Chapter 7H: Workshop Exercises – Help, Hints and Code

Objective: This Addendum to the Addendum Manual contains extra hints and code for exercises that accompany Workshop Lab Chapters.

TABLE OF CONTENTS

7.20 (HINTS AND CODE) – CLUES TO HELP WITH EXERCISES

Following are extra hints for the exercises along with specific code to insert.
This is intended to help you work through the exercises and understand how project was completed to give you the steps necessary to solve your own projects.
If you choose to jump right to the end of each exercise, the final solutions are provided in the Key directory.

7.20-1 EX01HINTS - BLINK AN LED (FIRST STEP OF "BLINK AND PRINT STATE")

Add the following lines of code into app_task in BlinkLEDandPrint.c after /*** Enter Exercise Code Here ***/

```
if( GPIO_PIN_OUTPUT_HIGH == wiced_hal_gpio_get_pin_output( WICED_GPIO_PIN_LED_1 ) )
{
    wiced_hal_gpio_set_pin_output( WICED_GPIO_PIN_LED_1, GPIO_PIN_OUTPUT_LOW );
}
else
{
    wiced_hal_gpio_set_pin_output( WICED_GPIO_PIN_LED_1, GPIO_PIN_OUTPUT_HIGH );
}
```

==========

7.20-2 EX02HINTS – DEBUG PRINTING (SECOND STEP OF "BLINK AND PRINT STATE")

In the application_start in BlinkLEDandPrint.c, add the following:

```
wiced_set_debug_uart( WICED_ROUTE_DEBUG_TO_PUART );
WICED_BT_TRACE("**** CYW20819 App Start **** \n\r");
```

Note that without routing to PUART, the messages may come through the HCI UART COM Port.
Messages streaming through the HCI UART will inhibit debug and re-programming.

If you have issues downloading to the kit, follow the steps below -

Press and hold the 'Recover' button on the kit.
Press and hold the 'Reset' button on the kit.
Release the 'Reset' button.
After one second, release the 'Recover' button.

Add two lines of code to send messaged into app_task in BlinkLEDandPrint.c after /*** Enter Exercise Code Here ***/

```
    if( GPIO_PIN_OUTPUT_HIGH == wiced_hal_gpio_get_pin_output( WICED_GPIO_PIN_LED_1 ) )
    {
        wiced_hal_gpio_set_pin_output( WICED_GPIO_PIN_LED_1, GPIO_PIN_OUTPUT_LOW );
        WICED_BT_TRACE( "LED ON\r\n" );
    }
    else
    {
        wiced_hal_gpio_set_pin_output( WICED_GPIO_PIN_LED_1, GPIO_PIN_OUTPUT_HIGH );
        WICED_BT_TRACE( "LED OFF\r\n" );
        }
```

==========

7.20-3 EX03HINTS – SET LED FROM BUTTON ACTION VIA INTERRUPT

In the function wiced_bt_dev_status_t, replace the wiced_thread_t and wiced_rtos_init_thread lines with wiced_hal_gpio_configure_pin and wiced_hal_gpio_register_pin_for_interrupt lines:

From:

```
/* The stack is safely up - create a thread to test out peripherals */
wiced_thread_t* peripheral_test_thread = wiced_rtos_create_thread();

wiced_rtos_init_thread(
    peripheral_test_thread,     // Thread handle
    4,                          // Medium Priority
    "App Task",                 // Name
    app_task,                   // Function
    1024,                       // Stack space for the app_task…
    NULL );                     // Function argument (not used)
```

To:

```
/* Configure the button to trigger an interrupt when pressed */
wiced_hal_gpio_configure_pin(WICED_GPIO_PIN_BUTTON_1, ( GPIO_INPUT_ENABLE | GPIO_PULL_UP |
GPIO_EN_INT_FALLING_EDGE ), GPIO_PIN_OUTPUT_HIGH );
wiced_hal_gpio_register_pin_for_interrupt( WICED_GPIO_PIN_BUTTON_1, button_cback, 0 );
```

Replace app_task with button_cback

From:

```
/*****************************************************************
* Function Name: void app_task(uint32_t)
*****************************************************************/
void app_task( uint32_t arg )
{
  while( 1 )
  {
    /*** Enter Exercise Code Here ***/
    /* Send the thread to sleep for a specified number milliseconds*/
    wiced_rtos_delay_milliseconds(SLEEP_250MS, ALLOW_THREAD_TO_SLEEP);
  }
}
```

To:

```
/*****************************************************************
* Function Name: void button_cback( void *data, uint8_t port_pin )
*****************************************************************/
void button_cback( void *data, uint8_t port_pin )
{
      if( GPIO_PIN_OUTPUT_HIGH == wiced_hal_gpio_get_pin_output( WICED_GPIO_PIN_LED_1 ) )
      {
            wiced_hal_gpio_set_pin_output( WICED_GPIO_PIN_LED_1, GPIO_PIN_OUTPUT_LOW );
      }
      else
      {
            wiced_hal_gpio_set_pin_output( WICED_GPIO_PIN_LED_1, GPIO_PIN_OUTPUT_HIGH );
      }
}
```

Replace the initiation of app_task with an initiation for button_task

From:
~~void app_task(uint32_t);~~

To:
```
void button_cback( void *data, uint8_t port_pin );
```

==========

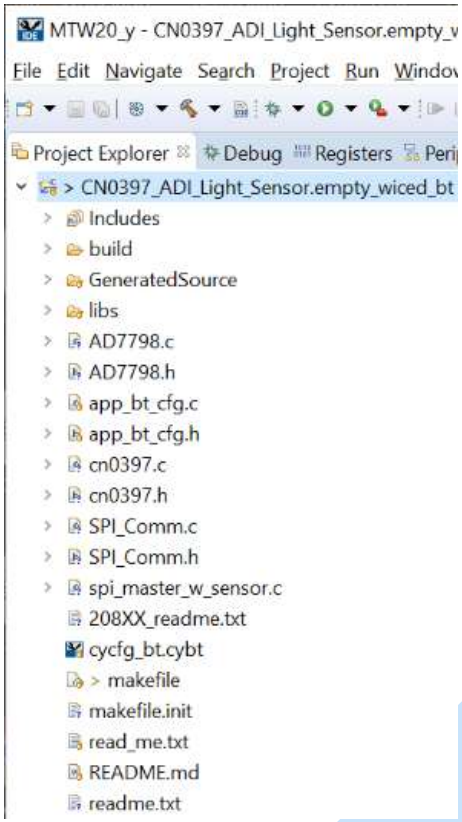
7.20-4 EX04HINTS – RGB LIGHT SENSOR W/ADI SHIELD - KEY

1. 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

2. In the Includes section of spi_master_w_sensor.c: Add include for Peripheral SPI Header File.
#include "wiced_hal_pspi.h"

3. In the Constants section of spi_maste_w_sensor.c: Define the correct pins on the Arduino Shield
/*SPI 1 defines*/
#define CLK_1 WICED_P09
#define MISO_1 WICED_P17
#define MOSI_1 WICED_P06
#define CS_1 WICED_P15

4. Also in the Constants section: Define the GPIO Configuration for SPI
/* SPI register configuration macro*/
#define GPIO_CFG(CS_1,CLK_1,MOSI_1,MISO_1)
((((UINT32)CS_1&0xff)<<24)|((UINT32)CLK_1&0xff)<<16)|(((UINT32)MOSI_1&0xff)<<8)|((UINT32)MISO_1)



5. In the Initialize_app Function; after button_state is set: Initialize the SPI Hardware

```
/* Init the SPI Hardware - MSB First and Mode 3 are required for the CN0397 */
wiced_hal_pspi_init(SPI1,
                    SPI_MASTER,
                    INPUT_PIN_PULL_UP,
                    GPIO_CFG(CS_1,CLK_1,MOSI_1,MISO_1),
                    DEFAULT_FREQUENCY,
                    SPI_MSB_FIRST,
                    SPI_SS_ACTIVE_LOW,
                    SPI_MODE_3,
                    CS_1);
```

6. In the Includes section of spi_master_w_sensor.c: Add include for the ADI Shield Header File, CN0397.h and initialize the ADI Shield.

```
#include "cn0397.h"

/* Init CN0397 and read ID */
CN0397_Init();
```

7. Read CN0397

```
/* Read the data from from the CN0397 and Display the data */
CN0397_SetAppData();
CN0397_DisplayData();
```

8. Add a single line of code (CY_RECIPE_EXTRA_LIBS+=-lgcc) to the makefile

```
################################################################################
# Paths
################################################################################
CY_RECIPE_EXTRA_LIBS+=-lgcc
# Path (absolute or relative) to the project
CY_APP_PATH=.
```

9. Program the kit, then follow instruction in a console window set to PUART port at 115200 baud.

==========

==========

7.20-5 EX05HINTS – WEIGHT SENSOR W/ADI SHIELD

To Be Developed….

==========

7.20-6 EX06HINTS – MESH 1ST NETWORK WITH DIMMABLE LED

From ModusToolbox, this is a pre-loaded Standard Example – No Edits needed.
It is suggested that device name in light_dimmable.c be set to a unique name that can be easily identified as belonging to you when attempting to “Add Device” using the Smartphone BLE_Mesh App.

To the extent, creating a mesh network with one LED is not clearly explained: Please send suggestions for Modifications to the material or Hints for this section to gcarson@arrow.com

==========

7.20-7 EX07HINTS – MESH ADD LIGHTS AND CREATE GROUPS

To increase the number of elements available to control, additional LEDs were added to a version of the BLE_Mesh_LightDimmable project specifically for the CYBT-213043-Mesh kit. The project is called BLE_Mesh_3LEDs. The CYBT-213043-Mesh kit has three LEDs, RGB, in a single component.
If NOT using the CYBT-213043-Mesh kit, code controlling those LEDs which aren’t available will need to be removed or commented out. Alternatively, the standard BLE_Mesh_LightDimmable project may be used.

This project is available in its complete form in the Templates – no changes needed.
It is suggested that device name in light_dimmable.c be set to a unique name that can be easily identified as belonging to you when attempting to “Add Device” using the Smartphone BLE_Mesh App.

To the extent, creating groups is not clearly explained: Please send suggestions for Modifications to the material or Hints for this section to gcarson@arrow.com

==========

7.20-8 EX08HINTS – MESH ON/OFF SWITCH

From ModusToolbox, this is a pre-loaded Standard Example – No Edits needed.
It is suggested that device name in on_off_switch.c be set to a unique name that can be easily identified as belonging to you when attempting to "Add Device" using the Smartphone BLE_Mesh App.

To the extent, this on/off switch exercise is not clearly explained: Please send suggestions for Modifications to the material or Hints for this section to gcarson@arrow.com

==========

7.20-9 EX09HINTS – MESH DIMMER SWITCH

From ModusToolbox, this is a pre-loaded Standard Example – No Edits needed.
It is suggested that device name in dimmer.c be set to a unique name that can be easily identified as belonging to you when attempting to "Add Device" using the Smartphone BLE_Mesh App.

To the extent, this dimmer switch exercise is not clearly explained: Please send suggestions for Modifications to the material or Hints for this section to gcarson@arrow.com

==========

Options:

1) If you're an experienced coder, attempt to add the second element on your own
   Use resources in Cypress' tools including "WICED API Reference"
2) Follow suggestions below to modify in light_dimmable.c and led_control.c
   Copy code into appropriate locations
   Replace code where there's a From: / To:
3) Review the highlighted code with comments found in the Key files:
   led_control_c_Highlighted_Changes.docx
   light_dimmable_Highlighted_Changes.docx
4) Import the Key directly, in place of the Template
   Key/Key_Ex10_Mesh_2nd_Element_Red_plus_Yel_or_Blue_LED

Update light_dimmable.c

Add a global variable to use the element_idx outside the mesh_app_attention routine

```
/*****************************************************
 *          Variables Definitions
 *****************************************************/
uint32_t global_element_index = 0;
```

Set variable in mesh_app_attention routine

```
global_element_index = element_idx;
```

3.a. Add element

```
wiced_bt_mesh_core_config_model_t   mesh_element2_models[] =
```

```
{
    WICED_BT_MESH_MODEL_LIGHT_LIGHTNESS_SERVER,
};
#define MESH_APP_NUM_MODELS_GREEN  (sizeof(mesh_element2_models) / sizeof(wiced_bt_mesh_core_config_model_t))
```

3.a. Add element structure

```
    {
        .location = MESH_ELEM_LOC_MAIN,                                 // location description as defined in the GATT Bluetooth
Namespace Descriptors section of the Bluetooth SIG Assigned Numbers
        .default_transition_time = MESH_DEFAULT_TRANSITION_TIME_IN_MS,  // Default transition time for models of the element in
milliseconds
        .onpowerup_state = WICED_BT_MESH_ON_POWER_UP_STATE_RESTORE,     // Default element behavior on power up
        .default_level = 0,                                             // Default value of the variable controlled on this
element (for example power, lightness, temperature, hue...)
        .range_min = 1,                                                 // Minimum value of the variable controlled on this
element (for example power, lightness, temperature, hue...)
        .range_max = 0xffff,                                            // Maximum value of the variable controlled on this
element (for example power, lightness, temperature, hue...)
        .move_rollover = 0,                                             // If true when level gets to range_max during move
operation, it switches to min, otherwise move stops.
        .properties_num = 0,                                            // Number of properties in the array models
        .properties = NULL,                                             // Array of properties in the element.
        .sensors_num = 0,                                               // Number of sensors in the sensor array
        .sensors = NULL,                                                // Array of sensors of that element
        .models_num = MESH_APP_NUM_MODELS_GREEN,                        // Number of models in the array models
        .models = mesh_element2_models,                                 // Array of models located in that element. Model data is
defined by structure wiced_bt_mesh_core_config_model_t
    },
```

Note: There's line from light_dimmable.c template that's not in the key:

```
        .max_lpn_num          = 4           // Max number of Low Power Nodes with established friendship. Must be > 0 if Friend feature is supported.
```

Extend variable into an array:

From:

```
uint8_t last_known_brightness = 0;
```

To:

```
uint8_t last_known_brightness[2] = {0};
```

Change Device Name to something unique:

From:

```
wiced_bt_cfg_settings.device_name = (uint8_t *)"Dimmable Light";
wiced_bt_cfg_settings.gatt_cfg.appearance = APPEARANCE_LIGHT_CEILING;
```

To:

```
wiced_bt_cfg_settings.device_name = (uint8_t *)"2 Elements Key";
wiced_bt_cfg_settings.gatt_cfg.appearance = APPEARANCE_LIGHT_CEILING;
```

Stop passing variable to LED Control Initialization:

From:

```
led_control_init(LED_CONTROL_TYPE_LEVEL);
```

To:

```
led_control_init();
```

Pass new “global_element_index” variable when initializing timer:

From:

```
wiced_init_timer(&attention_timer, attention_timer_cb, 0, WICED_SECONDS_PERIODIC_TIMER);
```

To:

```
wiced_init_timer(&attention_timer, attention_timer_cb, global_element_index, WICED_SECONDS_PERIODIC_TIMER);
```

Pass split variable of “RED” and “GREEN” from “MESH_LIGHT_LIGHTNESS_SERVER_ELEMENT_INDEX” to separate light initializations:

From:

```
// 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);
```

To:

```
// 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(RED, mesh_app_message_handler, is_provisioned);
wiced_bt_mesh_model_light_lightness_server_init(GREEN, mesh_app_message_handler, is_provisioned);
```

5.a. Extend commands setting brightness level from one element to multiple (occurs multiple places):

From:

```
led_control_set_brighness_level(last_known_brightness);
```

To:

```
led_control_set_brighness_level(last_known_brightness[element_idx], element_idx);
```

Or:

```
led_control_set_brighness_level(last_known_brightness[global_element_index], (uint8_t)arg);
```

From:

```
attention_brightness = (last_known_brightness != 0) ? 0 : 100;
```

To:

```
attention_brightness = (last_known_brightness[element_idx] != 0) ? 0 : 100;
```

From:

```
led_control_set_brighness_level(attention_brightness);
```

To:

```
led_control_set_brighness_level(attention_brightness, element_idx);
```

Or:

```
led_control_set_brighness_level(attention_brightness, (uint8_t)arg);
```

Update led_control.c

Split PWM channels:

From:

```
#define PWM_CHANNEL          PWM0
```

To:

```
#define PWM_CHANNELR          PWM0
#define PWM_CHANNELG          PWM1
```

Split led pins:

From:

```
wiced_bt_gpio_numbers_t led_pin = WICED_GPIO_PIN_LED_2;
```

To:

```
wiced_bt_gpio_numbers_t led_pin_r = WICED_GPIO_PIN_LED_2;
wiced_bt_gpio_numbers_t led_pin_g = WICED_GPIO_PIN_LED_1;
```

Update LED Control Initialization:

From:

```
void led_control_init(uint8_t control_type)
{
    pwm_config_t pwm_config;

    if (control_type == LED_CONTROL_TYPE_ONOFF)
        return;

    else if (control_type == LED_CONTROL_TYPE_LEVEL)
    {
        /* configure PWM */
#ifdef CYW20719B1
        wiced_hal_pwm_configure_pin(led_pin, PWM_CHANNEL);
#endif

#if ( defined(CYW20819A1) || defined(CYW20735B1) )
        wiced_hal_gpio_select_function(WICED_GPIO_PIN_LED_2, WICED_PWM0);
#endif
        wiced_hal_aclk_enable(PWM_INP_CLK_IN_HZ, ACLK1, ACLK_FREQ_24_MHZ);
        wiced_hal_pwm_get_params(PWM_INP_CLK_IN_HZ, 0, PWM_FREQ_IN_HZ, &pwm_config);
        wiced_hal_pwm_start(PWM_CHANNEL, PMU_CLK, pwm_config.toggle_count, pwm_config.init_count, 1);
    }
    else if (control_type == LED_CONTROL_TYPE_COLOR)
    {
        // TBD
```

```
    }
}
```

To:

```
void led_control_init(void)
{
    pwm_config_t pwm_config;

    /* configure PWM */
#ifdef CYW20719B1
    wiced_hal_pwm_configure_pin(led_pin_r, PWM_CHANNELR);
    wiced_hal_pwm_configure_pin(led_pin_g, PWM_CHANNELG);
#endif

#ifdef CYW20819A1
    wiced_hal_gpio_select_function(WICED_GPIO_PIN_LED_2,   WICED_PWM0);
    wiced_hal_gpio_select_function(WICED_GPIO_PIN_LED_1, WICED_PWM1);

#endif
    wiced_hal_aclk_enable(PWM_INP_CLK_IN_HZ, ACLK1, ACLK_FREQ_24_MHZ);
    wiced_hal_pwm_get_params(PWM_INP_CLK_IN_HZ, 0, PWM_FREQ_IN_HZ, &pwm_config);
    wiced_hal_pwm_start(PWM_CHANNELR, PMU_CLK, pwm_config.toggle_count, pwm_config.init_count, 1);
    wiced_hal_pwm_start(PWM_CHANNELG, PMU_CLK, pwm_config.toggle_count, pwm_config.init_count, 1);

}
```

Update LED brightness control

From:

```
void led_control_set_brighness_level(uint8_t brightness_level)
{
    pwm_config_t pwm_config;

    WICED_BT_TRACE("set brightness:%d\n", brightness_level);

    // ToDo.  For some reason, setting brightness to 100% does not work well on 20719B1 platform. For now just use 99% instead of 100.
    if (brightness_level == 100)
        brightness_level = 99;
```

```
    wiced_hal_pwm_get_params(PWM_INP_CLK_IN_HZ, brightness_level, PWM_FREQ_IN_HZ, &pwm_config);
    wiced_hal_pwm_change_values(PWM_CHANNEL, pwm_config.toggle_count, pwm_config.init_count);
}
```

To:

```
void led_control_set_brighness_level(uint8_t brightness_level, uint8_t element_idx)
{
    pwm_config_t pwm_config;

    WICED_BT_TRACE("set brightness:%d\n", brightness_level);

    // ToDo.  For some reason, setting brightness to 100% does not work well on 20719B1 platform. For now just use 99% instead of
100.
    if (brightness_level == 100)
        brightness_level = 99;

    wiced_hal_pwm_get_params(PWM_INP_CLK_IN_HZ, brightness_level, PWM_FREQ_IN_HZ, &pwm_config);

    switch(element_idx)
    {
    case RED:
        wiced_hal_pwm_change_values(PWM_CHANNELR, pwm_config.toggle_count, pwm_config.init_count);
     break;
    case GREEN:
        wiced_hal_pwm_change_values(PWM_CHANNELG, pwm_config.toggle_count, pwm_config.init_count);
     break;
    }

}
```

Add routine to turn LED on or off:

```
/*
 * Turn LED on or off
 */
void led_control_set_onoff(uint8_t onoff_value)
```

```
{
    WICED_BT_TRACE("set onoff:%d\n", onoff_value);

    if (onoff_value == 1)           // led is on
    {
        wiced_hal_gpio_configure_pin(led_pin, GPIO_OUTPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);
    }
    else if (onoff_value == 0)      // led is off
    {
        wiced_hal_gpio_configure_pin(led_pin, GPIO_OUTPUT_ENABLE, GPIO_PIN_OUTPUT_HIGH);
    }
}
```

Update led_control.h

From:

```
#define LED_CONTROL_TYPE_ONOFF   0
#define LED_CONTROL_TYPE_LEVEL   1
#define LED_CONTROL_TYPE_COLOR   2

/*
 * Initialize LED control of a specific type
 */
void led_control_init(uint8_t control_type);

/*
 * Set LED brightness level 0 to 100%
 */
void led_control_set_brighness_level(uint8_t brightness_level);

/*
 * Turn LED on or off
 */
void led_control_set_onoff(uint8_t onoff_value);
```

To:

```
typedef enum {
	RED,
	GREEN,
} led_control_t;

void led_control_init(void);
void led_control_set_brighness_level(uint8_t brightness_level, uint8_t element_idx);
```

Change to modus.mk

The new version of modus.mk is missing the readme.txt and has a continuation "\" without anything following. It is not known what or if this change affects. The project appears to work the same either way.

From:

```
CY_APP_SOURCE =  \
  ./light_dimmable.c \
  ./led_control.c \
  ./led_control.h \
  ./readme.txt
```

To:

```
CY_APP_SOURCE =  \
  ./light_dimmable.c ./led_control.c ./led_control.h \
```

==========

7.20-11 EX11HINTS – MESH TEMPERATURE SENSOR

From ModusToolbox, this is a pre-loaded Standard Example – No Edits needed.
It is suggested that device name in sensor_temperature.c be set to a unique name that can be easily identified as belonging to you when attempting to "Add Device" using the Smartphone BLE_Mesh App.

To the extent, this temperature sensor exercise is not well explained: Please send suggestions for Modifications to the material or Hints for this section to gcarson@arrow.com

==========

7.20-12 EX12HINTS – MESH ADI RED SENSOR + TEMPERATURE SENSOR

Options:

1) If you're a highly experienced coder; Start from the Temperature Sensor Code Example and the Red Sensor element on your own.
   Start with the embedded example BLE_Mesh_SensorTemperature
   Use resources in Cypress' tools including "WICED API Reference"
   Use resources from ADI for CN0397
2) If you're an experienced coder; Start with the Template that has suggestions within the file sensor_temperature.c in the Template/Ch06/Mesh_Temp_plus_Red_Sensor
   Hint: Search for "/*** ADI CN0397"
3) If you prefer to start with a working example, then reverse engineer; Import the Key directly Key/Key_Ex12_Mesh_Temp_plus_Red_Sensor
   Search for "/*** ADI CN0397" to see all changes made to the base BLE_Mesh_SensorTemperature example.