Chapter 7C: Bluetooth Mesh in WICED

Time 3 Hours

This chapter covers mesh code examples and mesh firmware architecture. It also covers client applications that can be used to provision and interact with mesh devices.

7C.1 Firmware Architecture

7C.1.1 Core Mesh Functionality

The main BT Mesh functionality is implemented in a set of files located in libraries/mesh_app_lib. The user does not typically need modify any of these files. The files are:

mesh_application.c	Top-level file containing application_start, stack initialization, Bluetooth management callback function, and other helper functions.
mesh_app_gatt.c	GATT database setup, GATT callback function, and all other GATT related functions.
wiced_bt_cfg.c	Bluetooth configuration including advertisement settings.
mesh_app_hci.c	Functions used for sending/receiving data to/from a host when the mesh device is used in HCI mode.
mesh_app_provision_server.c	Functions that allow the mesh device to be provisioned onto a network.

The API for the lower level core BT mesh library functions can be found in header functions that are found in the ModusToolbox installation directory at:

libraries/bt_20819A1-1.0/components/BT-SDK/common/include

Note that the easiest way to get to these files from inside the IDE are to right-click on an item such as a datatype for a structure and select "Open Declaration" (F3).

7C.1.2 User Application

The user application includes the Bluetooth and Mesh header files, sets up several variables and structures to configure mesh behavior, then registers and defines callback functions for various events that will be called from the mesh firmware. All user application functionality is handled in the various callback functions which will be discussed below.

Includes

The user application starts with at least the following includes to get access to the Bluetooth, mesh, trace, and HCI library functions:

```
#include "wiced_bt_ble.h"
#include "wiced_bt_gatt.h"
#include "wiced_bt_mesh_models.h"
#include "wiced_bt_trace.h"
#include "wiced_bt_mesh_app.h"

#ifdef HCI_CONTROL
#include "wiced_transport.h"
#include "hci_control_api.h"
#endif
```

There may be other header files included depending on the application's functionality.

Configuration Variables

After the include files, there is a set of variables that are used to configure the application. These variables configure names, models, properties, elements, and features such as relay, GATT proxy, friend, and low power.

Top Level Variables

The first set of variables set up the device name, appearance, manufacturer name, model number, etc. as shown with an example here:

```
char   *mesh_dev_name                                                    = "Dimmable Light";
uint8_t mesh_appearance[WICED_BT_MESH_PROPERTY_LEN_DEVICE_APPEARANCE]            = { BIT16_TO_8(APPEARANCE_LIGHT_CEILING) };
uint8_t mesh_mfr_name[WICED_BT_MESH_PROPERTY_LEN_DEVICE_MANUFACTURER_NAME]       = { 'C', 'y', 'p', 'r', 'e', 's', 's', 0 };
uint8_t mesh_model_num[WICED_BT_MESH_PROPERTY_LEN_DEVICE_MODEL_NUMBER]           = { 'A', '1', '9', 0 };
uint8_t mesh_prop_fw_version[WICED_BT_MESH_PROPERTY_LEN_DEVICE_FIRMWARE_REVISION] = { '0', '6', '.', '0', '2', '.', '0', '5' };
// this is overwritten during init
uint8_t mesh_system_id[8]                                          = { 0xbb, 0xb8, 0xa1, 0x80, 0x5f, 0x9f, 0x91, 0x71 };
```

Models

The next item is an array that specifies the models that will be implemented in the device. In fact, you need one of these arrays for each element in the device. So, if the device has two elements, you would have two arrays which each specify the models used in one of the two elements.

For the primary element, the array must contain WICED_BT_MESH_DEVICE. It will also contain other models to implement the required functionality of the primary element. For example, if the primary element has a user property server, the array will have an entry for WICED_BT_MESH_MODEL_USER_PROPERTY_SERVER. Secondary elements (if there are any) do not require WICED_BT_MESH_DEVICE but will contain any other models required for the element's functionality.

An example of a model array for a primary element with three models is shown here:

```
wiced_bt_mesh_core_config_model_t   mesh_element1_models[] =
{
    WICED_BT_MESH_DEVICE,
    WICED_BT_MESH_MODEL_USER_PROPERTY_SERVER,
    WICED_BT_MESH_MODEL_LIGHT_LIGHTNESS_SERVER,
};
```

Sensors

If your device implements sensor models, then you will need an array for each element containing sensors to configure each sensor. Each array has one structure entry per sensor in the given element. If your device does not implement any sensor models, then these arrays are not required and can be deleted.

An example for a configuration array for a single temperature sensor is shown here:

```
wiced_bt_mesh_core_config_sensor_t mesh_element1_sensors[] =
{
    {
        .property_id = WICED_BT_MESH_PROPERTY_PRESENT_AMBIENT_TEMPERATURE,
        .prop_value_len = WICED_BT_MESH_PROPERTY_LEN_PRESENT_AMBIENT_TEMPERATURE,
        .descriptor =
        {
            .positive_tolerance = MESH_TEMPERATURE_SENSOR_POSITIVE_TOLERANCE,
            .negative_tolerance = MESH_TEMPERATURE_SENSOR_NEGATIVE_TOLERANCE,
            .sampling_function  = MESH_TEMPERATURE_SENSOR_SAMPLING_FUNCTION,
            .measurement_period = MESH_TEMPERATURE_SENSOR_MEASUREMENT_PERIOD,
            .update_interval    = MESH_TEMPERATURE_SENSOR_UPDATE_INTERVAL,
        },
        .data = (uint8_t*)&mesh_sensor_sent_value,
        .cadence =
        {
            // Value 1 indicates that cadence does not change depending on the
measurements
            .fast_cadence_period_divisor = 1,
            .trigger_type_percentage     = WICED_FALSE,
            .trigger_delta_down          = 0,
            .trigger_delta_up            = 0,
            .min_interval                = (1 << 12), // minimum interval for sending
data by default is 4 seconds
            .fast_cadence_low            = 0,
            .fast_cadence_high           = 0,
        },
        .num_series     = 0,
        .series_columns = NULL,
        .num_settings   = 1,
        .settings       = sensor_settings,
    },
};
```

Properties

If any of the elements have properties associated with them, they are specified in an array of property structures. Each entry in the array specifies one property.

Each element in the device that contains properties has its own array to specify those properties just like with the model and sensor arrays. If there are no properties for a given element, then this array is not required and can be deleted.

An example of a property array for an element with one property (firmware revision) is shown here:

```
wiced_bt_mesh_core_config_property_t mesh_element1_properties[] =
{
    {
        .id          = WICED_BT_MESH_PROPERTY_DEVICE_FIRMWARE_REVISION,
        .type        = WICED_BT_MESH_PROPERTY_TYPE_USER,
        .user_access = WICED_BT_MESH_PROPERTY_ID_READABLE,
        .max_len     = WICED_BT_MESH_PROPERTY_LEN_DEVICE_FIRMWARE_REVISION,
        .value       = mesh_prop_fw_version
    },
};
```

Elements

This is one array that configures the elements in the device. There is one entry in the array for each element in the device. The entries define the element's location, default transition time, power up state, etc. Each entry also points to the model array, sensor array (if there is one) and property array (if there is one) for that element.

An example element array for a device with one element is shown here. Note the pointers provided to mesh_element1_properties and mesh_element1_models. In this case there is no sensor so sensors_num is set to 0 and NULL is specified for the pointer. The code example applications include comments explaining what each element means.

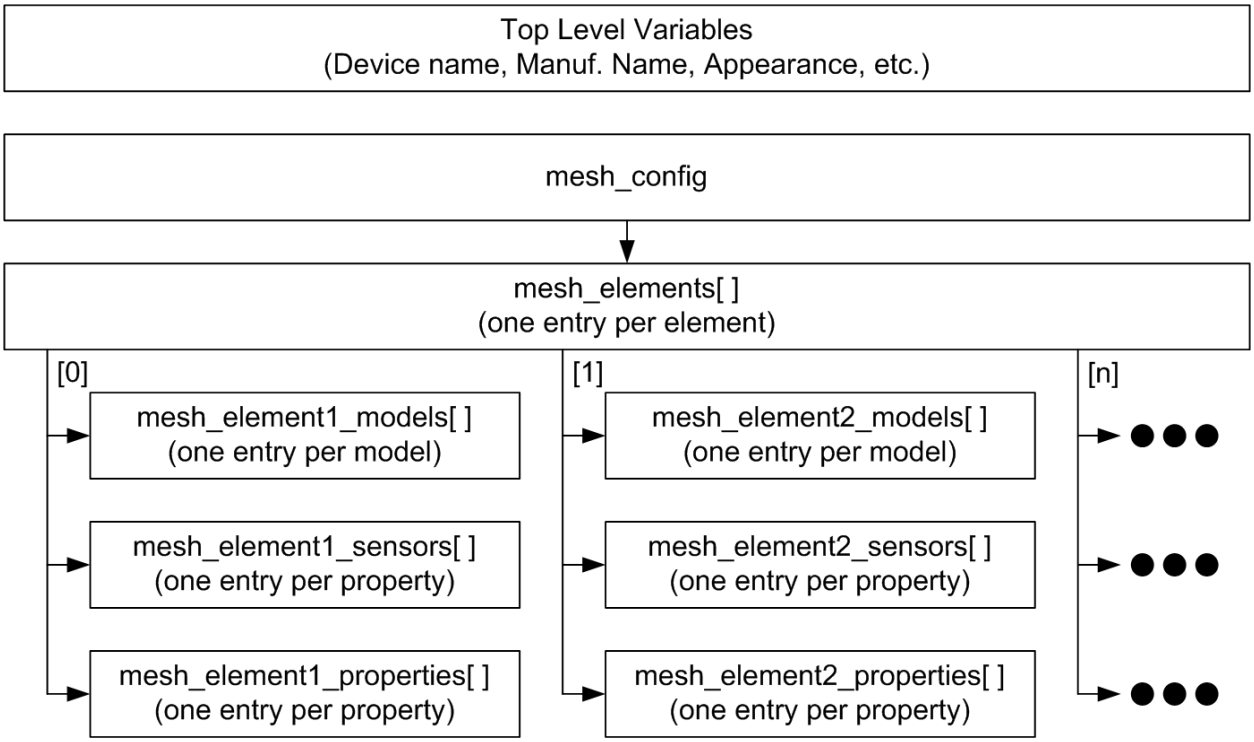
```
wiced_bt_mesh_core_config_element_t mesh_elements[] =
{
    {
        .location = MESH_ELEM_LOC_MAIN,
        .default_transition_time = MESH_DEFAULT_TRANSITION_TIME_IN_MS,
        .onpowerup_state = WICED_BT_MESH_ON_POWER_UP_STATE_RESTORE,
        .default_level = 0,
        .range_min = 1,
        .range_max = 0xffff,
        .move_rollover = 0,
        .properties_num = MESH_APP_NUM_PROPERTIES,
        .properties = mesh_element1_properties,
        .sensors_num = 0,
        .sensors = NULL,
        .models_num = MESH_APP_NUM_MODELS,
        .models = mesh_element1_models,
    },
};
```

The final structure contains information such as product ID, vendor ID, etc., as well as features such as relay, GATT proxy, friend, and low power. This structure also specifies how many replay caches are available, which limits how many other devices can send application messages to this device. If the node is a friend node or a low power node then those settings are specified in this structure. Finally, a pointer to the element array is provided.

An example mesh configuration structure is shown here. Note the pointer provided to the mesh_elements array. The code example firmware includes comments explaining what each element means.

```
wiced_bt_mesh_core_config_t  mesh_config =
{
    .company_id         = MESH_COMPANY_ID_CYPRESS,
    .product_id         = MESH_PID,
    .vendor_id          = MESH_VID,
    .replay_cache_size  = MESH_CACHE_REPLAY_SIZE,
    .features           = WICED_BT_MESH_CORE_FEATURE_BIT_FRIEND |
                          WICED_BT_MESH_CORE_FEATURE_BIT_RELAY |
                          WICED_BT_MESH_CORE_FEATURE_BIT_GATT_PROXY_SERVER,
    .friend_cfg         =
    {
        .receive_window        = 200,
        .cache_buf_len         = 300
    },
    .low_power          =
    {
        .rssi_factor           = 0,
        .receive_window_factor = 0,
        .min_cache_size_log    = 0,
        .receive_delay         = 0,
        .poll_timeout          = 0
    },
    .gatt_client_only          = WICED_FALSE,
    .elements_num  = (uint8_t)(sizeof(mesh_elements) / sizeof(mesh_elements[0])),
    .elements      = mesh_elements
};
```

The figure below illustrates how the different variables interact with each other to configure the mesh device.



User Function Callbacks

After the configuration variables, there must be a structure defined that contains a table of pointers to the functions that will be called by various mesh events. The structure looks like the following. Note that NULL can be specified for any of the functions that are not required by the user application functionality. Each of the items from the table are discussed in more detail below.

```
wiced_bt_mesh_app_func_table_t wiced_bt_mesh_app_func_table =
{
    mesh_app_init,          // application initialization
    NULL,                   // Default SDK platform button processing
    NULL,                   // GATT connection status
    mesh_app_attention,     // attention processing
    NULL,                   // notify period set
    NULL,                   // WICED HCI command
    NULL,                   // LPN sleep
    NULL                    // factory reset
};
```

Application Initialization

This function is called once the mesh stack is running and it is ready for the user application to perform any initialization that it requires. The initialization function takes one wiced_bool_t parameter called is_provisioned which is passed in by the stack. That is, the prototype for the application initialization function is:

```
typedef void(*wiced_bt_mesh_app_init_t)(wiced_bool_t is_provisioned);
```

In addition to any user initialization, the application initialization function should initialize any models, sensors, or property servers that the device requires (other than the required core mesh models). For the element model example shown in the previous section, the following would be placed in the application initialization function:

```
// Initialize Light Lightness Server and register a callback which will be executed when
// it is time to change the brightness of the bulb
wiced_bt_mesh_model_light_lightness_server_init(MESH_LIGHT_LIGHTNESS_SERVER_ELEMENT_INDEX,
       mesh_app_message_handler, is_provisioned);

// Initialize the Property Server.  We do not need to be notified when Property is set,
// because our only property is readonly
wiced_bt_mesh_model_property_server_init(MESH_LIGHT_LIGHTNESS_SERVER_ELEMENT_INDEX, NULL,
       is_provisioned);
```

The arguments to these initialization functions are:

- Element Index: This is the index of the element in the element array.
- Callback function: The function that is called when a message is received for the model (use NULL if the user application doesn't need a message callback)
- is_provisioned: This indicates whether a device has been provisioned or not. If a device has not been provisioned then configuration data for the model is deleted from NVRAM.

Button processing

The function specified here is a callback to the user application to configure button functionality. Typically, the user would configure the buttons with interrupts and then would use interrupt callback functions to process button events. The prototype for this function is:

```
typedef void(*wiced_bt_mesh_app_hardware_init_t)(void);
```

If the platform being used as the target device contains a push button but this callback is specified as NULL, the button is configured in the mesh_application.c library file to perform a factory reset on the device whenever the button is pressed and then released. Factory reset will return the device to an unprovisioned state.

GATT connection status

The function specified here is called when a GATT connection event occurs. That is, when there is a connection up or down GATT callback event. It is passed a structure with the GATT connection status of type wiced_bt_gatt_connection_status_t. That is, the prototype for the function is:

```
typedef void(*wiced_bt_mesh_app_gatt_conn_status_t)(wiced_bt_gatt_connection_status_t *p_status);
```

Attention processing

This function is called by the stack whenever it wants the application to alert the user of something. It is up to the hardware to determine how that alert should be done – e.g. blink and LED, sound a buzzer, etc. The stack provides the index of the element that should be used and a length of time for the alert in seconds from 0 (stop alert) up to 0xFF. The prototype for the alert function is:

```
typedef void (*wiced_bt_mesh_app_attention_t)(uint8_t element, uint8_t time);
```

Since the stack provides a length of time for the attention signal to occur, it is typical for the callback function to use a seconds timer to control when the attention signal stops.

Notify Period Set

Depending on the model, this function may be called to indicate to the application that it needs to periodically send updates. The function prototype is:

```
typedef wiced_bool_t (*wiced_bt_mesh_app_notify_period_set_t)(uint8_t element,
                                                              uint16_t company_id,
                                                              uint16_t model_id, uint32_t time);
```

The arguments provided by the stack are the element, company_id, model_id, and the time for periodic updates to be sent in seconds. If the model is a BT SIG defined model, the company_id will be MESH_COMPANY_ID_BT_SIG. If it is a vendor specific model, the company_id will be specific to that vendor (e.g. MESH_COMPANY_ID_CYPRESS).

If the value of time indicated by the stack is 0 then the user application should NOT send periodic updates.

The return value is a wiced_bool_t which should be returned as WICED_TRUE if the application will take care of periodic publications for the specified element, company_id, and model_id. This allows the application to handle periodic status updates for some models but not others.

It is mandatory to implement this functionality on sensors (i.e. WICED_BT_MESH_MODEL_SENSOR_SERVER). If a device does not have sensors, support for this callback is optional. If a device does not implement this callback it can just specify NULL. Applications that do not want to handle this callback should report all changes to the local states to the Mesh Models library. This will be discussed in a minute.

WICED HCI command

This function is used if the application needs to interpret HCI commands received from a host device.

```
typedef uint32_t (*wiced_bt_mesh_app_proc_rx_cmd_t)(uint16_t opcode, uint8_t *p_data,
                                                    uint32_t length);
```

As you can see, the function takes three arguments: the command opcode, a pointer to the command parameters, and the length of the command parameters.

Low Power Node (LPN) Sleep

The stack will call this function when it is safe to enter HID_OFF mode. The stack provides the length of time in milliseconds that it does not require any processing, so the application can go into HID_OFF for up to that length of time. It is up to the user application to determine whether to enter HID_OFF mode or not, and to wake up at the appropriate time. The prototype for this function is:

```
typedef void (*wiced_bt_mesh_app_lpn_sleep_t)(uint32_t duration);
```

Factory Reset

If a function is provided here, that function will be executed before a factory reset is performed. This is necessary if the user application has NVRAM data that needs to be erased when a factory reset is done. After that function returns, a factory reset will be performed to return the device to an unprovisioned state. The prototype for this function is:

```
typedef void (*wiced_bt_mesh_app_factory_reset_t)(void);
```

Sending Messages

The model initialization functions described above allow the user to specify a callback function when a message is received. On the other hand, when the application has a message that it needs to send, it must call a function to send the appropriate message. For example, a Generic OnOff client may send a message using:

```
wiced_result_t wiced_bt_mesh_model_onoff_client_set(uint8_t element_idx,
                                                    wiced_bt_mesh_onoff_set_data_t* p_data);
```

Likewise, a sensor server may send a status message using:

```
wiced_result_t wiced_bt_mesh_model_sensor_server_data(uint8_t element_idx, uint16_t property_id,
                                                      void *p_ref_data);
```

More examples of messages will be seen in the mesh demo starter applications in the next section.

Notifying the Mesh Model Library of Local Changes

In addition to sending messages over the network, it is also necessary in some cases to notify the Mesh Model Library of changes to local values. That is, if a value can be changed locally, the change should be reported so that the stack will be able to send the correct value when a client on the network requests it. For example, if a light on a light lightness server device can be turned on or off by the local hardware, it should use the following function to keep the library synchronized whenever the local value changes:

```
void wiced_bt_mesh_model_light_lc_onoff_changed(uint8_t element_idx,
                                                wiced_bt_mesh_onoff_set_data_t *p_status);
```

This is not required for sensor models because in that case the stack will call a report callback function whenever it receives a get message from a client, so it will always get the most recent sensor value before sending it out.

7C.2 MESH Demo Starter Applications

ModusToolbox contains a wealth of example apps and snips for mesh networking. There are four demo starter applications built into the Bluetooth SDK as well as dozens of online examples available on GitHub.

This section coves details about the four mesh demo starter applications and explains how they interact with each other.

7C.2.1 BLE_MESH_LightDimmable (Server)

Most of the code snippets shown as examples up to now have been from the BLE_MESH_LightDimmable application so most of the configuration is already familiar to you. This application controls an LED which can be turned on/off and can also be dimmed via a PWM.

Configuration

The configuration sets up models, properties and features using the variables and structures as described in the previous section. The models, sensors, properties and features implemented are as follows:

Models	WICED_BT_MESH_DEVICE WICED_BT_MESH_MODEL_USER_PROPERTY_SERVER WICED_BT_MESH_MODEL_LIGHT_LIGHTNESS_SERVER
Sensors	N/A
Properties	WICED_BT_MESH_PROPERTY_DEVICE_FIRMWARE_REVISION
Features	WICED_BT_MESH_CORE_FEATURE_BIT_FRIEND WICED_BT_MESH_CORE_FEATURE_BIT_RELAY WICED_BT_MESH_CORE_FEATURE_BIT_GATT_PROXY_SERVER

User Application Functionality

The top-level file for the user application is light_dimmable.c. There are helper functions to control the LED in led_control.c/h.

Two of the mesh callback functions are registered for this application: application initialization (mesh_app_init) and attention processing (mesh_app_attention).

mesh_app_init: The application initialization function does the following:

- Set the firmware version
- Initialize a 1 second timer (including specifying the timer callback function (attention_timer_cb) that will be used for attention processing
- Call a function to set up a PWM that will be used to drive the LED (led_control_init)
- Initialize the two models
    - The light lightness server model registers a message callback function (mesh_app_message_handler)
    - The property server model does not register a callback because the only property is read only so there is no action required for received messages. That is, the stack will handle sending the fixed property value when it is requested.

mesh_app_message_handler: The message handler for the light lightness server model only responds to one event - WICED_BT_MESH_LIGHT_LIGHTNESS_SET. For that event, the message handler calls the function mesh_app_process_set_level which takes the value of lightness_actual_present and converts it to a percentage brightness as an 8-bit value. The value is then sent to the function led_control_set_brighness_level which sets the PWM duty cycle to achieve the desired brightness.

mesh_app_attention: The attention callback starts the timer (or stops it if the attention time is 0). Every 1 second, the timer callback is called, and it does whatever is necessary including decrementing the remaining time.

Button Processing: Since no callback function is provided for button processing, the default behavior is used. That is, if the target kit has a button defined in its platform, pressing and releasing the button will result in a factory reset that removes all mesh provisioning information from the device.

7C.2.2 BLE_MESH_OnOffSwitch (Client)

This application demonstrates a basic on/off client. It can be used in conjunction with a kit that is running the BLE_MESH_LightDimmable application so that it can control the LED on that kit over the mesh network.

Configuration

The configuration sets up models, properties and features using the variables and structures as described in the previous section. The models, sensors, properties and features implemented are as follows:

Models	WICED_BT_MESH_DEVICE WICED_BT_MESH_MODEL_ONOFF_CLIENT
Sensors	N/A
Properties	N/A
Features	WICED_BT_MESH_CORE_FEATURE_BIT_LOW_POWER

User Application Functionality

The top-level file for the user application is on_off_switch.c.

Two of the mesh callback functions are registered for this application: application initialization (mesh_app_init) and button processing (mesh_app_hardware_init).

mesh_app_init: The application initialization function in this case only needs to do one thing. That is, it initializes the onoff client model. The onoff client model has no callback function because it does not check the result of transmission messages or status messages.

mesh_app_hardware_init: This function sets up the default button on the kit to have an interrupt on both edges with the callback function button_interrupt_handler. It also stores the default state of the button to a global variable. All other application functionality is handled by the interrupt callback.

button_interrupt_handler: This function does the following:

- Check if the button state has changed. If it has:
  - If the button is pressed, increment a counter
  - If the button is released, calculate the time that it was pressed:
    - If the button was pressed more than 15 seconds, do a factory reset.
    - Otherwise, call the process_button_push function.

process_button_push: This function sends a mesh message to toggle the on/off state of whatever device(s) it is controlling. It uses the default transition time and a delay of 0.

7C.2.3 BLE_MESH_Dimmer (Client)

This application demonstrates a level client. It can be used in conjunction with a kit that is running the BLE_MESH_LightDimmable application so that it can control both on/off and dimming of the LED on that kit over the mesh network.

Configuration

The configuration sets up models, properties and features using the variables and structures as described in the previous section. The models, sensors, properties and features implemented are as follows:

Models	WICED_BT_MESH_DEVICE WICED_BT_MESH_MODEL_LEVEL_CLIENT
Sensors	N/A
Properties	N/A
Features	WICED_BT_MESH_CORE_FEATURE_BIT_LOW_POWER

User Application Functionality

The top-level file for the user application is dimmer.c. There are helper functions to handle the button in button_control.c/h

Two of the mesh callback functions are registered for this application: application initialization (mesh_app_init) and button processing (button_hardware_init).

mesh_app_init: The application initialization function calls the helper function button_control_init and then initializes the level client model. The model has a callback function called mesh_app_message_handler that is used to act on messages that come back from the server.

mesh_app_message_handler: The message callback function for the level client handles two events:

> **WICED_BT_MESH_TX_COMPLETE**: This event occurs when the light lightness model responds that a transmission is complete.
>
> **WICED_BT_MESH_LEVEL_STATUS**: This event occurs when a status message is received from the light lightness model. The callback function prints out the current level, the target level, and the time remaining in the transition.

button_control_init: This function initializes a timer and registers the callback function button_timer_callback. It then saves the current button state.

button_hardware_init: This function just sets up the default button on the kit to have an interrupt on both edges with the callback function button_interrupt_handler. All other application functionality is handled by the interrupt callback.

button_interrupt_handler: This function does the following:

- If the button state has not changed, just return.

- If the button is pressed, start a 500 ms timer.
- If the button was released after being held down for less than 500 ms, call button_set_level to send a message to toggle the LED on or off.
- If the button has been held down for more than 500ms but less than 15 seconds, call button_set_level to send a final message to adjust the brightness of the LED.

button_timer_callback: The 500 ms timer callback calculates the next brightness value for the LED and calls button_set_level to send a message to the server to adjust the brightness of the LED .

button_set_level: This function sends a level message to the server using wiced_bt_mesh_model_level_client_set.

7C.2.4 BLE_Mesh_SensorTemperature

This application demonstrates a sensor server. In this case, the sensor measures temperature. It sends status messages to report values on a periodic basis.

Configuration

The configuration sets up models, properties and features using the variables and structures as described in the previous section. The models, sensors, properties and features implemented are as follows:

Models	WICED_BT_MESH_DEVICE WICED_BT_MESH_MODEL_SENSOR_SERVER
Sensors	WICED_BT_MESH_PROPERTY_PRESENT_AMBIENT_TEMPERATURE
Properties	N/A
Features	WICED_BT_MESH_CORE_FEATURE_BIT_LOW_POWER

User Application Functionality

The top-level file for the user application is sensor_ temperature.c.

Four of the mesh callback functions are registered for this application:

- Application initialization (mesh_app_init)
- Notify period set (mesh_app_notify_period_set).
- Low power node sleep (mesh_app_lpn_sleep)
- Factory reset (mesh_app_factory_reset)

mesh_app_init: The application initialization function:

- Sets up a pointer to the sensor configuration structure
- Sets the FW version
- If the device isn't provisioned yet, that's all it does.
- If the device is provisioned, it:
    - Calls thermistor_init to initialize the ADC
    - Reads the temperature using mesh_sensor_get_temperature_8
    - Initializes a timer and registers the callback function **mesh_sensor_publish_timer_callback**
    - Reads the stored sensor cadence value from NVRAM
    - Initializes the sensor server and registers two callback functions – one for the sensor report handler and one for the configuration change handler
        - **mesh_sensor_server_report_handler**
        - **mesh_sensor_server_config_change_handler**
    - Sends the initial temperature value by calling **wiced_bt_mesh_model_sensor_server_data**

mesh_sensor_publish_timer_callback: The timer callback function is used to send periodic temperature updates. It uses the cadence value set by the client to determine how often to send an update.

mesh_sensor_server_report_handler: This is the sensor server callback function for report messages. It only implements the case **WICED_BT_MESH_SENSOR_GET**. For that case, it gets the temperature value using **mesh_sensor_get_temperature_8** and sends it to the client using **wiced_bt_mesh_model_sensor_server_data**.

mesh_sensor_server_config_change_handler: This is the sensor server callback function for configuration change messages. It implements two cases:

> **WICED_BT_MESH_SENSOR_CADENCE_SET:** This event occurs when the client requests a change to the sensor cadence. For this case, the user application calls **mesh_sensor_server_process_cadence_changed**. That function updates the cadence, stores its value in NVRAM, and then restarts the publish timer so that cadence changes take effect.
>
> **WICED_BT_MESH_SENSOR_SETTING_SET:** This event occurs when the client requests a change to the sensor settings. For this case, the user application calls **mesh_sensor_server_process_setting_changed** which just prints a UART message that the change has been received.

mesh_app_notify_period_set: This function is called when the client requests a notify period change. The function first makes sure that the request is for the correct element (MESH_TEMPERATURE_SENSOR_INDEX), company ID (MESH_COMPANY_ID_BT_SIG), and model (WICED_BT_MESH_CORE_MODEL_ID_SENSOR_SRV). If it is, the publish period is updated and the timer is restarted.

mesh_app_lpn_sleep: This function calls **wiced_sleep_enter_hid_off** with the timeout period that is provided by the mesh stack so that the device wakes up at the appropriate time.

mesh_app_factory_reset: This function deletes the cadence information from NVRAM before the factory reset occurs.

7C.3 Client Applications

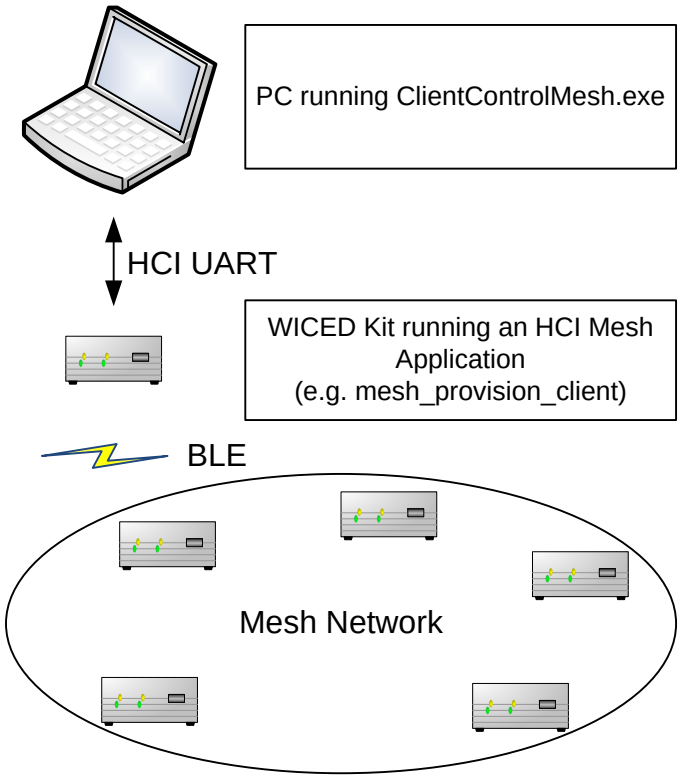
To provision devices on a mesh network and to interact with mesh devices, client applications are required. There are two Windows PC applications and one Android application provided by Cypress for these purposes. There are also 3rd party lighting applications that can be used on Android.

7C.3.1 Client Control Mesh (Windows)

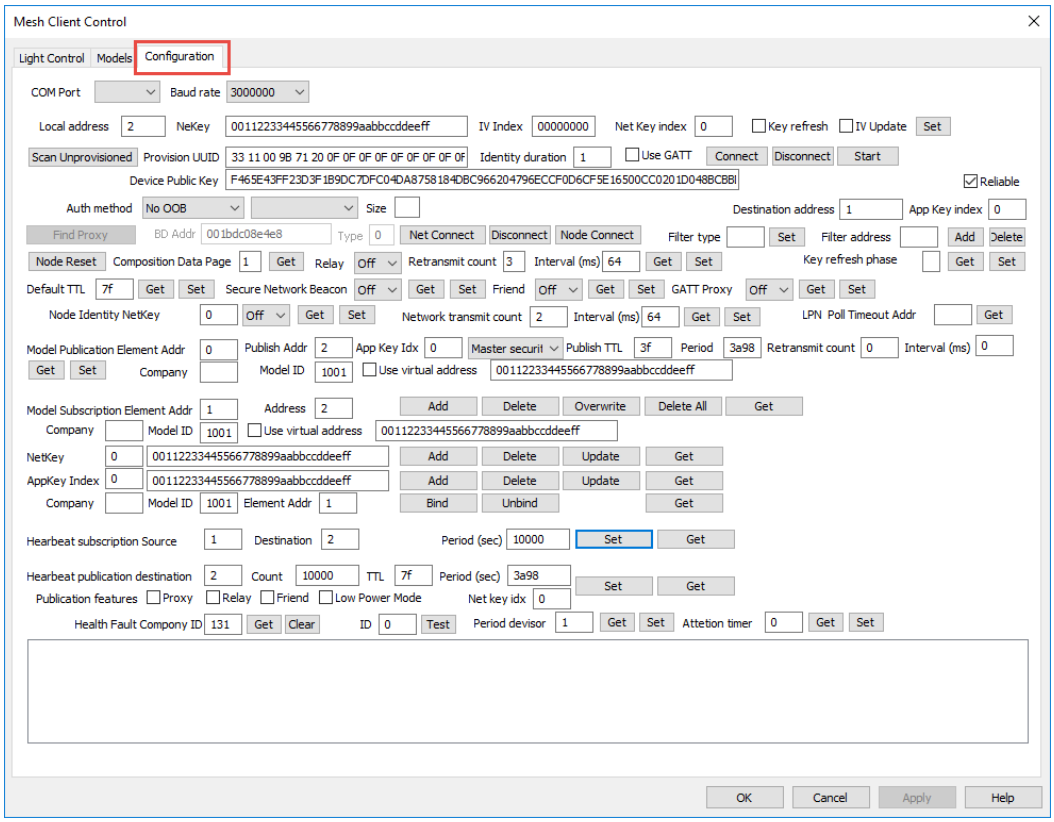
This application is an extension of the Client Control application that can be used for other WICED Bluetooth applications. It is located at:

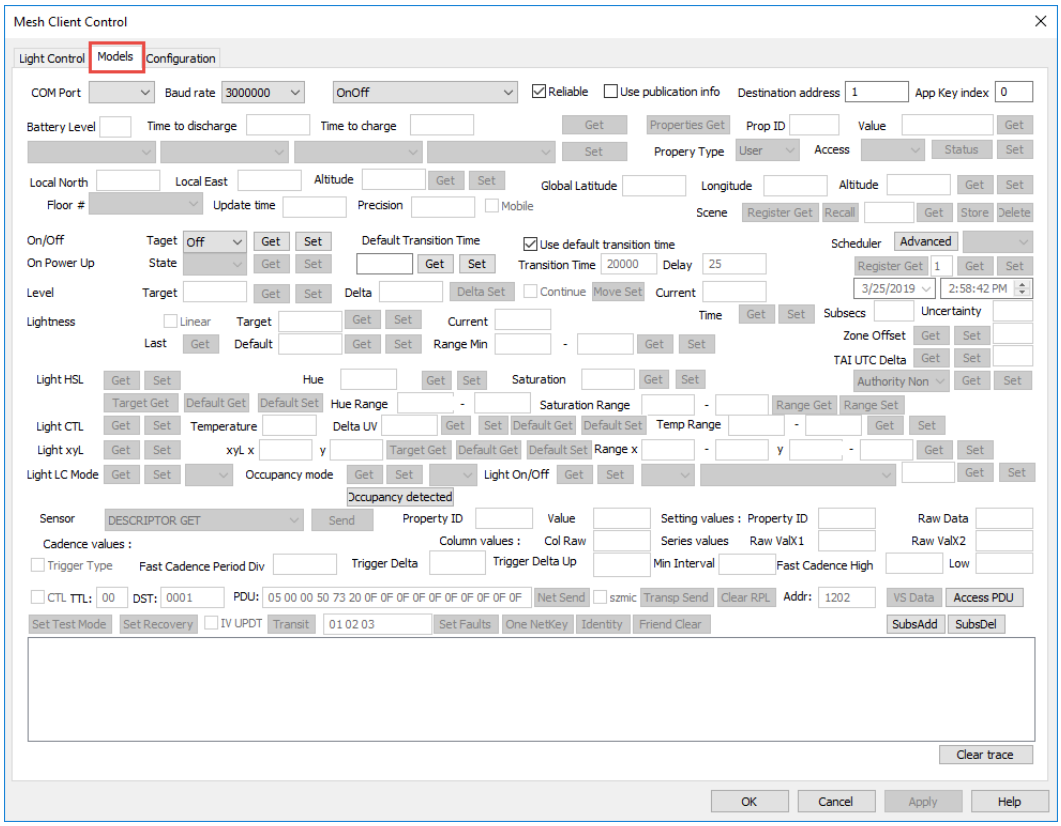
> <Install Folder>/ModusToolbox_1.1/libraries/bt_20819A1-1.0/components/BT-SDK/common/apps/snip/mesh/ClientControl/Release/ClientControlMesh.exe

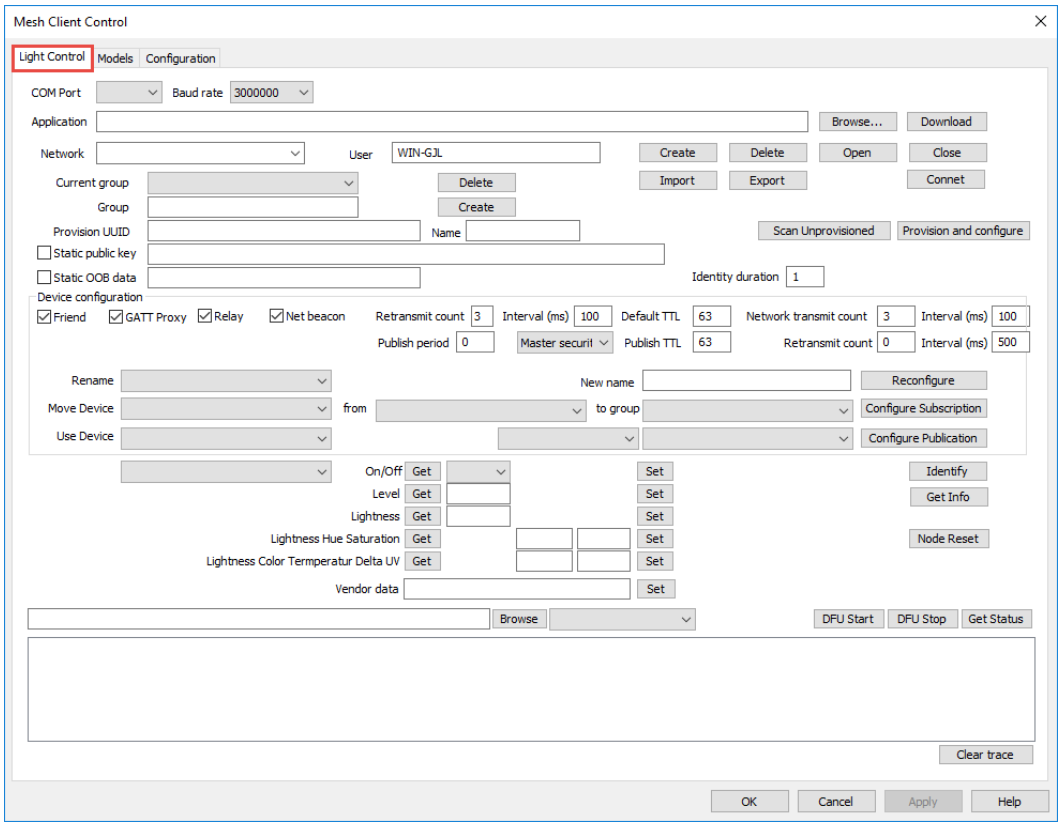
To use it, you must first program a kit with an application that does mesh operations such as provisioning, onoff client, etc. For example, the mesh_provision_client application supports a provisioning client, onoff client, level client, light lightness client, etc. That kit will receive HCI commands from the PC over UART and in turn sends the appropriate messages to the mesh network.



The application has tabs for Configuration (provisioning, etc.), Models (for interacting with any model), and Light Control (specific for lighting applications). It allows very low-level interaction with the mesh network as you can see in the screenshots below.



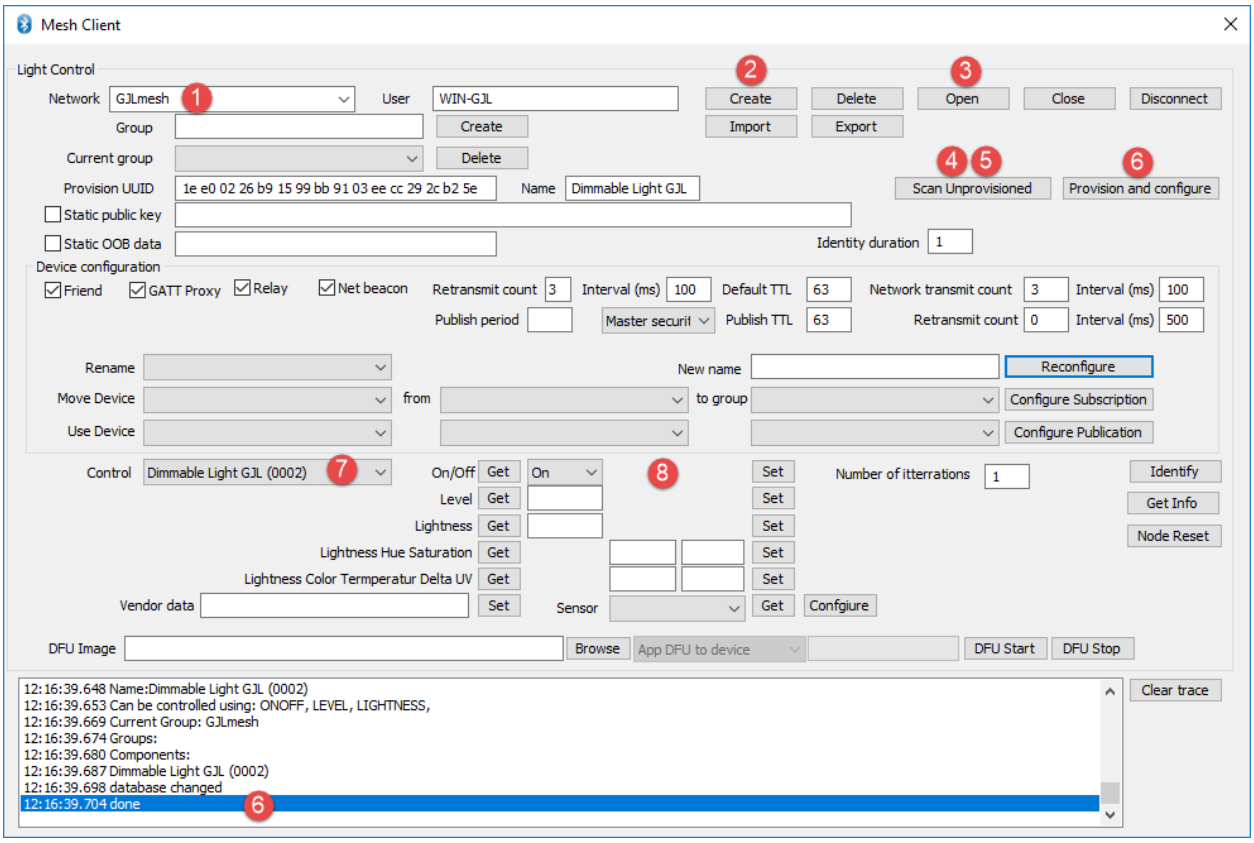

Mesh Client Control
Light Control
Models
Configuration
COM Port
Baud rate
3000000
OnOff
Reliable
Use publication info
Destination address
App Key index
Battery Level
Time to discharge
Time to charge
Prop ID
Value
Propery Type
User
Access
Status
Local North
Local East
Altitude
Global Latitude
Longitude
Floor #
Update time
Precision
Mobile
Scene
Register Get
Recall
Store
Delete
On/Off
Taget
Off
On Power Up
State
Default Transition Time
Use default transition time
Transition Time
20000
Delay
25
Scheduler
Advanced
Level
Target
Delta
Delta Set
Continue
Move Set
Current
3/25/2019
2:58:42 PM
Lightness
Linear
Time
Subsecs
Uncertainty
Last
Default
Range Min
Zone Offset
TAI UTC Delta
Light HSL
Hue
Saturation
Authority Non
Target Get
Default Get
Default Set
Hue Range
Saturation Range
Range Get
Range Set
Light CTL
Temperature
Delta UV
Temp Range
Light xyL
xyL x
y
Range x
Light LC Mode
Occupancy mode
Light On/Off
Occupancy detected
Sensor
DESCRIPTOR GET
Send
Property ID
Setting values : Property ID
Raw Data
Cadence values :
Column values :
Col Raw
Series values
Raw ValX1
Raw ValX2
Trigger Type
Fast Cadence Period Div
Trigger Delta
Trigger Delta Up
Min Interval
Fast Cadence High
Low
CTL TTL:
00
DST:
0001
PDU:
05 00 00 50 73 20 0F 0F 0F 0F 0F 0F 0F 0F 0F
Net Send
szmic
Transp Send
Clear RPL
Addr:
1202
VS Data
Access PDU
Set Test Mode
Set Recovery
IV UPDT
Transit
01 02 03
Set Faults
One NetKey
Identity
Friend Clear
SubsAdd
SubsDel
Clear trace
OK
Cancel
Apply
Help


Mesh Client Control
Light Control
Models
Configuration
COM Port
Baud rate
3000000
Application
Browse...
Download
Network
User
WIN-GJL
Create
Delete
Open
Close
Import
Export
Connet
Current group
Group
Provision UUID
Name
Scan Unprovisioned
Provision and configure
Static public key
Static OOB data
Identity duration
1
Device configuration
Friend
GATT Proxy
Relay
Net beacon
Retransmit count
Interval (ms)
Default TTL
Network transmit count
Publish period
Master securit
Publish TTL
Rename
New name
Reconfigure
Move Device
from
to group
Configure Subscription
Use Device
Configure Publication
On/Off
Level
Lightness
Lightness Hue Saturation
Lightness Color Termperatur Delta UV
Vendor data
Identify
Get Info
Node Reset
Browse
DFU Start
DFU Stop
Get Status
Clear trace
OK
Cancel
Apply
Help

7C.3.2 Mesh Client (Windows)

This application communicates with the specified mesh network directly using the BLE radio of the computer. Note that support for BLE was added in Windows 10 so you can't use this with earlier versions of Windows. It can create mesh networks, provision devices and can control lighting devices. It is located at:

<Install Folder>/ModusToolbox_1.1/libraries/bt_20819A1-1.0/components/BT-SDK/common/apps/snip/mesh/peerapps/Windows/MeshClient/Release/x8x/MeshClient.exe



The basic flow for using the application is:

1. Enter a name for your network
2. Click *Create*
3. Click *Open*
4. Click *Scan Unprovisioned*
5. Wait until your device appears in the list and click *Stop Scanning*
    a. If there are multiple unprovisioned devices you may need to stop and restart multiple times until you see the device you are looking for.
6. Click *Provision and configure*
    a. This step will take a few seconds – wait until it is complete before continuing.
7. Select your device in the *Control* dropdown.
8. Use *On/Off Get*, *On/Off Set*, *Level Get*, *Level Set*, etc. to control your device.

7C.3.3 Mesh Lighting Controller (Android)

This app communicates with the mesh network directly using the phone's BLE capabilities. It is provided by Cypress as an example application. Source code is available for customers to use in creating their own apps if they desire.

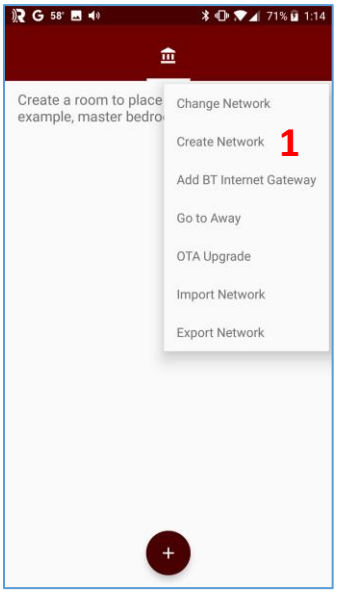
The app can create mesh networks, provision devices and can control lighting devices. The installable file is located at:

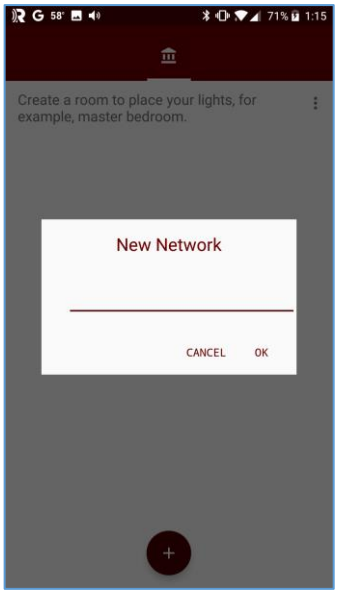
> <Install Folder>/ModusToolbox_1.1/libraries/bt_20819A1-1.0/components/BT-SDK/common/apps/snip/mesh/peerapps/Android/src/bin/MeshLIghtingController.apk

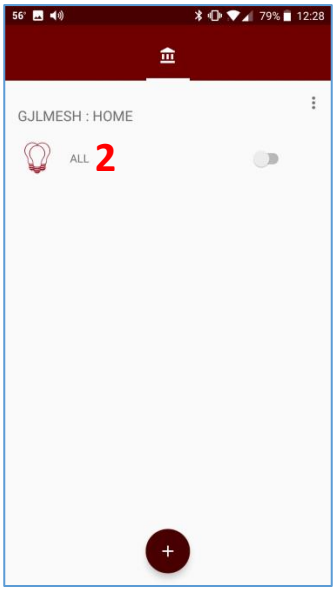
Since this is not on the Android Play Store, it is necessary to install it manually by dragging the .apk file onto the phone's filesystem and then executing it to install the app. You will need to allow installation of 3rd party applications for this to work.

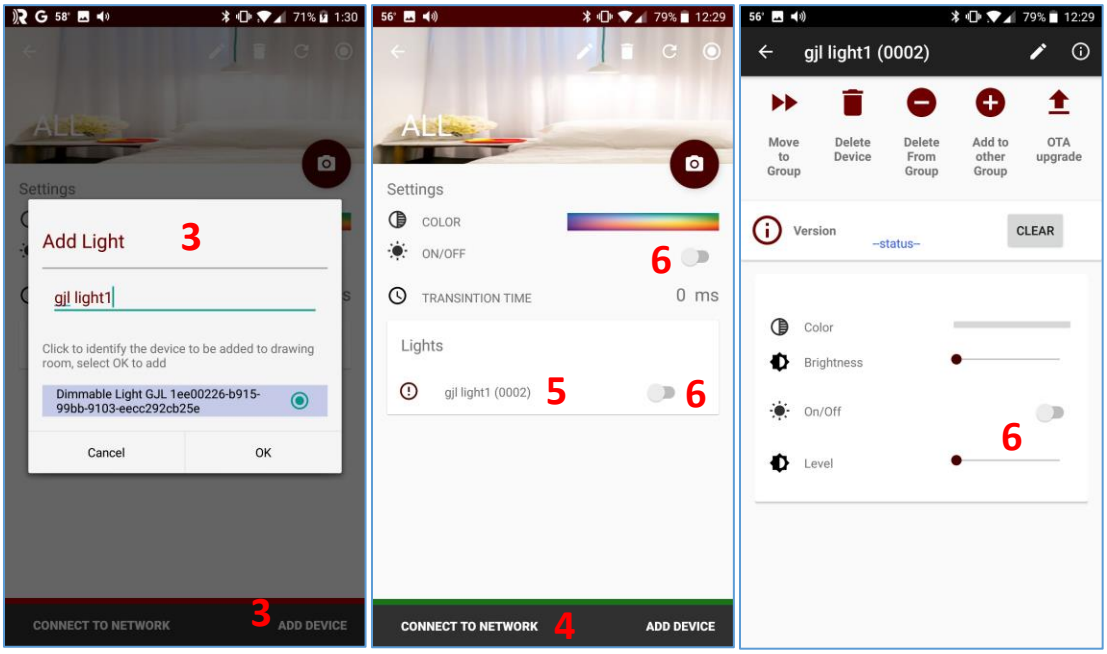
The basic flow for using the application is:

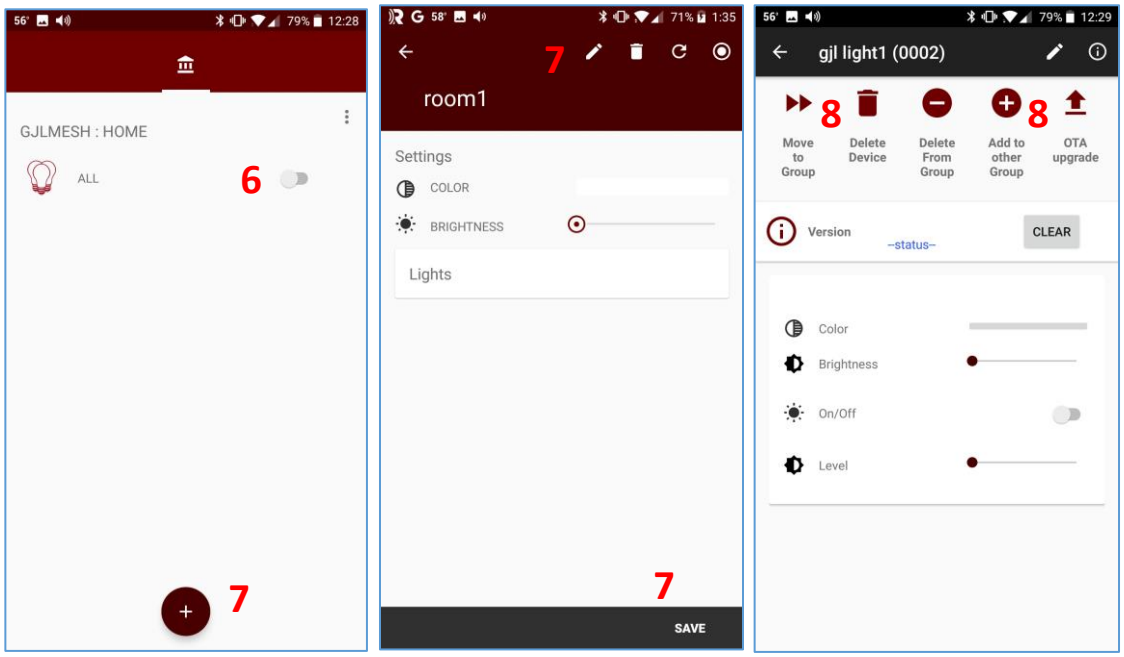
1. Create a network
2. Select a group (ALL is created by default)
3. Add a device to the group
    a. This will take a few seconds. Wait until it completes before proceeding
4. Connect to the network if it doesn't happen automatically (bar will be green when connected)
5. Select the device
6. Control the device
    a. Note: you can control all devices simultaneously at the group level or individually at the device level
7. Optional: Add additional Groups (i.e. Rooms)
8. Optional: Move or Add devices to other Groups








Settings
Add Light
3
gjl light1
Click to identify the device to be added to drawing room, select OK to add
Dimmable Light GJL 1ee00226-b915-99bb-9103-eecc292cb25e
Cancel
OK
CONNECT TO NETWORK
3
ADD DEVICE
Settings
COLOR
ON/OFF
6
TRANSINTION TIME
0 ms
Lights
gjl light1 (0002)
5
6
CONNECT TO NETWORK
4
ADD DEVICE
gjl light1 (0002)
Move to Group
Delete Device
Delete From Group
Add to other Group
OTA upgrade
Version
--status--
CLEAR
Color
Brightness
On/Off
6
Level


GJLMESH : HOME
ALL
6
7
room1
7
Settings
COLOR
BRIGHTNESS
Lights
7
SAVE
gjl light1 (0002)
8
8
Move to Group
Delete Device
Delete From Group
Add to other Group
OTA upgrade
Version
--status--
CLEAR
Color
Brightness
On/Off
Level

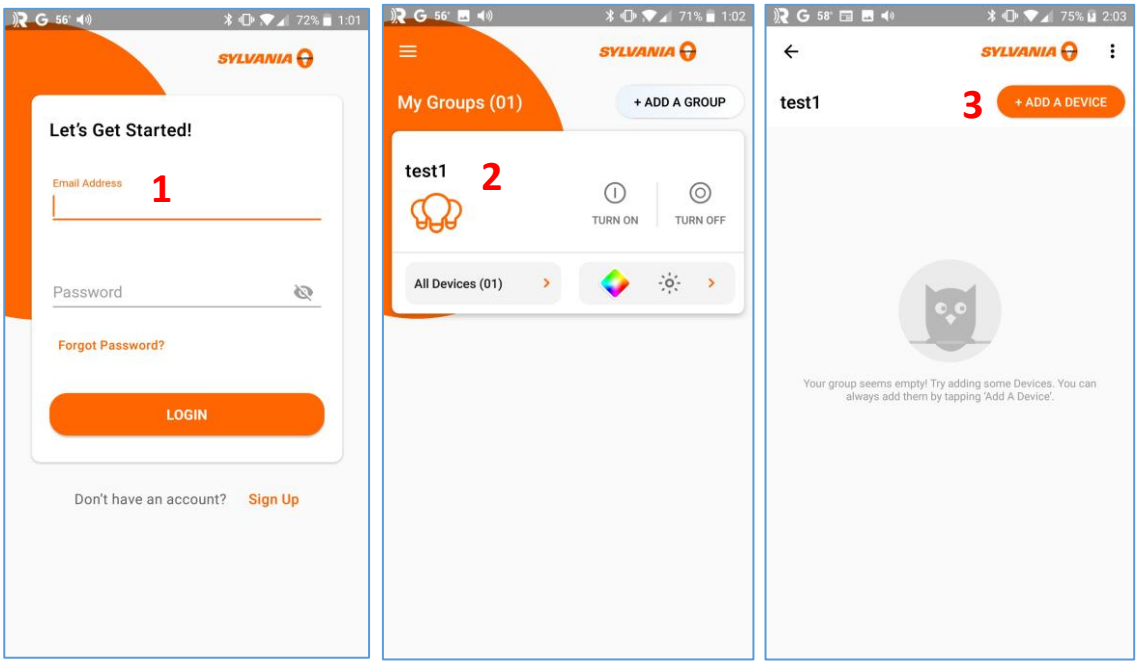
7C.3.4 Sylvania Smart Home by LedVance (Android)

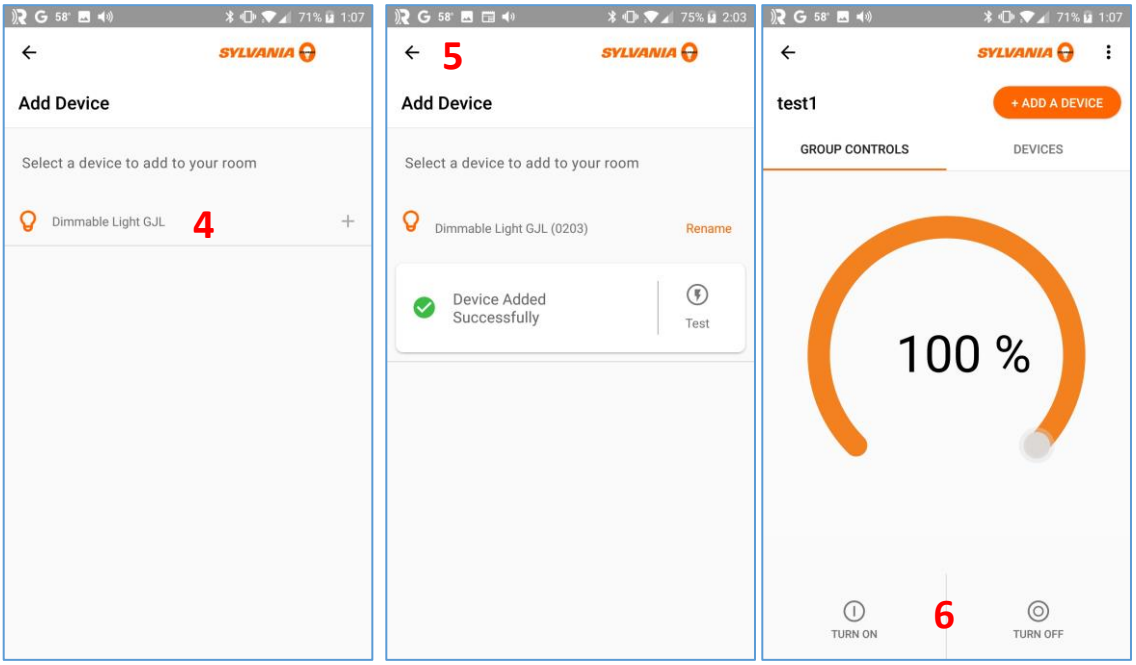
The Sylvania Smart Home application created by LedVance is available for Android from the Google Play Store. It can create mesh networks, provision devices and can control lighting devices. Note: there is a version of the app for iOS, but it is based on the Apple HomeKit solution, so it won't work with our starter applications.

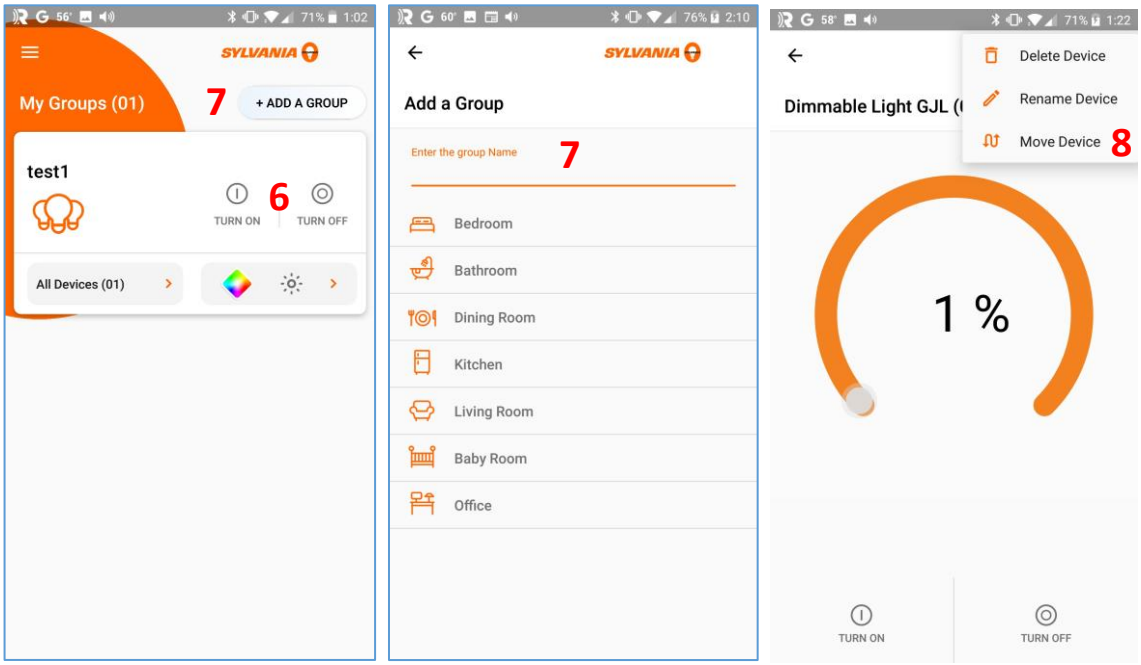
The basic flow for using the application is:

1. Register for an account
2. Select the *test1* group
3. Click *ADD A DEVICE*
4. Select your device from the list
    a. This will take a few seconds. Wait until it completes before proceeding
5. Go back to the test 1 group page
6. Control the device
    a. Note: you can control all devices simultaneously at the group level or individually at the device level
7. Optional: Add additional Groups (i.e. Rooms) from the My Groups screen.
    a. You can choose from a predefined list of rooms or enter your own name.
8. Optional: Move devices to other Groups

Note: Do NOT use the dimmer controls in the app since it is not compatible with the demo firmware. If you try dimming from the Sylvania Smart Home app, you will need to factory reset and re-provision your kit to get it to control the LED again.




Add Device
Select a device to add to your room
Dimmable Light GJL
4
5
Dimmable Light GJL (0203)
Rename
Device Added Successfully
Test
test1
+ ADD A DEVICE
GROUP CONTROLS
DEVICES
100 %
TURN ON
6
TURN OFF


My Groups (01)
7
+ ADD A GROUP
test1
TURN ON
6
TURN OFF
All Devices (01)
Add a Group
Enter the group Name
7
Bedroom
Bathroom
Dining Room
Kitchen
Living Room
Baby Room
Office
Delete Device
Rename Device
Move Device
8
1 %
TURN ON
TURN OFF

7C.4 (Advanced) OTA

The mesh library GATT definition includes the OTA upgrade service by default. Therefore, it is possible to update the firmware over BLE on an unprovisioned mesh device.

7C.5 (Advanced) Apple HomeKit

Apple HomeKit uses a completely different protocol. It is (currently) not based on Bluetooth Mesh but rather on WiFi and BLE. The Cypress mesh library will (in the future) allow devices that support mesh to also support HomeKit in the following ways:

1. (Short Term): Allow a single device to advertise as both an unprovisioned mesh device and an unpaired HomeKit device. The user can then choose to use it as a mesh device or a HomeKit device but not both at the same time.
2. (Longer Term): Allow a single device to be controlled simultaneously by Mesh clients and HomeKit clients.

7C.6 Exercises

Exercise 7C.1 Create Network with a LightDimmable Device

In this exercise you will create your own (very small) mesh network.

1. Perform a factory reset (i.e. remove provisioning and configuration information) on your mesh kit with the LightDimmable application programmed into it from the chapter 7A exercise. You can do this by pressing and releasing the user button on the kit. Alternately, you can reprogram LightDimmable onto the kit again.
2. Run a Windows or Android application to provision the device.
    a. Hint: The method of scanning for unprovisioned devices in the Windows app is not conducive to finding one device out of many, so it is probably not the best choice for this exercise.
3. Use the app to turn the LED on/off.
    a. Hint: if you are using the Windows or Cypress Android app, you can also try dimming the LED. If you are using the Sylvania Smart Home app, do NOT try dimming since it is not compatible with the demo firmware. If you try dimming from the Sylvania Smart Home app, you will need to factory reset and re-provision your kit to get it to control the LED again.

Exercise 7C.2 Add an OnOff Switch to the Network

In this exercise you will add a second device to your mesh network. This new device will be an on/off switch that can control the LED on the LightDimmable kit.

1. Create a new application for:
    a. Target Hardware: CYBT-213043-MESH
    b. Starter Application: BLE_Mesh_OnOffSwitch
2. Open the file "on_off_switch.c" and find the "mesh_dev_name". Change the name so that it has your initials in it (e.g. "<Inits> Switch")
3. Program the project into a second mesh kit.
    a. Hint: Unplug the kit (or move it to another power source) with LightDimmable programmed onto it first to make sure you don't overwrite the firmware in that kit.
    b. Hint: You may want to label the kits to keep track of which one is programmed with each project. Remember if you accidentally press the user button on the LightDimmable kit, it will perform a factory reset.
4. Once programming is done, plug in the LightDimmable kit.
5. Provision the OnOff Switch kit to the same network as the previous exercise.
6. Press the user button on the OnOff Switch kit to toggle the LED on the LightDimmable kit.
7. Use the app to toggle the LED on the LightDimmable kit.

Exercise 7C.3 Add a Dimmer to the Network

In this exercise you will add a dimmer device to your mesh network. This new device will be able to turn the LED on/off as well as control the brightness of the LED on the LightDimmable kit. Note that the OnOff Switch from the previous exercise will be able to control the same LED.

1. Create a new application for:
    a. Target Hardware: CYBT-213043-MESH
    b. Starter Application: BLE_Mesh_Dimmer
2. Open the file "dimmer.c" and find the "mesh_dev_name". Change the name so that it has your initials in it (e.g. "<Inits> Dimmer")
3. Program the project into a third mesh kit.
    a. Hint: Unplug the kits (or move them to another power source) with LightDimmable and Switch programmed onto them first to make sure you don't overwrite the firmware in those kits.
    b. Hint: You may want to label the kits to keep track of which one is programmed with each project. Remember if you accidentally press the user button on the LightDimmable kit, it will perform a factory reset.
4. Once programming is done, plug in the LightDimmable and Switch kits.
5. Provision the Dimmer kit to the same network as the previous exercises.
6. Press the user button on the Dimmer kit to toggle the LED on the LightDimmable kit.
7. Press and hold the user button on the Dimmer kit to adjust the brightness of the LED.
    a. Hint: If you hold the button for longer than 15 seconds a factory reset will be performed and the Dimmer kit will no longer be associated with the mesh network.
8. Verify that the OnOff switch kit and the app can still control the LED.

Exercise 7C.4 Add a second LightDimmable to the Network and Create/Modify Groups

In this exercise you will add a second light to the network. You will experiment with associating devices to different groups.

1. Program your LightDimmable application into a 4th mesh kit.
    a. Hint: Unplug all other kits first or move them to an alternate power source.
2. Provision the new LightDimmable kit.
3. Optional: Rename the new kit in the app so that you can distinguish it from the other LightDimmable kit.
4. Control both LightDimmable kits with the OnOff switch and Dimmer.
5. Move one LigthDimmable kit and the OnOff switch to one group and the other LightDimmable kit and the Dimmer to another group.
6. Notice how you can now independently control each LightDimmable kit.

Exercise 7C.5 (Advanced) Add a 2nd Element for the Green LED to LightDimmable

In this exercise, you will add a new element to the LightDimmable application so that you can control the Red and Green LEDs on the kit individually. To do this:

1. Use the app to remove the LightDimmable device from your network that you are going to reprogram.
2. Create a new application:
    a. Target Hardware: CYBT-213043-MESH
    b. Starter Application: BLE_Mesh_LightDimmable
    c. Application Name: BLE_Mesh_LightDimmable_ch7c_ex05
3. In the light_dimmable.c file:
    a. Add another element to the design. This new element will have one WICED_BT_MESH_MODEL_LIGHT_LIGHTNESS_SERVER model and no properties.
        i. Hint: You will need to create the mesh_element2_models array and add a set of entries to the mesh_elements array for the new element.
    b. In the mesh_app_init function, initialize the new light lightness server.
        i. Hint: You can use the same callback for all three light lightness servers since it is passed the element index when it is called.
4. In led_control.c:
    a. Add a define for another PWM channel and add code to led_control_init to initialize the new PWM.
        i. Hint: use PWM1.
        ii. Hint: you can use the same pwm_config structure for both PWMs – just call wiced_hal_gpio_select_function and wiced_hal_pwm_start two times each – once for each PWM.
    b. Add an additional parameter to the function led_control_set_brighness_level so that it knows which element a message is intended for.
        i. Hint: unit8_t element_idx.
        ii. Hint: remember to update the function prototype in led_control.h too.
    c. Update the led_control_set_brighness_level function so that it looks at the element_idx input and updates the appropriate PWM.
5. Back in light_dimmable.c:
    a. Search for calls to led_control_set_brighness_level (yes, the 't' is missing in brightness) and add the element_idx parameter.
        i. Hint: The timer callback function (attention_timer_cb) doesn't have access to element_idx, but when you init the timer you can set it to pass a uint32_t to the callback. Therefore, if you set up a global uint32_t and assign it to element_idx just before starting the timer you can pass that value to the timer callback.
        ii. Hint: If you don't want to deal with the above, you can just hard code the element_idx to 0 in mesh_app_attention and attention_timer_callback.
6. Program your kit.
7. Provision your device onto your network.

a. Hint: You should see 2 devices show up instead of just one.

8. Control each of the LEDs from the app individually using the two devices and together by using the group on/off control.
    a. Note that the OnOff Switch and Dimmer control both LEDs simultaneously because they control everything in the group at once.

Exercise 7C.6 (Advanced) Update LightDimmable to use the HSL Model

In this exercise, we will change the light lightness model to the Light HSL model. This model extends the light lightness model by adding the ability to control Hue and Saturation. We will use all 3 LEDs in the RGB LED for this exercise.

1. Use the app to remove the LightDimmable device from your network that you are going to reprogram.
2. Create a new application:
    a. Target Hardware: CYBT-213043-MESH
    b. Starter Application: BLE_Mesh_LightDimmable
    c. Application Name: BLE_Mesh_LightDimmable_ch7c_ex05
3. In the light_dimmable.c file:
    a. Remove the mesh_app_attention callback functionality to simplify the changes required.
    b. Change the model from WICED_BT_MESH_MODEL_LIGHT_LIGHTNESS_SERVER to WICED_BT_MESH_MODEL_LIGHT_HSL_SERVER in the appropriate places.
    c. Add 2 additional elements each with one model. The required models are:
        i. WICED_BT_MESH_MODEL_LIGHT_HSL_HUE_SERVER
        ii. WICED_BT_MESH_MODEL_LIGHT_HSL_SATURATION_SERVER
    d. Change the wiced_bt_mesh_model_lightness_server_init function call to wiced_bt_mesh_model_light_hsl_server_init.
    e. In the message handler, change the WICED_BT_MESH_LIGHT_LIGHTNESS_SET event to WICED_BT_MESH_LIGHT_HSL_SET.
    f. Change the function mesh_app_process_set_level to mesh_app_process_set_hsl and make the changes to get the Hue, Saturation and Lightness values from p_status.
        i. Hint: the data provided in p_status will be of type wiced_bt_mesh_light_hsl_status_data_t.
        ii. Hint: Use "Open Declaration" on that datatype to find out what it contains.
        iii. Hint: Change the calls to led_control_set_brighness_level to pass the Hue, Saturation, and Lightness instead of last_known_brightness.
            1. Hint: Hue, Saturation and Lightness are of type unit16_t.
4. In led_control.c:
    a. Set up two additional PWMs – one for the Green LED and one for the Blue LED.
        i. Hint: Use PWM1 and PWM2.
        ii. Use the following function to convert the HSL values to RGB values and then use the RGB values to control the three LEDs.

1. Hint: The HSL inputs can be passed in directly from what the model provides. The outputs are provided as pointers to three uint8_t variables (r, g, and b).

```
/* Convert HSL values to RGB values */
/* Inputs: hue (0-360), sat (0-100), and lightness (0-100) */
/* Outputs: r (0-100), g (0-100), b (0-100) */
void HSL_to_RGB(uint16_t hue, uint16_t sat, uint16_t light, uint8_t* r,
uint8_t* g, uint8_t* b)
{
       uint16_t v;

    /* Formulas expect input in the range of 0-255 so we need to convert
       the input ranges which are H: 0-360, S: 0-100, L: 0-100 */
    hue = (hue*255)/360;
    sat = (sat*255)/100;
    light = (light*255)/100;

    v = (light < 128) ? (light * (256 + sat)) >> 8 :
          (((light + sat) << 8) - light * sat) >> 8;
    if (v <= 0) {
        *r = *g = *b = 0;
    } else {
        int m;
        int sextant;
        int fract, vsf, mid1, mid2;

        m = light + light - v;
        hue *= 6;
        sextant = hue >> 8;
        fract = hue - (sextant << 8);
        vsf = v * fract * (v - m) / v >> 8;
        mid1 = m + vsf;
        mid2 = v - vsf;

        // Convert output range of 0-255 to 0-100
        v = (v*100)/255;
        m = (m*100)/255;
        mid1 = (mid1*100)/255;
        mid2 = (mid2*100)/255;

        switch (sextant) {
           case 0: *r = v; *g = mid1; *b = m; break;
           case 1: *r = mid2; *g = v; *b = m; break;
           case 2: *r = m; *g = v; *b = mid1; break;
           case 3: *r = m; *g = mid2; *b = v; break;
           case 4: *r = mid1; *g = m; *b = v; break;
           case 5: *r = v; *g = m; *b = mid2; break;
        }
    }
}
```

5. Program your kit.
6. Provision your device onto your network.
7. Use the app to adjust the light color and intensity. Also note that if you turn the light on/off using the switch in the app, the device remembers the last value so that when you turn it back on the color and brightness are the same.