを構成でき、Configurator の変更を保存した後、構成コードが自動的に生成されます。次のセクションで、生成される構成コードについて説明します。

enter_DefaultMode_from_RESET は、さまざまな MCU とペリフェラル・ブロックを構成するために特別に設計された関数のセットを呼び出します。

4.1.2　Bluetooth クロック

クロック設定は CMU_enter_DefaultMode_from_RESET 関数内で初期化されます。クロック設定には、使用されるクロック・ソース（HFXO、LFXO、PLFRCO、および LFRCO）、チューニングなどのクロック・パラメータ、ペリフェラルのクロック・ソースなどが含まれます。

HFCLK

HFCLK は無線プロトコル・タイマ（PROTIMER）に使用されます。HFCLK は高周波数クロックです。精度は ±50 ppm 以上である必要があり、このクロックが十分な精度を維持するには外部水晶が必要です（HFXO）。

HFXO の初期化により、タイミングが重視される接続とスリープ管理を確保できるように外部水晶を構成します。HFXO は、高周波数クロック（HFCLK）として設定し、Wireless Gecko の HFXO 入力ピンに物理的に接続する必要があります。

LFCLK

デバイスが HFCLK を停止し、スリープ・モードになるためには、別のクロックが必要になります。これが LFCLK クロックです。

デバイスがスリープ・モードになる際、PROTIMER の現在の状態が保存されます。デバイスは、ウェイク・アップすると、経過したスリープ・クロックのティック数を計算し、それに応じて PROTIMER を調整します。無線ボードでは、PROTIMER が動作を続けていたように見えます。

このクロックの精度は、デバイスの動作モードによって異なります。アドバタイズやスキャンの場合、精度はあまり重要ではありませんが、接続がオープンである場合は、±500 ppm 以上の精度が必要です。このクロックは、精度要件に応じて、LFXO、PLFRCO、または LFRCO のいずれかで駆動できます。

デフォルトの構成では、LFXO が Wireless Gecko に接続され、低周波数クロック（LFCLK）として設定されます。設計上 LFXO に接続できない場合、または PLFRCO を使用できない場合は、アプリケーションからスリープを明示的に無効にし、LFRCO がクロック・ソースとして使用されるように設定する必要があります。セクション 4.2.3 スリープで説明するように、LFCLK の精度が十分でない場合、Bluetooth スタックを正確に動作させるために、スリープ・モードを無効にする必要があります。

HFXO CTUNE

下の例では、水晶チューニング（CTUNE）設定が、Silicon Labs の Bluetooth モジュール、基準設計、および無線ボードのすべてで動作するようにデフォルトで設定されています。ただし、最終製品の設計では、デバイスごとに、または設計ごとに、個別に水晶校正が必要になる場合があります。CTUNE 値 uint_16 ctuneStedyState は、CMU_HFXOInit_TypeDef hfxoInit 構造内で設計に応じて調整できます。

```
// $[High Frequency Clock Setup]
/* Initializing HFXO */
CMU_HFXOInit_TypeDef hfxoInit = CMU_HFXOINIT_DEFAULT;
hfxoInit.autoStartEm01     = 1;    // HFXO started automatically
hfxoInit.ctuneSteadyState     = 322;     // HFXO CTUNE configuration
CMU_HFXOInit(&hfxoInit);
```

HFXO と LFXO の構成の詳細については、『EFR32 リファレンス・マニュアル』の第 12 章を参照してください。

注：Bluetooth スタックでは 38.4 MHz HFXO 周波数のみをサポートしています。その他の HFXO 周波数はサポートされていません。

4.1.3 DC-DC 構成

DCDC 構成は EMU_enter_DefaultMode_from_RESET 関数で設定します。SDK の例では、DC-DC 構成が Silicon Labs の Bluetooth モジュール、無線ボード、および基準設計で動作するように設定されていますが、カスタム設計では個別に DC-DC 設定を行う必要がある場合があります。

```
// $[EMU Initialization]
/* Initialize DCDC regulator */
EMU_DCDCInit_TypeDef dcdcInit = EMU_DCDCINIT_DEFAULT;

dcdcInit.powerConfig         = emuPowerConfig_DcdcToDvdd;
dcdcInit.dcdcMode            = emuDcdcMode_LowNoise;
dcdcInit.mVout               = 1800;
dcdcInit.em01LoadCurrent_mA     = 15;
dcdcInit.em234LoadCurrent_uA       = 10;

dcdcInit.maxCurrent_mA          = 200;
dcdcInit.anaPeripheralPower     = emuDcdcAnaPeripheralPower_DCDC;
dcdcInit.reverseCurrentControl     = 160;

EMU_DCDCInit(&dcdcInit);
```

DCDC の構成の詳細については、『EFR32 リファレンス・マニュアル』の第 11 章と、『*AN0948：電力構成と DC-DC*』を参照してください。

4.2 gecko_stack_init() を使用した Bluetooth スタック構成

gecko_stack_init() 関数は、スリープ・モード構成、接続用のメモリ割り当て、OTA 構成などを含む、Bluetooth スタックを構成するために使用されます。Bluetooth スタックが構成されていない場合は、どの Bluetooth スタック関数も使用することはできません。

Bluetooth スタック構成の例：

```
/* Gecko configuration parameters (see gecko_configuration.h) */
static const gecko_configuration_t config = {
  .config_flags=0,
  .sleep.flags=SLEEP_FLAGS_DEEP_SLEEP_ENABLE,
  .bluetooth.max_connections=MAX_CONNECTIONS,
  .bluetooth.heap=bluetooth_stack_heap,
  .bluetooth.heap_size=sizeof(bluetooth_stack_heap),
  .bluetooth.sleep_clock_accuracy = 100, // ppm
  .gattdb=&bg_gattdb_data,
  .ota.flags=0,
  .ota.device_name_len=3,
  .ota.device_name_ptr="OTA",
  .pti = &ptiInit,
};
```

gecko_stack_init() 関数の構成オプションは、スリープの有効化/無効化、Bluetooth 接続カウント、ヒープ・サイズ、GATT データベース、OTA 構成、および PTI です。クロックなど、その他の初期化は、enter_DefaultMode_from_RESET() 関数で構成され、この関数は部品と設定されたオプションに基づいて Hardware Configurator で自動的に生成されます。

関数 gecko_stack_init() が呼び出された後、使用される各スタック・コンポーネントを個別に初期化する必要があります。個別に行うことで、不要なスタック・コンポーネントを含めずにメモリを最適化することができます。

以下の API は、スタック・コンポーネントを個別に初期化するために使用できます。

gecko_bgapi_class_dfu_init()	デバイス・ファームウェア・アップグレード (dfu) API を有効にします。
gecko_bgapi_class_system_init()	ローカル・デバイス（システム）API を有効にします。
gecko_bgapi_class_le_gap_init()	汎用アクセス・プロファイル (gap) API を有効にします。
gecko_bgapi_class_le_connection_init()	接続 API を介して、接続の確立、パラメータ設定、および切断手順を管理できます。
gecko_bgapi_class_gatt_init()	gatt API を介して、リモート GATT サーバ内の属性を参照/管理できます。
gecko_bgapi_class_gatt_server_init()	gatt_server API を介して、ローカル GATT データベース内の属性を参照/管理できます。
gecko_bgapi_class_endpoint_init()	エンドポイント API を介して、エンドポイントの作成と削除、およびデータ・ルーティングの構成できます。
gecko_bgapi_class_hardware_init()	ハードウェア API を介して、システム・ハードウェアとペリフェラルのアクセスと構成を有効にします。
gecko_bgapi_class_flash_init()	フラッシュ・メモリのユーザ・データを管理するために使用できる固定ストア・コマンド（フラッシュ）API を有効にします。
gecko_bgapi_class_test_init()	DTM テスト API を有効にします。
gecko_bgapi_class_sm_init()	セキュリティ・マネージャ (sm) API を有効にします。
gecko_bgapi_class_util_init()	*atoi*、*itoa* などのユーティリティ関数 API を有効にします。

4.2.1 CONFIG_FLAGS

現在のフラグ：

GECKO_CONFIG_FLAG_USE_LAST_CTUNE	1= CTUNE をオーバーライドして、PS ストアに最後に保存された値を使用します。
GECKO_CONFIG_FLAG_SUPERVISOR	0= 通常のアプリケーション。OTA 設定をフラッシュに保存する必要があります。 1= アプリケーションは OTA のスーパーバイザで、保存されている OTA 設定を使用します。
GECKO_CONFIG_FLAG_RTOS	1= アプリケーションは RTOS を使用します。クロック、ベクタ、TEMPDRV、またはスリープは RTOS から提供されるためスタックでは構成しません。

4.2.2 Mbedtls

これは、スタックが使用する暗号化ライブラリを構成します。Mbedtls は、暗号化演算のハードウェア・アクセラレーションに使用する、クラス 1（sl_*.c）およびクラス 2（slcl_*.c）のプラグインを提供します。この 2 つのプラグイン・クラスは相互互換性がありません。Bluetooth スタックはクラス 2 プラグインを使用するため、アプリケーションで暗号化関数を使用する場合は、クラス 2 プラグインも使用する必要があります。

暗号化演算を使用する前に、暗号化のハードウェア・アクセラレーションを初期化する必要があります。この初期化を行うには、使用する暗号化デバイス用に mbedtls_device_init() と mbdedtls_device_set_instance() を呼び出します。デバイス・コンテキストの初期化は 1 回だけ行う必要があります。GECKO_MBEDTLS_FLAGS_NO_MBEDTLS_DEVICE_INIT フラグが構成オプション .mbedtls.flags で設定されていない場合、Bluetooth スタックは、使用する暗号化デバイスの初期化を自動的に行います。設定されている場合は、暗号化デバイスの自動初期化が無効になります。アプリケーションで暗号化デバイスの初期化を処理する場合、このフラグを設定する必要があります。

使用可能な暗号化デバイスが複数ある場合、構成オプション .mbedtls.dev_number で Bluetooth スタックが使用する暗号化デバイスを定義します。デフォルトは 0 です。

```
.mbedtls.flags = 0,             // GECKO_MBEDTLS_FLAGS_NO_MBEDTLS_DEVICE_INIT disable automatic
.mbedtls.dev_number = 0,              // initialization of crypto device
```

4.2.3 スリープ

Wireless Gecko のスリープ・モード EM2（エネルギー・モード 2）は gecko_init() 関数で有効にする必要があります。スリープ・フラグは gecko_configuration_t 構造体の一部です。SLEEP_FLAGS_DEEP_SLEEP_ENABLED フラグを、スリープを有効にするように設定する必要があります。セクション 第 5 章 Bluetooth スタックのイベント処理 で説明するように、ブロッキング・イベントではスリープ・モードはスタックにより自動的に処理されます。

gecko_configuration_t 構造体（main.c）でスリープを有効にする例：

```
.sleep.flags     = SLEEP_FLAGS_DEEP_SLEEP_ENABLE     // EM sleeps enabled
```

スリープ・モードを使用するには、正確な 32 kHz 低周波数クロック（LFCLK）がハードウェアに必要です。Bluetooth スタックで正確なスリープ・クロックを使用できない場合、低消費電力スリープ・モードにすることはできません。低消費電力スリープ・モードを必要としないアプリケーションの場合、LFXO を省略することができますが、この場合は gecko 構成構造体のスリープ・フラグを次のように設定する必要があります。

```
  .sleep.flags              = 0,                                  // Sleeps disabled
```

4.2.4 Bluetooth 接続

スタックで許容される Bluetooth 同時接続の最大数は、接続管理で割り当てられているメモリ容量によって制限されます。メモリは gecko_init() で初期化時に割り当てられ、定義 #define MAX_CONNECTIONS により決定されます。接続が 1 つ増えるごとに、RAM 使用量は約 800 バイト増加します。

Bluetooth 接続を 1 つに制限する例。

```
#define MAX_CONNECTIONS 1
```

4.2.5 OTA 構成

ファームウェア・アップグレードの一部は OTA スーパーバイザ・アプリケーションにより処理されるため、Bluetooth 無線 (OTA) ファームウェア・アップグレードを利用できます。

OTA には、OTA フラグを介して構成される構成オプションがいくつかあります。

Note: Gecko ブートローダを使用する場合は、セキュリティに OTA フラグを使用しないでください。代わりに、Gecko ブートローダを介してセキュリティを構成します。

それ以外の場合は、OTA フラグを使用して、OTA サービスで使用される OTA 制御ポイントおよび OTA データ特性セキュリティ・プロパティを制御します。これにより、OTA ファームウェア・アップデートを実行するために必要な Bluetooth セキュリティのレベルを制御します。未承認のデバイスによって新しいファームウェア・イメージがデバイスにアップロードされるのを防ぐために、OTA フラグでセキュリティを使用することを強くお勧めします。

```
.ota.flags            = 0,      // 0x0 No restrictions
                                // 0x200 Authenticated write
                                // 0x100 Encrypted write
                                // 0x400 Bonded write
```

Wireless Gecko のデバイス名、デバイスがいつスーパーバイザの OTA モードになるか、およびデバイス名の長さは、gecko 構成構造体を使用して構成できます。

```
.ota.device_name_len        = 3,     // OTA name length
.ota.device_name_ptr        = "OTA",    // OTA Device Name
```

最後に、デバイスを OTA DFU モードに確実に設定して、信頼できるデバイスのみに機能を制限する必要があります。

OTA ファームウェア・アップデートの詳細については、『*AN1045：EFR32xG1 および BGM11x シリーズ製品の Bluetooth 無線デバイス・ファームウェア・アップデート*』および『*UG266：Silicon Labs Gecko ブートローダ・ユーザ・ガイド*』を参照してください。

4.2.6 PTI

gecko_init() の PTI 構造体は、インターフェイス・クロック速度、DCLK のピン、DFRAMEDOUT、およびモードを含む、パケット・トレース出力のインターフェイス構成を設定するために使用されます。

各無線ボードの boards_pti.h には、その特定の無線ボードの正しい構成セットがあります。カスタム設計では、ボード・レイアウトに合わせて構造体を修正する必要があります。

4.2.7 Bluetooth 5 アドバタイズ・セット

アドバタイズ・セットの最大数を定義する必要があります。これらのセットを使用すると、新しい bt5_set_mode コマンドを使用して複数のアドバタイザを開始できます。コンテキストごとに約 60 バイトの RAM が割り当てられます。

```
. uint8_t max_advertisers;  //!< Maximum number of advertisers to support, if 0 defaults to 1
```

第 5 章　Bluetooth スタックのイベント処理

Wireless Gecko 用の Bluetooth スタックはイベント駆動型アーキテクチャで、イベントはメイン while ループで処理されます。

5.1　ブロッキング・イベント・リスナ

gecko_wait_event() は、ブロッキング待機関数の実装であり、イベントがイベント・キューに入るまで待って、イベントをイベント・ハンドラに返します。この動作モードでは、デバイスと接続の同期を維持しながら、スリープを最も効率よく自動的に管理できるため、Bluetooth スタックではこのモードが推奨されます。

- gecko_wait_event() 関数は、イベントが受信されるまで、内部メッセージ・キューを処理します。
- 処理する保留中のイベントやメッセージがない場合、デバイスは EM1 または EM2 スリープ・モードになります。
- この関数は、受信したイベントを保持しながら、gecko_cmd_packet 構造へのポインタを返します。

以下のコード・スニペットは、iBeacon の例の簡単なメインの while ループを示しています。ここでは gecko_wait_event() を使用して、起動後にアドバタイズを設定します。

```
/* Main loop */
while (1) {
    struct gecko_cmd_packet* evt;

    /* Wait (blocking) for a Bluetooth stack event. */
    evt = gecko_wait_event();

    /* Run application and event handler. */
    switch (BGLIB_MSG_ID(evt->header))
    {
    /*  This boot event is generated when the system is turned on or reset. */
    case gecko_evt_system_boot_id:

      /* Initialize iBeacon ADV data */
      bcnSetupAdvBeaconing();
      break;

    /* Ignore other events  */
    default:
      break;
    }
```

5.2　ノンブロッキング・イベント・リスナ

この動作モードでは、より多くの手動調整が必要です。たとえば、スリープ管理はアプリケーションで行う必要があります。一部の使用事例では、ノンブロッキング動作が必要になります。

- gecko_peek_event() 関数は、イベントが受信されるまで、またはすべてのメッセージが処理されるまで、内部メッセージ・キューを処理します。
- この関数は、受信したイベントを保持しながら gecko_cmd_packet 構造へのポインタを返します。またはキューにイベントがない場合は NULL を返します。

5.2.1 スリープとノンブロッキング・イベント・リスナ

アプリケーションがノンブロッキング gecko_peek_event() 関数を使用してイベント・ハンドラを作成する場合、スリープ実装も異なります。アプリケーションは、gecko_can_sleep_ms() を使用してデバイスをスリープ状態にできる時間についてスタックに対してクエリを実行し、次に gecko_sleep_for_ms() 関数を使用してその時間スリープ状態になるように設定する必要があります。gecko_can_sleep_ms() 関数または gecko_sleep_for_ms() 関数を呼び出す前に割り込みを無効にし、関数が実行された後に割り込みを有効にする必要があります。

以下の例は、ノンブロッキング・イベント処理を使用する場合のスリープ管理の実装方法を示しています。

CORE_ENTER_ATOMIC()

```
/* Main loop */
while (1) {
    struct gecko_cmd_packet* evt;
    CORE_DECLARE_IRQ_STATE;

    /* Poll (non-blocking) for a Bluetooth stack event. */
    evt = gecko_peek_event();

    /* Run application and event handler. */
    switch (BGLIB_MSG_ID(evt->header))
    {
    /* This boot event is generated when the system is turned on or reset. */
    case gecko_evt_system_boot_id:

      /* Initialize iBeacon ADV data */
      bcnSetupAdvBeaconing();
      break;

    /* Ignore other events */
    default:
      break;
    }
     CORE_ENTER_ATOMIC();                    // Disable interrupts

      /* Check how long the stack can sleep */
      uint32_t durationMs = gecko_can_sleep_ms();
      /* Go to sleep. Sleeping will be avoided if there isn't enough time to sleep */
      gecko_sleep_for_ms(durationMs);

    CORE_EXIT_ATOMIC();                     // Enable interrupts

}
```

第 6 章　割り込み

無線割り込みまたは IO ピンからの割り込みによって、それぞれの割り込みハンドラでイベントが生成されます。イベントは、メッセージ・キューからのメイン・イベント・ループで後で処理されます。アプリケーションは、常に割り込みハンドラ内での処理時間を最小限にし、イベント・コールバックやメイン・ループで処理を行う必要があります。

一般に、割り込みスキームはイベント・ベースのプログラミング・アーキテクチャに従いますが、Bluetooth スタックには、他にはない重要な例外がいくつかあります。

- BGAPI コマンドは、割り込みコンテキストから呼び出せません。
- 割り込みコンテキストから呼び出せるのは、gecko_external_signal() 関数のみです。
- 前のコード例に示したように、gecko_sleep_for_ms(...) を呼び出す前に、割り込みを無効にする必要があります。

6.1　外部イベント

外部イベントは、すべてのペリフェラル割り込みを外部信号として取り込み、メイン・イベント・ループに渡してループ内で処理するために使用されます。外部イベント割り込みは、IO、コンパレータ、ADC など、さまざまなペリフェラル割り込みソースから発生します。どのような外部割り込みが発生したかをイベント・ハンドラに通知するために、信号ビット配列が使用されます。

- 外部信号の主な目的は、割り込みコンテキストからメイン・イベント・ループに対してイベントをトリガすることです。
- BGAPI イベント system_external_signal は、void gecko_external_signal(uint32 signals) 関数を呼び出すことにより生成できます。
- 関数 gecko_external_signal は割り込みコンテキストから呼び出せます。
- gecko_external_signal 関数の signals パラメータは system_external_signal イベントに渡されます。

```
/**
* Main
*/
void main()
{
  ...

      //Event loop
  while(1)
  {
    ...

      //External signal indication (comes from the interrupt handler)
    case gecko_evt_system_external_signal_id:
      // Handle GPIO IRQ and do something
      // External signal command's parameter can be accessed using
     // event->data.evt_system_external_signal.extsignals
      break;
    ...
  }
}

/**
* Handle GPIO interrupts and trigger system_external_signal event
*/
void GPIO_ODD_IRQHandler()
{
  static bool radioHalted = false;

  uint32_t flags = GPIO_IntGet();
  GPIO_IntClear(flags);

  //Send gecko_evt_system_external_signal_id event to the main loop
  gecko_external_signal(...);
  }
}
```

6.2 優先度

無線割り込みの優先度を最も高くすることを強くお勧めします。これはデフォルトの構成で、その他の割り込みはこれより低い優先度で処理されます。

次の表では、Bluetooth スタック内の動作コンテキストが異なる 3 つのコンポーネントと、各コンポーネントの接続を確保するために割り込みを無効にする最大時間を説明します。

Component	Description	Timing accuracy	Operating Context	Maximum IRQ disable	What happens if timing requirements are ignored
Radio	Time-critical low level TX/RX radio control	Microseconds	Radio IRQ	< ~10 µs	Packets are not transmitted or received, which will eventually cause supervision timeout and Bluetooth link loss.
Link layer	Time-critical connection management procedures and encryption	Milliseconds	PendSV IRQ	< ~20 ms	If the link control procedure is not handled in time, Bluetooth link loss may happen. Slave-side channel map update and connection update timings are controlled by master.
Host Stack	Bluetooth Host Stack, Security Manager, GATT	Seconds	Application	< 30 s	SMP and GATT have a 30 s timeout and if operations are not handled within that timeout Bluetooth link loss will occur.

第 7 章　Wireless Gecko のリソース

Bluetooth スタックで使用する Wireless Gecko のリソースの一部は、アプリケーションには使用できません。以下の表に、これらのリソースを示し、スタックがこれらをどのように使用するかを説明します。最初の 4 つのリソース（赤で表示）は、常に Bluetooth スタックで使用されます。

Category	Resource	Used in software	Notes
PRS	PRS7	PROTIMER RTC synchronization	PRS7 always used by the Bluetooth stack.*
Timers	RTCC	EM2 timings	Used for sleep timings. Both channels are always reserved. The application can only read the RTC value, but cannot write it or use RTCC.
	PROTIMER	Bluetooth	The application does not have access to PROTIMER.
Radio	RADIO	Bluetooth	Always used and all radio registers are reserved for the Bluetooth stack.
GPIO	NCP	Host communication.	2 to 6 x I/O pins can be allocated for the NCP usage depending on used features (UART, RTS/CTS, wake-up and host wake-up). Optional to use, and valid only for NCP use case.
	PTI	Packet trace	2 to N x I/O pins. Optional to use.
	TX enable	TX activity indication	1 x I/O pin. Optional to use.
CRC	GPCRC	PS Store	Can be used in application, but application should always reconfigure GPCRC before use, and GPCRC clock must not be disabled in CMU.
Flash	MSC	PS Store	Can be used by application, but MSC must not be disabled.
CRYPTO	CRYPTO	BLE link encryption	The CRYPTO peripheral can only be accessed through the mbedTLS crypto library, not through any other means. The library should be able to do the scheduling between the stack and application access.

* SDK 2.3 では Hardware Configurator によって不要な PRS 0、1、2、3 および 11 チャンネルが予約されています。これらの PRS チャンネルはスタックによって使用されませんが、GUI で構成することはできません。

7.1 フラッシュ

Bluetooth スタックはフラッシュ・メモリから実行され、現在は（v.2.0.0 以降のソフトウェア）フラッシュの約半分を占めます。フラッシュは、次の図に示すように、ブートローダ用、Bluetooth スタック用、GATT データベース用、および固定ストア（PS ストア）用のブロックに分割できます。

- ブートローダは、Bluetooth スタックとアプリケーションのアップグレード性を有効にするために必要です。ブートローダは将来に対応した設計となっており、改良を行い機能を追加できます。
- Bluetooth スタックのブロックには、リンク・レイヤ、GAP、SM、ATT、GATT レイヤなどの実際の Bluetooth ファームウェアに加え、OTA のアップグレード可能性を確保するための小型のスーパーバイザ・アプリケーションと、Bluetooth スタックに必要なハードウェア固有のライブラリも含まれています。
- GATT データベースにはアプリケーションで使用される Bluetooth サービスと特性が含まれ、データベースのサイズは、そのアプリケーションと、GATT サービスおよび特性がいくつ含まれているかによって異なります。
- PS ストアは不揮発性のデータ・ストアで、ここには Bluetooth スタックとアプリケーションの両方が Bluetooth 結合キー、アプリケーションの構成データ、ハードウェア構成などの永久データを保存できます。PS ストアはフラッシュの最後の 4 kB にあります。

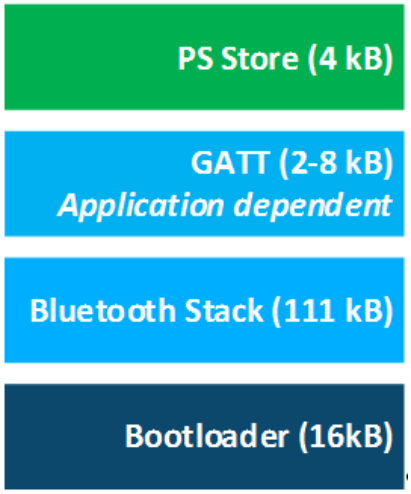


図 7.1. フラッシュの使用状況

次の表に、フラッシュの使用状況と各ブロックのアドレス範囲を示します。推定値は、使用事例、構成、またはアプリケーションのリソース、および SDK バージョンにより変わる可能性があります。

Bluetooth stack version	Block	Allocated Flash	Address range-EFR32xG1	Address range-EFR32xG12
2.0.0	Bootloader	16 kB	0x00000000-0x00003FFF	N/A
2.01	Stack	111 kB	0x00004000-0x0001F7FF	N/A
2.0.2	PS Store	4 kB	0x0003F000-0x0003FFFF	N/A
2.1.0	Bootloader	16 kB	0x00000000-0x00003FFF	N/A
	Stack	116 kB	0x00004000-0x00020FFF	N/A
	PS Store	4 kB	0x0003F000-0x0003FFFF	N/A
2.3.0	Bootloader	16 kB	0x00000000-0x00003FFF	0x0FE10000-0x0FE13FFF
	Stack	120 kB	0x00004000-0x0001F7FF	0x00000000-0x000FEFFF
	PS Store	4 kB	0x0003F000-0x0003FFFF	0x00100000-0x00100FFF

Bluetooth スタックのバージョン 2.4.0 以降では、フラッシュはプラットフォームにより変わります。

		EFR32MG1X	EFR32MG12X	EFR32MG13X
ブートローダ	サイズ	16	16	16
	アドレス	0 ～ 0x3FFF	0x0FE10000 ～ 0x0FE13FFF	0x0FE10000 ～ 0x0FE13FFF
スタック	サイズ	117	122	123
	アドレス	0x4000 ～ 0x20FFF	0 ～ 0x1E800	0 ～ 0x1EC00
PS ストア	サイズ	4	4	4
	アドレス	0x3F000 ～ 0x3FFFF	0xFF000 ～ 0xFFFFF	0x7F000 ～ 0x7FFFF

7.2 リンク方法

Bluetooth スタックはプリリンク済みのバイナリ・オブジェクトとして提供されます。アプリケーションは、Bluetooth スタックを個別のバイナリ・イメージとしてリンクします。インプレース OTA アップデートの場合、このリンク方法は、Bluetooth スタックがアプリケーションに依存しないことを意味し、これにより Bluetooth スタックはアップデート中も独立して動作できます。Bluetooth スタックは OTA をサポートしていますが、アプリケーションがアップデート・プロセスを開始する必要があります。

OTA アップデートと、アップデートを有効にする方法の詳細については、『*AN1045：EFR32xG1 および BGM11x シリーズ製品の Bluetooth 無線デバイス・ファームウェア・アップデート*』、『*UG266：Silicon Labs Gecko ブートローダ・ユーザ・ガイド*』、および『*AN1086：Silicon Labs Bluetooth アプリケーションでの Gecko ブートローダの使用*』を参照してください。

7.3 RAM

Bluetooth スタックは Wireless Gecko の RAM の一部を予約し、アプリケーションのために未使用の RAM を確保しています。

Bluetooth 機能による RAM の消費は、以下のように分けられます。

- Bluetooth スタック
- Bluetooth 接続プール
- Bluetooth GATT データベース
- C STACK
- C HEAP

以下の表に RAM 使用量の詳細を示します。

Component	Allocated RAM
Bluetooth stack	12 kB
Bluetooth connection pool	2976 + Number of connections * 844 bytes
Bluetooth GATT database	Application-dependent (20 to 200 bytes)
C STACK	1.5 kB
C HEAP	0 kB

7.3.1 Bluetooth スタック

Bluetooth スタックには 12 kB 以上の RAM が必要です。これには、低レベル無線ドライバを組み込んだ Bluetooth スタック・ソフトウェア (binstack.o) とアプリケーション・プログラミング・インターフェイス (bgapi.o) が含まれます。

7.3.2 Bluetooth 接続プール

Bluetooth スタックは、動的メモリ割り当てにスタックの静的メモリ・プールを使用します。割り当てられるメモリ・プールのサイズは、並列接続の数によって異なります。この数は、gecko_init() 関数の .bluetooth.max_connections パラメータで設定します。

*Bluetooth 接続プール・サイズ = 2976 + 接続数 * 844 バイト*

7.3.3 Bluetooth GATT データベース

Bluetooth GATT データベースは RAM を使用します。使用される RAM の容量は、ユーザ定義の GATT データベースにより異なり、一般化できません。書き込み可能なすべての特性は、定義された長さに応じた容量の RAM を使用します。さらに、GATT 内の各属性には、属性の権限を維持するために、数バイトの RAM が必要です。一般的な RAM 使用量は、およそ 20 ～ 200 バイトです。セクション 第 8 章 付録：フラッシュおよび RAM 消費量の計算の例を参照してください。

7.3.4 C STACK

Bluetooth スタックには、CSTACK 用に 1.5 kB 以上の RAM が予約されている必要があります。アプリケーションの開発者は、スタックに必要な 1.5 kB に加え、アプリケーションの CSTACK 用に RAM を割り当てる必要があります。

7.3.5 C HEAP

アプリケーション HEAP も、アプリケーション要件に基づいて予約する必要があります。Bluetooth スタックは、動的メモリ割り当てにスタックの静的メモリ・プールを使用し、C HEAP をまったく使用しません。

7.4 RTCC

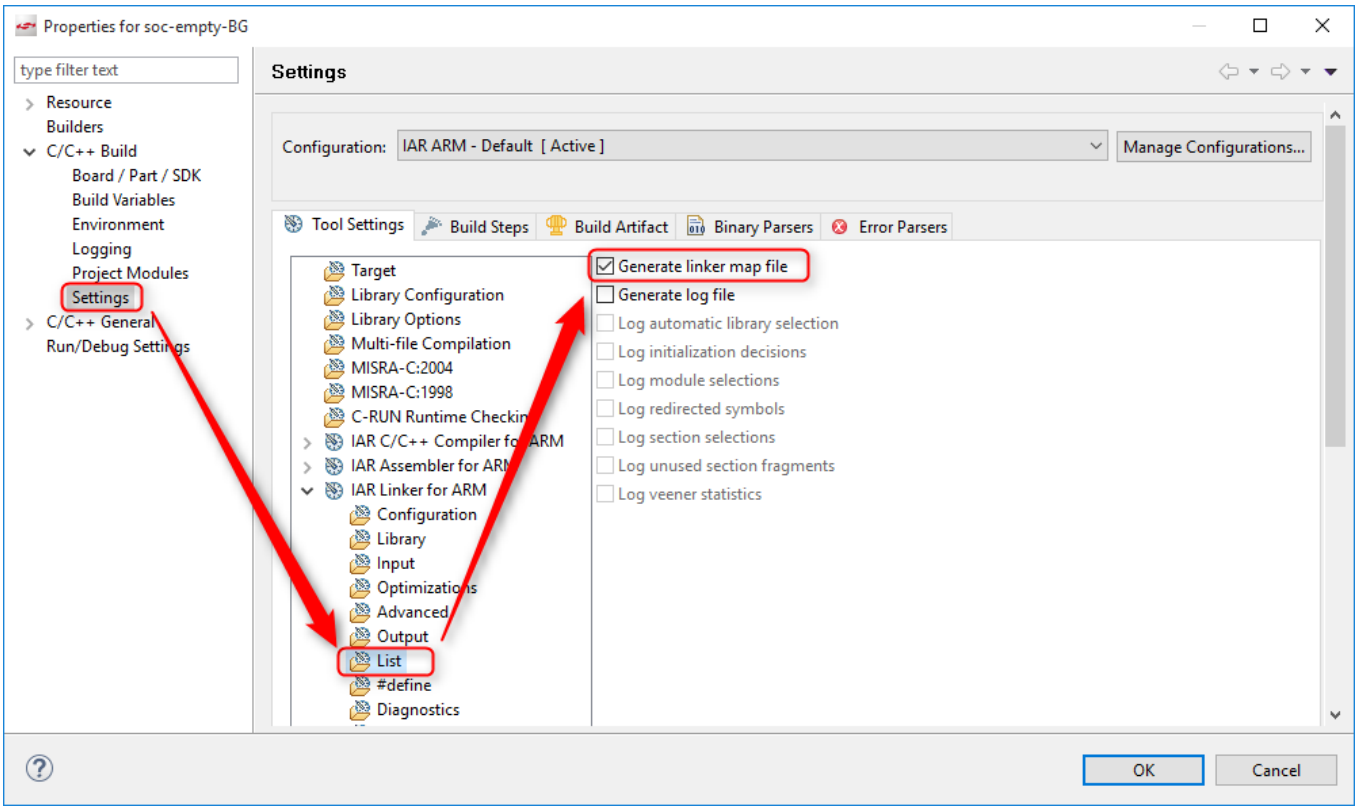
ハードウェア RTCC（リアルタイム・クロックおよびカレンダー）は、Bluetooth スタックによりカウンタ・モードで実行するよう設定され、スタックが使用するため予約されています。ただし、RTC 値はアプリケーションが読み取ることはできますが、アプリケーションが書き込むことはできません。RTC 値は、デバイスが起動するたびにリセットされます。

アプリケーションに RTCC のような機能が必要な場合は、次のようなアプリケーション・コードを作成します。

1. 現在の時刻をスマート･フォンなどの外部デバイスから取得するメカニズムを構築します。一部のスマート･フォンには Bluetooth 時間プロファイルが実装されており、時刻と日付の値を読み取るために使用できます。
2. 時刻を「基準時点からの通算秒」に変換します（たとえば、stdlib の mktime を使用）。
3. Bluetooth スタックの API hardware_get_time() を使用して、リセットからの経過秒を取得します。
4. 「基準時点からの通算秒」とリセットからの経過秒の差を計算し、この差を PS-key などに保存します。
5. 現在のカレンダー時刻を取得するには、hardware_get_time を使用して現在の RTC 値を取得し、この値に PS-key の値を加算し、次に stdlib の localtime を使用して現在のカレンダー時刻を取得します。

第 8 章　付録：フラッシュおよび RAM 消費量の計算

RAM と ROM の正確な消費量を確認するには、リンク後に *.map* ファイルを生成します。次の図に示すように、IAR のプロジェクトの設定ページでこのオプションをオンに切り替える必要があります。



コンパイル後、ワークスペースの *IAR ARM - Default* フォルダに map ファイルが生成されています。

ファイルの最初の配置概要の部分をご覧ください。ここに、各オブジェクトの正確なサイズとメモリ場所が記載されています。

SDK v2.3.0 の例では、レガシ OTA ブートローダ（binbootloader.o）は、アドレスが 0x00000000 で、フラッシュから 0x31c（796）バイトを消費しています。

```
 Section Kind Address Size Object
------- ---- ------- ---- ------
".text_bootloader": 0x4000
 bootloader 0x00000000 0x4000 <Block>
 .binbootloader const 0x00000000 0x31c binbootloader.o [1]
 bootloader const 0x0000031c 0x3ce4 <Block tail>
```

BLE スタック（binstack.o）は、アドレスが 0x00004000 で、0x1c9e4（117220）バイトを消費しています。

```
 Section Kind Address Size Object
 ------- ---- ------- ---- ------
".text_stack": 0x1c9e4
 stack 0x00004000 0x1c9e4 <Block>
 .binstack const 0x00004000 0x1c9e4 binstack.o [1]
 - 0x000209e4 0x1c9e4
```

SDK v.2.3 0 以降では、bgapi はスタックとは別のライブラリで、アプリケーション・スペースでリンクされます。bgapi はいくつかの bgapi クラスにリファクタリングされています。これらのクラスを個別に使用することで、未使用のクラスでメモリが消費されないようにすることができます。bgapi クラスのオブジェクトは、.text_app セクションに記載されています。

```
 Section Kind Address Size Object
 ------- ---- ------- ---- ------
".text_app": 0x4cae
 app 0x00021000 0x4cae <Block>
...
.rodata const 0x00021584 0x28 gecko_bgapi_dfu.c.obj [9]
.rodata const 0x000215ac 0x38 gecko_bgapi_endpoint.c.obj [9]
.rodata const 0x000215e4 0x30 gecko_bgapi_flash.c.obj [9]
.rodata const 0x00021614 0xa0 gecko_bgapi_gatt.c.obj [9]
.rodata const 0x000216b4 0x48 gecko_bgapi_gatt_server.c.obj [9]
.rodata const 0x000216fc 0x78 gecko_bgapi_hardware.c.obj [9]
.rodata const 0x00021774 0x28 gecko_bgapi_le_connection.c.obj [9]
.rodata const 0x0002179c 0x70 gecko_bgapi_gap.c.obj [9]
.rodata const 0x0002180c 0x80 gecko_bgapi_sm.c.obj [9]
.rodata const 0x0002188c 0x38 gecko_bgapi_system.c.obj [9]
.rodata const 0x000218c4 0x30 gecko_bgapi_test.c.obj [9]
.rodata const 0x000218f4 0x18 gecko_bgapi_util.c.obj [9]
.rodata const 0x0002190c 0xc bgapi_sm.c.obj [9]
.rodata const 0x00021918 0xc bgapi_sm.c.obj [9]
.rodata const 0x00021924 0x18 adc.c.obj [9]
.rodata const 0x0002193c 0x54 uart.c.obj [9]
.rodata const 0x00021990 0xc uart.c.obj [9]
.rodata const 0x0002199c 0x28 uart.c.obj [9]
...
.text ro code 0x00023e2c 0x58 gecko_bgapi_le_connection.c.obj [9]
.text ro code 0x00023e84 0x58 gecko_bgapi_gatt.c.obj [9]
.text ro code 0x00023edc 0x58 gecko_bgapi_gatt_server.c.obj [9]
.text ro code 0x00023f34 0x58 gecko_bgapi_endpoint.c.obj [9]
.text ro code 0x00023f8c 0x58 gecko_bgapi_hardware.c.obj [9]
.text ro code 0x00023fe4 0x58 gecko_bgapi_flash.c.obj [9]
.text ro code 0x0002403c 0x58 gecko_bgapi_test.c.obj [9]
.text ro code 0x00024094 0x58 gecko_bgapi_sm.c.obj [9]
.text ro code 0x000240ec 0x58 gecko_bgapi_util.c.obj [9]
.text ro code 0x00024144 0x50 bgapi_dfu.c.obj [9]
.text ro code 0x00024194 0xf6 bgapi_system.c.obj [9]
..
.text ro code 0x0002428c 0x248 bgapi_gap.c.obj [9]
.text ro code 0x000244d4 0xd4 bgapi_le_connection.c.obj [9]
.text ro code 0x000245a8 0x454 bgapi_gatt_lib.c.obj [9]
.text ro code 0x000249fc 0x192 bgapi_endpoint.c.obj [9]
..
.text ro code 0x00024b90 0x184 bgapi_hardware.c.obj [9]
.text ro code 0x00024d14 0x154 bgapi_ps.c.obj [9]
.text ro code 0x00024e68 0x200 bgapi_test_lib.c.obj [9]
.text ro code 0x00025068 0x2b8 bgapi_sm.c.obj [9]
.text ro code 0x00025320 0x14c bgapi_util.c.obj [9]

...etc.
```

CSTACK と HEAP の使用量も確認できます。この例では、CSTACK は 0x20003038 から、HEAP は 0x20003818 から開始され、両方とも 2 K です。

```
 Section Kind Address Size Object
 ------- ---- ------- ---- ------

 CSTACK 0x20003038 0x800 <Block>
 CSTACK uninit 0x20003038 0x800 <Block tail>
 - 0x20003838 0x800

 HEAP 0x20003818 0x800 <Block>
 HEAP uninit 0x20003818 0x800 <Block tail>
 - 0x20004018 0x1000
```

GATT データベースの RAM 使用量を計算するには、.data セクションと .bss セクションの gatt_db.o で専有されているメモリ・スペースを加算します。以下の例で、empty-soc-example からの GATT データベースの RAM 使用量を示します。この例では 20 バイトの RAM を使用しています。

```
Section Kind Address Size Object
------- ---- ------- ---- ------
"P3", part 1 of 3: 0x34
 P3-1 0x20003000 0x34 <Init block>
..
 .data inited 0x20003004 0x10 gatt_db.o [2]
...
"P3", part 3 of 3: 0x1c20
..
 .bss zero 0x20005424 0x4 gatt_db.o [2]
..
```

フラッシュと RAM の全体の使用量を確認するには、各メモリ・セクションのモジュール概要の最初の行で、場所とサイズを調べます。

```
 Section Kind Address Size Object
 ------- ---- ------- ---- ------
".text_app": 0x4cae
 app 0x00021000 0x4cae <Block>
```

各セクションのこれらの値を加算し、フラッシュと RAM の全体の使用量を計算します。

次のセクションはフラッシュを消費します。

- .text_bootloader
- .text_stack
- .text_app

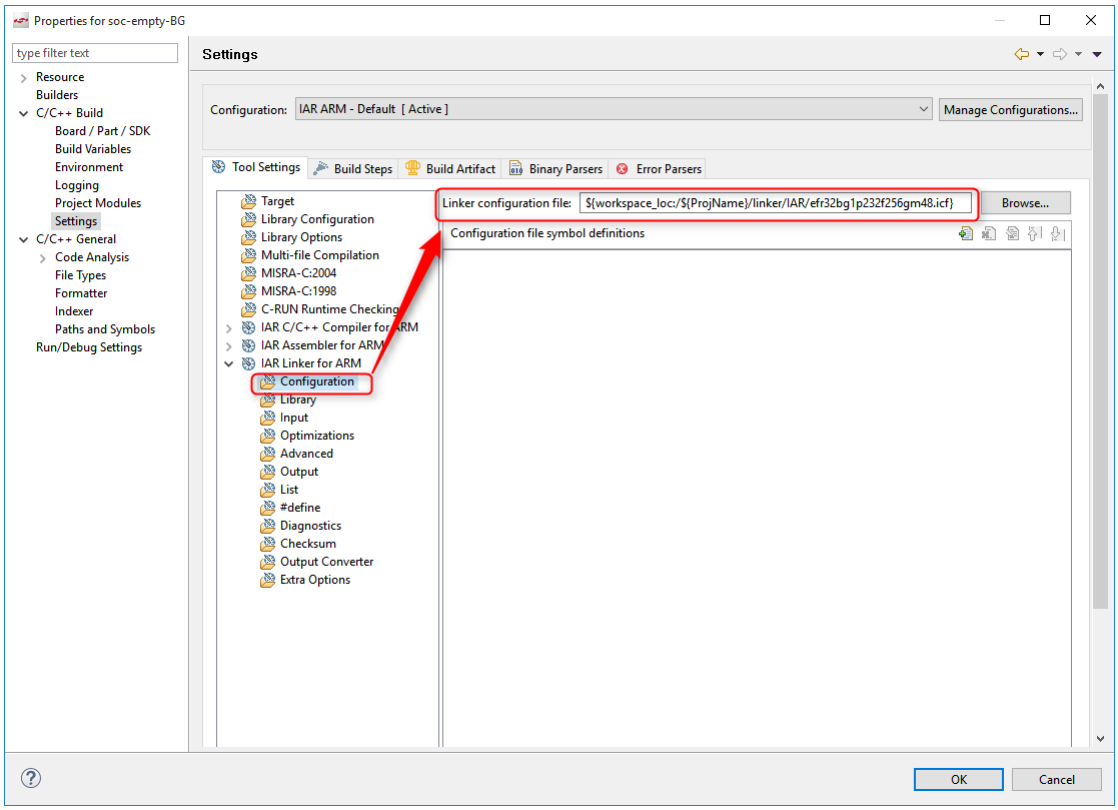
次のセクションは RAM を消費します。

- P3.data
- P3.CSTACK
- P3.HEAP
- P3.bss

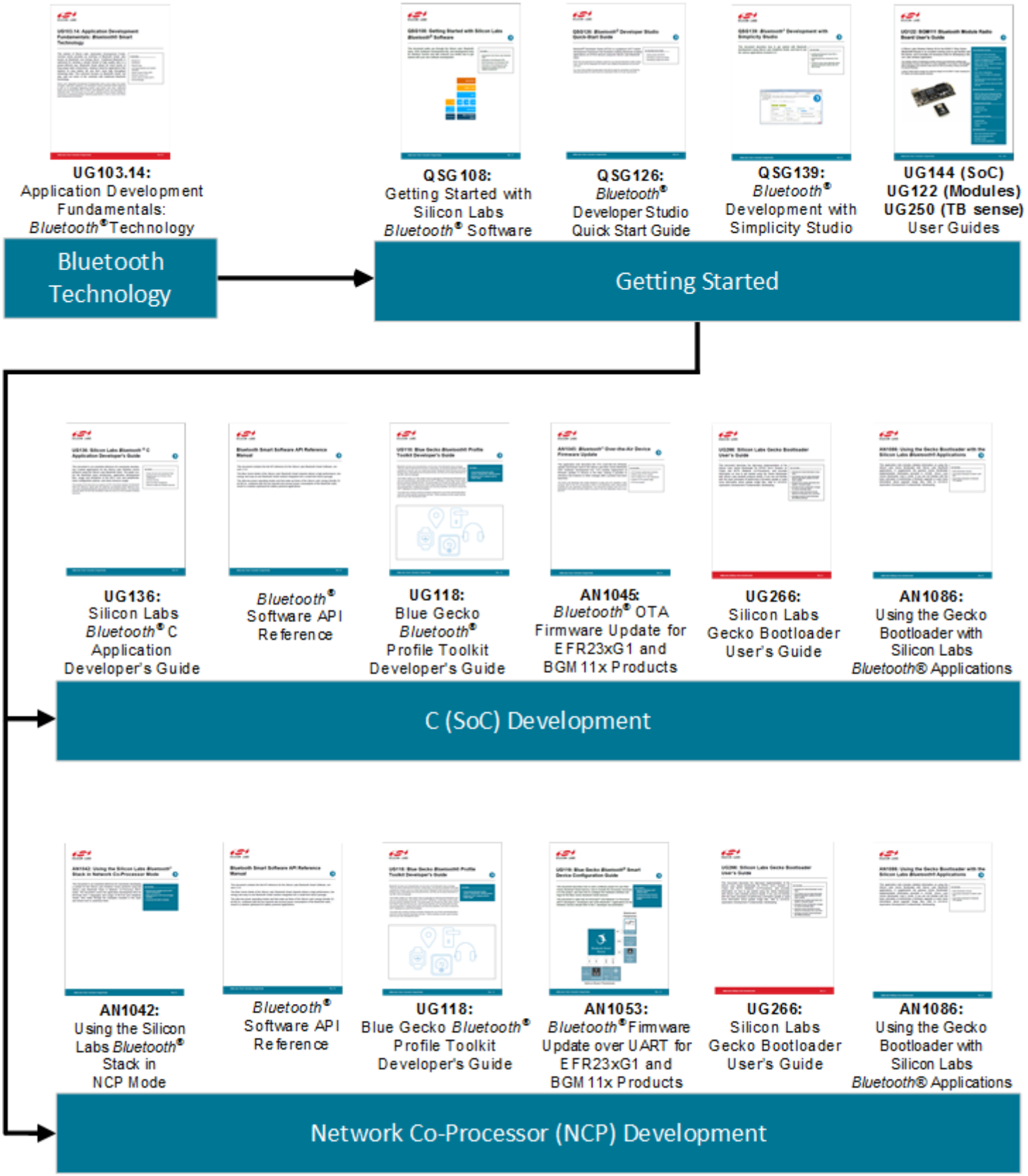
Bluetooth スタック用に予約されている追加の 12 K RAM セクションは *.map* ファイルにはリストされず、IAR リンカ・スクリプト・ファイルにリストされます。

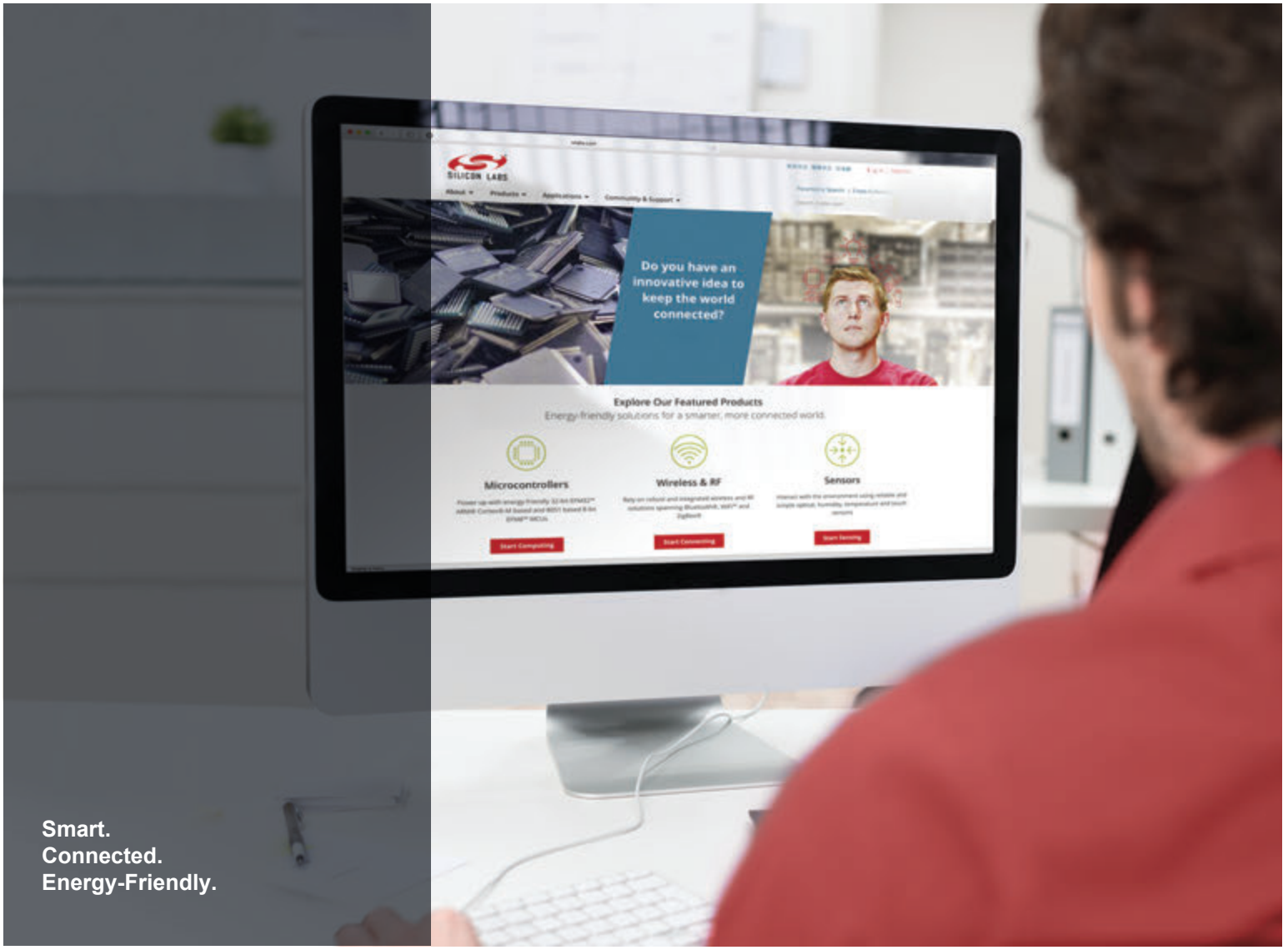
```
/*12KiB ram for stack*/
define symbol blob_ram_size = 0x3000;
define region RAM_region = mem:[from __ICFEDIT_region_RAM_start__+blob_ram_size to __ICFEDIT_region_RAM_end__];
```

このファイルはプロジェクト・ワークスペースにありますが、ターゲットによって異なります。使用されている *.icf* ファイルを見つけるには、プロジェクト・プロパティを確認してください。


Properties for soc-empty-BG
type filter text
Resource
Builders
C/C++ Build
Board / Part / SDK
Build Variables
Environment
Logging
Project Modules
Settings
C/C++ General
Code Analysis
File Types
Formatter
Indexer
Paths and Symbols
Run/Debug Settings
Settings
Configuration: IAR ARM - Default [Active]
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
Target
Library Configuration
Library Options
Multi-file Compilation
MISRA-C:2004
MISRA-C:1998
C-RUN Runtime Checking
IAR C/C++ Compiler for ARM
IAR Assembler for ARM
IAR Linker for ARM
Configuration
Library
Input
Optimizations
Advanced
Output
List
#define
Diagnostics
Checksum
Output Converter
Extra Options
Linker configuration file: ${workspace_loc:/${ProjName}/linker/IAR/efr32bg1p232f256gm48.icf}
Browse...
Configuration file symbol definitions
OK
Cancel

第 9 章　資料


UG103.14:
Application Development Fundamentals: Bluetooth® Technology
QSG108:
Getting Started with Silicon Labs Bluetooth® Software
QSG126:
Bluetooth® Developer Studio Quick Start Guide
QSG139:
Bluetooth® Development with Simplicity Studio
UG144 (SoC)
UG122 (Modules)
UG250 (TB sense)
User Guides
Bluetooth Technology
Getting Started
UG136:
Silicon Labs Bluetooth® C Application Developer's Guide
Bluetooth® Software API Reference
UG118:
Blue Gecko Bluetooth® Profile Toolkit Developer's Guide
AN1045:
Bluetooth® OTA Firmware Update for EFR23xG1 and BGM11x Products
UG266:
Silicon Labs Gecko Bootloader User's Guide
AN1086:
Using the Gecko Bootloader with Silicon Labs Bluetooth® Applications
C (SoC) Development
AN1042:
Using the Silicon Labs Bluetooth® Stack in Network Co-Processor Mode
Bluetooth® Software API Reference
UG118:
Blue Gecko Bluetooth® Profile Toolkit Developer's Guide
AN1053:
Bluetooth® Firmware Update over UART for EFR23xG1 and BGM11x Products
UG266:
Silicon Labs Gecko Bootloader User's Guide
AN1086:
Using the Gecko Bootloader with Silicon Labs Bluetooth® Applications
Network Co-Processor (NCP) Development



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS®

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com