UG136: Silicon Labs *Bluetooth* ® C Application Developer's Guide

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。该版本适用于 Silicon Labs Bluetooth SDK 2.9.x 及更高版本。

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 4
   1.1 关于此版本 4
   1.2 必备条件 4
2. 应用开发流程 5
   2.1 应用构建流程. 6
3. 项目结构. 7
   3.1 Bluetooth 文件 7
   3.2 GATT 数据库 9
   3.3 设备固件升级. 9
   3.4 RTOS 支持 10
   3.5 多协议支持 10
   3.6 硬件支持 11
4. 配置 Bluetooth 协议栈和 Wireless Gecko 设备 12
   4.1 Wireless Gecko MCU 和外围设备配置 12
      4.1.1 自适应跳频 12
      4.1.2 Bluetooth 时钟. 12
      4.1.3 DC-DC 配置 13
      4.1.4 LNA 13
      4.1.5 PTI 13
      4.1.6 发射功率 13
      4.1.7 Wi-Fi 共存 14
   4.2 使用 gecko_stack_init() 配置 Bluetooth 14
      4.2.1 CONFIG_FLAGS 15
      4.2.2 Mbedtls 15
      4.2.3 多协议优先级配置 15
      4.2.4 睡眠模式 15
      4.2.5 Bluetooth 协议栈配置 16
      4.2.6 OTA 配置 16
      4.2.7 PA 17
      4.2.8 软件定时器 17
      4.2.9 RF 路径增益 17
5. Bluetooth 协议栈事件处理 18
   5.1 阻塞事件侦听器 18
   5.2 非阻塞事件侦听器 18
      5.2.1 睡眠和非阻塞事件侦听器 19
   5.3 用于 Micrium OS 的事件侦听器 19
      5.3.1 来自多个任务的命令 20
6. 中断 21
   6.1 外部事件 21

6.2 优先级 22
7. Wireless Gecko 资源 23
7.1 闪存 24
7.2 链接 25
7.3 RAM 25
7.3.1 Bluetooth 协议栈 25
7.3.2 Bluetooth 连接池 25
7.3.3 Bluetooth GATT 数据库 26
7.3.4 调用协议栈 26
7.3.5 堆内存 26
7.4 RTCC 26
8. 应用 ELF 文件 27
9. 文档 28

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 关于此版本

Silicon Labs Bluetooth SDK 的当前版本为 2.9.x。

当前支持的编译器和 IDE 版本为：

- **IDE：**Simplicity Studio 4.1.7 或更新版本
- **编译器：**IAR v7.80.4 和 GCC 7.2.1

2.8.x 版本中的重要变更：

将项目从 Bluetooth 协议栈 2.8.x 版本移动到 2.9.x 及更高版本时，必须要在项目中处理 Bluetooth 协议栈中的以下变更。

更改了命令 cmd_system_set_tx_power 的行为

命令 **cmd_system_set_tx_power** 设置允许的最大 TX 功率，之前它只设置使用的功率。这是 tx 功率补偿后的值，即它是天线的实际输出功率。

命令 cmd_le_gap_start_advertising 行为已更改

在使用传统广播的默认广播配置下，如果可连接模式允许且广播或扫描响应数据超过 31 个字节，则 Bluetooth 协议栈将自动切换为扩展广播。

1.2 必备条件

本文档假设 Silicon Labs 的 Bluetooth SDK 当前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。

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。

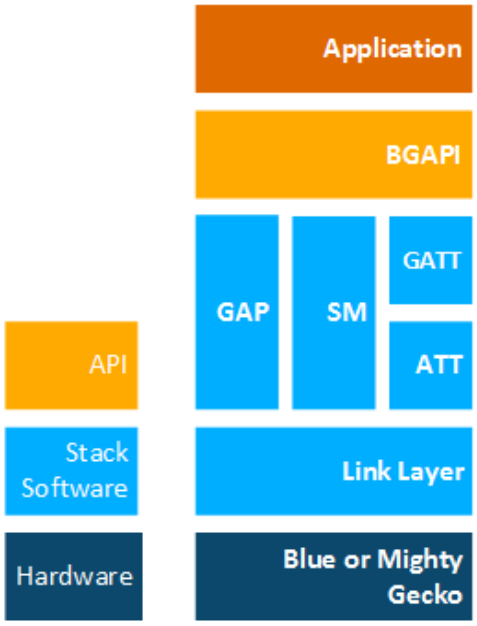


Figure 2.1. Bluetooth 协议栈体系结构方框图

Bluetooth 协议栈包含以下块。

- **Bootloader** – Gecko Bootloader 不是协议栈的一部分，但随 Bluetooth SDK 一起提供。请参阅*《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。有关引导装载的一般信息，请参阅*《UG103.06: 引导装载基础知识》*。
- **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。
- **Bluetooth AppLoader** – 在引导装载程序之后启动的应用。它会检查用户应用是否有效，如果检查有效，AppLoader 便会启动应用。如果应用映像无效，AppLoader 便会启动 OTA 进程以尝试接收有效的应用映像。这需要使用 Gecko Bootloader。

2.1 应用构建流程

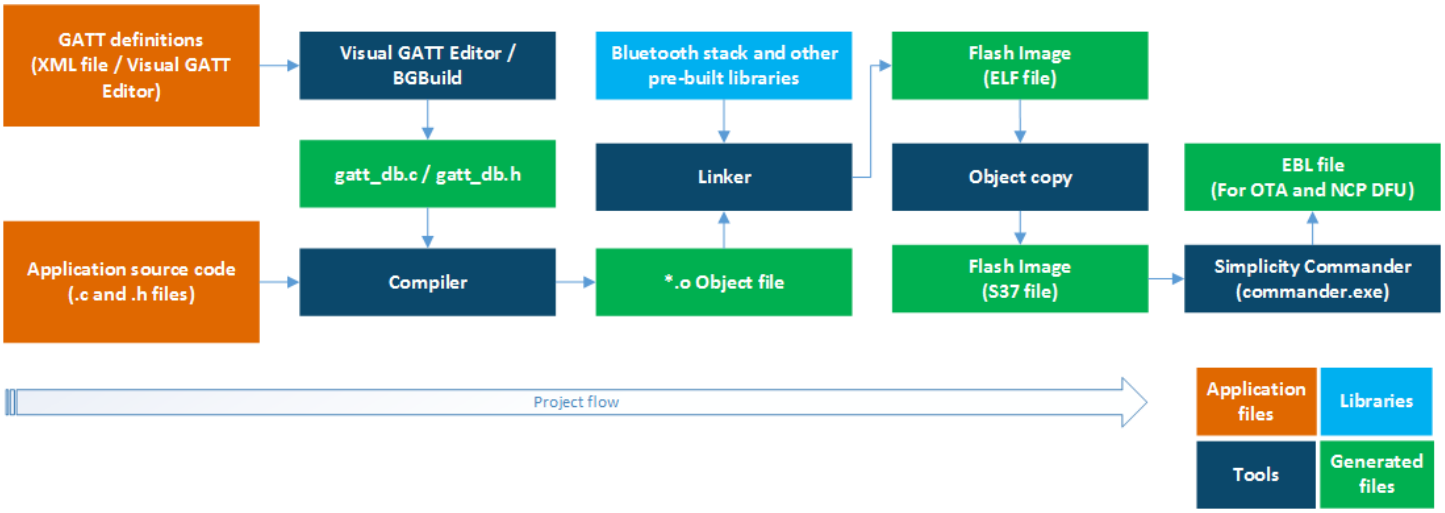


Figure 2.2. Bluetooth 项目构建流程

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包含在项目中的必要和可选资源。

3.1 Bluetooth 文件

库文件

Bluetooth 协议栈库是：

- **binapploader.o**：Bluetooth AppLoader 的二进制映像，提供可选的 OTA（无线）功能。
- **libbluetooth.a**：Bluetooth 协议栈库。
- **libmbedtls.a**：Bluetooth 协议栈的 mbed TLS 加密库。
- **libcoex.a**：Bluetooth 协议栈的 Wi-Fi 和 Bluetooth 共存仲裁特性。使用该文件需要硬件支持。
- **libpsstore.a**：Bluetooth 协议栈的 PSStore 功能。
- **librail.a**：Bluetooth 协议栈的 RAIL（无线电抽象接口层）库。

RAIL

Bluetooth 协议栈使用 RAIL 访问无线电，RAIL 库需要与 Bluetooth 协议栈链接。RAIL 为每个设备产品系列以及单协议和多协议环境提供单独的库。Gecko SDK 中提供了 RAIL 库。有关更多信息，请参阅*《UG103.13：RAIL 基础知识》*和其他 RAIL 文档。

EMLIB 和 EMDRV

Bluetooth 协议栈使用 EMLIB 和 EMDRV 库访问 EFR32 硬件。源代码中提供了 EMLIB 和 EMDRV 外围设备库，需要将这些库包含到项目中。EMLIB 和 EMDRV 是 Gecko SDK 的一部分。有关 EMLIB 和 EMDRV 的更多信息，请参阅<Simplicity Studio Gecko SDK> \platform\bootloader\documentation\Gecko_Bootloader_API_Reference\index.html 中的 Gecko Bootloader API 参考信息，以及 <Simplicity Studio Gecko SDK>\platform\ 下相应文件夹中的文档。

头文件

bg_version.h

该文件包含 Bluetooth 协议栈版本。

API 头文件

这些文件定义了 Bluetooth 协议栈 API。有三个用于不同使用案例的不同文件。只须包括其中一个文件。*native_gecko.h* 与裸机 Bluetooth 应用一起使用。*ncp_gecko.h* 在构建用于支持 NCP 的 SoC 应用时使用。*rtos_gecko.h* 与 Micrium RTOS 一起使用。

这些文件有两个用途：第一，它们包含实际的 Bluetooth 协议栈 API 以及该协议栈的命令和事件；第二，它们向 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。第 4.2 使用 gecko_stack_init() 配置 Bluetooth 节中更详细地讨论了配置选项和 gecko_init() 的使用方法。应用必须调用 gecko_init() 以初始化 Bluetooth 协议栈。

此函数是出于方便目的而提供。它初始化 Bluetooth 协议栈中的所有功能。要更细致地进行配置，请使用 gecko_stack_init()，如下所述。

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。

native_gecko.h

该文件用于不具有 RTOS 的应用中。它使用直接函数调用向 Bluetooth 协议栈提供 IPC（进程间通信）。

ncp_gecko.h

此文件在为主机提供 NCP 功能的应用中使用。 它使用 NCP 头作为函数调用来向 Bluetooth 协议栈提供 IPC。

rtos_gecko.h

当为 Micrium OS 构建应用时，使用 rtos_gecko.h。Bluetooth 协议栈是 Micrium OS 的独立任务，并使用其电源、睡眠和内存管理。rtos_gecko.h 通过 Micrium OS 中任何任务的 Bluetooth 协议栈为 IPC 提供封装程序。该文件包含 Bluetooth 协议栈 API 及该协议栈的命令和事件，以及一个 Bluetooth 协议栈配置 API。

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》*。

gatt_db.c 和 gatt_db.h

gatt_db.c 由 BGBuild.exe 或 Visual GATT 编辑器自动生成，用于定义 GATT 数据库的结构和内容。gatt_db.h 包含此数据库以及本地特性和服务的句柄。从 gatt_db_def.h 至 gatt_db.h，GATT 的类型定义将自动包含在内。

3.3 设备固件升级

设备固件升级 (DFU) 是指通过串行链路或无线链路 (OTA) 升级应用的过程。在这两种情况下，应用都需要添加以下文件来启用对 DFU 的支持。

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
```

在 Bluetooth AppLoader 中使用 OTA 进程时，应用属性结构体需要驻留在紧跟应用矢量表后的位置。使用 Bluetooth 协议栈提供的链接器文件时，将自动启用此对象。

3.4 RTOS 支持

Bluetooth 协议栈也可以在 Micrium RTOS 上运行。在这种情况下，*native_gecko.h* 被替换为 *rtos_gecko.h*，并且将以下文件添加到项目：*rtos_bluetooth.c* 和 *rtos_bluetooth.h*。

rtos_bluetooth.c 和 rtos_bluetooth.h

rtos_bluetooth.c 和 rtos_bluetooth.h 用于 Micrium OS 任务与 Bluetooth 协议栈及其他 Micrium OS 任务之间的 IPC（进程间通信）。使用 Micrium OS 时，还需要包括在下面介绍的 rtos_gecko.h 头文件。它提供了从任何 Micrium OS 任务使用 Bluetooth 协议栈的 API IPC 封装。

需要在 gecko_configuration_t 结构体中为 Bluetooth 协议栈配置 RTOS 支持。config_flags 字段需要有 GECKO_CONFIG_FLAG_RTOS 集。这导致 Bluetooth 协议栈需要依靠 Micrium OS 进入睡眠模式，而不是直接睡眠。应合理配置 scheduler_callback 和 stack_schedule_callback，以调用正确的函数。这些回调函数用于唤醒对应的任务。

与 Micrium OS 一同使用的 Bluetooth 协议栈配置如下：

```
.config_flags = GECKO_CONFIG_FLAG_RTOS, .scheduler_callback = BluetoothLLCallback, .stack_schedule_callback = BluetoothUpdate,
```

调用 gecko_stack_init() 后可以调用 bluetooth_start_task()。

```
void bluetooth_start_task(OS_PRIO ll_priority, OS_PRIO stack_priority);
```

它将任务优先级当作参数。ll_priority 用于链路层，stack_priority 用于 Bluetooth 协议栈。链路层优先级必须为系统中的最高优先级，其及时执行对系统性能至关重要。

3.5 多协议支持

在多协议环境中使用 Bluetooth 协议栈时，必须通过以下函数启用 Bluetooth 协议栈中的多协议功能：

```
gecko_init_multiprotocol(const void *config);
```

config 参数目前始终设置为 NULL，它被保留为用于将来的扩展。

此外，要在多协议环境中使用 Bluetooth，还必须使用支持多协议的 RAIL 库。

3.6 硬件支持

以下文件是 Gecko SDK 的一部分，它们添加了对硬件特定功能的支持。

hal-config.h

此头文件包含 MCU 外围设备的初始化设置，例如时钟和电源管理以及外围设备（例如 UART、SPI 等）的初始化设置。请注意，此文件仅包含外围设备的非板特定设置，例如 UART 的波特率，不包含板特定设置，例如 UART 的输入/输出引脚等。

init_mcu.c 和 init_mcu.h

这些文件包含用于初始化 MCU 的内部设置（如时钟和电源管理）的设备初始化函数。

init_board.c 和 init_board.h

这些文件包含用于初始化板上的外部部件的板初始化函数。例如，它启用 GPIO，并初始化无线电板上的外部闪存。

init_app.c 和 init_app.h

这些文件包含根据应用初始化 WSTK 上的外部部件的应用初始化函数。例如，它启用 WSTK 上的 VCOM、传感器和 LCD 显示器。

pti.c 和 pti.h

这些文件包含用于启用数据包追踪接口的 PTI 初始化函数。

hal-config-board.h

此头文件包含板初始化设置，例如按钮和 LED 引脚、UART 和 SPI 引脚等等。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。

bspconfig.h / bsphalconfig.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。如果使用硬件配置器工具，那么示例使用 bsphalconfig.h。否则，使用 bspconfig.h。

mx25flash_spi.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。

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

initMcu()

initMCU() 函数用于初始化 MCU 内核。此函数启动振荡器并配置设备的能量模式。可将独立于板设置的外围设备初始化（例如，定时器初始化）添加到此函数中。必须在 main() 刚开始时调用此函数。

initBoard()

initBoard() 函数用于初始化板功能，例如初始化外部闪存。可将取决于板设计的外围设备初始化（例如，GPIO 初始化或 UART 初始化）添加到此函数中。必须在 initMcu() 后调用该函数。

initApp()

initApp() 函数用于初始化应用特定功能，例如启用 WSTK 上的 SPI 显示。必须在 initBoard() 后调用此函数。

4.1.1 自适应跳频

Bluetooth 协议栈根据 ETSI EN 300 328 标准实现自适应跳频 (AFH)。使用 +10 dBm 及以上的发射功率时需要 AFH。AFH 还可以避免信道拥塞，从而提高性能。

要启用 Bluetooth 协议栈中的 AFH，必须调用以下初始化函数：

```
void gecko_init_afh();
```

在主从连接中，两端均可以独立使用 AFH。主机可以是非自适应性的，但从机可能仍然需要是自适应性的。该标准允许在阻塞的信道上使用控制传输。出于合规性方面的原因，如果从机检测到阻塞的信道正在使用中，它将只在该信道上发送一个数据包以避免连接超时。

Note: 传统广播不使用自适应跳频。传统广播使用 3 个频道，而 AFH 至少需要 15 个频道才能满足 ETSI 标准的要求。为 AFH 启用广播时需要使用扩展广播。

4.1.2 Bluetooth 时钟

时钟设置在 initMcu_clocks() 函数中初始化。时钟设置包括使用参数（例如，调谐）初始化振荡器（HFXO、LFXO 和 LFRCO）、初始化时钟（HFCLK、LFCLK、LFA、LFB、LFE）以及向振荡器分配时钟。注意：此函数中不启用外围设备时钟（如 GPIO 时钟、定时器时钟）。这些时钟必须在初始化外围设备时启用。

HFCLK

HFCLK 用于无线电协议定时器 (PROTIMER)。HFCLK 是高频时钟，精度必须为至少 ±50 ppm。此时钟要求外部晶体足够精确 (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 或 LFRCO 驱动，具体取决于精度要求。

时钟精度在 Bluetooth 协议栈配置结构体中定义，请参阅 4.2.5 Bluetooth 协议栈配置。

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，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 4.2.4 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。

CTUNE

在示例中，HFXO 和 LFXO 的晶体调谐（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。可根据设计使用 initMcu_clocks() 函数中的 hfxoInit.ctuneSteadyState 和 lfxoInit.ctune 设置来调整 CTUNE 值。

```
// Initialize HFXO
CMU_HFXOInit_TypeDef hfxoInit = BSP_CLK_HFXO_INIT;
hfxoInit.ctuneStartup = BSP_CLK_HFXO_CTUNE;
hfxoInit.ctuneSteadyState = BSP_CLK_HFXO_CTUNE;
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 initMCU() 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。这些自定义设置可在 *hal-config-board.h* 中设置。

```
#define BSP_DCDC_INIT                                                  \
  {                                                                    \
    emuPowerConfig_DcdcToDvdd,    /* DCDC to DVDD */                   \
    emuDcdcMode_LowNoise,         /* Low-noise mode in EM0 */          \
    1800,                         /* Nominal output voltage for DVDD mode, 1.8V  */  \
    15,                           /* Nominal EM0/1 load current of less than 15mA */ \
    10,                           /* Nominal EM2/3/4 load current less than 10uA  */ \
    200,                          /* Maximum average current of 200mA
                                     (assume strong battery or other power source) */      \
    emuDcdcAnaPeripheralPower_DCDC,/* Select DCDC as analog power supply (lower power) */ \
    160,                          /* Maximum reverse current of 160mA */                   \
    emuDcdcLnCompCtrl_1uOF,       /* 1uF DCDC capacitor */                                 \
  }
```

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》*。

4.1.4 LNA

低噪声放大器（LNA）是一种电子放大器，可放大极低功耗信号，同时不会显著降低信噪比。LNA 可提高 RF 灵敏度。

某些 MGM12P 模块具有板载 LNA。为使用这些模块，应正确配置并启用 LNA。在 *hal-config-board.h* 中使用前缀 BSP_LNA_ 配置 LNA。

如果板支持 LNA，则 LNA 在 initBoard() 函数的 module_initLna() 中初始化。

4.1.5 PTI

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。

PTI 在 initApp() 函数的 configEnablePti() 中初始化。可使用 HAL_PTI_BAUD_RATE 定义在 *hal-config.h* 中设置波特率，而使用带有 BSP_PTI_ 前缀的定义在 *hal-config-board.h* 中配置引脚。

从 Bluetooth 2.6.x 版本开始，使用 RAIL 提供的函数配置 PTI。

4.1.6 发射功率

Bluetooth 的发射功率取决于无线电允许的最大功率、软件配置、RF 路径增益补偿、以及自适应跳频（AFH）的使用。

ETSI EN 300 328 标准要求发射机功率为 +10 dBm 及以上时要使用 AFH。

如果受自适应性要求的限制，最大允许功率限制为 +10 dBm 以下。ETSI 标准要求至少有 15 个通道用于 AFH。在以下情况下，此要求禁止使用 +10 dBm 及以上：传统广播、扫描响应、和连接中 - 没有足够的信道可用时。

4.1.7 Wi-Fi 共存

Wi-Fi 共存 (COEX) 是一种协议，由 Bluetooth 和 Wi-Fi 仲裁哪个协议可使用无线传输。启用后，它可以提高 Wi-Fi 和 Bluetooth 的性能。在 *hal-config-board.h* 中使用带有前缀 BSP_COEX_ 和 HAL_COEX_ 的定义配置 COEX。

为启用 COEX，在调用 gecko_stack_init() 后调用以下函数：

```
gecko_initCoexHAL();
```

COEX 为 Wi-Fi IC 实现 GPIO 接口。它依赖于 EMLIB em_gpio.c 和 EMDRV gpiointerrupt.c，需要在项目中包含这两个文件。

4.2 使用 gecko_stack_init() 配置 Bluetooth

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：

```
uint8_t bluetooth_stack_heap[DEFAULT_BLUETOOTH_HEAP(MAX_CONNECTIONS)];
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",
  .max_timers=4
};
```

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、睡眠时钟准确度、GATT 数据库、OTA 配置和 PA 配置。

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()	已过时，为满足向后兼容性而提供。
gecko_bgapi_class_hardware_init()	支持访问和配置软件定时器。
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_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 多协议优先级配置

Bluetooth 协议栈和多协议环境中的其他协议一起使用时，可能需要更改 RAIL 的 Bluetooth 优先级设置以优化部分使用情况。

应用需要分配配置结构并将其提供给 Bluetooth 协议栈：

```
gecko_bluetooth_ll_priorities custom_priorities; static const gecko_configuration_t config = { // .bluetooth.linklayer_priorities = &custom_priorities, // };
```

gecko_bluetooth_ll_priorities 结构必须由 GECKO_BLUETOOTH_PRIORITIES_DEFAULT 常数初始化为默认状态。

gecko_bluetooth_ll_priorities 结构含有以下字段：

- scan_min & scan_max – 扫描操作的优先级范围。
- adv_min & adv_max – 广播操作的优先级范围。
- conn_min & conn_max – 连接包的优先级范围。
- init_min & init_max – 连接初始化的优先级范围。
- threshold_coex – 提高优先级信号时的阈值级别，仅在启用 coex 时使用。
- rail_mapping_offset – Bluetooth 优先级所在的 RAIL 优先级。
- rail_mapping_range – Bluetooth 优先级所在的 RAIL 优先级范围。

对于每个优先级范围，0 是最大优先级，0xff 是最小优先级。Bluetooth 优先级与 RAIL 优先级不同。也就是说，在 0 到 0xff 之间的所有 Bluetooth 优先级中，Bluetooth 有它自己的空间。要将 Bluetooth 优先级与 RAIL 优先级映射，则必须使用字段 rail_mapping_offset 和 rail_mapping_range 中的值形成单度方程：

```
RAIL_priority=(BT_priority/0xFF)*rail_mapping_range+rail_mapping_offset
```

4.2.4 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
```

4.2.5 Bluetooth 协议栈配置

协议栈内存

Bluetooth 协议栈使用内存管理程序为每个连接和内部数据缓冲区分配内存。此内存需要由应用分配并传递给 Bluetooth 协议栈。内存大小取决于连接数。C 宏 DEFAULT_BLUETOOTH_HEAP() 计算所需内存的默认大小（以字节为单位）。

分配 bluetooth_stack_heap 数组并将其传递给 Bluetooth 协议栈的示例：

```
uint8_t bluetooth_stack_heap[DEFAULT_BLUETOOTH_HEAP(MAX_CONNECTIONS)]; static const
        gecko_configuration_t config = { // .bluetooth.heap = bluetooth_stack_heap,
        .bluetooth.heap_size = sizeof(bluetooth_stack_heap), // };
```

连接数

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。在 gecko_init() 中进行初始化期间分配内存。可定义 C-define MAX_CONNECTIONS 来设置连接数。然后使用相同的定义来计算 Bluetooth 协议栈的内存大小，如上所述。然后，在配置结构体的 .bluetooth.max_connections 字段中将 MAX_CONNECTIONS 进一步传递给 Bluetooth 协议栈。

将 Bluetooth 连接限制为一 (1) 个的示例。

```
#define MAX_CONNECTIONS 1
```

有关连接 RAM 使用情况的更多信息，请参阅 7.3.2 Bluetooth 连接池。

睡眠时钟精度

Bluetooth 协议栈使用 .sleep_clock_accuracy 来优化从睡眠中唤醒的时间。单位是 ppm（百万分之一）。如果该值太大，则 Bluetooth 协议栈会过早从睡眠状态中唤醒以等待实际事件，从而导致耗电过多。如果该值太小，则 Bluetooth 协议栈唤醒太晚并错过连接事件，从而导致连接断开。

如果不定义该值或将其设置为 0，则使用默认值 250 ppm。

设置睡眠时钟精度的示例：

```
.bluetooth.sleep_clock_accuracy = 100, // ppm
```

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播发器。每个上下文分配约 60 字节的 RAM。

```
.bluetooth.max_advertisers = 5;  //!< Maximum number of advertisers to support, if 0 defaults to 1
```

4.2.6 OTA 配置

支持 Bluetooth 无线 (OTA) 固件升级，因为固件升级的一部分是由 Bluetooth AppLoader 应用处理的。

OTA 模式使用 .ota.flags 配置字段。它目前有一个选项，即 GECKO_OTA_FLAGS_RANDOM_ADDRESS，此选项将 OTA 设置为使用静态随机地址而不是公共地址。

当 Wireless Gecko 处于 AppLoader 的 OTA 模式时，可通过 Gecko 配置结构体配置其设备名称和设备名称长度。

```
  .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 固件更新的更多信息，请参阅*《UG266：Silicon Labs Gecko Booloader 用户指南》*和*《AN1086：将 Gecko Bootloader 与 Silicon Labs Bluetooth 应用一起使用》*。

4.2.7 PA

在基于 EFR32 SoC 的设计上，PAVDD（功率放大器稳压器 VDD 输入）可由 DC/DC 输出供电或直接由 3.3 V 电源供电。

Bluetooth 协议栈配置默认使用 DC/DC 作为 PAVDD 输入。如果 PAVDD 由 3.3 V 电源供电，那么应在 Bluetooth 配置结构体中加入以下两行：

```
.pa.config_enable = 1, .pa.input = GECKO_RADIO_PA_INPUT_VBAT,
```

4.2.8 软件定时器

可配置最多可用的软件定时器数量。各定时器需要协议栈资源才能运行。在某些使用案例中，增加软定时器的数量可能会导致性能下降。

```
.max_timers = 4; // 软件定时器的最大数量，最多 16 个，默认值：4
```

4.2.9 RF 路径增益

应用可以分别为 RX 和 TX 定义 RF 路径增益值。

调整发射机功率时，Bluetooth 协议栈会将 TX RF 路径增益考虑在内。然后天线辐射的功率与应用请求匹配。例如，如果应用请求的最大功率为 +10 dBm，路径损耗为 -1 dBm，则该引脚的实际功率为 +11 dBm。

使用 RX RF 路径增益补偿来自 Bluetooth 协议栈的 RSSI 报告。

```
.rf.tx_gain = -20; // RF TX 路径增益单位 0.1 dBm .rf.rx_gain = -18; // RF RX 路径增益单位 0.1 dBm
```

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。

```
/* 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

}
```

5.3 用于 Micrium OS 的事件侦听器

该应用在 Micrium OS 下运行，使用一个不同的程序来接收事件。该应用需要等待 Micrium OS 标志，而不是调用函数来接收事件。仅可接收单一任务的事件。

bluetooth_event_flags 中的 Micrium OS 标志用于向不同的任务通知 Bluetooth 协议栈的状态。该应用仅使用 BLUETOOTH_EVENT_FLAG_EVT_WAITING 和 BLUETOOTH_EVENT_FLAG_EVT_HANDLED。

该应用事件处理程序需要等待 BLUETOOTH_EVENT_FLAG_EVT_WAITING。

```
OSFlagPend(&bluetooth_event_flags, (OS_FLAGS)BLUETOOTH_EVENT_FLAG_EVT_WAITING 0,OS_OPT_PEND_BLOCKING + OS_OPT_PEND_FLAG_CONSUME, NULL
,&os_err);
```

输入事件随后添加至 bluetooth_evt。

```
switch
            (BGLIB_MSG_ID(bluetooth_evt->header)) { ... }
```

事件得到处理后，应释放该事件以接收下一个事件。此过程通过发布以下标志通知 Bluetooth 任务完成：BLUETOOTH_EVENT_FLAG_EVT_HANDLED。

```
OSFlagPost(&bluetooth_event_flags, (OS_FLAGS)BLUETOOTH_EVENT_FLAG_EVT_HANDLED, OS_OPT_POST_FLAG_SET, &os_err);
```

Note: 当应用挂起时，睡眠和电源管理自动由 Micrium OS 代替应用处理。

5.3.1 来自多个任务的命令

可以从多个 Micrium OS 任务发送 Bluetooth 命令。这要求各个任务在发送和释放命令之前获得排他性。

为了方便起见，Bluetooth 协议栈提供了两种函数。BluetoothPend 用于获得 Micrium OS 互斥量，BluetoothPost 用于释放互斥量。

以下代码块在 Bluetooth 命令前获得 Bluetooth 互斥量，之后又将其释放。

```
BluetoothPend(&err); //acquire mutex for Bluetooth stack gecko_cmd_gatt_server_send_characteristic_notification(0xff, gattdb_temp_measurement, 5, temp_buffer); BluetoothPost(&err);//release mutex
```

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。无线电的默认中断优先级为 1，链路层的优先级为 2，USART 中断的优先级为 3，其他中断的默认优先级为 4。

如果应用需要禁用中断，则推荐使用 BASEPRI 寄存器，而非 PRIMASK 寄存器。BASEPRI 寄存器按照中断优先级进行禁用，而 PRIMASK 将会禁用所有中断。EMLIB 内核可配置为使用 BASEPRI 寄存器，且之后可配合 CORE_ENTER_ATOMIC() 和 CORE_EXIT_ATOMIC() 宏命令使用。

如果没有 RTOS，链路层将使用 PendSV 实现高于应用软件的优先级。如果有 RTOS，链路层不使用 PendSV，但链路层任务将具有比应用任务更高的优先级。RTOS 计划程序随后将向链路层任务提供高于应用任务优先级的优先级。

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	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.

* PendSV 中断仅在没有 RTOS 的情况下使用

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. RTCC is only reserved in EFR32BG1X and EFR32BG12X. For more information see 7.4 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.

7.1 闪存

应用和 Bluetooth 协议栈从闪存中执行。闪存可分成供引导装载程序、Bluetooth AppLoader、应用以及永久性存储（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。对于有独立引导装载程序闪存的设备，它位于闪存处。
- Bluetooth AppLoader 为应用提供 OTA 升级能力。这是可选功能，但使用此功能时还要求同时使用引导装载程序。
- 应用位于 Bluetooth AppLoader 和 PS 存储之间。Bluetooth 协议栈是与应用链接的库。Bluetooth 协议栈包括实际 Bluetooth 固件，其中包括链路层以及 GAP、SM、ATT 和 GATT 层。
- 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 存储位于闪存的最后 4 kB。

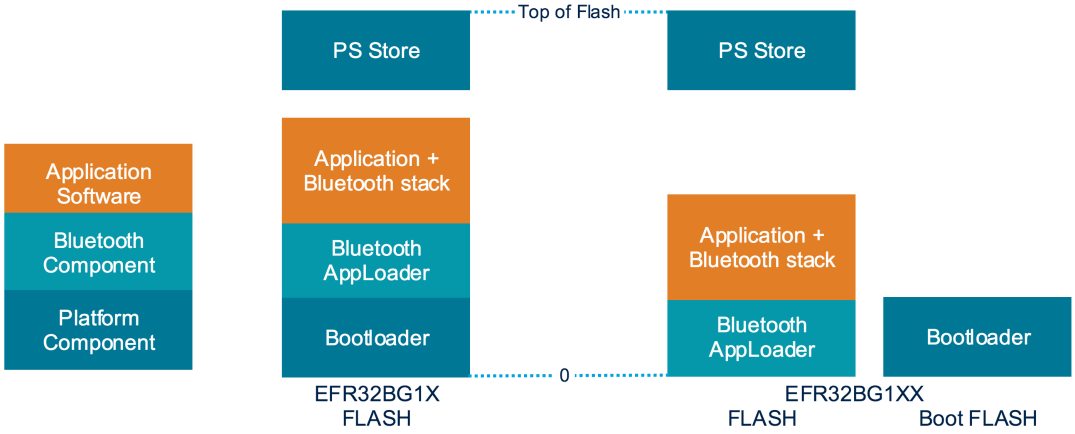


Figure 7.1. 有无独立引导装载程序闪存的闪存使用情况

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

	编译器	EFR32BG1X	EFR32BG12X	EFR32BG13x
引导装载程序		16	16	16
Bluetooth AppLoader		34	36	36
soc-empty*	GCC	125	128	130
	IAR	126	129	130
soc-thermometer*	GCC	127	130	132
	IAR	128	131	132
PSStore		4	4	4

**soc-empty* 和 *soc-thermometer* 是 Bluetooth SDK 中提供的示例应用。它们使用高优化程度进行编译。GCC 使用 -Os 标志， IAR 使用 -Ohz 标志。

7.2 链接

Bluetooth 协议栈以一组库文件的形式提供。应用将 Bluetooth 协议栈库与应用的其余部分链接。链接器将随后创建包含可供加载到闪存中的应用代码和数据的 ELF 文件。

为了生成 OTA DFU 文件，应用代码和数据必须链接到它们在 ELF 文件中的部分。这是通过 Bluetooth 协议栈中提供的链接器文件自动完成的。

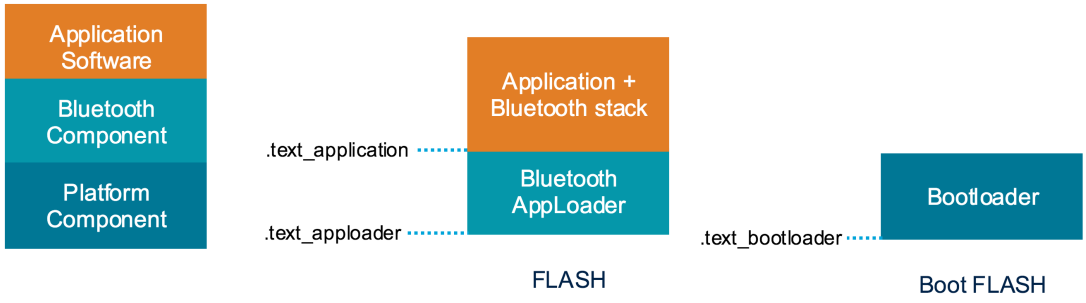


Figure 7.2. Sections Defined in the Linker File and Their Placement

链接器文件定义了两个内存区域，一个针对主闪存，另一个针对引导装载程序闪存。如果不存在单独的引导装载程序闪存，链接器文件将从引导装载程序的主闪存中保留一些内存。Bluetooth AppLoader 位于主闪存的开头，应用以及所有库从下一个可用的闪存页面启动。

有关 OTA 更新及其启用方法的更多信息，请参阅*《UG266: Silicon Labs Gecko Bootloader 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	4824 + Number of connections * 436 bytes
Bluetooth GATT database	Application-dependent (20 to 200 bytes)
Call stack	2 kB
Heap memory	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和应用编程接口的 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 = 4824 + 连接数 * 436 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 使用量为约 20 到 200 字节。

7.3.4 调用协议栈

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

调用协议栈大小定义的位置取决于编译器和启动文件。默认的调用协议栈大小为 2 kB。可使用以下命令行选项覆盖此值：

编译器	命令行选项	注释
IAR	--config_def __STACK_SIZE=*<size>*	调用协议栈在链接器文件中定义。此参数需要传递给链接器。
GCC	-D __STACK_SIZE=*<size>*	调用协议栈在启动代码中定义。需要为编译器定义此对象。

7.3.5 堆内存

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

堆大小定义的位置取决于编译器和启动文件。默认的堆大小为 3328 (0xD00) 个字节。可通过以下命令行选项覆盖此值：

编译器	命令行选项	注释
IAR	--config_def __HEAP_SIZE=*<size>*	调用协议栈在链接器文件中定义。此参数需要传递给链接器。
GCC	-D __HEAP_SIZE=*<size>*	调用协议栈在链接器文件中定义。需要为编译器定义此对象。

7.4 RTCC

本章仅适用于 EFR32BG1X 和 EFR32BG12X。EFR32BG13X 具有用于 Bluetooth 协议栈的单独定时器，并且应用可自由使用 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 localtime 获取当前日历时间。

8. 应用 ELF 文件

ELF（可执行和可链接格式）是可执行文件的标准文件格式。本章介绍 ELF 文件中与应用和 Bluetooth 协议栈相关的区段。

一些链接接器提供用于说明所消耗闪存的输出，但其中包含的内容并不明显。Bluetooth 项目可能包含引导装载程序和 Bluetooth AppLoader。设备可能具有用于引导装载程序的单独闪存。ELF 文件提供有关 RAM 和闪存使用情况的确切信息。

Simplicity Studio 提供 GCC 工具链，其中包含 *objdump* 命令行工具。此工具可用于从 ELF 文件中获取区段信息。

objdump 要求使用输入 ELF 文件。如果使用 -h 参数，*objdump* 将转储区段头信息。

IAR

为示例应用从命令行中调用 *objdump*：

```
arm-none-eabi-objdump -h IAR\ ARM\ -\ Default/soc-thermometer-iar-mg1p.out
```

objdump ，然后提供以下输出：

```
Sections: Idx Name Size VMA LMA File off Algn 0 .text_apploader rw 00008fc0 00004000 00004000 00000034 2**11 CONTENTS, ALLOC, LOAD, R
EADONLY, DATA 1 .text_application us 0001e3d3 0000d000 0000d000 00008ff4 2**11 CONTENTS, ALLOC, LOAD, READONLY, CODE 2 A1 rw 0000080
0 20000000 20000000 000273c8 2**3 ALLOC 3 P3 rw 00000246 20000800 20000800 000273c8 2**2 ALLOC, CODE 4 P3 ui 00000d00 20000a48 20000a
48 000273c8 2**3 ALLOC 5 P3 zi 00002b60 20001748 20001748 000273c8 2**8
```

.text_apploader 包含 Bluetooth AppLoader。

.text_application 包含应用代码和只读数据。在此示例中，应用大小为 0x1e3d3（十六进制数）和 123859（十进制数）个字节。

有关其余区段的说明，请参阅 IAR 文档。

GCC

为示例应用从命令行中调用 *objdump* ：

```
arm-none-eabi-objdump -h GNU\ ARM\ v4.9.3\ -\ Default/soc-thermometer-gcc-mg1p.axf
```

objdump 然后提供以下输出：

```
Sections: Idx Name Size VMA LMA File off Algn 0 .text_bootloader 00000000 00000000 00000000 000306d0 2**0 CONTENTS 1 .text_apploader
00009000 00004000 00004000 00004000 2**0 CONTENTS, ALLOC, LOAD, READONLY, DATA 2 .text_application 0001e4c4 0000d000 0000d000 0000d00
0 2**8 CONTENTS, ALLOC, LOAD, READONLY, CODE 3 .text_application_ARM.exidx 00000008 0002b4c4 0002b4c4 0002b4c4 2**2 CONTENTS, ALLOC,
LOAD, READONLY, DATA 4 .stack_dummy 00000400 20000000 20000000 000306d0 2**3 CONTENTS 5 .text_application_data 000002d0 20000400 0002
b4cc 00030400 2**2 CONTENTS, ALLOC, LOAD, CODE 6 .bss 00002a88 20000700 0002b800 00030700 2**8 ALLOC 7 .heap 00000c00 20003188 200031
88 00030ad0 2**3 CONTENTS
```

.text_bootloader 包含引导装载程序。在此示例中，它是单独装载的，并且区段为空。

.text_apploader 包含 Bluetooth AppLoader。

.text_application 包含应用代码和只读数据。在此示例中，应用大小为 0x1e3c3（十六进制数）和 124100（十进制数）个字节。

.text_application_ARM.exidx 用于调试目的

.stack_dummy 是调用协议栈的占位符区段。

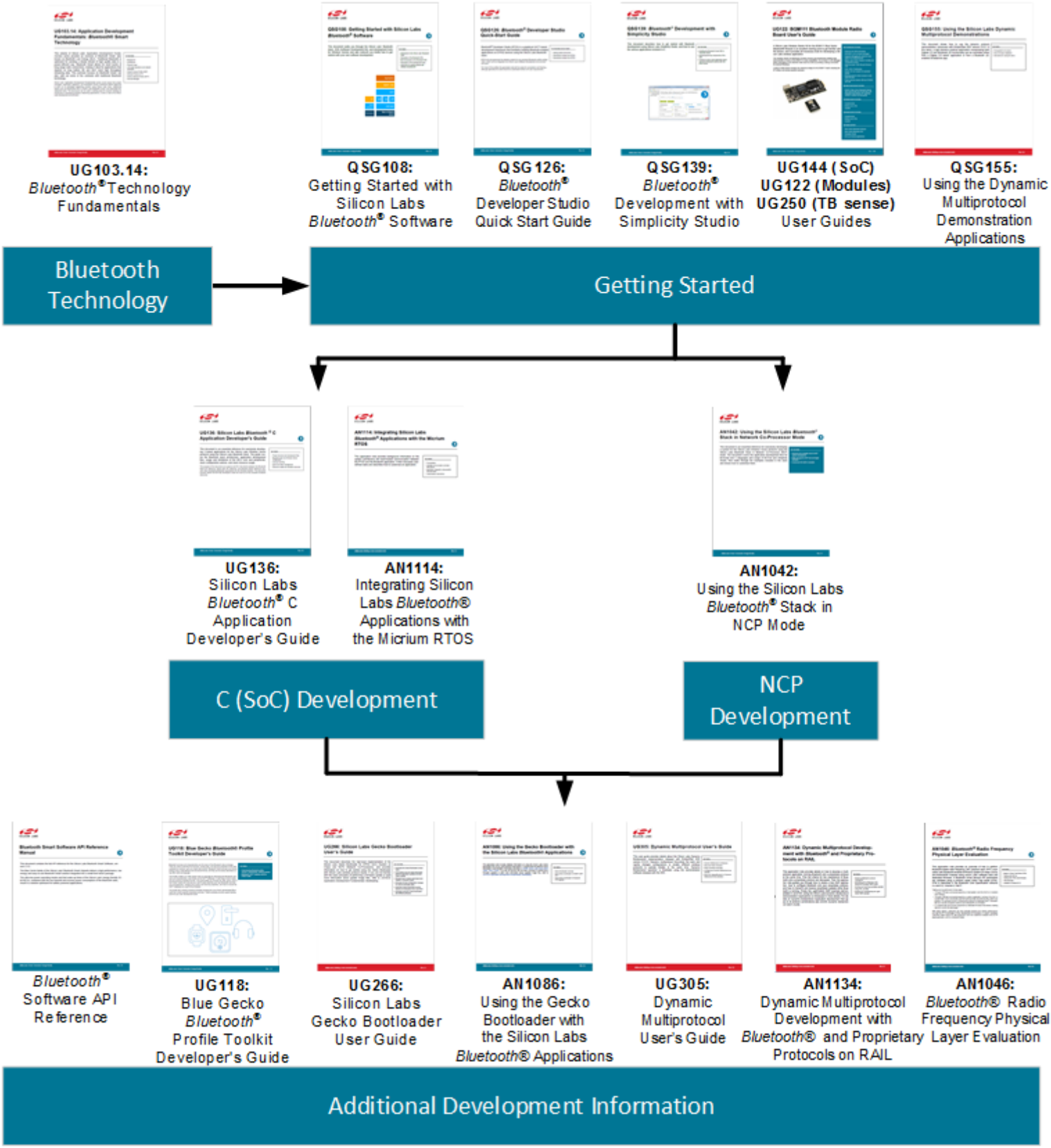
.text_application_data 是已初始化变量的 RAM 区段。

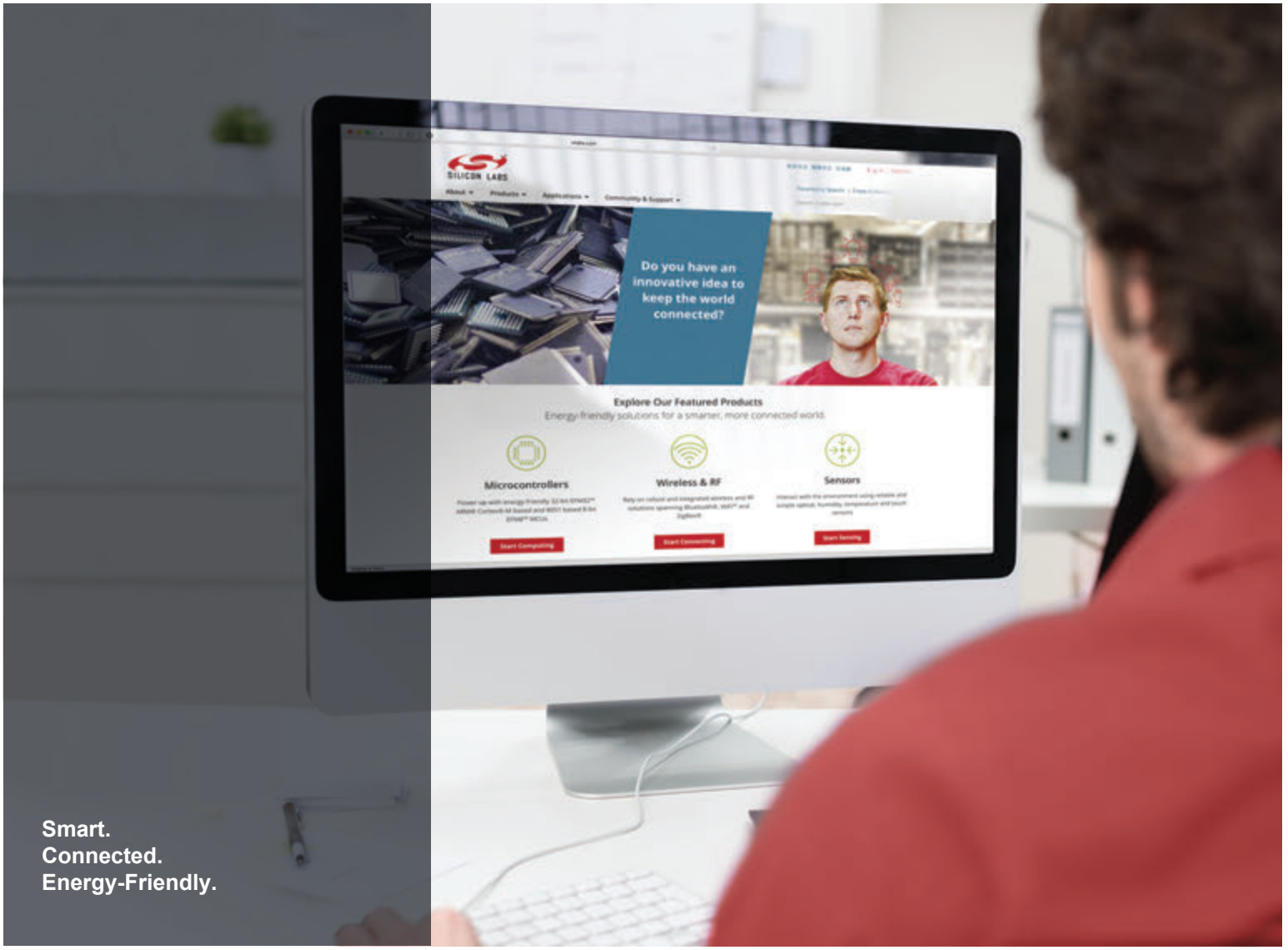
.bss 是未初始化变量的 RAM 区段。

.heap 是堆的 RAM 区段。

有关其余区段的说明，请参阅 GCC 文档。

9. 文档


UG103.14:
Bluetooth® Technology Fundamentals
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
QSG155:
Using the Dynamic Multiprotocol Demonstration Applications
Bluetooth Technology
Getting Started
UG136:
Silicon Labs Bluetooth® C Application Developer's Guide
AN1114:
Integrating Silicon Labs Bluetooth® Applications with the Micrium RTOS
AN1042:
Using the Silicon Labs Bluetooth® Stack in NCP Mode
C (SoC) Development
NCP Development
Bluetooth® Software API Reference
UG118:
Blue Gecko Bluetooth® Profile Toolkit Developer's Guide
UG266:
Silicon Labs Gecko Bootloader User Guide
AN1086:
Using the Gecko Bootloader with the Silicon Labs Bluetooth® Applications
UG305:
Dynamic Multiprotocol User's Guide
AN1134:
Dynamic Multiprotocol Development with Bluetooth® and Proprietary Protocols on RAIL
AN1046:
Bluetooth® Radio Frequency Physical Layer Evaluation
Additional Development Information



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