UG136：Silicon Labs *Bluetooth* ® C 应用开发人员指南

本文档是使用 Silicon Labs Bluetooth 协议栈为 Silicon Labs Wireless Gecko 产品开发基于 C 语言的应用的开发人员的必要参考。本指南的内容涵盖 Bluetooth 协议栈体系结构、应用开发流程、MCU 内核和外围设备使用和限制、协议栈配置选项以及协议栈资源使用。

本文档的目的是发现并弥补 Bluetooth 协议栈 API 参考、Gecko SDK API 参考和 Wireless Gecko 参考手册之间在开发 Wireless Gecko 所用 Bluetooth 应用方面的空白。本文档中呈现的细节可帮助开发人员充分利用可用的硬件资源。

内容要点

- 项目结构和开发流程
- Bluetooth 协议栈和 Wireless Gecko 配置
- 中断处理
- 事件和睡眠管理
- 资源使用和可用资源

目录

1. 介绍

本文档是 C 语言开发人员的 Silicon Labs Bluetooth 协议栈指南。

本文档的内容涵盖各种开发角度，是使用 Bluetooth 协议栈为 Wireless Gecko 产品开发基于 C 语言的应用的开发人员的重要参考。

本文档的内容涵盖以下主题：

- 章节 2. 应用开发流程 讨论了应用开发流程和项目结构。
- 章节 4. 配置 Bluetooth 协议栈和 Wireless Gecko 设备 解释了项目，包括库和应用代码中的实际 Wireless Gecko 配置。
- 章节 5. Bluetooth 协议栈事件处理 对使用 Silicon Labs Bluetooth 协议栈进行开发的每位开发人员都很重要，因为它解释了应用如何在一个基于事件的体系结构中与协议栈同步运行。
- 章节 6. 中断 和章节 7. Wireless Gecko 资源 涉及的主题是外围设备和芯片组资源，内容涵盖为协议栈使用预留的资源、正确的中断处理方法，以及协议栈的内存占用和应用的可用内存。

1.1 必备条件

本文档假设 Silicon Labs 的 Bluetooth SDK 2.3.0 或更新版本已被正确安装到开发计算机（Windows、MAC OSX 或 Linux）上，且读者熟悉快速入门指南和 SDK 示例。此外，读者应对 Bluetooth 技术有基本的了解。有关更多信息，请参见 *《UG104.13：应用开发基础知识：Bluetooth 技术》*。

有关安装和工具说明，请参见 *《QSG108：Silicon Labs Bluetooth 软件入门指南》*。有关在 Silicon Labs Simplicity Studio 开发环境中开始使用示例应用的说明，请参见 *《QSG139：使用 Simplicity Studio 进行 Bluetooth 开发》*。

最后，Silicon Labs 提供了 Bluetooth Developer Studio 的插件，有助于快速生成代码 GATT 数据库。请参见 *《QSG126：Bluetooth® Developer Studio 快速入门指南》* 获取关于插件的快速入门指南。

目前支持的编译器和 IDE 版本如下表所示。

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)

2. 应用开发流程

下图描述了高级固件结构。开发人员在协议栈基础之上创建应用，该协议栈是 Silicon Labs 提供的预编译对象文件，可实现终端设备所需的 Bluetooth 连接性。

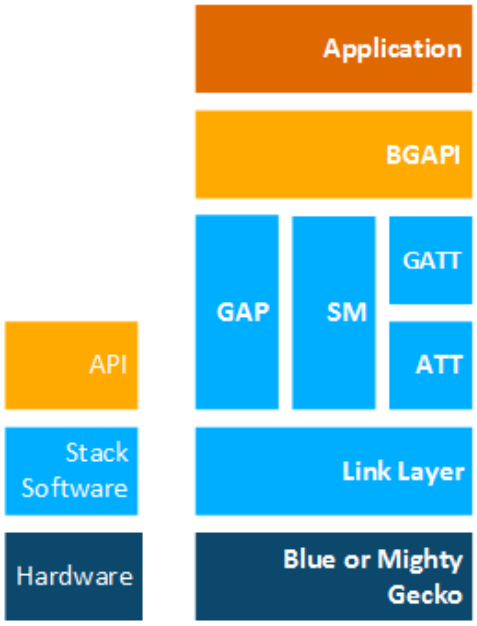


Figure 2.1. Bluetooth 协议栈体系结构方框图

Bluetooth 协议栈包含以下块。

- **引导装载程序**——当前提供三种引导装载程序。请参见*《UG103.06：应用开发基础知识：引导装载》*了解更多信息。
  - 旧版 OTA（无线）固件更新
  - 旧版 UART DFU（设备固件更新）
  - Gecko Bootloader。请参见*《UG266：Gecko Bootloader 用户指南》*和*《AN1086：将 Gecko Bootloader 与 Silicon Labs Bluetooth 应用一起使用》*了解更多信息。
- **Bluetooth 协议栈**——Bluetooth 功能包括链路层、通用访问配置文件、安全管理器、属性协议和通用属性配置文件。
- **OTA 管理程序**——在引导装载程序启动之后启动的应用。它会检查用户应用是否有效，如果检查有效，管理程序便会启动应用。如果应用映像无效，管理程序便会启动配置的引导装载程序以尝试接收有效的应用映像。这需要使用旧版 OTA 引导装载程序或 Gecko Bootloader。

2.1 应用构建流程

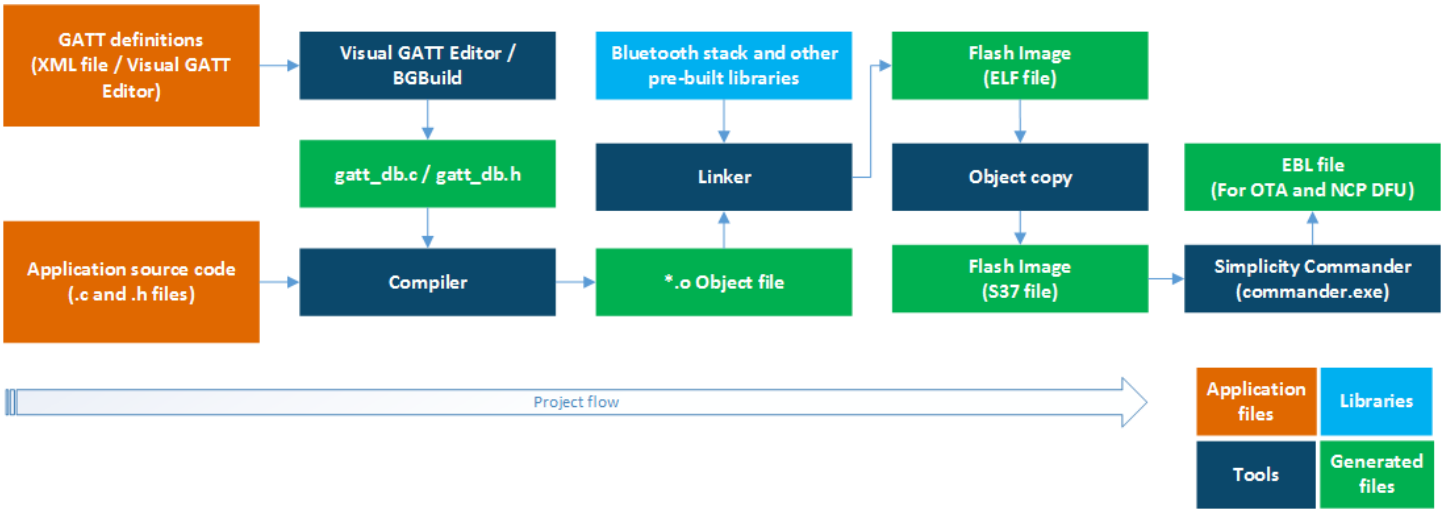


Figure 2.2. Bluetooth 项目构建流程

要构建项目，首先必须定义 Bluetooth 服务和特性（GATT 定义），并通过 Silicon Labs 提供的示例或任何空项目模板编写应用源代码，如《*QSG139：使用 Simplicity Studio 开发 Bluetooth 应用*》中所述。

SDK v2.1.0 和更新版本提供了两种 Bluetooth 服务和特性的定义方式。第一个选项是 Simplicity Studio 中的 Visual GATT Editor GUI。这是一个图形工具，用于设计 GATT 和生成 *gatt_db.c* 与 *gatt_db.h*。此外，它可以导入 *.xml* 和 *.bgproj* GATT 定义文件。Visual GATT Editor 是 Simplicity Studio 项目中的默认 GATT 定义和生成工具。

第二个选项是创建 *.xml* 或 *.bgproj*（根据《*UG118：Blue Gecko Bluetooth® Profile Toolkit 开发人员指南*》），然后将 BGBuild 可执行文件用作预编译步骤，以便将 GATT 定义文件转换为 .c 和 .h。IAR Embedded Workbench 项目使用的就是这种方法。

编译项目会生成一个对象文件，该文件随后会链接至 SDK 中提供的预编译库。该链接的输出是一个闪存映像，可以编程到支持的 Wireless Gecko 设备中。

3. 项目结构

本节解释了应用项目结构以及必须被包含在项目中的必要和可选资源。

3.1 Bluetooth 协议栈和 Bootloader 库

Bluetooth 协议栈库是：

- **binbootloader.o**: Bootloader 映像（预编译旧版 OTA 引导装载程序）。仅受 EFR32xG1 设备支持的旧版引导装载程序需要该文件。自 v2.3.0 SDK 起，可以使用支持所有 EFR32 设备的统一 Gecko Bootloader。请参见*《UG266：Silicon Labs Gecko Bootloader 用户指南》*了解更多详情。
- **binstack.o**：Bluetooth 协议栈、管理程序以及 Bluetooth 协议栈所用库的二进制映像。
- **stack.a**：与应用共享的 Bluetooth 协议栈、EMLIB 和 EMDRV 的导出符号。
- **bgapi.a**：执行非核心功能的 BGAPI 命令。自 SDK v2.3.0 起，*bgapi.a* 必须被包含在所有项目中。对于较早的 SDK，它只需被包含在 NCP 应用中。

3.2 GATT 数据库

GATT（通用属性配置文件）数据库是一个描述 Bluetooth 设备的 Bluetooth 配置文件、服务和特性的标准化方法。借助 Silicon Labs Bluetooth 协议栈，GATT 定义可以直接在 Simplicity Studio 中的 Visual GATT Editor GUI 中编辑或以 XML 格式编写，并作为预构建任务传递至 BGBuild 可执行文件。有关如何创建 GATT 数据库和语法的更多信息，请参阅*《UG118：Blue Gecko Bluetooth® Smart Profile Toolkit 开发人员指南》*。

3.2.1 gatt_db.h

gatt_db.h 头文件表示的是由 BGBuild.exe 或 Visual GATT Editor 自动生成的 GATT 数据库结构。从 gatt_db_def.h 至 gatt_db.h，GATT 的类型定义被自动包含在内。

3.3 EMLIB 和 EMDRV 外围设备源代码

EMLIB 是一个低级外围设备支持库，为 Silicon Laboratories 中的所有 EFM32、EZR32 和 EFR32 MCU 及 SoC 提供一个统一 API。EMDRV 是适用于 EFR32 片上外围设备的一组功能特定的高性能驱动程序。这些驱动程序通常基于 DMA，且使用所有可用的低能耗功能。该 API 为大多数驱动程序同时提供同步和异步函数。

开发人员可以选择将哪些外围设备驱动程序包含在项目中。必要的 EMDRV 源代码文件可以被包含在项目中，但要记得不要使用为协议栈预留的资源，也不要为无线电中断分配最高优先级。章节 6. 中断 和章节 7. Wireless Gecko 资源 更详细地讨论了中断和资源限制。

有关 EMLIB 和 EMDRV 的更多详情，请参阅 Gecko SDK API 文档和各种应用说明。

3.4 aat.h

AAT（应用地址表）是固件更新文件格式 EBL 的支持结构（元数据），用于定义程序在升级映像中的大小和位置。它是应用映像中的第一个项目，且有指向应用矢量表的指针。协议栈管理程序将会初始化应用矢量表，并在调用应用之前，先从矢量表中调用协议栈位置。

ATT 虽然并不定义应用在 GBL 升级映像中的大小和位置，但是仍然必不可少。因此，AAT 是所有项目的必要部分，且应用必须包含 aat.h 头文件。

3.5 application_properties.c

该文件包含应用属性结构体，其中包含有关应用映像的信息，如类型、版本和安全性。该结构体在 Gecko Bootloader API 中的 application_properties.h 中进行定义（请参见 <Simplicity Studio Gecko SDK>\platform\bootloader\documentation\Gecko_Bootloader_API_Reference\index.html 中的 Gecko Bootloader API 参考）。Simplicity Studio 项目中包含一个预生成文件，可对其进行修改以包含应用特定属性。可使用 Gecko Bootloader API 访问这些应用属性。通过更改 define 语句，可以更新以下成员：

```
// 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
```

如果使用 OTA 管理程序，则应用属性的 app.capabilities 结构体成员必须包含 Bluetooth 协议栈版本号：

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

如果使用 Gecko Bootloader，则应用属性结构体是必要的，且可驻留在闪存中的任何位置。

3.6 native_gecko.h

该文件有两个用途：第一，它包含实际的 Bluetooth 协议栈 API 以及该协议栈的命令和事件；第二，它向 Bluetooth 协议栈提供配置、事件和睡眠管理 API。

配置、事件和睡眠管理 API 在下文中有介绍。

void gecko_init(const gecko_configuration_t*config)

此函数需要单一参数，即指向 gecko_configuration_t 结构体的指针。它的目的在于使用结构体中提供的参数配置和初始化 Bluetooth 协议栈。下一节更详细地讨论了配置选项和 gecko_init() 使用方法。应用必须调用 gecko_init() 以初始化 Bluetooth 协议栈。

自 SDK v2.3.0 起，此函数已经过时，但仍可使用以维持向后兼容性。

void gecko_stack_init(const gecko_configuration_t*config)

此函数需要单一参数，即指向 gecko_configuration_t 结构体的指针。它的目的在于使用结构体中提供的参数配置和初始化 Bluetooth 协议栈。一旦调用函数 gecko_stack_init()，就必须单独初始化使用的各个协议栈组件。这种单独初始化不包含那些不必要的协议栈组件，从而可实现内存优化。

可以使用以下 API 来单独初始化协议栈组件：

- 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)

这是一个阻塞函数，等待 Bluetooth 协议栈发出事件并进行阻塞，直至收到事件。一旦收到事件，将返回指向 gecko_cmd_packet 结构体的指针。

如果已在 Bluetooth 协议栈配置中启用 EM 睡眠模式，当未收到 Bluetooth 协议栈发出的任何事件时，设备将自动进入 EM1 或 EM2 模式。要确保设备只要有可能就进入功耗最低的睡眠模式，最简单的方法就是使用 gecko_wait_event()。

章节 5. Bluetooth 协议栈事件处理 详细讨论了 Bluetooth 协议栈的事件处理。

struct gecko_cmd_packet* gecko_peek_event(void)

这是一个非阻塞函数，用于请求 Bluetooth 协议栈发出 Bluetooth 事件。当请求了事件且事件队列不为空时，将返回指向 gecko_cmd_packet 结构体的指针。如果事件队列中没有任何事件，则返回 NULL。

当使用此非阻塞事件侦听器时，必须由应用代码管理 EM 睡眠模式，因为 Bluetooth 协议栈不会自动管理它们。睡眠模式管理是使用 gecko_can_sleep_ms() 和 gecko_sleep_for_ms() 函数实现的，稍后会有对这两个函数的讨论。

章节 5. Bluetooth 协议栈事件处理 详细讨论了协议栈的事件处理。

int gecko_event_pending(void)

此函数可检查事件队列中是否有任何待处理 Bluetooth 协议栈事件。如果发现待处理 Bluetooth 事件，此函数会返回一个非零值，表明事件应由 gecko_peek_event() 或 gecko_wait_event() 处理。如果未发现任何事件，则返回零。

uint32 gecko_can_sleep_ms(void)

此函数用于确定 Bluetooth 协议栈可以进入睡眠模式的时长。返回值是协议栈可在下一次 Bluetooth 操作必须出现之前进入睡眠模式的毫秒数。如果无法进入睡眠模式，则返回零。此函数只能与非阻塞 gecko_peek_event() 事件处理一起使用。

uint32 gecko_sleep_for_ms(uint32 max)

此函数用于使协议栈进入 EM 睡眠模式，持续时间为此函数的单一参数中设置的最大毫秒数。返回值为实际处于睡眠模式的毫秒数。协议栈可能会因为外部事件而被唤醒。此函数只能与非阻塞 gecko_peek_event() 事件处理一起使用。

3.7 InitDevice.h

该头文件包含设备初始化函数，协议栈初始化除外。例如，初始化函数不仅可以初始化时钟和电源管理，而且可以初始化外围设备硬件接口，如 USART、I2C、ADC 和 GPIO。

章节 4. 配置 Bluetooth 协议栈和 Wireless Gecko 设备 **更详细地介绍了 Bluetooth 协议栈和外围设备初始化。**

3.8 无线电板特定内容

Bluetooth SDK 中提供的某些示例使用 WSTK 开发套件的特定功能，如 LCD 显示屏或相对湿度和温度传感器。已经提供用于使用这些功能的简化函数，以便使用入门套件快速开发原型设计。以下头文件可简化特定无线电板上的功能配置。

bspconfig.h

BSP（板支持包）头包括无线电板特定配置，这些配置被用作 WSTK 特定函数的参数，如在 WSTK 上切换 IO，或在入门套件上驱动 LCD 显示屏。

flashpwr.h

该头文件包含用于将某些无线电板（例如，BRD4100A）上的 SPI 闪存芯片配置为低功耗模式的函数。例如，这在测量睡眠电流时很有用，因为若 SPI 闪存未处于低功耗模式，则达不到最小的 EM2、EM3 或 EM4 电流。

pti.h / boards_pti.h

PTI（数据包追踪接口）是 Wireless Gecko SoC 中的内置块，用于将传入和传出无线电数据包作为元数据路由至调试接口。Simplicity Studio 的 Network Analyzer 随后会捕捉并显示这些数据包。Network Analyzer 配备一个适用于 Bluetooth 数据包的解码器，可用于调试、分析和测量 Bluetooth 网络。

需要在代码中初始化 PTI 以启用 HW 块，并且需要将 PTI 引脚路由至对应的无线电板引脚。

4. 配置 Bluetooth 协议栈和 Wireless Gecko 设备

要在 Wireless Gecko 上运行 Bluetooth 协议栈和应用，必须正确配置 MCU 及其外围设备。一旦硬件完成初始化，还必须使用 gecko_init() 函数对协议栈进行初始化。

4.1 Wireless Gecko MCU 和外围设备配置

4.1.1 enter_DefaultMode_from_RESET()

enter_DefaultMode_from_RESET 函数用于基于板设计初始化 MCU 内核和外围设备设置。必须在 main() 刚开始时调用此函数。

在为 Silicon Labs 的无线电板开发应用时，这些设置在 SDK 中提供的示例中进行了正确设置，但为自定义硬件设计创建应用的开发人员需要相应地配置这些设置。Simplicity Studio 中的 Hardware Configurator 会启用一个图形用户界面以配置外围设备，且会在保存 configurator 更改后，自动生成配置代码。后面几节讨论了生成的配置代码。

enter_DefaultMode_from_RESET 会调用专为配置不同 MCU 和外围设备块设计的一组函数。

4.1.2 Bluetooth 时钟

时钟设置在 CMU_enter_DefaultMode_from_RESET 函数中完成初始化。时钟设置包括使用的时钟源（HFXO、LFXO、PLFRCO 和 LFRCO）、时钟参数（如调谐）、外围设备的时钟源等。

HFCLK

HFCLK 用于无线电协议定时器 (PROTIMER)。HFCLK 是高频时钟，精度必须至少达到 ±50 ppm，需要一个外部晶体才能足够精准 (HFXO)。

HFXO 初始化会配置外部晶体以实现时间关键型连接和睡眠管理。HFXO 必须被设置为高频时钟 (HFCLK)，且必须与 Wireless Gecko 的 HFXO 输入引脚进行物理连接。

LFCLK

要让设备停止 HFCLK 并进入睡眠模式，需要另一个时钟，即 LFCLK 时钟。

当设备进入睡眠模式时，会保存 PROTIMER 的当前状态。当设备被唤醒时，它会计算睡眠时钟发出滴答声的次数，并相应地调整 PROTIMER。至于无线电，PROTIMER 似乎持续不断地发出滴答声。

该时钟的精度取决于设备的运行模式。当广播或扫描时，精度并不是那么重要，但当连接打开时，精度必须至少达到 ±500 ppm。此时钟可由 LFXO、PLFRCO 或 LFRCO 驱动，具体取决于精度要求。

默认配置是，LFXO 连接到 Wireless Gecko，并被设置为低频时钟 (LFCLK)。如果设计不支持连接 LFXO 或无法使用 PLFRCO，则必须明确禁用应用的睡眠模式，且 LFRCO 必须被设置为用作时钟源。如章节 4.2.3 睡眠模式 中所述，如果 LFCLK 不够精确，则必须禁用 Bluetooth 协议栈的睡眠模式以便正确运行。

HFXO CTUNE

在示例中，晶体调谐 (CTUNE) 设置是默认值，以与所有 Silicon Labs 的 Bluetooth 模块、参考设计和无线电板搭配使用。不过，在某些情况下，最终产品设计需要特定的晶体校准，要么在每个设备上进行，要么在每个设计上进行。CTUNE 值 uint_16 ctuneStedyState 可根据 CMU_HFXOInit_TypeDef hfxoInit 结构的内部设计进行调整：

```
// $[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);
```

有关配置 HFXO 和 LFXO 的更多信息，请参阅《EFR32 参考手册》第 12 章。

注意：Bluetooth 协议栈仅支持 38.4 MHz HFXO 频率；不支持任何其他 HFXO 频率。

4.1.3 DC-DC 配置

DCDC 配置在 EMU_enter_DefaultMode_from_RESET 函数中设置。在 SDK 中的示例中，DC-DC 配置被设置为与 Silicon Labs 的 Bluetooth 模块、无线电板和参考设计搭配使用，但自定义设计可能需要特定的 DC-DC 设置。

```
// $[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);
```

有关配置 DCDC 的更多信息，请参阅《EFR32 参考手册》第 11 章和*《AN0948：电源配置和 DC-DC》*。

4.2 使用 gecko_stack_init() 配置 Bluetooth 协议栈

gecko_stack_init() 函数用于配置 Bluetooth 协议栈，包括睡眠模式配置、分配给连接的内存、OTA 配置等。没有一个 Bluetooth 协议栈函数可在 Bluetooth 协议栈配置好之前使用。

Bluetooth 协议栈配置示例：

```
/* 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() 函数中的配置选项是：睡眠启用/禁用、Bluetooth 连接计数、堆大小、GATT 数据库、OTA 配置和 PTI。时钟等初始化的其余部分都在 enter_DefaultMode_from_RESET() 函数中配置，该函数基于部分或整套选项在 Hardware Configurator 中自动生成。

一旦调用函数 gecko_stack_init()，就必须单独初始化使用的各个协议栈组件。这种单独初始化不包含不必要的协议栈组件，从而可实现内存优化。

可以使用以下 API 来单独初始化协议栈组件：

gecko_bgapi_class_dfu_init()	启用设备固件升级 (dfu) API。
gecko_bgapi_class_system_init()	启用本地设备（系统）API。
gecko_bgapi_class_le_gap_init()	启用通用访问配置文件 (gap) API。
gecko_bgapi_class_le_connection_init()	允许通过连接 API 管理连接建立、参数设置和断开程序。
gecko_bgapi_class_gatt_init()	有助于通过 gatt API 浏览和管理远程 GATT 服务器中的属性。
gecko_bgapi_class_gatt_server_init()	有助于通过 gatt_server API 浏览和管理本地 GATT 数据库中的属性。
gecko_bgapi_class_endpoint_init()	允许通过端点 API 创建和删除端点以及配置数据路由。
gecko_bgapi_class_hardware_init()	支持通过硬件 API 访问和配置系统硬件和外围设备。
gecko_bgapi_class_flash_init()	启用可用于管理闪存中的用户数据的永久性存储命令（闪存）API。
gecko_bgapi_class_test_init()	启用 DTM 测试 API。
gecko_bgapi_class_sm_init()	启用安全管理器 (sm) API。
gecko_bgapi_class_util_init()	启用效用函数 API，如 *atoi* 和 *itoa*。

4.2.1 CONFIG_FLAGS

当前标志：

GECKO_CONFIG_FLAG_USE_LAST_CTUNE	1=将 CTUNE 重写为 PS 存储中最后保存的值。
GECKO_CONFIG_FLAG_SUPERVISOR	0=正常应用。需要将 OTA 设置保存至闪存。 1=应用是 OTA 管理程序，且使用保存的 OTA 设置。
GECKO_CONFIG_FLAG_RTOS	1=应用使用 RTOS。协议栈并不配置时钟、矢量、TEMPDRV 或睡眠，因为它们是由 RTOS 提供的。

4.2.2 Mbedtls

它可配置协议栈使用的加密库。Mbedtls 提供 1 类（sl_*.c）和 2 类（slcl_*.c）插件，用于实现加密操作的硬件加速。这两类插件互不兼容。Bluetooth 协议栈使用 2 类插件，因此应用中使用的加密功能也必须使用 2 类插件。

在进行加密操作时，必须初始化加密的硬件加速。为此，需要为使用的加密设备调用 mbedtls_device_init() 和 mbdedtls_device_set_instance()。设备上下文的初始化应只进行一次。Bluetooth 协议栈会自动为其使用的加密设备这样做，除非 GECKO_MBEDTLS_FLAGS_NO_MBEDTLS_DEVICE_INIT 标志已在配置选项 **.mbedtls.flags** 中进行设置，这会禁用加密设备的自动初始化。如果应用处理加密设备初始化，则需要设置此标志。

配置选项 **.mbedtls.dev_number** 用于定义 Bluetooth 协议栈在有多个设备可用时使用哪个加密设备；默认值是 0。

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

4.2.3 睡眠模式

必须在 gecko_init() 函数中启用 Wireless Gecko 的睡眠模式 EM2（能源模式 2）。睡眠标志是 gecko_configuration_t 结构体的一部分。必须设置 SLEEP_FLAGS_DEEP_SLEEP_ENABLED 标志以启用睡眠模式。如果出现阻塞事件，协议栈会自动处理睡眠模式，如章节 5. Bluetooth 协议栈事件处理 中所述。

在 gecko_configuration_t 结构体（main.c）中启用睡眠模式的示例：

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

睡眠模式要求硬件中存在精准的 32 kHz 低频时钟（LFCLK）。如果 Bluetooth 协议栈没有精准的睡眠时钟可用，则无法进入低功耗睡眠模式。对于不需要低功耗睡眠模式的应用，可以忽视 LFXO，但必须如下所示设置 Gecko 配置结构体中的睡眠标志：

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

4.2.4 Bluetooth 连接

协议栈允许的最大同步 Bluetooth 连接数受到分配给连接管理的内存大小限制。内存是在 gecko_init() 中的初始化期间分配的，并由定义函数 #define MAX_CONNECTIONS 确定。每个连接会将 RAM 使用量增大约 800 字节。

将 Bluetooth 连接限制为一（1）个的示例。

```
#define MAX_CONNECTIONS 1
```

4.2.5 OTA 配置

Bluetooth 无线（OTA）固件升级可用，因为固件升级有一部分是由 OTA 管理程序应用处理的。

OTA 有几个通过 OTA 标志配置的配置选项。

Note: 当使用 Gecko Bootloader 时，为确保安全性，请勿使用 OTA。应该通过 Gecko Bootloader 配置安全性。

否则，**OTA 标志**可被用于控制 OTA 服务中使用的 OTA Control Point 和 OTA Data 特性安全性属性。这会控制执行 OTA 固件更新时所需的 Bluetooth 安全级别。强烈建议确保 OTA 标志的安全性，从而阻止未经授权的设备将新的固件映像上传至设备。

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

可以通过 Gecko 配置结构体配置 Wireless Gecko 的设备名称（当它处于管理程序的 OTA 模式中时）和设备名称长度。

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

最后，应确保将设备设置为 OTA DFU 模式，以便仅可信设备才具备该能力。

有关 OTA 固件更新的更多详情，请参阅*《AN1045：适用于 EFR32xG1 和 BGM11x 系列产品的 Bluetooth 无线设备固件更新》*和*《UG266：Silicon Labs Gecko Booloader 用户指南》*。

4.2.6 PTI

gecko_init() 中的 PTI 结构体用于设置数据包追踪输出的接口配置，包括接口时钟速度、DCLK 引脚、DFRAMEDOUT 和模式。

每个无线电板的 boards_pti.h 为该无线电板设置了正确的配置。对于自定义设计，必须修改结构体以契合板布局。

4.2.7 Bluetooth 5 广播组

必须定义最大广播组数。这些广播组可用于启动多个使用 bt5_set_mode 命令的播发器。每个上下文分配约 60 字节的 RAM。

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

5. Bluetooth 协议栈事件处理

Wireless Gecko 所用的 Bluetooth 协议栈是一个事件驱动型体系结构，其中事件在主 while 循环中得到处理。

5.1 阻塞事件侦听器

gecko_wait_event() 是阻塞等待函数的实现，该函数会等待事件出现在事件队列中，并将它们返回至事件处理程序。这是一种推荐的 Bluetooth 协议栈运行模式，因为它可以最高效且自动地管理睡眠，同时让设备和连接保持同步。

- gecko_wait_event() 函数会处理内部消息队列，直至收到事件。
- 如果没有任何待处理事件或待处理消息，设备会进入 EM1 或 EM2 睡眠模式。
- 该函数会返回指向保留接收的事件的 gecko_cmd_packet 结构的指针。

下面的代码片段显示了使用 gecko_wait_event() 的 iBeacon 示例中的简单主 while 循环，该循环会在启动后设置广播。

```
/* 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 非阻塞事件侦听器

该运行模式需要更多手动调整，例如，睡眠管理需要由应用完成。在某些使用案例中，非阻塞运行是必需的。

- gecko_peek_event() 函数会处理内部消息队列，直至收到事件或所有消息都得到处理。
- 该函数会返回指向保留接收的事件的 gecko_cmd_packet 结构的指针，如果队列中没有任何事件，则返回 NULL。

5.2.1 睡眠和非阻塞事件侦听器

当应用使用非阻塞 gecko_peek_event() 函数创建事件处理程序时，睡眠实现也会有所不同。应用必须使用 gecko_can_sleep_ms() 询问协议栈设备可以进入睡眠模式多久，然后使用 gecko_sleep_for_ms() 函数将睡眠时长设为该值。在调用 gecko_can_sleep_ms() 或 gecko_sleep_for_ms() 函数前，必须禁用中断，且一旦执行了函数，必须立即启用中断。

下面的示例显示了如何在使用非阻塞事件处理时实现睡眠管理。

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

}
```

6. 中断

中断会在其各自的中断处理程序中创建事件，无论是无线电中断还是 IO 引脚中断。这些事件稍后会在消息队列的主事件循环中得到处理。应用应始终最大限度缩短中断处理程序中的处理时间，并暂停处理以实现事件回调，或将处理交给主循环。

一般而言，中断机制根据的是任何基于事件的编程体系结构，但 Bluetooth 协议栈存在少数独特而重要的例外：

- 无法从中断上下文中调用 BGAPI 命令。
- 从中断上下文中仅可调用 gecko_external_signal() 函数。
- 在调用 gecko_sleep_for_ms(...) 之前，必须禁用中断，如前面的代码示例中所示。

6.1 外部事件

外部事件用于捕捉所有外围设备中断，将其作为要传递至主事件循环并在该循环中得到处理的外部信号。外部事件中断可来自任何外围设备中断源，例如 IO、比较器或 ADC 等。信号位数组用于通知事件处理程序已发出了哪些外部中断。

- 外部信号的主要目的在于触发中断上下文向主事件循环发出事件。
- BGAPI 事件 system_external_signal 可通过调用 void gecko_external_signal(uint32 signals) 函数生成。
- 函数 gecko_external_signal 可从中断上下文中调用。
- gecko_external_signal 函数的 signals 参数被传递至 system_external_signal 事件。

```
/**
* 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 优先级

强烈建议无线电应具有最高优先级中断。这是默认配置，其他中断滞后处理。

下表描述了 Bluetooth 协议栈的三种运行在不同操作上下文中的不同组件，以及它们为确保各个组件的连接而禁用中断的最大时长。

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.

7. Wireless Gecko 资源

Bluetooth 协议栈使用的某些 Wireless Gecko 资源不可用于应用。下表列出了各个资源并描述了协议栈对它们的使用。前四个资源（红色）总是被 Bluetooth 协议栈使用。

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.

* 在 SDK 2.3 中，Hardware Configurator 会不必要地预留 PRS 0、1、2、3 和 11 通道。这些 PRS 通道并不会被协议栈使用，但无法在 GUI 中配置。

7.1 闪存

Bluetooth 协议栈从闪存中执行，当前（v.2.0.0 软件或更高版本）需要占用约一半闪存。闪存可分成供引导装载程序、Bluetooth 协议栈、GATT 数据库以及永久性存储（PS 存储）使用的多个块，如下图所示。

- 要启用 Bluetooth 协议栈和应用可升级性，引导装载程序必不可少。引导装载程序的设计不会过时，支持引导装载程序改进和功能添加。
- Bluetooth 协议栈块包括实际的 Bluetooth 固件，其中包括链路层、GAP、SM、ATT 和 GATT 层，还有用于提供 OTA 可升级性的小管理程序应用以及 Bluetooth 协议栈要求的硬件特定库。
- GATT 数据库包含应用使用的 Bluetooth 服务和特性，且它的大小取决于应用以及包含的 GATT 服务和特性数。
- PS 存储是一种非易失数据存储，其中 Bluetooth 协议栈和应用均可存储永久数据，如 Bluetooth 绑定密钥、应用配置数据、硬件配置等。PS 存储位于闪存中的最后 4 kB。

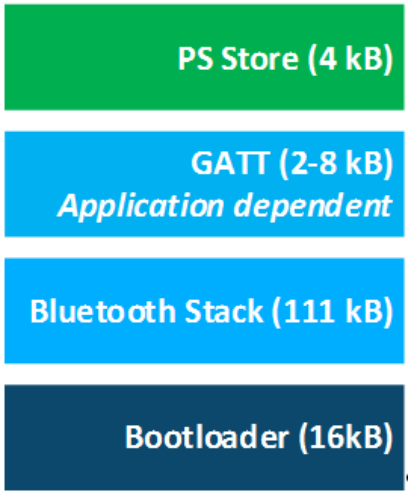


Figure 7.1. 闪存使用量

下表显示了各个块的闪存使用量和地址范围。这些估计值可能会因使用案例、配置或应用资源及 SDK 版本而异。

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

自 Bluetooth 协议栈版本 2.4.0 起，闪存因平台而异。

		EFR32MG1X	EFR32MG12X	EFR32MG13X
引导装载程序	大小	16	16	16
	地址	0-0x3FFF	0x0FE10000-0x0FE13FFF	0x0FE10000-0x0FE13FFF
协议栈	大小	117	122	123
	地址	0x4000-0x20FFF	0-0x1E800	0-0x1EC00
PS 存储	大小	4	4	4
	地址	0x3F000-0x3FFFF	0xFF000-0xFFFFF	0x7F000-0x7FFFF

7.2 链接

Bluetooth 协议栈作为预链接二进制对象交付。应用作为单独的二进制映像链接至 Bluetooth 协议栈。对于就地 OTA 更新，这种链接方法意味着 Bluetooth 协议栈完全独立于应用，这让 Bluetooth 协议栈可在更新期间独立运行。Bluetooth 协议栈为 OTA 提供支持，但应用需要启动更新进程。

有关 OTA 更新及其启用方法的更多信息，请参阅*《AN1045：适用于 EFR32xG1 和 BGM11x 系列产品的 Bluetooth 无线设备固件更新》*、*《UG266：Silicon Labs Gecko 用户指南》*和*《AN1086：将 Gecko Bootloader 与 Silicon Labs Bluetooth 应用一起使用》*。

7.3 RAM

Bluetooth 协议栈会预留 Wireless Gecko 的部分 RAM，并将未使用的 RAM 留给应用。

Bluetooth 功能的 RAM 消耗量被分为：

- Bluetooth 协议栈
- Bluetooth 连接池
- Bluetooth GATT 数据库
- C STACK
- C HEAP

下表显示了 RAM 使用量详情。

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 协议栈

Bluetooth 协议栈需要至少 12 kB RAM。它包括配备低级无线电驱动程序 (binstack.o) 和应用编程接口 (bgapi.o) 的 Bluetooth 协议栈软件。

7.3.2 Bluetooth 连接池

Bluetooth 协议栈将它自己的静态内存池用于动态内存分配。分配的内存池大小取决于并行连接数。该数使用 gecko_init() 函数中的 .bluetooth.max_connections 参数设置。

*Bluetooth 连接池大小 = 2,976 + 连接数 * 844 字节*

7.3.3 Bluetooth GATT 数据库

Bluetooth GATT 数据库使用 RAM。RAM 使用量取决于用户定义的 GATT 数据库，不能泛化。启用写入功能的所有特性与其定义的长度使用同样多的 RAM。此外，GATT 中的每个属性都需要几个字节的 RAM 来维持属性许可。典型的 RAM 使用量是 20 至 200 字节左右。请参见章节 8. 附录：计算闪存和 RAM 消耗量 中的示例。

7.3.4 C STACK

Bluetooth 协议栈需要至少预留 RAM 中的 1.5 kB CSTACK。应用开发人员在预留协议栈需要的 1.5 kB 之外，还必须为应用 CSTACK 分配 RAM。

7.3.5 C HEAP

还必须根据应用要求预留应用 HEAP。Bluetooth 协议栈将它自己的静态内存池用于动态内存分配，完全不使用 C HEAP。

7.4 RTCC

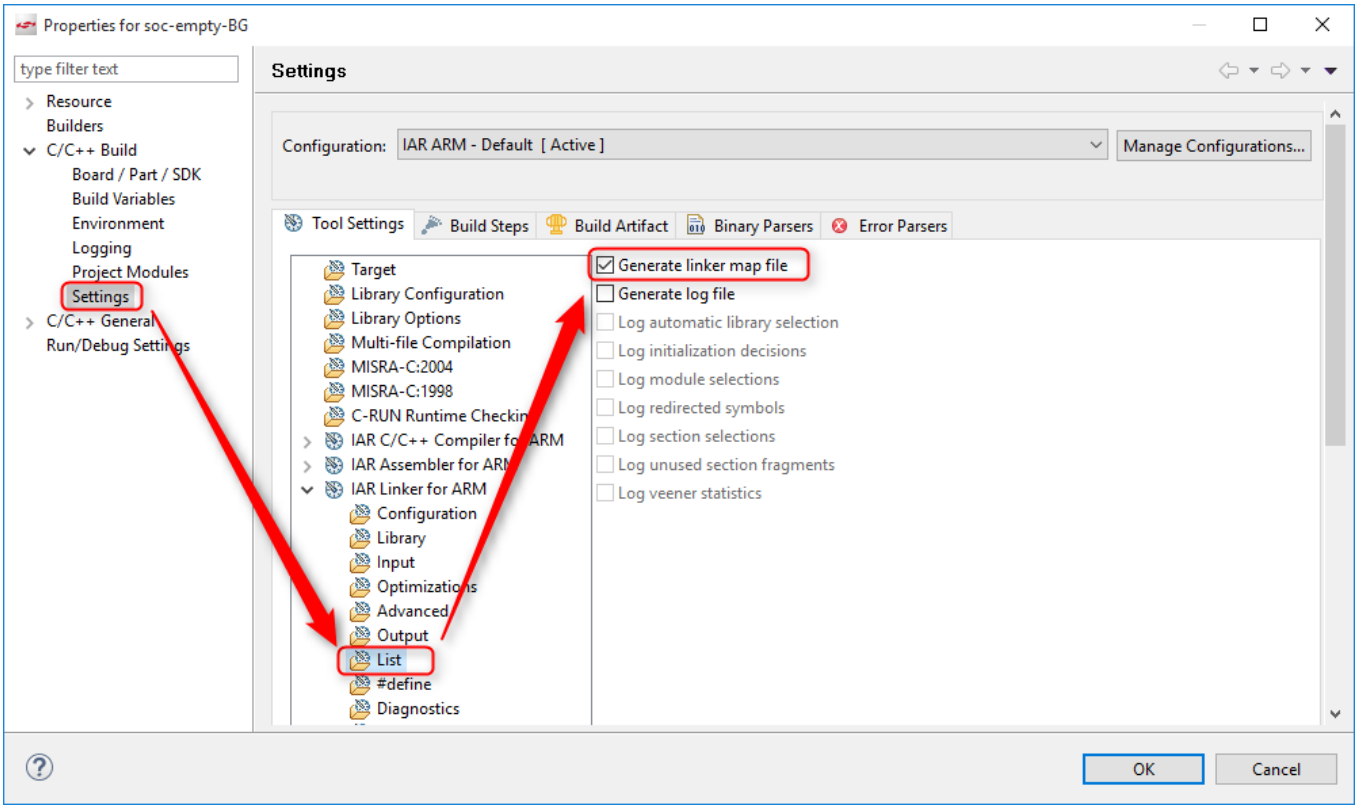
硬件 RTCC（实时时钟和日历）被 Bluetooth 协议栈设置为在计数器模式下运行，且被预留给协议栈使用。不过，应用可以读取但无法写入 RTC 值。每当设备启动时，RTC 值都会复位。

如果应用需要像 RTCC 这样的功能，可以开发以下应用模式代码：

1. 构建一个用于从外部设备（如智能手机）中检索当前时间的机制。某些智能手机可执行 Bluetooth 定时器配置文件，而该文件可用于读取时间和日期值。
2. 将时间转换为“自 Epoch 以来的秒数”（例如，使用 stdlib 中的 mktime）。
3. 使用 Bluetooth 协议栈的 API hardware_get_time() 获取自复位以来的秒数。
4. 计算“自 Epoch 以来的秒数”和自复位以来的秒数之差，并将其存储到 PS 密钥等处。
5. 若想获取当前日历时间，请使用 hardware_get_time 获取当前 RTC 值，并将 PS 密钥中的值和它相加，然后使用 stdlib 中的 local time 获取当前日历时间。

8. 附录：计算闪存和 RAM 消耗量

要得到准确的 RAM 和 ROM 消耗量结果，应在链接之后生成一个 *.map* 文件。在 IAR 中，必须在项目设置页面开启此选项，如下图所示。



在编译后，可在工作空间的 *IAR ARM - 默认*文件夹中找到映射文件。

在文件开头部分找到位置摘要部分。在此可以找到每个对象的准确大小和内存位置。

例如，在 SDK v2.3.0 中，旧版 OTA 引导装载程序（binbootloader.o）位于 0x00000000 地址，并消耗 0x31c（796）字节闪存。

```
 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 协议栈（binstack.o）位于 0x00004000，并消耗 0x1c9e4（117220）字节。

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

自 SDK v.2.3 0 起，bgapi 便是协议栈中的独立库，并链接到应用空间中。该 bgapi 被重构成若干个 bgapi 类。这些类可单独使用，所以未使用的类不会消耗内存。可在 .text_app 部分找到 bgapi 类的对象。

```
 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.
```

也可查看 CSTACK 和 HEAP 使用量。在这个示例中，CSTACK 从 0x20003038 开始，HEAP 从 0x20003818 开始，二者都是 2 K。

```
 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
```

要计算 GATT 数据库的 RAM 使用量，请加上 .data 和 .bss 部分由 gatt_db.o 占用的内存空间。下面的示例显示了 empty-soc-example 中的 GATT 数据库的 RAM 使用量。它使用 20 字节的 RAM。

```
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]
..
```

要查看每个内存部分的总体闪存和 RAM 使用量，请查看模块摘要的第一行，找到位置和大小。

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

加上每个部分的这些数值以计算总体闪存和 RAM 使用量。

以下部分消耗闪存：

- .text_bootloader
- .text_stack
- .text_app

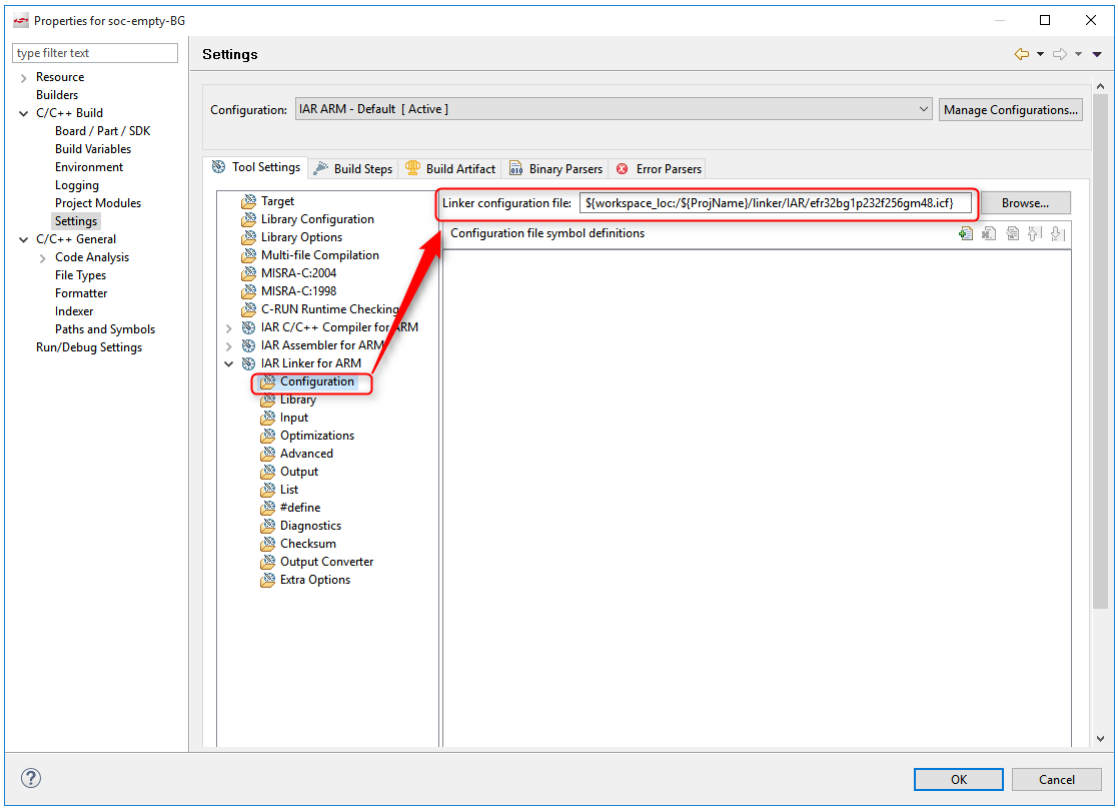
以下部分消耗 RAM：

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

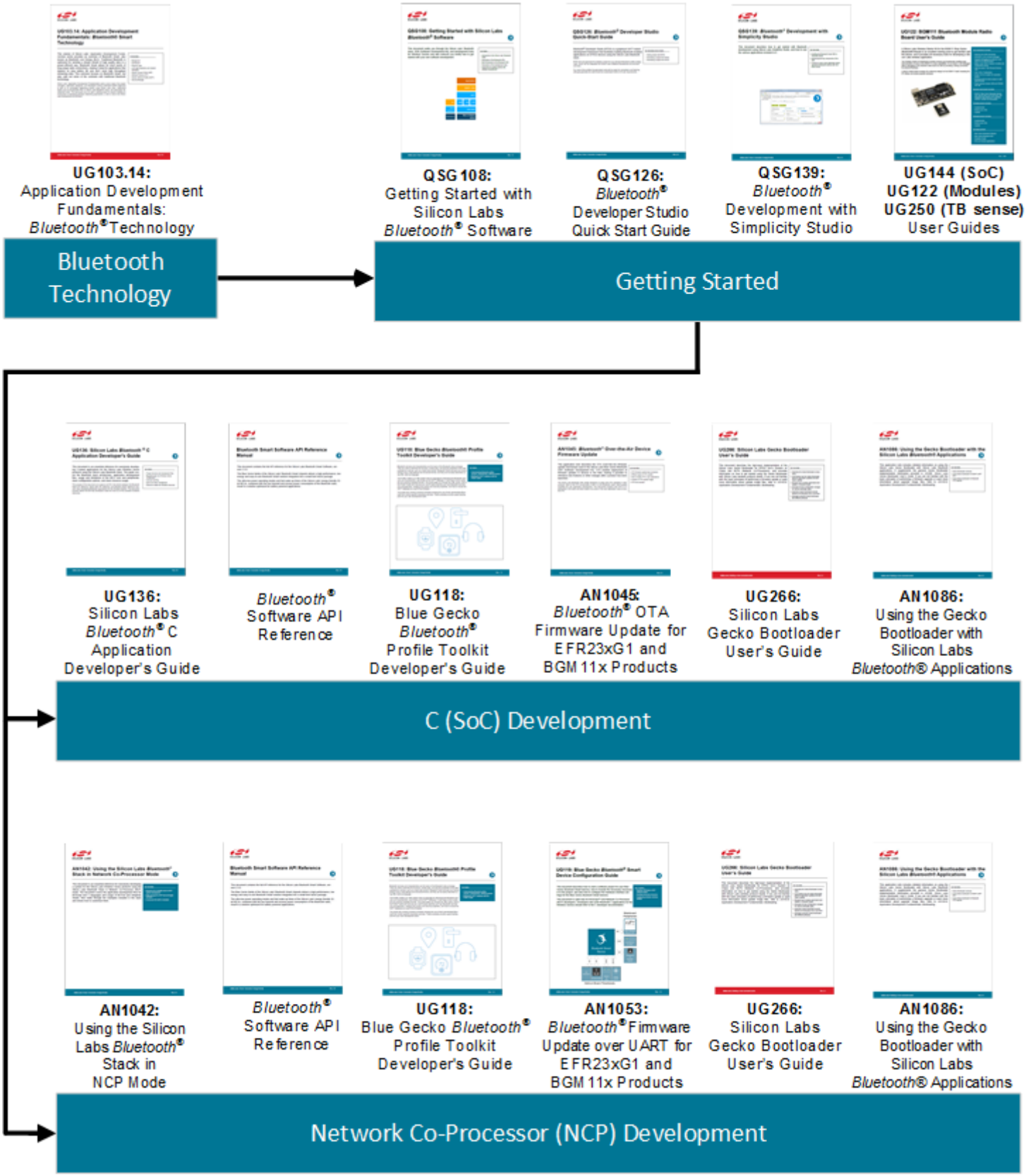
为 Bluetooth 协议栈预留的额外 12 K RAM 部分并未列在 *.map* 文件中，而是列在 IAR 链接器脚本文件中。

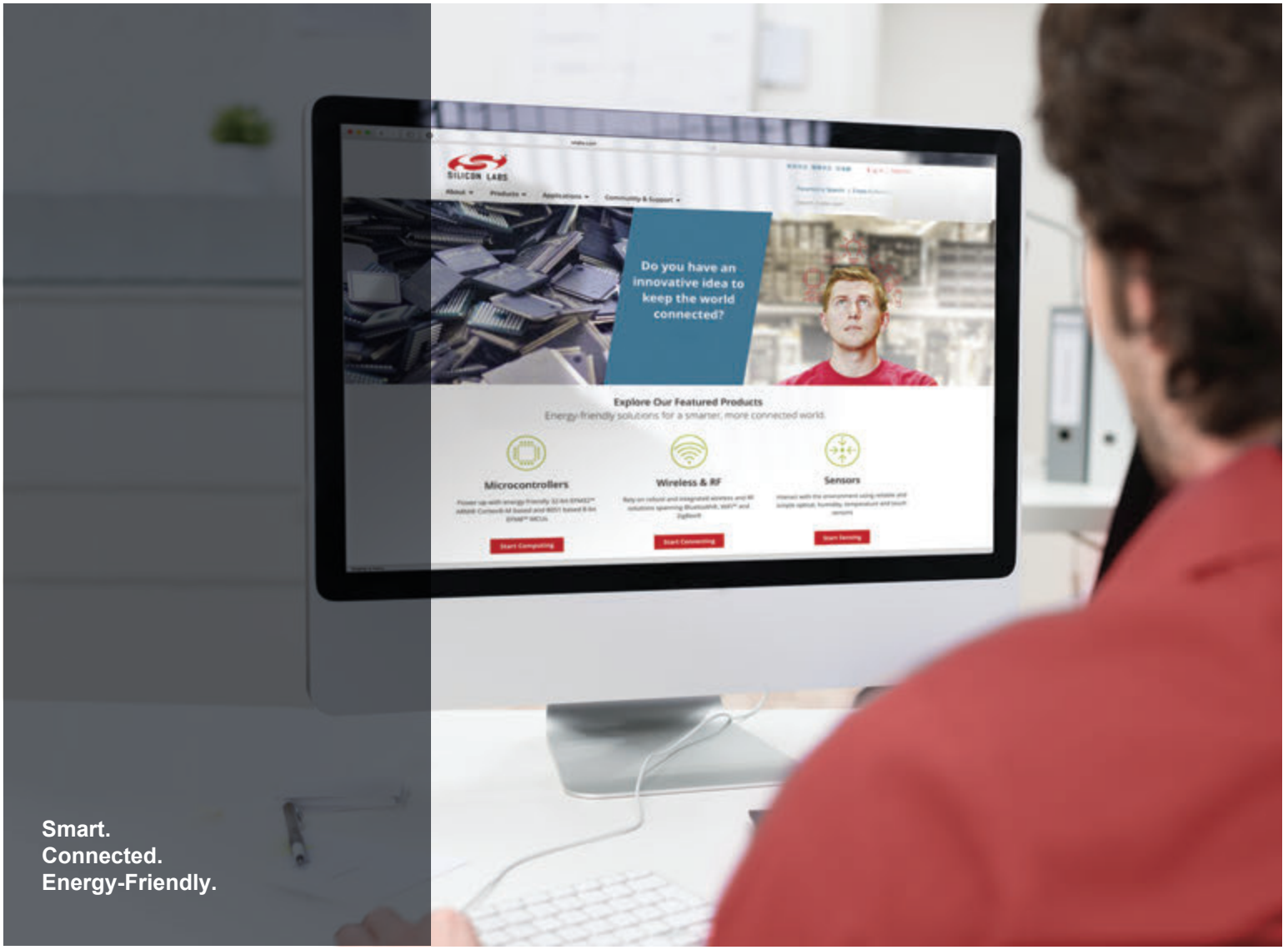
```
/*12KiB ram for stack*/
define symbol blob_ram_size = 0x3000;
define region RAM_region = mem:[from __ICFEDIT_region_RAM_start__+blob_ram_size to __ICFEDIT_region_RAM_end__];
```

该文件可在项目工作空间内找到，但因目标而异。请查看项目属性以找到使用的 *.icf* 文件。


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

9. 文档


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 NCP 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