Chapter 7: Workshop Exercises

Objective: This Addendum Manual contains exercises to accompany Workshop Lab Chapters.

TABLE OF CONTENTS

EXERCISE 7.1– BLINK AN LED 2
EXERCISE 7.2 – ADD DEBUG PRINTING 3
EXERCISE 7.3 – SET LED FROM BUTTON ACTION VIA INTERRUPT 4
EXERCISE 7.4 – RGB LIGHT SENSOR W/ADI SHIELD 5
EXERCISE 7.5 – WEIGHT SENSOR W/ADI SHIELD 7
EXERCISE 7.6 – MESH 1ST NETWORK WITH DIMMABLE LED 9
EXERCISE 7.7 – MESH LIGHTS (MULTI-ELEMENT NODES) AND GROUPS 13
EXERCISE 7.8 – MESH ON/OFF SWITCH 14
EXERCISE 7.9 – MESH DIMMER SWITCH 14
EXERCISE 7.10 – MESH 2ND ELEMENT – RED AND YELLOW OR BLUE LEDS 15
EXERCISE 7.11 – MESH TEMPERATURE SENSOR 16
EXERCISE 7.12 – MESH ADI RED SENSOR + TEMPERATURE SENSOR 17

EXERCISE 7.1 – BLINK AN LED

This exercise ensures all software is installed and connections to the kit are correct.

Reference: Section 3.4.1 GPIO in PERIPHERALS section of Lab Manual for those who care to try to write their own GPIO interface code to drive the LED on and off.

Target Kits: CYW920819

In this exercise, create an application to blink LED_2 on the kit at 2 Hz and add a message to print through the Peripheral UART to a terminal program on your PC each time the LED changes state.

1. Open Modus Toolbox
2. Select a directory as your workspace
3. In the Quick Panel click "New Application".
4. Select your kit: CYW920819EVB-02 then press Next >
5. Select Empty-20819EVB02
6. Change the application name to **Blinky_Print_20819EVB02**.

   Note: The specific application name is up to you. It is recommended to use an alphabetically sortable prefix to help find each project when multiple projects are being developed simultaneously.

   Note: It is recommended to designate the target platform in the name to easily differentiate which projects can be programmed into which kits.

7. Click "Next ". Verify the presented device, board, and example, then click "Finish".

8. Delete the file empty_wiced_bt.c from the newly created project.
   Find BlinkLEDandPrint.c in the folder Template_Blink_LED_Print in the Template folder directory that was provided. Copy BlinkLEDandPrint.c into the newly created project. This file has `application_start(void)` and this is where the code will jump to after initializing the ThreadX RTOS.

9. Examine BlinkLEDandPrint.c to make sure you understand what it does.

   All WICED BLE applications are multi-threaded (the BLE stack requires it). There is an operating system (RTOS) that gets launched from the device startup code and you can use it to create your own threads. Each thread has a function that runs almost as though it is the only software in the system – the RTOS allocates time for all threads to execute when they need to. This makes it easier to write your programs without a lot of extra code in your main loop. The details of how to use the RTOS effectively are covered in the next chapter but, in these exercises, we will show you how to create a thread and associate it with a function for the code you will write (look in app_bt_management_callback()).

10. Add code in the app_task thread function of BlinkLEDandPrint.c to do the following:
    a. Read the state of WICED_GPIO_PIN_LED_1

       Hint: Open WICED API Reference in Documentation tab next to Quick Start tab in lower left corner of ModusToolbox. Components > Hardware Drivers > GPIO

       Hint: Search for /*** Enter Exercise Code Here ***/

       Reference: Section 3.4.1 GPIO in PERIPHERALS

    b. Drive the state of WICED_GPIO_PIN_LED_1 to the opposite value.

11. In the Quick Panel, look in the "Launches" section and click the line that has the project name followed by “Program".

12. Once the build and program operations are done, observe "Programming Complete" in the Console window and LED 1 on the kit periodically turning on and off at the rate designated in BlinkLEDandPrint.c.

EXERCISE 7.2 – ADD DEBUG PRINTING

Reference: Section 3.4.2 DEBUG PRINTING in PERIPHERALS section of Lab Manual

Target Kits: CYW920819

For this exercise, you will add a message to print through the Peripheral UART to a terminal program on your PC each time the LED changes state.

Optional step: The added capability in this exercise can be added directly to the project from the previous exercise or a new project can be created via cloning. To Clone, select “New Application”, select your kit “CYW920819-02”, click [Next], click [Import] then search for and select the project to be cloned, in this case select “Blinky_Print_20819EVB02” then [Select Folder], Change the project name, click [Next] then [Finish].

1. Add WICED_BT_TRACE calls to display “LED LOW” and “LED HIGH” at the appropriate times.
    a. Hint: Reference Debug Printing in the manual.
    b. Note: Verify that at the top of the file is `#include "wiced_hal_puart.h"` Anytime you want to access hardware feature, you need to include the wiced_hal_xxx.h so that the APIs are available.
    c. Hint: Remember to set the debug UART to WICED_ROUTE_DEBUG_TO_PUART. Although you will see comments in the template code encouraging you to put initialization code in the app_bt_management_callback() function so that it runs when the BLE stack starts up, we recommend doing it in application_start() instead. This is because the PUART is a special type of peripheral and you may want to print messages before even trying to start the stack!

   d. Hint: Remember to use \n\r to create a new line so that information is printed on a new line each time the LED changes.
2. Program your application to the board.
3. Open a terminal window (e.g. PuTTY or TeraTerm) with a baud rate of 115200 and observe the messages being printed.
   a. Hint: The PUART will be the larger number of the two WICED COM ports.
   b. Hint: if you don't have terminal emulator software installed, you can use putty.exe which is included in the class files under "Software_tools". To configure putty:
      i. Go to the Serial tab, select the correct COM port (you can get this from the Microsoft Windows program Device Manager under "Ports (COM & LPT)" as *"WICED USB Serial Port"*), and set the speed to 115200.
      ii. Go to the session tab, select the Serial button, and click on "Open".
      iii. If you want an automatic carriage return with a line feed in putty (i.e. add a \r for every \n) check the box next to "Terminal -> Implicit CR in every LF"

EXERCISE 7.3 – SET LED FROM BUTTON ACTION VIA INTERRUPT

Reference: Section 3.4.1 GPIO in PERIPHERALS section of Lab Manual

Target Kits: CYW920819

Note: This is an optional exercise, not covered in the workshop presentation.

In this exercise, rather than polling the state of the button in a thread, you will use an interrupt so that your firmware is notified every time the button is pressed. In the interrupt callback function, you will toggle the state of the LED.

1. Create another new application from empty-20819EVB02 and rename it **Interrup_Btn_20819EVB02**
2. Delete empty_wiced_bt.c from the new project.
3. Copy Interrupt_Bin.c from the Template files provided.
4. In the interrupt_Bin.c file:
   a. Remove the calls to wiced_rtos_create_thread() and wiced_rtos_init_thread().
   b. Delete the thread function.
2. In the BTM_ENABLED_EVT, set up a falling edge interrupt for the GPIO connected to the button and register the callback function.
   a. Hint: You will need to call wiced_hal_gpio_register_pin_for_interrupt and wiced_hal_gpio_configure_pin.
3. Create the interrupt callback function so that it toggles the state of the LED each time the button is pressed.
4. Program your application to the board and test it.

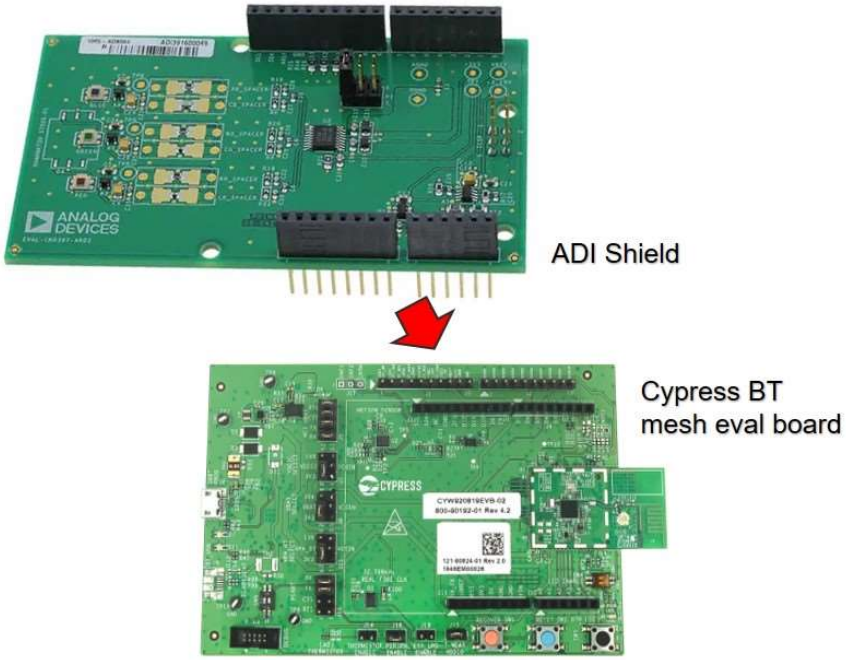
EXERCISE 7.4 – RGB LIGHT SENSOR W/ADI SHIELD

Reference:

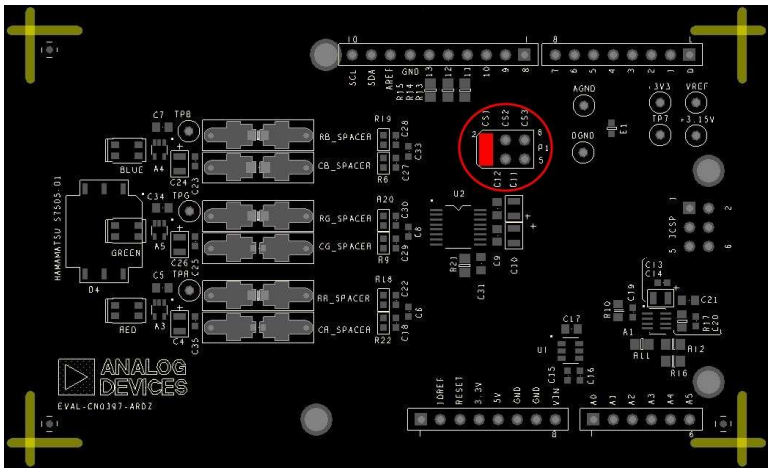
Target Kits: CYW920819 with ADI CN0397 Arduino shield

In this exercise, you will be reading and displaying data on the Debug UART from the Analog Devices CN0397 RGB Light Sensor Arduino shield. You will be interfacing to the AD7798 3-channel 16-bit Delta Sigma ADC as a SPI Slave.

1. Plug the CN0397 onto the CYW920819EVB-02 as shown.

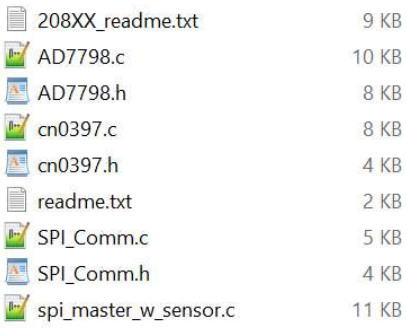


Make sure that the CS Jumper is set to CS1 (GPIO 10), the leftmost jumper



Chip Select	GPIO
CS1	10
CS2	9
CS3	8

2. Find the Project files in the Template directory provided: CN0397_ADI_Light_Sensor
3. Create a new application project based on an Empty-20819EVB02 and call it CN0397_ADI_Light_Sensor
4. Delete the empty_wiced_bt.c file from the new project
5. Copy the four .c code files, three .h header files and the two .txt files from the CN0397_ADI_Light_Sensor folder into the new project



6. In the spi_master_w_sensor.c file:
   a. Add a #include for the SPI functions - "wiced_hal_pspi.h" and for the header file cn0397.h
   b. For the CLK_1, MISO_1, MOSI_1 and CS_1 defines, put in the correct platform pins to match the CN0397 signals on D10 through D13.
      Hint: Use <Ctrl>F to search in spi_master_w_sensor.c for WICED_Pxx
   c. In the Initialize_app routine, initialize SPI1 after the Button GPIO is initialized

   See the API Documentation for the SPI Init call.

```
void wiced_hal_pspi_init ( spi_interface_t      spi,
                           UINT8                devRole,
                           UINT16               spiPinPullConfig,
                           UINT32               spiGpioCfg,
                           UINT32               clkSpeed,
                           SPI_ENDIAN           endian,
                           SPI_SS_POLARITY      polarity,
                           SPI_MODE             mode,
                           UINT8                csPin
                         )
```

   Hint: It is SP1 with a `SPI_MASTER` role. Pin Pull is INPUT_PIN_PULL_UP, and use the GPIO Macro for the configuration (`#define GPIO_CFG(CS_1,CLK_1,MOSI_1,MISO_1).` Speed also has a define that will set it to 1MHz. The SPI Endian is MSB First, SPI SS polarity is ACTIVE LOW, mode is 3 and the csPin is CS_1.

7. In the sensor thread initialize the CN0397

   a. Hint: the function calls are in CN0397.h, (make sure you include CN0397 so that the functions can be found)
   b. It should be put in the INIT_SENSOR state
8. In the sensor thread, add function calls to read the data and display the data.
   a. Hint: to read data you must call SetAppData()
9. The CN0397 math routines require floating points libraries. ModusToolbox 2.0 didn't handle this properly, so a line needed to be added to the makefile to the beginning of the makefile.

   ```
   CY_RECIPE_EXTRA_LIBS+=-lgcc
   ```

   If you are using floating point math in any of your projects to need to add this line into the makefile and add `#include <math.h>` to your source files that use floating point math.
10. Open the TeraTerm or Putty window (Baudrate 115,200).
11. Program the application to the board.
12. Follow the calibration procedure and see the results

EXERCISE 7.5 – WEIGHT SENSOR W/ADI SHIELD

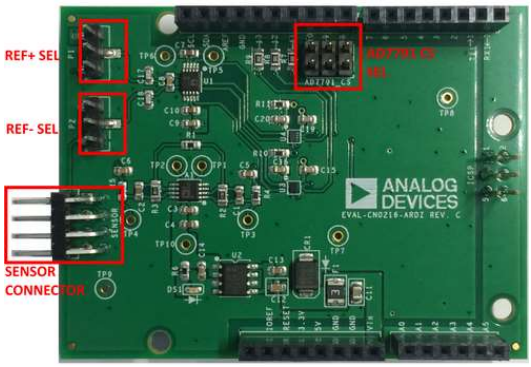
Reference:

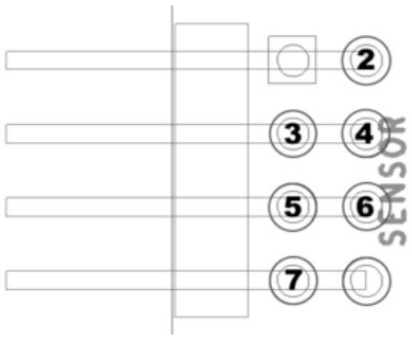
Target Kits: CYW920819 with ADI CN0216 ADI Weight Sensor Arduino shield and Load Cell

1. Plug the CN0216 ADI shield onto the CYW920819EVB02 kit
2. Make sure that the jumper for CS is the leftmost, Digtial Pin 10

CS - Arduino **Digital Pin 10**

3. Connect a load cell to CN0216 Sensor Connector
   -Green/Blue to Pin 3
   -Black/Purple to Pin 2
   -White/Yellow to Pin 6
   -Red/Orange to Pin 7





Note: Bar type load cells have four wires: Green/Black/White/Red. Extension wires have been added to simplify connection to the CN0216 shield. Extension wires may be a different color as indicated.

4. Connect a jumper between 5V and VIN on CN0216 Arduino Shield Power Connector
   - 5V (pin 5) to VIN (pin 8)
5. Create a **New Application** for target **CYW920819EVB-02** project using **Empty-208019EVB02** renamed to **CN0216_ADI_Weight_Sensor**

3. Delete the empty_wiced_bt.c from the new project

4. Copy all files from Template file CN0216_ADI_Weight_Scale into the new project

5. Add the following line to makefile (reference https://community.cypress.com/thread/50179)

   CY_RECIPE_EXTRA_LIBS+=-lgcc

6. Program the kit and follow instructions in the console to calibrate.

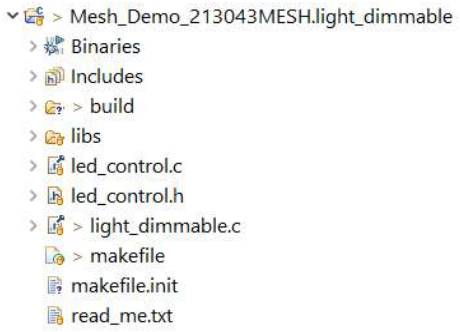
EXERCISE 7.6 – MESH 1ST NETWORK WITH DIMMABLE LED

Reference Chapter 4 or Cypress Workshop Chapter WBT101-07A

Target Kits: CYW920819 or CYBT-213043-MESH

In this exercise you will create a 1 node mesh network.

1. In ModusToolbox, Create Demo projects for your selected kit.
2. Expand Mesh Demo project ending in “light_dimmable” that includes the number of the selected kit.



3. Within the selected project, open "light_dimmable.c" and find where device_name is assigned. Change the name so that it has your initials in it (e.g. "<Inits> Dimmable Light").

Note: **The device_name can be a Maximum of 25 characters in length**. Any longer and the device name will not show up when trying to provision in the iOS or Android App.

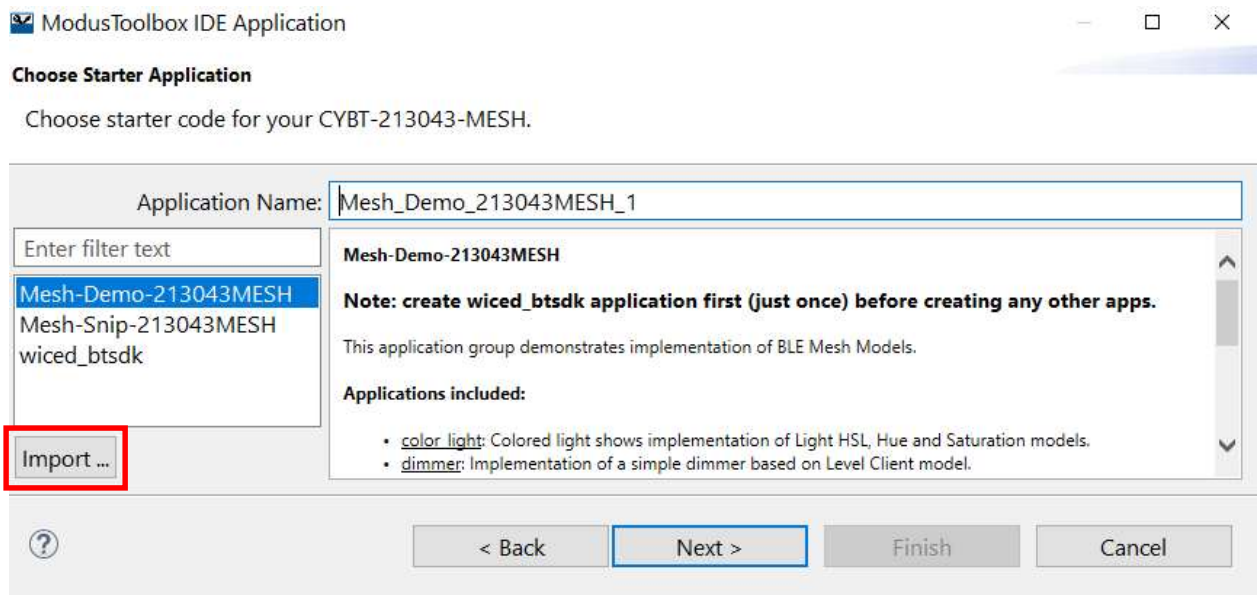
4. Program the project to your kit.
    a. Hint: Open a terminal window for the PUART to see messages.
    The default PUART baud rate for the mesh applications is 921600.
    Hint: If your terminal emulator does not support 921600:
    Add this line of code after you set the PUART to the debug UART
    `wiced_hal_puart_configuration(115200, PARITY_NONE, STOP_BIT_1);`

5. Run Mesh Client or a Mesh Lighting smartphone application to provision the device.
    a. Hint: If you don't see any devices listed after ~10 seconds, exit the app, stop/restart BLE and then try again.

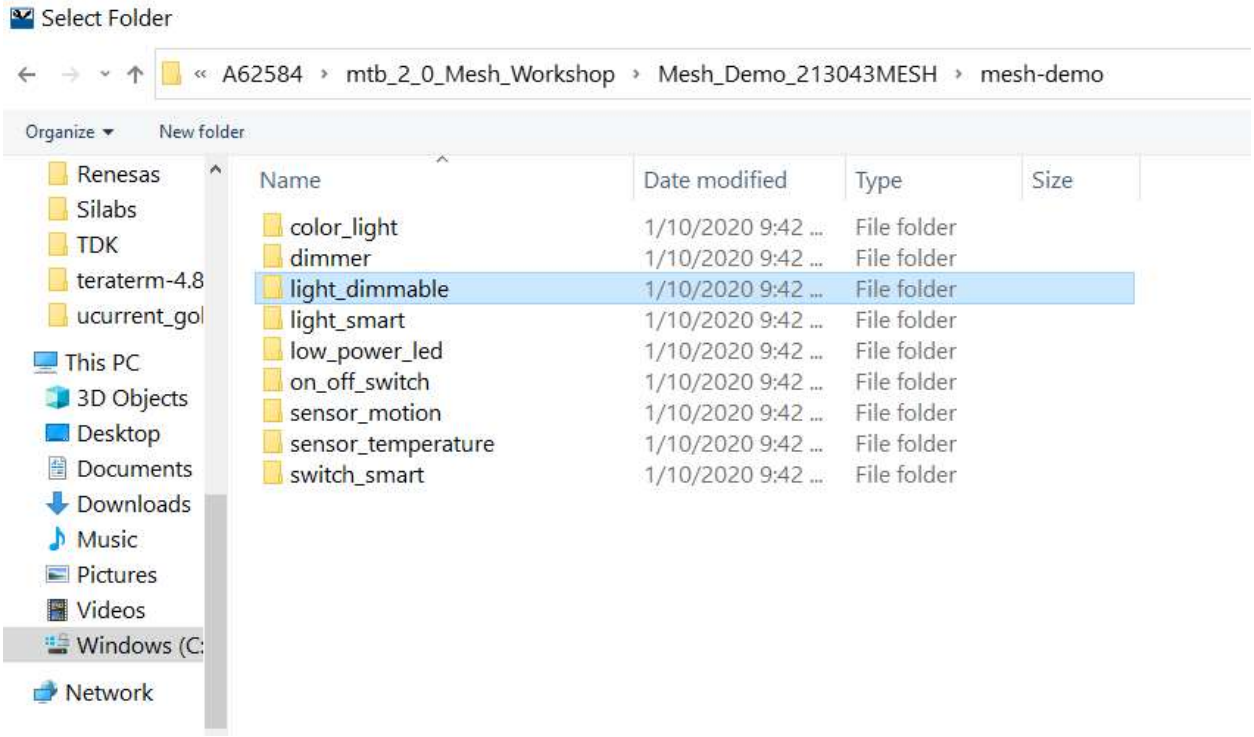
Optional Method: Cloning of a mesh project:

If you don’t want to modify the Mesh Demo project you can clone the Mesh Demo project that you want to use as a base application.

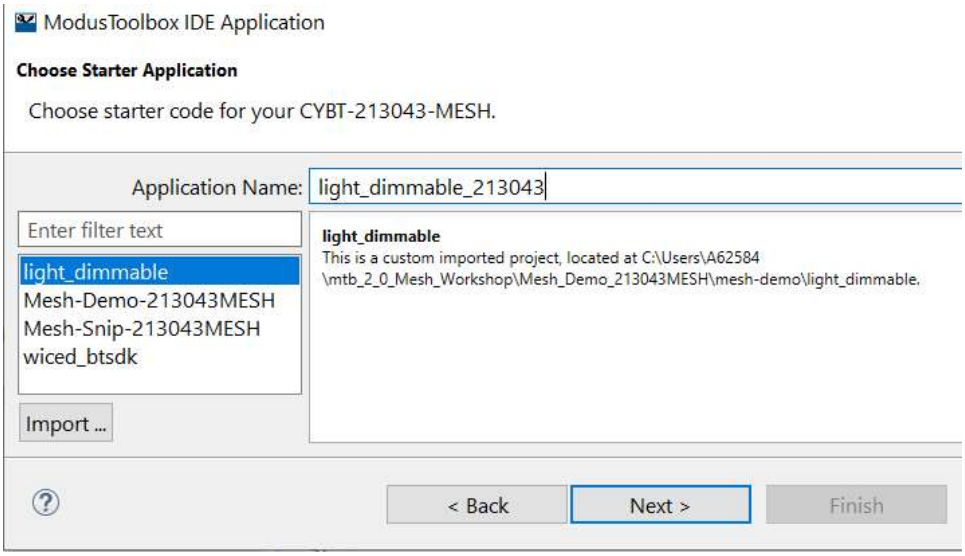
1. Click on New Application and select the target platform.
2. Click on Import



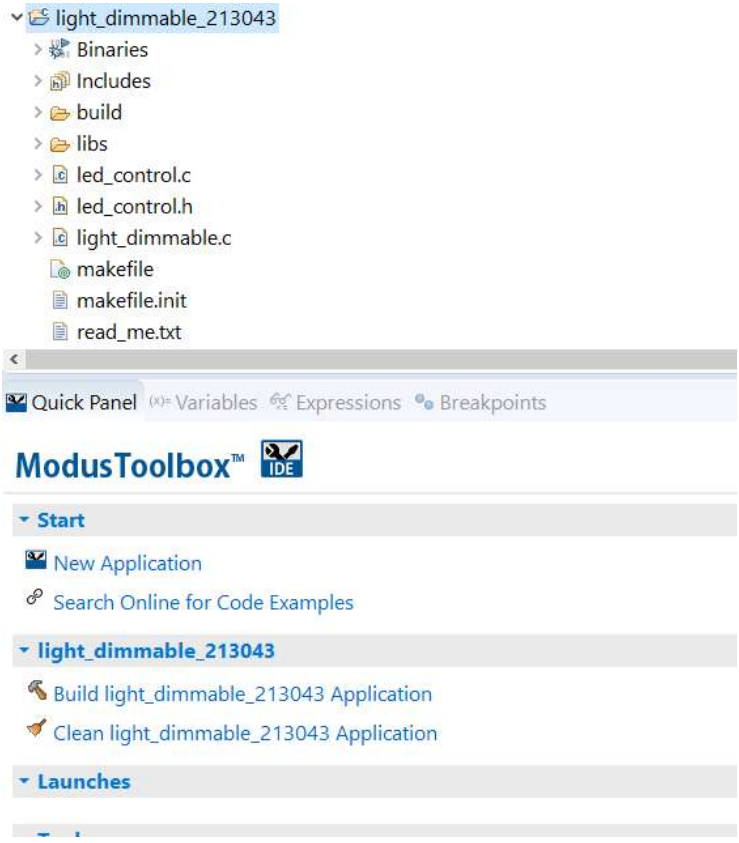
3. Find the directory with the Mesh Demo project that you want to clone. It will be in the Workspace directory under the Mesh Demos for the eval board that you are targeting. This is an example of the light_dimmable the CYBT213043-MESH eval board.



4. Change the name (maybe add the eval board name so that you remember which eval board that you are targeting), click Next and then click Finish.



After it is finish and you click on the application in the explorer window, you will see that the Launches are blank.



This is because the path in the makefile pointing to the wiced_btsdk is not correct. Cloning of Mesh Project may require an additional step to adjust a path pointer for the wiced_btsdk. Mesh projects in ModusToolbox are initially setup as a group, two levels down in the Project Explorer hierarchy. If a Mesh project is cloned by importing, it will be brought to the top level in the Project Explorer hierarchy. There's a path pointer to wiced_btsdk in the file "makefile" within the project. The pointer appears as either:

```
# Path (absolute or relative) to the bt-sdk folder (at repo root)

CY_SHARED_PATH=$(CY_APP_PATH)/../../../wiced_btsdk
```

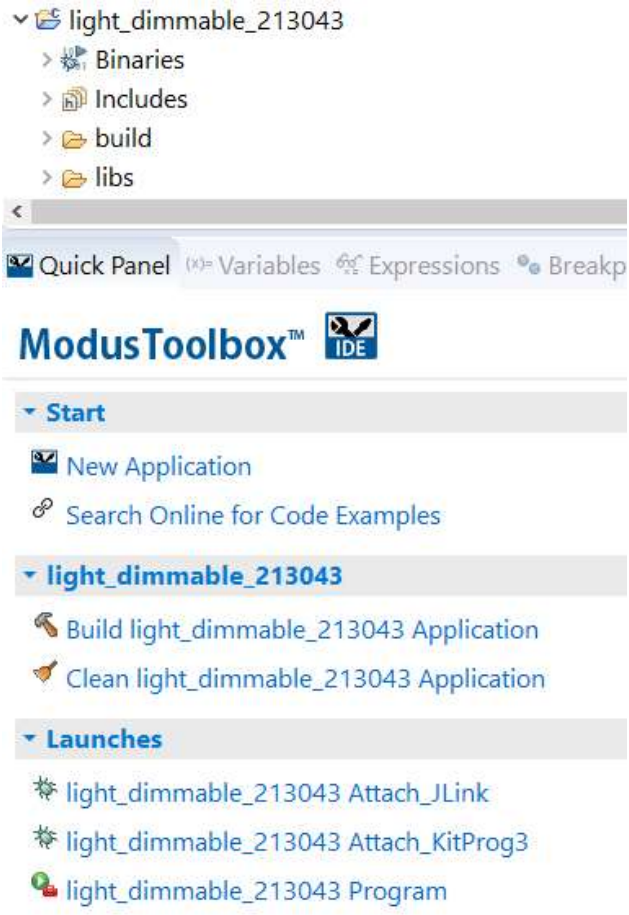
- or –

```
# Path (absolute or relative) to the bt-sdk folder (at repo root)

CY_SHARED_PATH=$(CY_APP_PATH)/../wiced_btsdk
```

Each "../" signifies to move up one level. Mesh projects that are two levels down require 3 of the "../" in their path to wiced_btsdk. Cloned projects are placed at the top level of projects and require only a single "../" in their path to find wiced_btsdk.

Once you change the path, do a build and then you will see the launches with what you expect.



EXERCISE 7.7 – MESH LIGHTS (MULTI-ELEMENT NODES) AND GROUPS

Reference Chapter 5 or Cypress Workshop Chapter WBT101-07B

Target Kits: CYBT-213043

In this exercise you will add additional lights to the network you created in the previous chapter. You will experiment with associating devices to different groups.

1. Prepare a project through one of the methods taught:
    - Create a new set of Demo projects
    - Clone a project – Import “Mesh_Three_light_dimmable” from Template directory
    - Modify an existing project
2. Modify the prepared project with an extended 3-LED version of light_dimmable
    a. Expand a project that will become a 3-LED Mesh Node
    b. If starting with a new Demo or modifying an existing project; **Copy** the three files (**led_control.c, led_control.h, light_dimmable.c**) from the provided Key/Key_Dimmable_3LEDs folder into the newly created project. Overwrite the existing files.
    c. Open light_dimmable.c and change device name to something unique – to distinguish it from other students performing the same exercise. (Ensure device_name does NOT exceed 25 characters)
    d. Hint: Plug only one kit at a time into your PC to ensure the desired kit is programmed.
    e. Hint: Programmed kits can be powered using an alternate power source.
3. Provision the Three_light_dimmable kit to a mesh network using a BLE_Mesh App.
4. Create multiple Rooms (i.e. Groups) and add one or more lights to each Room.
    a. Hint: Leave all the lights in the group "All". That is, just add them to the new rooms, don't move them.
5. Experiment with controlling all lights at once (All), one room at a time, and individually.
6. Optional: experiment with other room configurations. For example:
    a. Light 1 is in Room 1 and All
    b. Light 2 is in Room 2 and All
    c. Light 3 is in Room 1, Room2 and All
    d. Light 4 is only in All

EXERCISE 7.8 – MESH ON/OFF SWITCH

Reference Chapter 6 or Cypress Workshop Chapter WBT101-07C

Target Kits: CYW920819 or CYBT-213043

In this exercise you will add an on/off switch to your mesh network that can control LEDs on the light_dimmable kit(s).

1. Remove one of your light_dimmable kits from your mesh network using the BLE_Mesh app.
2. Within ModusToolbox Project Explorer; Expand a Demo project ending in “on_off_switch” that is designated for the selected target kit.
3. Open "on_off_switch.c", find the "mesh_dev_name" and change the name to something unique, possibly using your initials (e.g. "<Inits> Switch") (don’t exceed 25 characters)
4. Program the project into a mesh kit.
    a. Hint: You may want to label the kits to keep track of which one is programmed with each project.
    b. Note: Pressing the black user button SW3 on a kit programmed with a light_dimmable project will perform a factory reset so that kit will need to be re-provisioned.
5. Provision the OnOff Switch kit to your network.
6. Press the user button on the OnOff Switch kit to toggle the LEDs on the a kit programmed with a light dimmable project.
7. Experiment with changing the Assignment for the OnOff Switch to control different lights or rooms.
    a. Hint: You can't move switches to different rooms, rather you Assign the device or rooms that the switch will control.
8. Note that you can still control the lights using the app.

EXERCISE 7.9 – MESH DIMMER SWITCH

Reference Chapter 6 or Cypress Workshop Chapter WBT101-07C

Target Kits: CYW920819 or CYBT-213043

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 a kit programmed with a light_dimmable project. The OnOff Switch from the previous exercise will be able to control the same LED.

1. Remove one of your light_dimmable kits from your mesh network using the BLE_Mesh app.
2. Expand a Demo project ending in “dimmer” for the selected kit.
3. 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")
4. Program the project into the kit.
    a. 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 light_dimmable kit, it will perform a factory reset.
5. Provision the Dimmer kit to your network.

6. Press the user button on the Dimmer kit to toggle the LEDs on the light_dimmable kits.
7. Press and hold the user button on the Dimmer kit to adjust the brightness of the LEDs.
    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.

Experiment with Assigning the Dimmer to different lights or rooms.

EXERCISE 7.10 – MESH 2ND ELEMENT – RED AND YELLOW OR BLUE LEDS

Reference Arrow BLE-Mesh Chapter 6 or Cypress Workshop Chapter WBT101-07C

Target Kits: CYW920819 (Red and Yellow LEDs) or CYBT-213043 (Red and Blue LEDs)

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

1. Use the Smartphone App to remove one of the light_dimmable devices from your network.
2. Create a new application for a light_dimmable or modify an existing one
3. Expand the selected light_dimmable project.
4. 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.
        ii. Hint you will need to add a set of entries to the mesh_elements array for the new element.
        iii. Hint: Make sure you set the number of models and number of properties in the mesh_elements array to the correct values for this new element.
    b. In the mesh_app_init function, initialize the new light lightness server.
        i. Hint: You can use the same callback for both light lightness servers since it is passed the element index when it is called.
5. 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: The configurator was not used in the light_dimmable demo application so for consistency you can set up the PWM the same way.
        ii. Hint: Copy the code for PWM0 and update it to use PWM1.
        iii. Hint: the BSP has a #define for LED_GREEN that you can use.
        iv. 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.
6. 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, you can:
         1. Set up a global uint32_t to hold the index value.
         2. Pass that variable as an argument when you init the timer
         3. Update the value of the variable with the element_idx value just before starting the timer.
      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.
7. Program your kit.
8. Provision your device onto your network.
   a. Hint: You should see 2 devices show up for this kit instead of just one.

Control each of the LEDs from the app individually using the two separate devices
Note that the OnOff Switch and Dimmer control both LEDs simultaneously because they control everything in the group at once.

EXERCISE 7.11 – MESH TEMPERATURE SENSOR

Reference Chapter 6

Target Kits: CYW920819 (Note: Using the kit with ADI CN0397 RGB Light Sensor Arduino Shield for this exercise will get you ready for the next exercise, which specifically requests that kit)

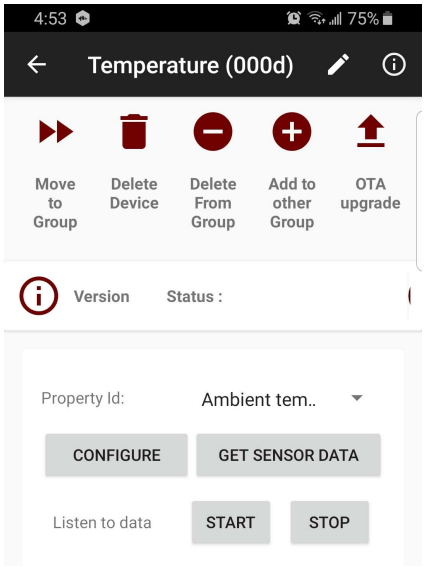
Setting up the initial Temperature Sensor Element (as basis for the next exercise)

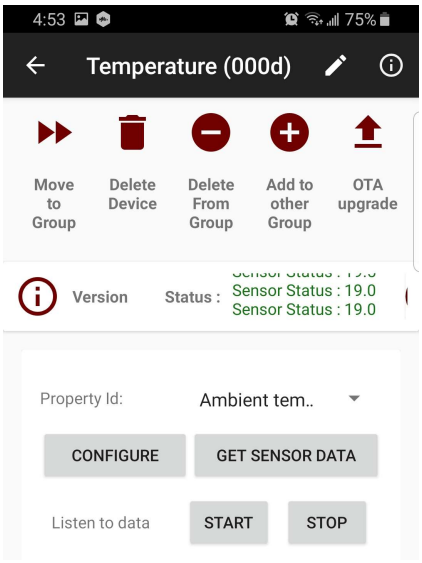
Enable visibility of the temperature sensor in your mesh network.

1. Select an existing or Import a new project:
   a. To Select a project, expand a project ending in “sensor_temperature” for your selected kit.
   b. To Import a project: Use “New Application” / Select target kit / [Next] / [Import]/ find Template\mesh-demo\sensor_temperature / change “`CY_SHARED_PATH=$(CY_APP_PATH)/../../../wiced_btsdk`” in makefile to “`CY_SHARED_PATH=$(CY_APP_PATH)/../wiced_btsdk`”
2. Open the file "sensor_temperature.c" and find where the device name is assigned. Change the name so that it has your initials in it (e.g. "<Inits> Temperature Sensor").
3. Program the project to your CYW920819EVB-02 kit.
   a. Hint: You should open a terminal window for the PUART to see messages.
4. Run the Mesh Lighting application to provision the device.

a. Hint: If you don't see any devices listed after ~10 seconds, exit the app, stop/restart BLE and then try again.

5. Press **GET SENSOR DATA** several times.
6. The temperature shows up as a number in a Status Window of the BLE_Mesh Smartphone AP





EXERCISE 7.12 – MESH ADI RED SENSOR + TEMPERATURE SENSOR

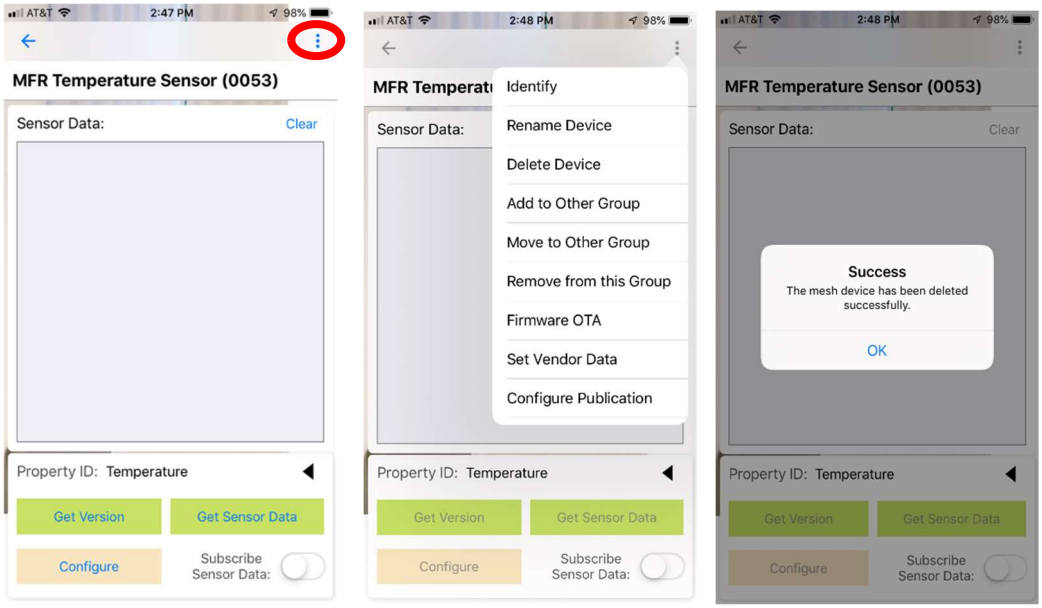
Reference Chapter 6

Target Kits: CYW920819 with ADI CN0397 RGB Light Sensor Arduino Shield

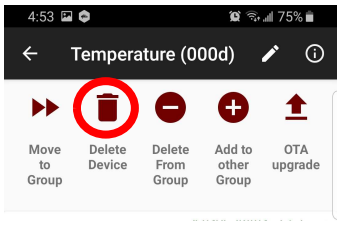
Adding a 2nd Sensor Element (the Red Light Sensor on the CN0397 Shield)

In this exercise, you will add a Red Light Sensor on the CN0397 Arduino Shield to the Temperature Sensor Application. Once provisioned, you will see 2 devices and be able to GET the temperature and Red Light data from the target board.

1. Remove your Temperature Sensor programmed CYW92018EVB-02 from your mesh network using the Android or iOS App (these are iOS screenshots). Select the sensor by clicking on Control. Click on the 3 vertical dots in the upper right and click on Delete Device



Note: For Android, select a Component from a Room list, then tap the Trash Can symbol to delete it. Leaving the Component screen by tapping the ← will show the Component is no longer in the Room list.



2. Select New, Import Clone or Import Template to create a new project:
   - Select New via "New Application"➔select target kit➔[Next]➔Mesh-Demo-20819EVB02 –> Use the "…sensor_temperature" project as the starting point.
   - Import Clone via "New Application"➔select target kit➔[Next]➔[Import]➔select an existing project that ends in <target kit number>.sensor_temperature
   - Import Template via "New Application"➔select target kit➔[Next]➔[Import]➔Template/**Mesh_temp_and_red_sensor_20819_template**
3. If using new or cloned project; Copy files from Template/**Mesh_Temp_plus_Red_Sensor** into the "….sensor_temperature" starting point project. Overwrite existing sensor_temperature.c file.
4. If using Imported Template; Open > makefile and verify or add the single line in the "Paths" section

   CY_RECIPE_EXTRA_LIBS+=-lgcc

   (reference https://community.cypress.com/thread/50179)

5. Open sensor_temperature.c and find "wiced_bt_cfg_settings.device_name". Change the device_name to something unique that's you'll recognize when adding to your mesh network… and less than 25 characters.
6. Add the model for element 2.
   Hint: this is element 1. It will be similar for element 2. The mesh device isn't needed for more that the first element, just the sensor server. Define the MESH_APP_NUM_MODELS_RED for the element configuration structure

```
wiced_bt_mesh_core_config_model_t mesh_element1_models[] =
{
    WICED_BT_MESH_DEVICE,
    WICED_BT_MESH_MODEL_SENSOR_SERVER,

};

#define MESH_APP_NUM_MODELS_TEMP    (sizeof(mesh_element1_models) /
sizeof(wiced_bt_mesh_core_config_model_t))
```

7. Create the #defines and variables for the Red Sensor Element
   Hint: These are the #defines for the Temperature Sensor

```
#define MESH_TEMPERATURE_SENSOR_POSITIVE_TOLERANCE      CONVERT_TOLERANCE_PERCENTAGE_TO_MESH(1)
#define MESH_TEMPERATURE_SENSOR_NEGATIVE_TOLERANCE      CONVERT_TOLERANCE_PERCENTAGE_TO_MESH(1)

#define MESH_TEMPERATURE_SENSOR_SAMPLING_FUNCTION       WICED_BT_MESH_SENSOR_SAMPLING_FUNCTION_UNKNOWN
#define MESH_TEMPERATURE_SENSOR_MEASUREMENT_PERIOD      WICED_BT_MESH_SENSOR_VAL_UNKNOWN
#define MESH_TEMPERATURE_SENSOR_UPDATE_INTERVAL         WICED_BT_MESH_SENSOR_VAL_UNKNOWN
```

   For the Red Light Sensor the tolerances are unspecified.
   WICED_BT_MESH_SENSOR_TOLERANCE_UNSPECIFIED
   (This is found in the wiced_bt_mesh_model_defs.h file)
   And the sampling function, measure period and update interval are the same as the Temperature Sensor.

   Create the data variable to send the Red Light Sensor
   Hint: the ambient light sensor model requires 3 octets

```
#define WICED_BT_MESH_PROPERTY_LEN_PRESENT_AMBIENT_LIGHT_LEVEL    3
```

   So your variable should be an array of 3 bytes. Something like this

```
uint8_t       mesh_red_sensor_sent_value[3] = {0,0,0};
```

8. Copy the Temperature Sensor `bt_mesh_core_config_sensor_t mesh_element1_sensors` structure. Rename it to element1 to element2 and change the values for the red light sensor The property ID is WICED_BT_MESH_PROPERTY_PRESENT_AMBIENT_LIGHT_LEVEL so that the Mesh App knows what to display. The prop_value_len was shown in the last paragraph
   Hint: .data is address of the red sensor sent value.

9. Add the 2nd Sensor to the `bt_mesh_core_config_element_t mesh_elements[]` structure

Hint: Copy the Temperature Element, add it to the structure. Change the .sensors and .models_num for element 2 and the red sensor.

10. In the mesh_app_init function in the section where it is provisioned

    Add the SPI port initialization for reading the CN0397 `wiced_hal_pspi_init`

    Add initialization of the CN0397 with CN0397_Init()

    Add a WICED Timer for the Red Sensor publish
    Hint: copy the Temperature Sensor Timer and change the
    `&mesh_sensor_publish_timer_callback to`
    `&mesh_red_sensor_publish_timer_callback`

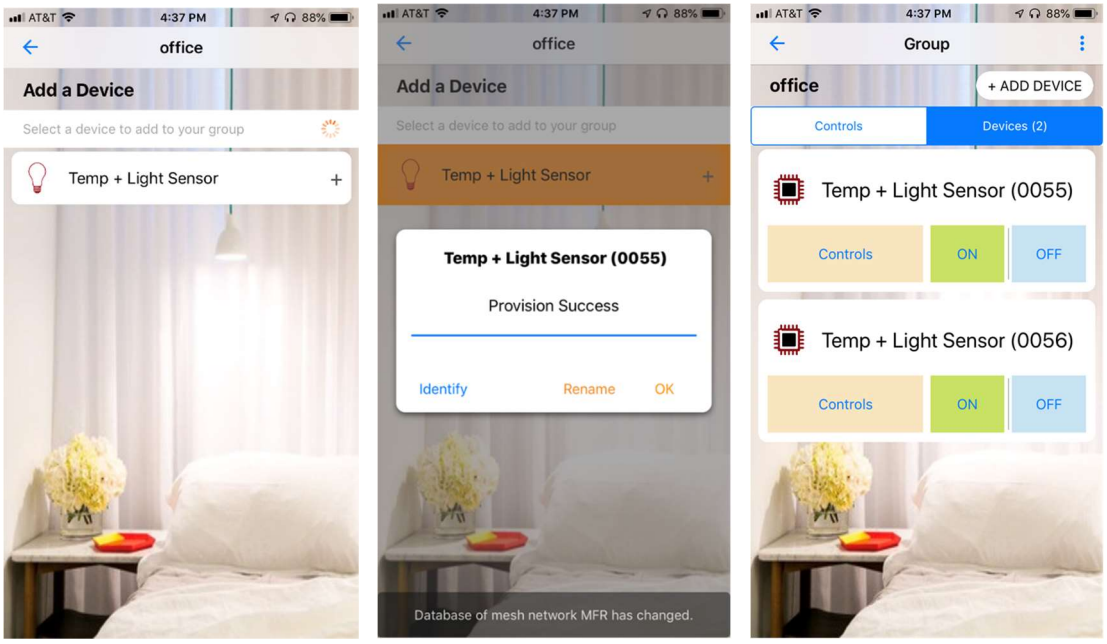
    Restore the cadence NVRAM
    Create a #define MESH_RED_SENSOR_CADENCE_VSID at VSID_START+100

    (put it with the RED SENSOR #defines at the top of the code)
    Copy the Temperature `wiced_hal_read_nvram` and replace
    `MESH_TEMPERATURE_SENSOR_CADENCE_NVRAM_ID` with the Red Sensor address

    Init the red sensor server
    `wiced_bt_mesh_model_sensor_server_init()` for the red sensor
    Hint: Copy the Temperature sensor server init and change the index
    `MESH_RED_SENSOR_INDEX`

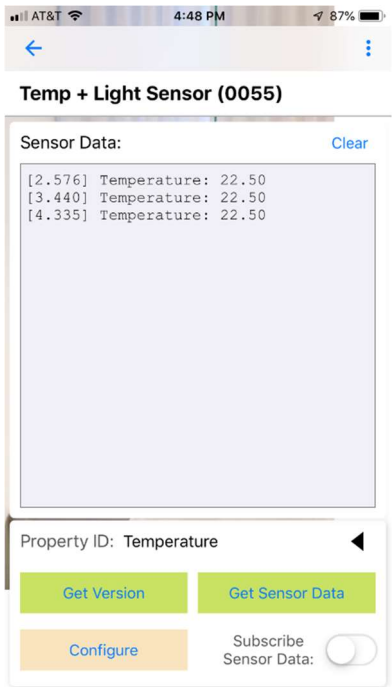
11. Build and program the CYW920819EVB-02

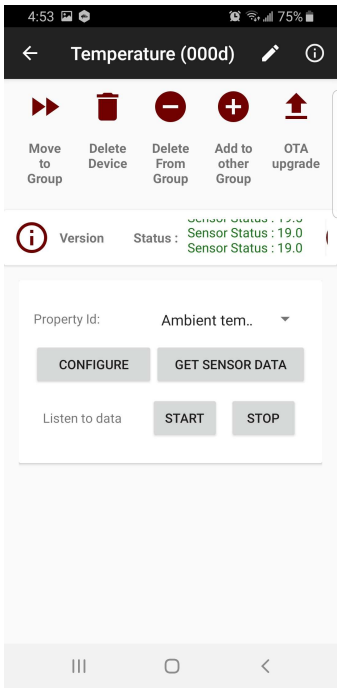
12. Provision your Temperature Sensor Kit to your Network. You should see 2 Devices, one Temp + Light Sensor (xxxx) and Temp + Light Sensor (xxxx+1)



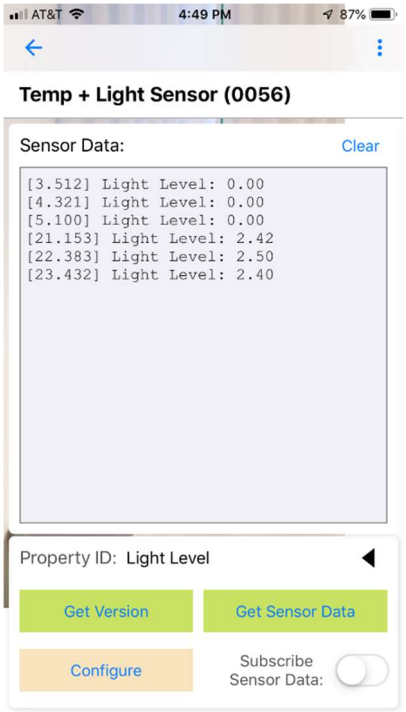
Note: The above images are for iOS. Android is similar; Provision one Device and get two Components, one with a higher number in brackets than the other.

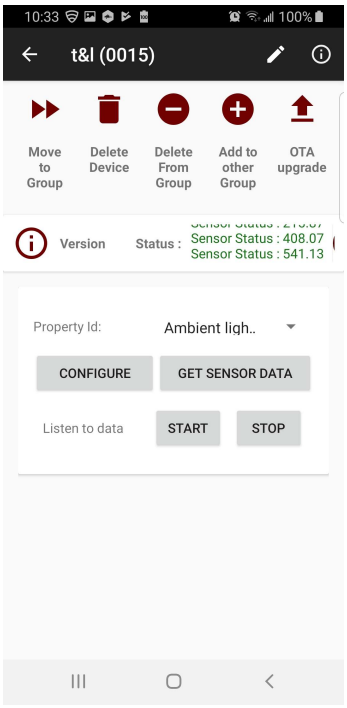
13. Read the Temp Sensor in the app (click on Controls for the of the top device and then on Get Sensor Data a few times). Put your finger on the thermistor and watch the temperature rise





14. Read the Red Light Sensor in the app (click on Controls for the Light Level Device and then on Get Sensor Data).





Ta da! You are done with this exercise.