AN1265: Dynamic Multiprotocol Development with *Bluetooth*® and OpenThread on SoCs in GSDK v3.x and Higher

This application note provides details on developing Dynamic Multiprotocol applications using Bluetooth and Silicon Labs OpenThread. It describes how to configure applications in Simplicity Studio 5 using the Silicon Labs OpenThread SDK. It then provides a detailed walkthrough on how the underlying code functions. For details on Dynamic Multiprotocol application development that apply to all protocol combinations, see *UG305: Dynamic Multiprotocol User's Guide*.

KEY POINTS

- Sample app configuration
- CLI commands
- Important Software Components and files

1 Introduction

This application note provides instructions on getting started with dynamic multiprotocol (DMP) applications using Silicon Labs OpenThread and Bluetooth running over the FreeRTOS real time operating system.

The sample application **ot-ble-dmp** is a test application that demonstrates the components that go into building a DMP application. It provides a command line interface (CLI) that allows the user to execute basic OpenThread and Bluetooth commands. It also demonstrates how the power manager component can be used to save power by allowing the device to enter low power (EM2) mode in between activities.

The term 'dynamic' in DMP refers to the fact that both protocols are operating simultaneously. The radio scheduler takes care of multiplexing the transmitted and received packets over the radio. For more information on how the radio scheduler works, see *UG305: Dynamic Multiprotocol User's Guide*.

This document assumes that you have installed Simplicity Studio 5 (SSv5) and the OpenThread and Bluetooth SDKs, and that you are familiar with SSv5 and configuring, building, and flashing applications. If not, see *QSG170: Silicon Labs OpenThread Quick Start Guide*.

1.1 Hardware Requirements

- An EFR32 part with at least 512 kB of flash.

2 Building the ot-ble-dmp Sample App

Precompiled demo application images are provided with the Gecko SDK Suite 3.0, compatible with:

- brd4180a
- brd4186c
- brd2703a
- brd4116a

To get started quickly, In the Simplicity Studio 5 (SSv5) Launcher Perspective go to the DEMOS tab. Find the **ot-ble-dmp** demo and click **RUN**. This uploads the application image to your board.

To build the **ot-ble-dmp** sample app from source, you will need SSv5 and the Gecko SDK 3.x development environment with the following packages installed:

- OpenThread SDK
- Bluetooth SDK

The GNU ARM toolchain is installed with SSv5. The IAR-EWARM toolchain is not compatible with OpenThread.

1. With your target development hardware connected, open SSv5's File menu and select New > Silicon Labs Project Wizard. The Target, SDK, and Toolchain Selection dialog opens. Your target hardware should be populated. Click **NEXT**.
2. The Example Project Selection dialog opens. Use the Technology Type and Keyword filters to search for a specific example, in this case **ot-ble-dmp**. Select it and click **NEXT**.

   Note that, if you do not see the application, your connected hardware may not be compatible. To verify, in the Launcher Perspectives My Products view enter EFR32MGxx and select one of the boards. Go to the Examples tab, filter by Thread technology and verify you can see the app.
3. The Project Configuration dialog opens. Here you can rename your project, change the default project file location, and determine if you will link to or copy project files. Note that if you change any linked resource, it is changed for any other project that references it. Unless you know you want to modify SDK resources, use the default selection. Click **FINISH.**

The Simplicity IDE opens with the ot-ble-dmp project open in the Project Configurator. You may now build the project. For those used to Simplicity Studio 4, no generation step is necessary because it is done automatically. The ot-ble-dmp.s37 image will be located in the GNU ARM directory, and may be uploaded to your board using an SSv5 tool such as the flash programmer or Simplicity Commander.

3 CLI Commands

If you have used the **ot-cli-ftd** sample application, the OpenThread commands available in the **ot-ble-dmp** app are identical. Type help at the prompt to see a list. A complete OpenThread CLI reference is available here:

https://github.com/openthread/openthread/blob/master/src/cli/README.md

A quick tutorial on using the CLI to form a two-node OpenThread network and send a ping is available here:

https://github.com/openthread/openthread/tree/master/examples/apps/cli

The ot-ble-dmp app adds a set of Bluetooth commands that can be used to exercise the bluetooth stack. Type "ble" at the prompt to see a list of subcommands:

- `get_address`
- `create_adv_set`
- `set_adv_timing`
- `set_adv_random_address`
- `start_adv`
- `stop_adv`
- `start_discovery`
- `set_conn_timing`
- `conn_open`
- `conn_close`

These commands are implemented in the bluetooth_cli.c file, and each of them calls a corresponding Bluetooth C API function. For detailed documentation on the underlying functions, see https://docs.silabs.com/bluetooth/latest. Note that the C API prefixes for the Bluetooth SDK changed from gecko_ to sl_bt_ in version 3.0. See *AN1255: Transitioning from the v2.x to the v3.x Bluetooth® SDK* for more information about this and other BGAPI changes.

`ble get_address`

- Prints out the public Bluetooth address.
- Example: `ble get_address`
- Calls `sl_bt_system_get_identity_address()`

`ble create_adv_set`

- Create an advertising set. Must be called to obtain a handle for use in the other advertising commands.
- Example: `ble create_adv_set`
- Calls `sl_bt_advertiser_create_set()`

`ble set_adv_timing <handle> <interval_min> <interval_max> <duration> <max_events>`

- Set the advertising timing parameters of the given advertising set.
- Example: `ble set_adv_timing 0 160 320 0 0`
- Calls `sl_bt_advertiser_set_timing()`

`ble set_adv_random_address <handle>`

- Set the advertiser on an advertising set to use a random address.
- Example: `ble set_adv_random_address 1`
- Calls `sl_bt_advertiser_set_random_address()`

`ble start_adv <handle> <discoverableMode> <connectableMode>`

- Starts advertising on a given advertising set with specified discoverable and connectable modes.
- Example: `ble start_adv 0 2 2`
- Calls `sl_bt_advertiser_start()`

`ble stop_adv`

- Stops advertising on the given handle.
- Example: `ble stop_adv`
- Calls `sl_bt_advertiser_stop()`

`ble start_discovery <mode>`

- Scans for advertising devices.
- Example: `ble start_discovery 1`
- Calls `sl_bt_scanner_start()`

`ble set_conn_timing <min_interval> <max_interval> <latency> <timeout>`

- Sets the default Bluetooth connection parameters.
- Example: `ble set_conn_timing 6 400 0 800`
- Calls `sl_bt_connection_set_default_parameters()`

`ble conn_open <address> <address_type>`

- Connects to an advertising device. Address type 0=public address, 1=random address. Initiating phy argument hard coded to 1.
- Example: `ble conn_open 80fd34a198bf 0`
- Calls `sl_bt_connection_open()`

`ble conn_close <handle>`

- Closes a Bluetooth connection.
- Example: `ble conn_close 0`
- Calls `sl_bt_connection_close()`

3.1 Establishing a Bluetooth Connection Between Two Nodes

To establish a Bluetooth connection, the client starts advertising on advertising set 0 with modes discoverable and connectable. The server connects using the client's public address.

CLIENT:

```
> ble create_adv_set
ble create_adv_set
success handle=0
>
> ble start_adv 0 2 2
ble start_adv 0 2 2
success
>
> ble get_address
ble get_address
BLE address: 90fd9f7b5d39
```

SERVER:

```
> ble conn_open 90fd9f7b5d39 0
ble conn_open 90fd9f7b5d39 0
success
>
> BLE connection opened handle=1 address=90fd9f7b5d39 address_type=1 master=1 advertising_set=255
BLE connection parameters handle=1 interval=40 latency=0 timeout=100 security_mode=0 txsize=27
BLE event: 0x40800a0
BLE event: 0x900a0
BLE connection parameters handle=1 interval=40 latency=0 timeout=100 security_mode=0 txsize=251
```

4 Important Software Components and Files

With the ot-ble-dmp project open in Project Configurator, click the Software Components tab to see all the software components in the Component Library. Filter on Installed Components to see the components used in the project. Four key component categories pertain to building a DMP application:

- Bluetooth components
- OpenThread components
- Rail Library Multiprotocol component (under Platform > Radio)
- FreeRTOS components (under Third Party)

You need to include these components in any project to add OpenThread and Bluetooth DMP functionality.

When the FreeRTOS component is added to a project, the Project Configurator automatically takes care of adding the CMSIS RTOS2-based adaptation layers necessary to run the OpenThread and Bluetooth stacks over FreeRTOS. The adaptation files for OpenThread and Bluetooth are located in the following Simplicity Studio 5 locations:

- developer/sdks/gecko_sdk_suite/<version>/protocol/openthread/platform-abstraction/rtos/sl_ot_rtos_adaptation.c
- developer/sdks/gecko_sdk_suite/<version>/protocol/bluetooth/src/sl_bt_rtos_adaptation.c

The three application source files for this project (the only source files that are not part of a Gecko Platform component) are stored at the top level of the project and are named:

- main.c
- app.c
- bluetooth_event_callback.c

4.1 The Main Function and Initialization

The ot-ble-dmp app uses the same main function definition as used by other OpenThread sample applications. The call to `sl_system_init()`, which is defined in `sl_system_init.c`, initializes the entire system, including calls to `sl_bt_rtos_init()` and `sl_ot_rtos_init()` that are responsible for creating the Bluetooth and OpenThread threads.

The application can use the `app_init()` function to perform any necessary initialization steps prior to starting the kernel. The call to `sl_system_kernel_start()` starts the FreeRTOS scheduler.

The OpenThread instantiation and CLI initialization is handled by the OpenThread initialization thread by calling `sl_ot_init()` defined in `sl_ot_init.c`. More details on the different threads and their priorities are provided in the next section.

4.2 FreeRTOS Tasks

The ot-ble-dmp app creates the following five RTOS threads by default:

- OpenThread initialization thread (priority 53)
- OpenThread main thread (priority 24)
- Bluetooth link layer thread (priority 52)
- Bluetooth stack event thread (priority 51)
- Bluetooth event handler thread (priority 50)

The OpenThread threads are created in sl_ot_rtos_adaptation.c, and the Bluetooth tasks are created in sl_bt_rtos_adaptation.c.

As mentioned earlier, the OpenThread initialization thread handles the OpenThread instantiation and CLI initialization. As the Bluetooth event callback `sl_bt_on_event()` utilizes OpenThread CLI for prints (discussed in section 4.3 Handling Bluetooth Events), the OpenThread initialization thread starts with the highest priority.

Once the initialization is complete, the initialization thread also creates the main (operating) thread for OpenThread. By default, the main thread uses a low priority compared to Bluetooth, thus enabling Bluetooth threads to take over.

The priorities for the different Bluetooth threads and the OpenThread main thread are configurable and are defined in sl_bt_rtos_config.h and sl_openthread_rtos_config.h.

Silicon Labs Bluetooth has a serialized API which allows for commands and events to be passed between RTOS tasks in a thread-safe manner. OpenThread does not have a serialized API. For this reason, it is most convenient for the application logic to run in the OpenThread task. An application tick callback is provided for this purpose, and is called from within the OpenThread task's run loop: `sl_ot_rtos_application_tick()`. The ot-ble-dmp app includes a simple implementation of the tick callback in the app.c file.

OpenThread API calls made from within the application tick are thread-safe because they are executed within the OpenThread task. Because the Bluetooth API is serialized, Bluetooth API calls may be made from any task. The Bluetooth task is responsible for consuming and processing these serialized events. This happens transparently to the application.

4.3 Handling Bluetooth Events

Bluetooth events are dispatched to the application via the `sl_bt_on_event()` callback. For the ot-ble-dmp app, an implementation of this callback is located in bluetooth_event_callback.c. This example handler simply prints out some information about the event. In a real application, these events would be processed by application handlers.

Bluetooth events are processed within a dedicated Bluetooth Event Handler thread. This is a separate thread whose sole purpose is to check for waiting Bluetooth events, and call `sl_bt_on_event()` when they become available. This thread is automatically created during initialization.

4.4 Power Manager Integration

The ot-ble-dmp application also includes the Power Manager component (under Platform > Service > System), which is responsible for putting the system to sleep when possible.

The Power Manager component includes seamless FreeRTOS integration. It runs automatically from within the FreeRTOS Idle Task. When all threads have suspended (because they are pending on some event to continue processing), the Idle Task runs, and the power manager code can then put the system to sleep.

The application informs the power manager what sleep level it would like by adding and removing energy requirements via the API calls `sl_power_manager_add_em_requirement()` and `sl_power_manager_remove_em_requirement()`. Adding an EM1 requirement tells the Power Manager that the lowest energy level allowed is EM1, which only idles the processor and does not go to sleep. Removing the EM1 requirement allows the power manager to enter energy level EM2, which is deep sleep. See the reference for your MCU on https://docs.silabs.com/ under Modules>Platform Services>Power Manager.

5 OpenThread Sleepy End Device Demo

The ot-ble-dmp app starts out by adding an EM1 requirement during initialization, in the `sl_ot_rtos_application_init()` callback. This prevents the device from going into EM2 sleep mode, so that the CLI is responsive, and the user can enter commands.

To demonstrate an OpenThread Sleepy End Device, first form a two node OpenThread network by following the instructions at: https://github.com/openthread/openthread/tree/master/examples/apps/cli.

Next, on the device that joined the network (not the leader), type the following commands:

```
> mode s
> pollperiod 1000
```

The `mode` command puts the device into sleepy child mode. The `pollperiod` command tells the child to send data polls once every second. At this point the child is still not sleeping, and the CLI is still responsive.

Pressing either button PB0 or PB1 on the WSTK development board will toggle the energy mode requirement. Specifically, the first time the button is pressed, the EM1 requirement will be removed, allowing EM2 sleep. The child will start sleeping in EM2 mode in between data polls, and the CLI will no longer be responsive. You can verify that the child is still able to send and receive messages by sending a ping from the leader node. There will be up to one second of latency due to the child's sleep cycle. Pressing either button again will add back the EM1 requirement, which will bring the device out of EM2 so that the CLI can be used.

To monitor the power consumption of the device while performing the above steps, use the Energy Profiler tool in Simplicity Studio to connect to the device and start an energy capture. See *UG343: Multi-Node Energy Profiler User's Guide* for more information about the Energy Profiler.

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

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

www.silabs.com