AN1133: Dynamic Multiprotocol Development with *Bluetooth*® and Zigbee

This application note provides details on developing Dynamic Multiprotocol applications using Bluetooth and Zigbee. Using the examples described in *QSG155: Using the Silicon Labs Dynamic Multiprotocol Demonstration Applications*, this document describes how to configure applications in Simplicity Studio using the EmberZNet PRO 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

- Generating and configuring Zigbee/Bluetooth example files.
- Details on the application User Interface.
- How the Zigbee example application functions.
- How the Bluetooth Application functions.

1 Introduction

QSG155: Using the Silicon Labs Dynamic Multiprotocol Demonstration Applications describes how to control a dynamic multiprotocol light application either from a protocol-specific switch application or from a Bluetooth-enabled smartphone app. This application note provides details about how those examples are designed and implemented, to be used when developing your own Zigbee/Bluetooth multiprotocol implementations.

Note: The Zigbee dynamic multiprotocol solution is currently only supported for SoC architectures (not NCP) and always-on devices. Support for NCP architecture and sleep mode support are not yet available. Please contact Silicon Labs Sales for more information on our multiprotocol software roadmap.

1.1 Resources

- *UG305: Dynamic Multiprotocol User's Guide* provides details on
  - Dynamic Multiprotocol Architecture
  - Radio Scheduler operation (with examples)
  - Task Priority management
- *AN1135: Using Third Generation Non-Volatile Memory (NVM3) Data Storage in Dynamic Multiprotocol Applications* explains how NVM3 can be used as non-volatile data storage in Dynamic Multiprotocol applications with Zigbee and Bluetooth.

1.2 Development Environment Requirements

The required hardware for the example implementation is described in *QSG155: Using the Silicon Labs Dynamic Multiprotocol Demonstration Applications.* Along with the Simplicity Studio development environment, required software tools are:

- EmberZNet SDK version 6.0.0 or higher
- Bluetooth SDK version 2.6.0 or higher
- Micrium OS-5 kernel.
- An EFR32 chip with at least 512 kB of flash (required to run all the necessary software components)
- IAR Embedded Workbench for ARM (IAR-EWARM) version compatible with your SDK.

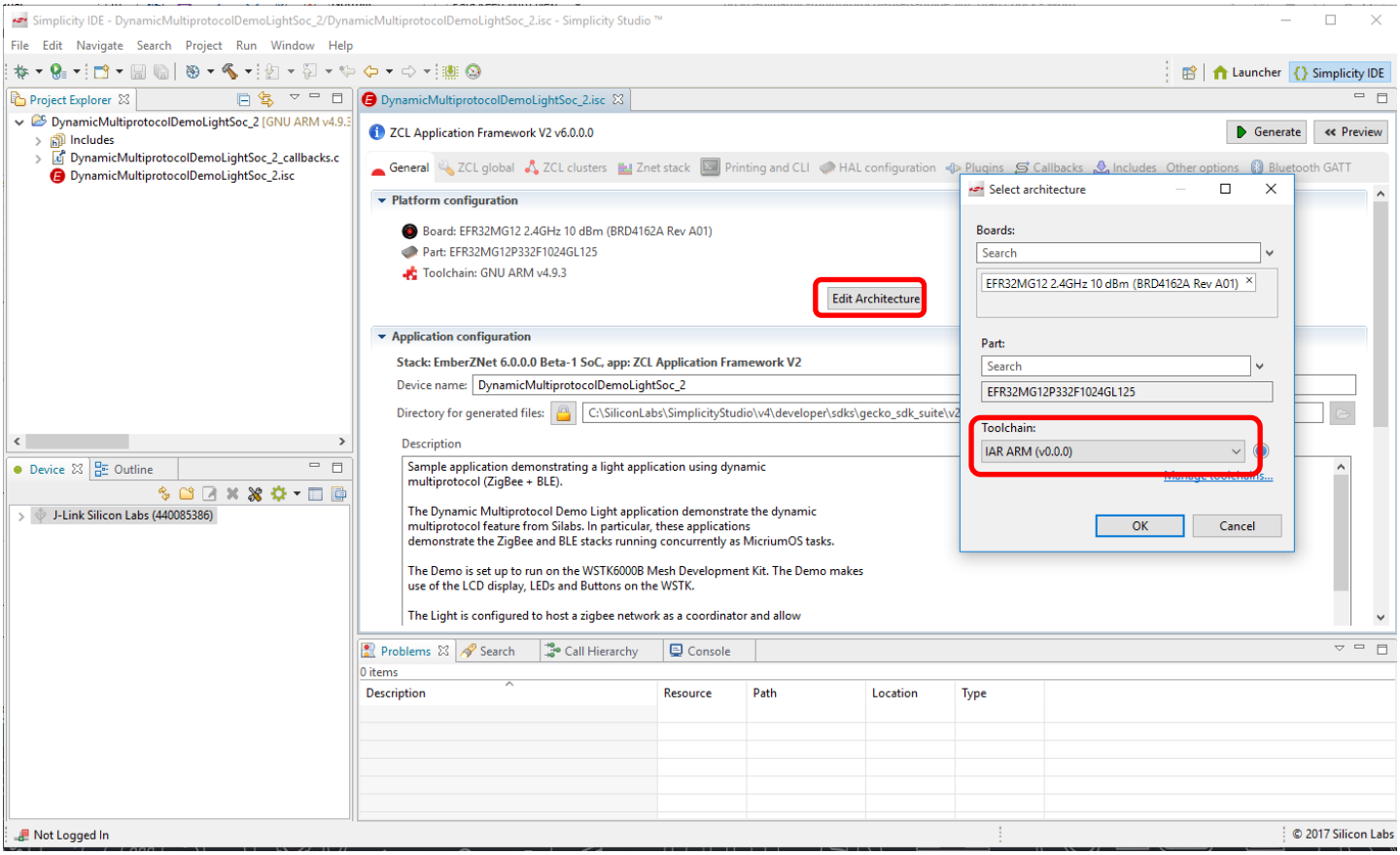
2 Working with the Zigbee/Bluetooth Example

2.1 Application Generation

To work with Zigbee/Bluetooth dynamic multiprotocol applications you must install both the EmberZNet SDK version 6.0.0.0 or higher, and the Bluetooth SDK version 2.6.0 or higher. The Micrium kernel is installed along with the EmberZNet SDK. IAR Embedded Workbench for ARM (IAR-EWARM) 7.80 must be installed and used as your compiler. See *QSG106: Getting Started with EmberZNet PRO* for information on installing the SDKs and IAR-EWARM.

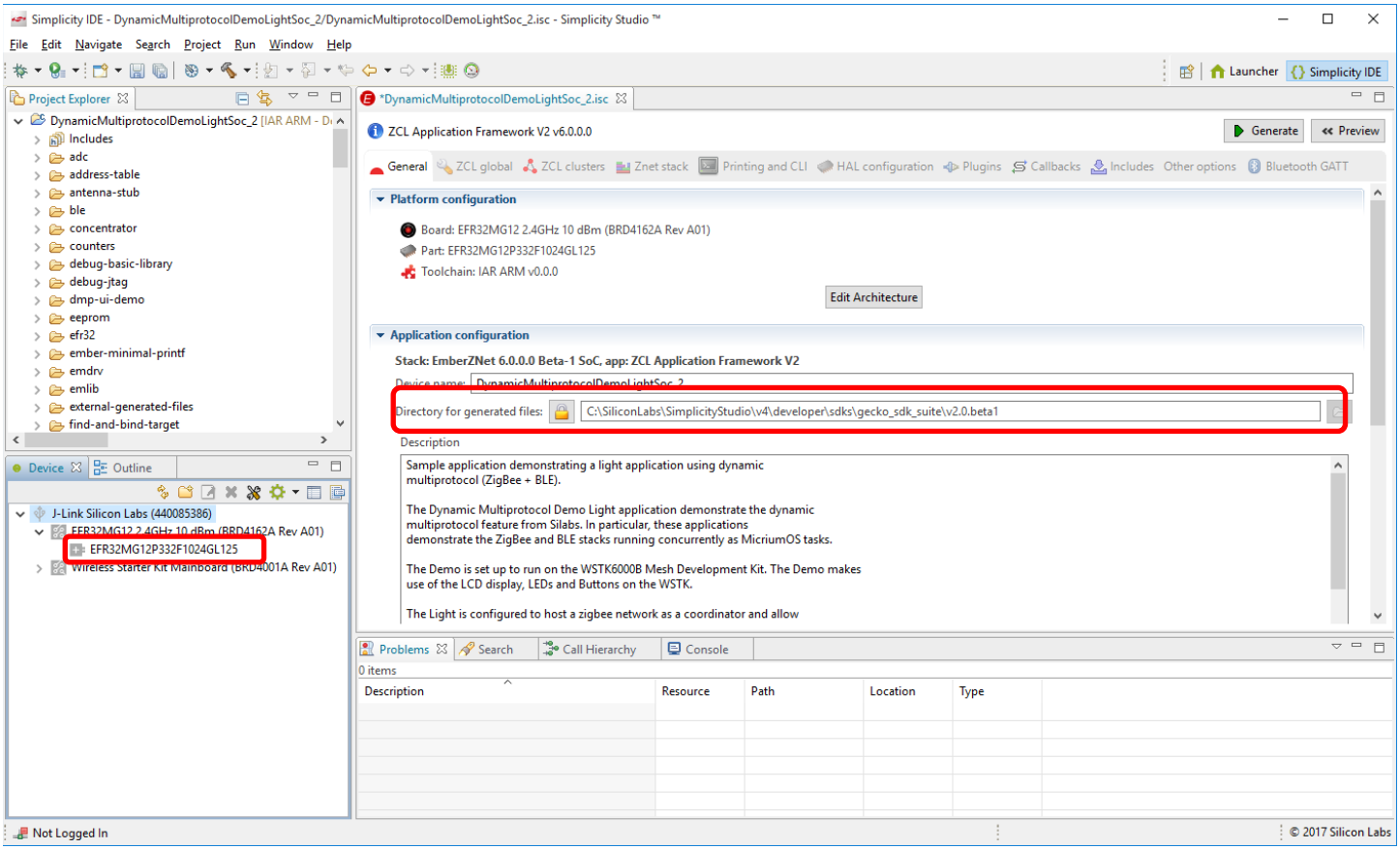
Dynamic multiprotocol applications are generated, built, and uploaded in the same way as other applications. If you are not familiar with these procedures, see *QSG106: Getting Started with EmberZNet PRO* for details. The following summary procedure uses the **DynamicMultiprotocolDemoLight** example application.

1. In Simplicity Studio, start a new project.
2. In the new project dialogs, select ZCL Application Framework V2, the EmberZNet SoC stack, then either check Start with a blank application or, as in this procedure, select the **DynamicMultiprotocolDemoLight** example.
3. Name the project. In project setup, if you have both compilers installed, uncheck GNU ARM, then click **Finish**.
4. If you forget to uncheck GNU ARM and your project General tab still shows it as a compiler, change to IAR EWARM.



5. Click **Generate** to generate project files.
6. Click  to build the application image.

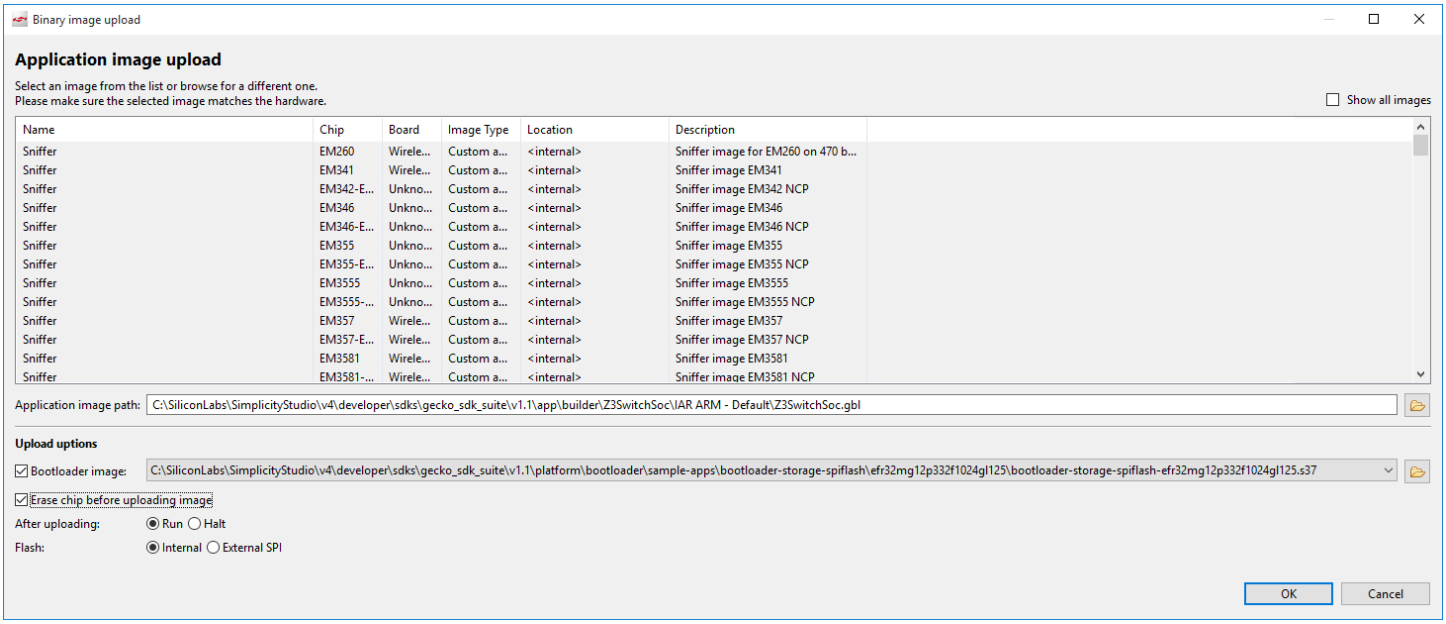
7. Note the part number for your device and the directory for generated files.



8. Right-click the target J-Link under Devices, and select Upload Application.
9. Browse to `<folder on General tab>\app\builder\<projectName>\IAR ARM - <qualifier>\<project name>` and select the .gbl file.
10. Silicon Labs strongly recommends that, if you have not already loaded a bootloader onto your device, you do so now. Check **Bootloader image**, then browse to the following folder and select the .s37 file corresponding to the radio board part number, for example 'bootloader-storage-spiflash-efr32mg12p432f1024gl125.s37'.:

```
C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\platform\bootloader\sample-apps\bootloader-storage-spiflash\
```

11. When both images are selected, the dialog should resemble the following figure. Click **OK**.



12. Application load success indicators are code-dependent. With the **DynamicMultiprotocolDemoLight** example, the LCD should display the following before changing over to the light bulb display:

2.2 EmberZNet Dynamic Multiprotocol Plugin Configurations

This section describes the configurations in a Zigbee application to implement Zigbee/Bluetooth LE Dynamic Multiprotocol functionality. The instructions assume you have started with a blank application.

In the Stack Libraries group:

- Enable **Bluetooth stack**.
  - Max number of Bluetooth connections: The maximum simultaneous connections in Bluetooth. The higher the number the more RAM is required for the application.
- Disable **Zigbee PRO Stack Library**
- Enable **Zigbee PRO BC table MbedTls Stack Library**. MbedTLS is required to arbitrate use of the encryption block, which both Zigbee and Bluetooth access for security.

In the Utility group:

- Enable **Micrium RTOS**. The stack then runs as a Micrium OS task.
  - CPU Usage Tracking: Checked by default. Uncheck to save RAM. .
  - Zigbee stack size: Defaults to 1000
  - Up to three **Application Tasks**. These are custom Micrium tasks other than the stack. Priority is the priority that Micrium will use to execute the tasks. The Zigbee stack and Bluetooth stack have priorities higher than 7 but they will yield to the application task. This is automatically managed by the Micrium kernel.
- Disable **Simple Main**.
- Enable **mbed TLS.**

- Enable **Idle/Sleep**. Dynamic Multiprotocol implementations use that logic to manage when the stack yields the radio.

In the RAIL group:

- Enable **RAIL Library Multiprotocol**, which provides RAIL library functionality with multiprotocol support.
- Disable **RAIL Library**

Note: The Multiprotocol group contains plugins used for a Switched Multiprotocol application. They are not enabled for a dynamic multiprotocol implementation.

3 About the Zigbee/Bluetooth LE Example

The Zigbee/Bluetooth LE Dynamic Multiprotocol example demonstrates a light that can be controlled from both Bluetooth and a Zigbee network. Software.is included both as compiled demonstrations and as example code in the EmberZNet SDK version 6.0.0. Demonstration functionality is illustrated in *QSG155: Using the Silicon Labs Dynamic Multiprotocol Demonstration*. The purpose of the example is show the way of implementing a dynamic multiprotocol application using the Silicon Labs EmberZNet stack.

The example is generated in the Simplicity IDE in the same way as any other Zigbee application is generated, but including some specific configuration options, as described in section 2.2 EmberZNet Dynamic Multiprotocol Plugin Configurations.

The Dynamic Multiprotocol Demo application has three main components.

1. User Interface (LCD and Buttons)
2. Zigbee application
3. Bluetooth application

3.1 User Interface

The user interface is developed specifically for the dynamic multiprotocol demonstration, and APIs to update the text and graphic on the LCD are called directly from Zigbee and Bluetooth event handlers. The implementation to manipulate the LCD is contained in the following files,

```
bitmaps.h //Contains the arrays containing the bitmap of the graphics drawn on the LCD
```

```
dmp_ui.c //Contains the functions to change the state of the display based on the state of the application
```

```
dmp_ui.h //Header file exporting functions implemented in the dmp_ui.c
```

The above uses the display driver library supplied by Silicon Labs to update the content on the LCD display mounted on the WSTK.

3.2 Zigbee Application

The example is set up to be light and a coordinator on the Zigbee network. The following cluster set is supported by the application.

Supported Clusters
Basic
Identify
Scenes
Groups
On/Off
ZLL Commissioning

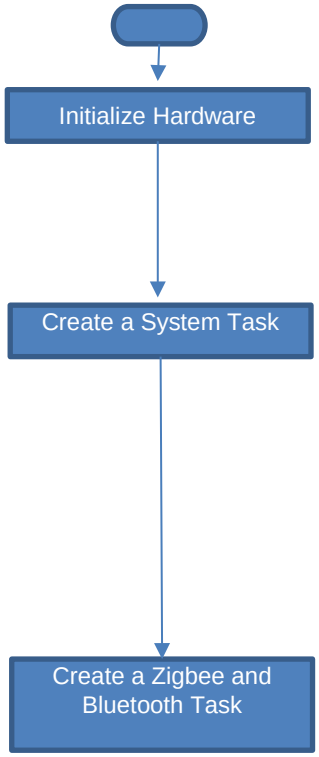
The example also supports Green Power Proxy Basic. Please note that the example was developed with a focus on demonstrating dynamic multiprotocol feature and may not be Zigbee-certifiable.

The On/Off cluster controls the LEDs and the bulb icon on the WSTK board to represent the state of the light.

The dynamic multiprotocol applications make use of Micrium OS and the Zigbee application is run as a task of Micrium OS.

The hardware and peripherals of the chip are initialized before any tasks are created. A system task is created after initialization, which then creates the application tasks including the Zigbee and Bluetooth task.

From: micrium-rtos-main.c



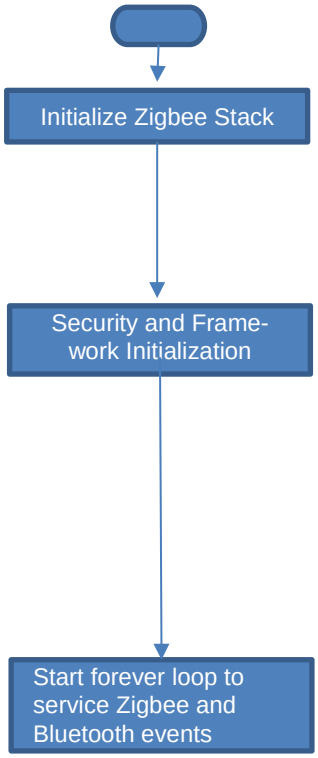
Micrium-rtos.main.c

```
halInit();
initMicriumCpu();
emberAfMainInit();
```

```
OSTaskCreate(&systemStartTaskControlBlock,
                "System Start",
                systemStartTask,
                NULL,
                SYSTEM_START_TASK_PRIORITY,
                &systemStartTaskStack[0],
                SYSTEM_START_TASK_STACK_SIZE / 10,
                SYSTEM_START_TASK_STACK_SIZE,
                0, // Not receiving messages
                0, // Default time quanta
                NULL, // No TCB extensions
                OS_OPT_TASK_STK_CLR | OS_OPT_TASK_STK_CHK,
                &err);
```

```
OSTaskCreate(&zigbeeTaskControlBlock,
                "Zigbee Stack",
                zigbeeTask,
                NULL,
                ZIGBEE_STACK_TASK_PRIORITY,
                &zigbeeTaskStack[0],

EMBER_AF_PLUGIN_MICRIUM_RTOS_ZIGBEE_STACK_SIZE / 10,

EMBER_AF_PLUGIN_MICRIUM_RTOS_ZIGBEE_STACK_SIZE,
                0, // Not receiving messages
                0, // Default time quanta
                NULL, // No TCB extensions
                OS_OPT_TASK_STK_CLR | OS_OPT_TASK_STK_CHK,
                &err);

bluetooth_start_task(BLE_LINK_LAYER_TASK_PRIORITY,
                     BLE_STACK_TASK_PRIORITY,
                     emberAfPluginBleGetConfig());
```

From: af-main-soc.c



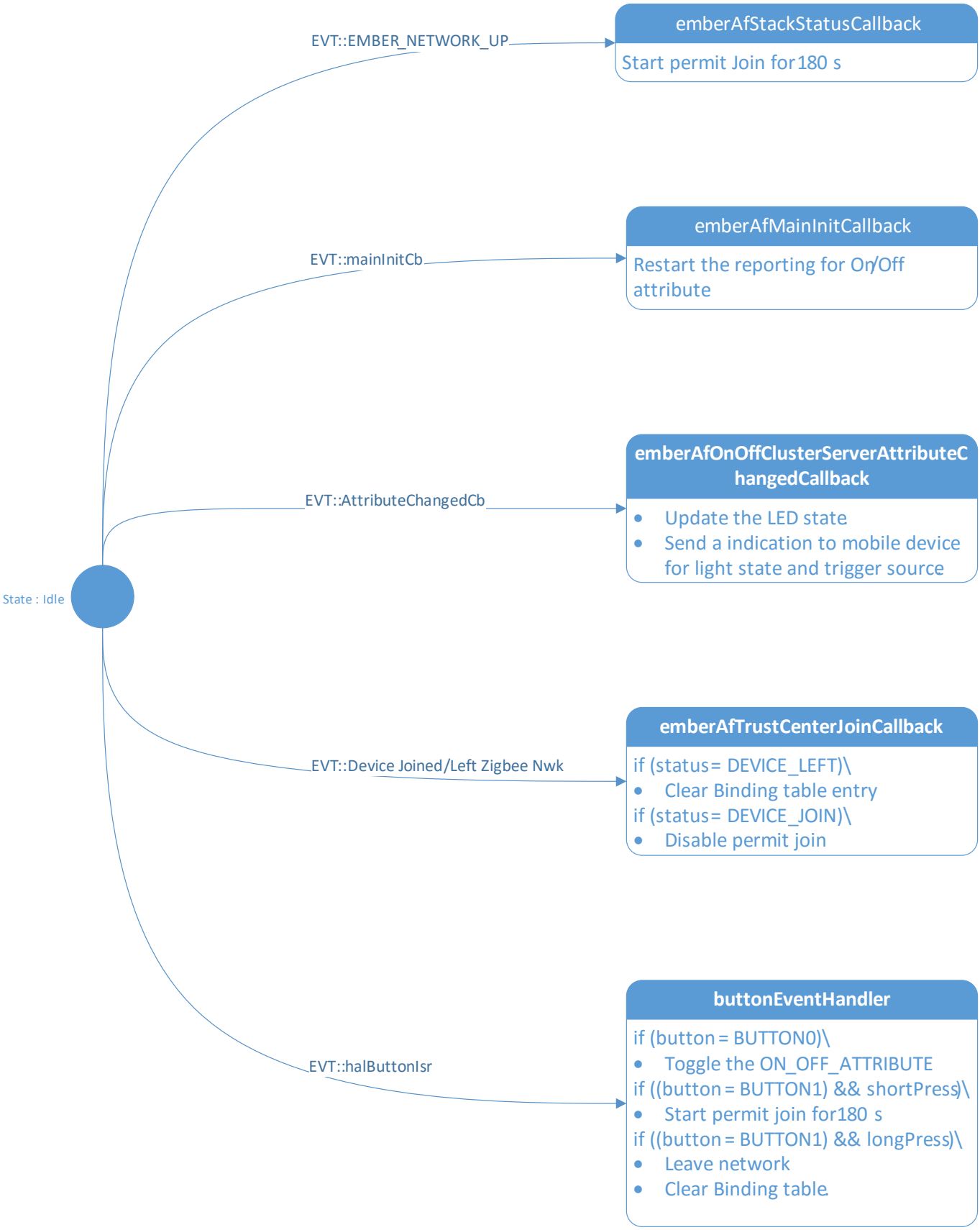
Af-main-soc.c

```
status=emberInit();
```

```
emAfInitializeNetworkIndexStack();
// Initialize messageSentCallbacks table
emAfInitializeMessageSentCallbackArray();
emberAfEndpointConfigure();
emAfInit();

// The address cache needs to be initialized and used with the
source routing
// code for the trust center to operate properly.
securityAddressCacheInit(EMBER_AF_PLUGIN_ADDRESS_TABLE_SIZE,
// offset
       EMBER_AF_PLUGIN_ADDRESS_TABLE_TRUST_CENTER_CACHE_SIZE);
// size

EM_AF_NETWORK_INIT();
```

```
while (true) {
    halResetWatchdog();    // Periodically reset the watch-
dog.
    emberTick();          // Allow the stack to run.
    // Allow the ZCL clusters and plugin ticks to run. This
should go
    // immediately after emberTick
    // Skip these ticks if a crypto operation is ongoing
    if (0 == emAfIsCryptoOperationInProgress()) {
      emAfTick();
    }

    emberSerialBufferTick();
    emberAfRunEvents();
}
```

Once the Zigbee stack is set up to run, subsequent interactions with the stack occurs via event handlers, as shown in the following figure.



3.3 Bluetooth Application

The Bluetooth application supports following services and characteristics. These are pre-selected in the GATT editor during project generation.

Service	Characteristic
Device Information	Manufacturer Name String Model Number String Serial Number String Firmware Revision String
Generic Access	Device Name Appearance
Silabs DMP Light	Light Trigger Source

3.3.1 Silabs DMP Light Service

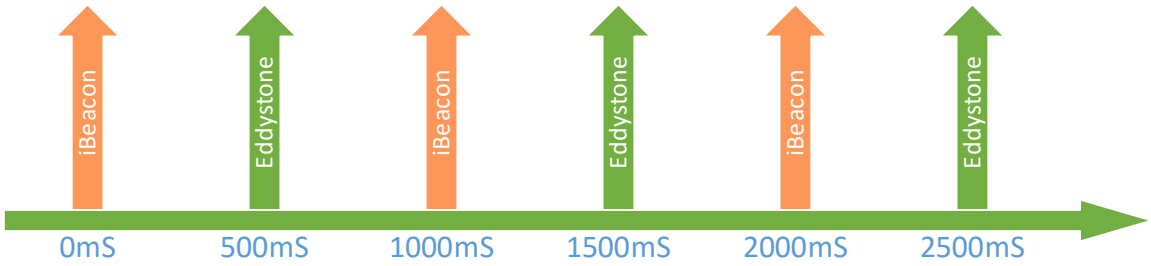
In the above table the Silabs DMP Light is a custom service with a UUID of `bae55b96-7d19-458d-970c-50613d801bc9`. This custom UUID is used to uniquely identify the Light by the Wireless Gecko application.

The Service has two characteristics,

Characteristic	Data Type	Description
Light	8bit Boolean	Used to get and set the light state 1 = Light On 0 = Light Off
Trigger Source	8bit enum	Indicates the source of the Light state change command. 0 = Bluetooth 1 = Zigbee 2 = Button Press

3.3.2 Beacons

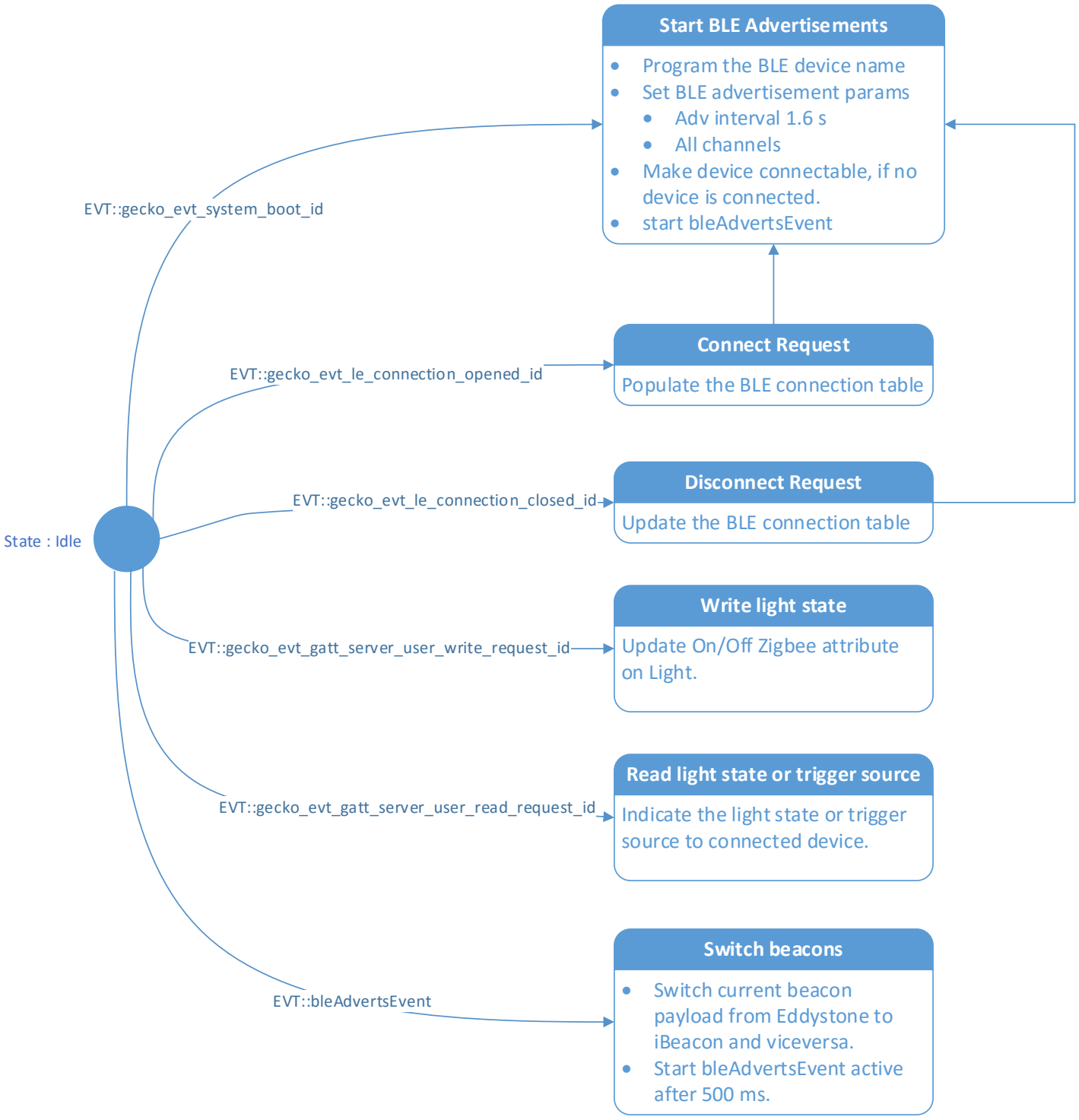
The application implements both iBeacons as well as Eddystone beacons. The default behavior is to transmit each beacon at alternating 500 mS interval as shown below



The beacons are set up in `bleAdvertsEventHandler` in the callbacks file.

3.3.3 Bluetooth Event Handling

The Bluetooth stack is initialized as part of the System Task, as shown in the Zigbee implementation section. The Bluetooth task handles the Bluetooth LE link layer messaging and management. The Bluetooth stack's interaction with the user application is through a framework plugin. A number of events that are called in the context of the Zigbee task allow the user application to interact with the Bluetooth stack. The following diagram describes the Bluetooth-related events.

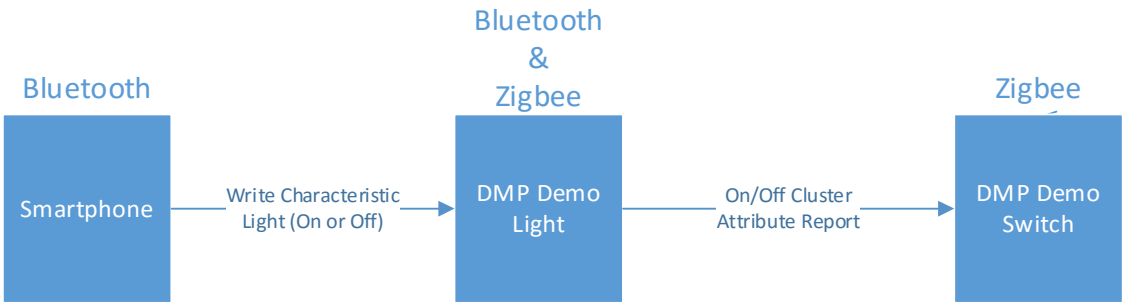


3.3.4 Bluetooth and Zigbee Interaction

The primary purpose of the example application is to show Zigbee and Bluetooth working together on a device. For this purpose, when the Light receives a command to change its state through one protocol, it executes the command and sends out a notification to the other devices using the other protocol to keep everything in sync.

Two basic operations are described below, first a write to Light characteristics from a Bluetooth connected device (shown in the following figure) and then a change in the Light state from a Zigbee device.

Write from the Bluetooth Connected Device



The application's services and characteristics are pre-selected in the GATT editor in Simplicity Studio. Upon generation the characteristics are #define in the gatt_db.h. Using the #define reference, the characteristics can then be coupled to read and write Bluetooth requests. For example the Light characteristic is reference from GATT as `gatt_light_state` which is then tied to an application specific write API of `writeLightState` in the `AppCfgGattServerUserWriteRequest` as shown below.

```
static const AppCfgGattServerUserWriteRequest_t appCfgGattServerUserWriteRequest[] =
{
  { gattdb_light_state, writeLightState },
  { 0, NULL }
};
```

The application implements the Zigbee attribute write and a Bluetooth write response in the `writeLightState` function as follows

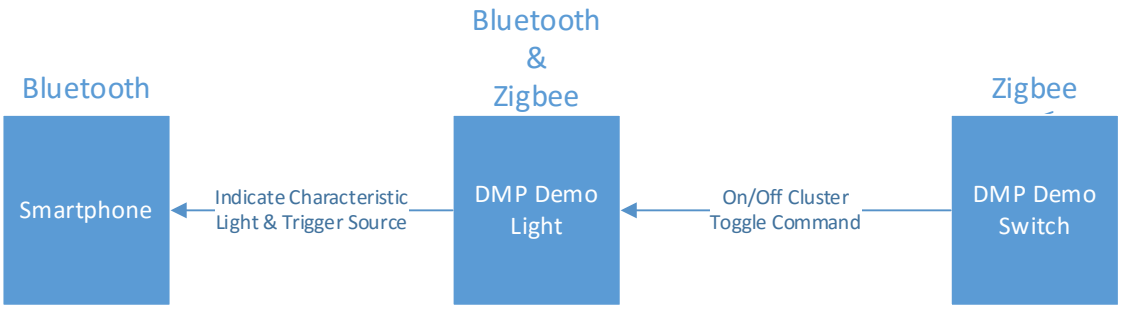
```
static void writeLightState(uint8_t connection, uint8array *writeValue)
{
  lightDirection = DMP_UI_DIRECTION_BLUETOOTH;
  emberAfWriteAttribute(emberAfPrimaryEndpoint(),
                        ZCL_ON_OFF_CLUSTER_ID,
                        ZCL_ON_OFF_ATTRIBUTE_ID,
                        CLUSTER_MASK_SERVER,
                        (int8u *)&writeValue->data[0],
                        ZCL_BOOLEAN_ATTRIBUTE_TYPE);
  gecko_cmd_gatt_server_send_user_write_response(
    connection,
    gattdb_light_state,
    ES_WRITE_OK
    );
}
```

The `emberAfWriteAttribute()` is used to write the attribute table of the Zigbee application with the value supplied by the Bluetooth connected device above. Since the on-off attribute of the on-off server cluster is a reportable attribute it is reported to all devices setup in the binding table of the Light.

The `emberAfOnOffClusterServerAttributeChangedCallback()` is then used to change the state of the LEDs and the LCD to indicate the state of the light on the WSTK main board.

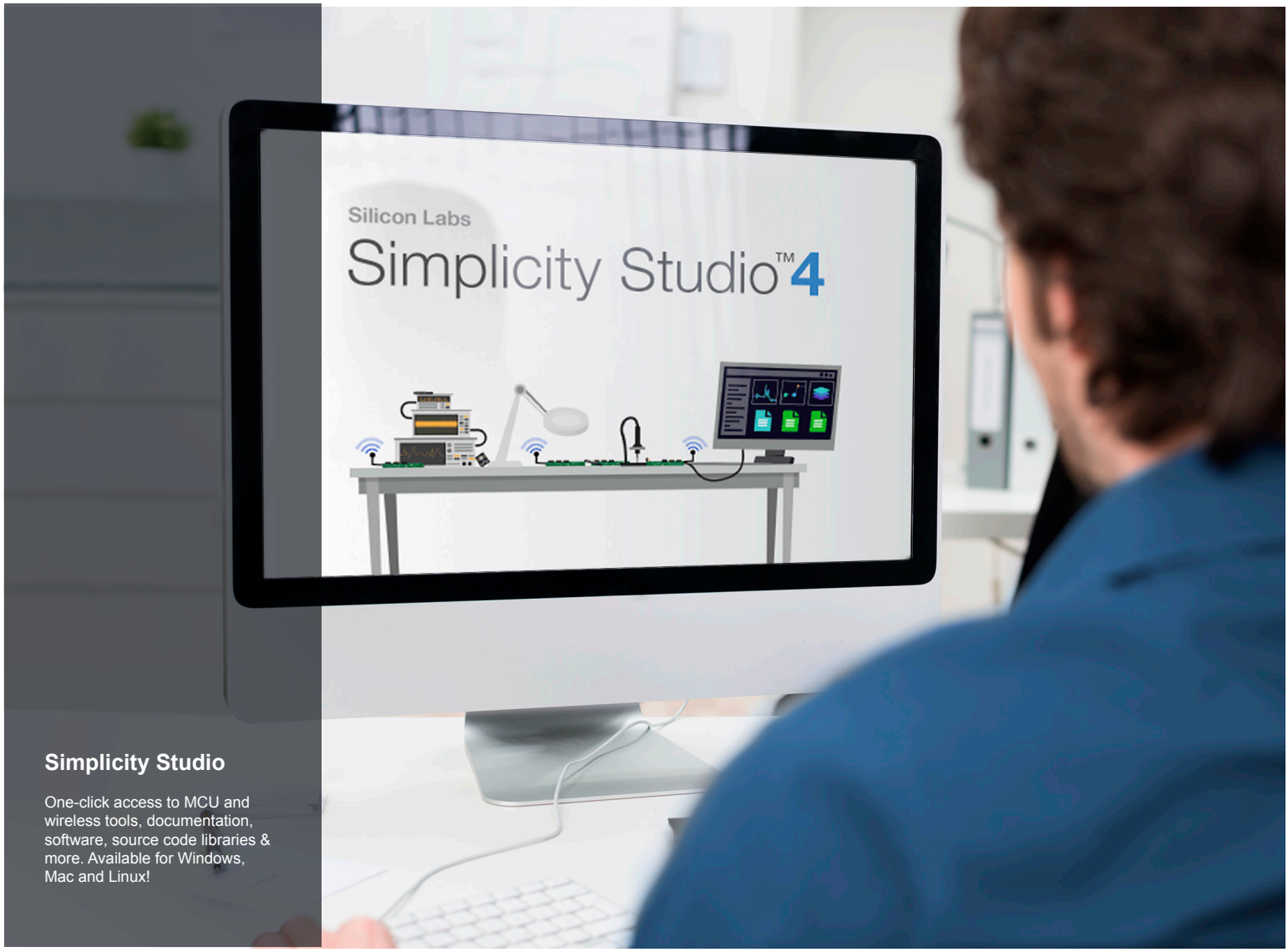
Write from the Zigbee Connected Device

The flow in the other direction, that is a change in the Light state from Zigbee connected device, is shown in the following figure.



Any on-off client on the same network as the Light can send an on-off cluster's On, Off or Toggle command to the Light to change its state. Once such a command is received over the Zigbee interface the Silicon Labs Zigbee framework interprets it and calls an appropriate handler to change the value of on-off attribute of the on-off server cluster. In the example **DynamicMultiprotocolDemoSwitch** application the on-off client sends a Toggle command to the Light, which toggles the value of the on-off attribute and triggers the `emberAfOnOffClusterServerAttributeChangedCallback()`. The callback is then used to change the state of the light as well as send notifications for both Trigger Source and Light characteristics to the connected Bluetooth devices and to update the LEDs and the LCD to indicate the change in the Light state.

```
void emberAfOnOffClusterServerAttributeChangedCallback(int8u endpoint,
                                                       EmberAfAttributeId attributeId)
{
  EmberStatus status;
  int8u data;

  if (attributeId == ZCL_ON_OFF_ATTRIBUTE_ID) {
    status =  emberAfReadAttribute(endpoint,
                                   ZCL_ON_OFF_CLUSTER_ID,
                                   ZCL_ON_OFF_ATTRIBUTE_ID,
                                   CLUSTER_MASK_SERVER,
                                   (int8u*)&data,
                                   sizeof(data),
                                   NULL);

    if (status == EMBER_ZCL_STATUS_SUCCESS) {
      if (data == 0x00) {
        halClearLed(BOARDLED0);
        halClearLed(BOARDLED1);
        dmpUiLightOff();
        notifyLight(currentConnection, 0);
      } else {
        halSetLed(BOARDLED0);
        halSetLed(BOARDLED1);
        notifyLight(currentConnection, 1);
        dmpUiLightOn();
      }
      if ( (lightDirection == DMP_UI_DIRECTION_BLUETOOTH)
           || (lightDirection == DMP_UI_DIRECTION_SWITCH) ) {
        dmpUiUpdateDirection(lightDirection);
      } else {
        lightDirection = DMP_UI_DIRECTION_ZIGBEE;
        dmpUiUpdateDirection(lightDirection);
      }
      ble_lastEvent = lightDirection;
      lightDirection = DMP_UI_DIRECTION_INVALID;

      if (ble_lastEvent != DMP_UI_DIRECTION_INVALID) {
        if    (    (ble_lightState_config    !=    GAT_RECEIVE_INDICATION)    &&    (ble_lastEvent_config    ==
GAT_RECEIVE_INDICATION)) {
          notifyTriggerSource(currentConnection, ble_lastEvent);
        }
      }
    }
  } else {
  }
}}
```



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 and Community
community.silabs.com

SILICON LABS

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

http://www.silabs.com