PSoC™ Roadshow Technical Workshop

Day 2 – PSoC™ 6 and ModusToolbox™

Objective: Create, debug, and program a dual core application featuring a Wi-Fi Access Point and HTTP Server hosted on the PSoC™ 6 MCU running FreeRTOS. Workshop will leverage the Eclipse IDE for ModusToolbox™, project creator, and library manager.

Attendees will understand the basic development flow of ModusToolbox, how to add additional middleware libraries and update the Makefile build system to successfully integrate the included libraries, and where to find key documentation to enable efficient development.

Prerequisites and Important Information

Installed Software:

- ModusToolbox™ Tools Package v3.0

Hardware being used:

- PSoC™ 6 Wi-Fi BT prototyping kit (CY8CPROTO-062-4343W)

Provided workshop collateral (Embedded World Workshop folder on desktop):

- **ModusToolbox Workshop Session 2 Lab Guide.pdf** (this document)

- Files to be included in workshop project
    - **html_web_page.h** – contains embedded webpage html
    - **http_server.c / http_server.h** – include file for http-server, derived from mtb-example-anycloud-wifi-web-server

- Files to be referenced during workshop
    - **main-cm4-freertos-template.c** – reference file containing initial CM4 main.c content
    - **PSoC_6_Code_Snippets.c** – helper file containing various code snippets to be added to project

A few key points regarding PSoC™ 62, Dual Core, and ModusToolbox v3.0

- The Cortex CM0 is the initial core to boot. The CM0 application calls a **Cy_SysEnableCM4(…)** function to enable the CM4 core. This function requires the offset address of the CM4 Vector Offset (pointer to start of flash)

- For the majority of the current ModusToolbox Code Examples (single core), only the CM4 is leveraged.
  - o For these Code Examples, the CM0 runs a **default image**, that powers on the CM4 and enters a deep sleep low power mode.

- For Dual Core projects there is a parent level **Application** structure that contains the common BSP, top level Makefile, and individual **Project** structures for the CM0 and CM4 respectively.
  - o **Building** always occurs at the **Application** level (meaning both the CM0and CM4 project is compiled)
  - o **Programming** always occurs at the **Application** level (the CM0 and CM4 images are combined into a single binary object for programming)
  - o **Debugging** can be done on a **specific core** (CM0 or CM4) or at the **Application** level, with the CM0 initiating the Debug session and then attaching to the CM4 project

- The **CYBSP_INIT** function is called within the CM0 and CM4 projects
  - o For single core projects (with the default CM0 firmware) the BSP initialization occurs on the CM4
  - o For dual core projects the BSP initialization occurs on the CM0

Section 1 – Creating a new dual core application and extending CM0 Flash size

1. Open **Eclipse IDE for ModusToolbox v3.0**
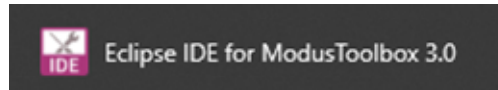
2. Create a New Application

   - Create new Workspace directory within User directory (example c:\users\username\workshop2)

   - From Quick Panel select **New Application**, this action will launch the Project Creator

   - Select the board support package: **PSoC™ 6 BSP** -> **CY8CPROTO-062-4343W**

   - Press **Next**

   - Check the template application: **Getting Started -> Dual-CPU Empty PSoC6 App**, and provide a unique name (example **MyApp**)

   - Select **Create** to clone the application, create the eclipse project, and import the required BSP files and libraries

3. Open **PSoC_6_Code_Snippets.c** file in Eclipse IDE Code Editor (or text editor of choice)

   Note: this file includes many of the functions that are to be added into the project and will be used to simplify copying the functions directly into the application.

4. Review Application and Sub-Project folder structure within Eclipse IDE Project Explorer

5. Extend FLASH size available to CM0 project

   - Open CM0 Linker Script (located within the BSP folder)
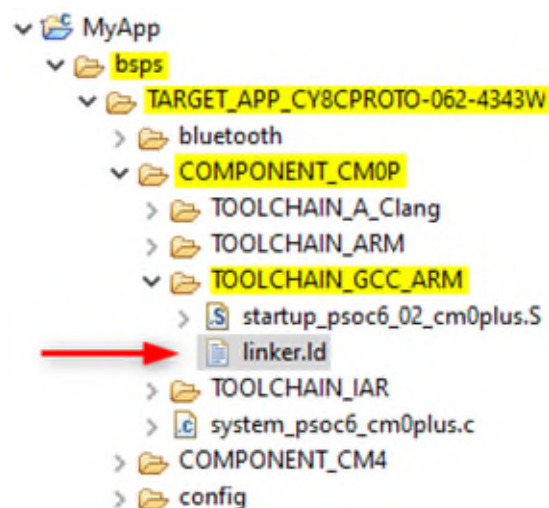
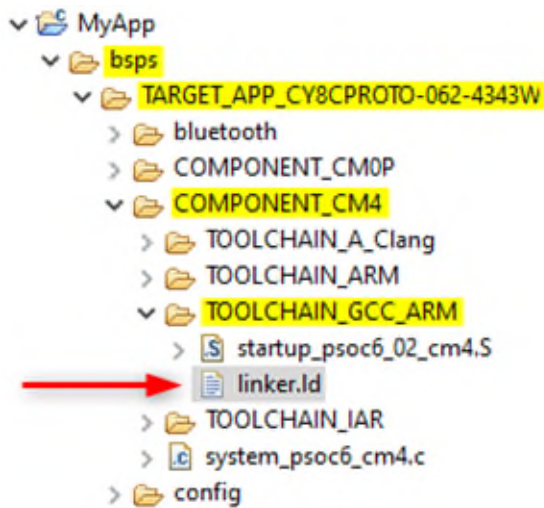   - Locate the **flash** region settings (line 67) within the MEMORY section

   - Change flash size to 0x8000

     ```
     flash              (rx)    : ORIGIN = 0x10000000, LENGTH = 0x8000
     ```

   - **Save** and close CM0 Linker Script file

6. Update CM4 flash size offset to account for increase in CM0 Flash

- Open CM4 Linker Script (located within the BSP folder)

- Locate **FLASH_CM0P_SIZE** parameter (line 51) and change to 0x8000

```
FLASH_CM0P_SIZE  = 0x8000;
```

- **Save** and close CM4 Linker Script file

7. Update CM0 Makefile with M4 Application Address

- Open **MyApp.proj_cm0p** -> **Makefile**
- **Update** CY_CORTEX_M4_APPL_ADDR define (line 76)

```
DEFINES+=CY_CORTEX_M4_APPL_ADDR=CY_FLASH_BASE+0x8000U
```

- **Save** and Close CM0 Makefile

8. Enable use of HAL Library on CM0

- Open **MyApp.proj_cm0p** -> **main.c**
- Add HAL header file

```
#include "cyhal.h"
```

- **Save** and close main.c

Section 2 – Converting the CM4 bare metal application to a FreeRTOS application

1. Highlight the parent application (MyApp) and open the **Library Manager** from the Quick Panel

2. Add **FreeRTOS** and **Retarget-IO** libraries to project
   - Highlight the "**proj_cm4 Libraries**" entry from the central panel and select the **Add Library** button
   - Place a check mark next to **Core** -> **freertos** and **Peripheral** -> **retarget-io**
   - Select **OK** to return to the main view and review changes.
   - Select **Update** to clone the requested middleware libraries and any dependencies into project
   - Select **Close** once update is completed

3. Integrate FreeRTOS into application, following guidance in the FreeRTOS Readme
   - Open mtb_shared / freertos / release-<version> / **README.md**
   - Follow steps in Quick Start Section
     - Add **COMPONENTS+=FREERTOS**
     - Copy, Paste, and Edit **FreeRTOSConfig.h** file, per readme guidance

4. Replace CM4 **main.c** file content with content from **main-cm4-freertos-template.c**
   - Open **main-cm4-freertos-template.c** in Eclipse IDE Code editor
   - Copy and Paste content, replacing all content in the existing CM4 project's main.c file

5. Review content in Retarget-IO Readme
   - Open mtb_shared / retarget-io / release-<version> / **README.md**
   - Follow steps in RTOS Integration and Enabling Conversion of '\n'
     - Add **DEFINES+=CY_RTOS_AWARE CY_RETARGET_IO_CONVERT_LF_TO_CRLF**

6. Open a Serial Terminal within the Eclipse IDE for ModusToolbox
   - Ensure development board is connected to PC using the USB cable
   - Select the Terminal panel (in lower section of IDE)
   - Select the Open a Terminal icon 🖥 (or press Ctrl-Alt-Shift-T)
     - Choose terminal: **Serial**
     - **Configure** Serial Port (likely the last entry in the list)
     - Baud rate: **115200** (default)
   - Recommended: Relocate the Terminal panel to a new location within the IDE

7. **Build and Program** the application to CY8CPROTO-062-4343W board
   - Ensure correct application project is selected (the active project within Eclipse)
   - Select the appropriate **<Project Name> Program Application (KitProg3_MiniProg4)** action from the Quick Panel. Observe the output in the terminal window

Section 3 – Leveraging Inter-Processor Communication (IPC) across cores

1. Review HAL documentation for IPC (Inter-Processor Communication) APIs

   - Open Hardware Abstraction Layer (HAL) from the Quick Panel API Documentation section

   - Navigate to **HAL API Reference** -> **HAL Drivers** -> **IPC (Inter-Processor Communication)**

   - Code used in this workshop was derived from **Snippet 2**

2. Edit CM0 application (main.c) to initialize IPC message queues, code to be copied from PSoC_6_Code_Snippets.c file.   Paste all code just before the Cy_SysEnableCM4

   - Reference code declares variable for a message queue and a message payload (led value)

   - Reference code creates pointers for a queue pool and queue handle, allocating space for these pointers within shared memory

   - Reference code initializes configures the shared queue structure and initialized the local message queue

3. Remove DeepSleep function within main FOR loop

4. Add IPC function to 'get' queue and store message payload into LED value variable (within FOR loop)

   ```
   cyhal_ipc_queue_get(&cm0_msg_queue, &cm0_led_value, CYHAL_IPC_NEVER_TIMEOUT);
   ```

5. Use HAL APIs for GPIO to initialize a GPIO to drive and LED and update the LED based on the received message

   - Initialize a GPIO as an OUTPUT with STRONG DRIVE using the following function:

   ```
   cyhal_gpio_init(CYBSP_USER_LED, CYHAL_GPIO_DIR_OUTPUT,
                   CYHAL_GPIO_DRIVE_STRONG, cm0_led_value);
   ```

   - Update the GPIO output value (after 'getting' the message queue) using the following function:

   ```
   cyhal_gpio_write(CYBSP_USER_LED, cm0_led_value);
   ```

6. Edit CM4 application (main.c) to retrieve message queue handle, code to be copied from PSoC_6_Code_Snippets.c file

   - Reference code declares a variable for the message queue object and retrieves the message queue handle from shared memory, this function can be called within the main_task.
     Look for `// TODO Get IPC Message Queue Handle`

7. Add IPC function to 'put' queue message with payload (new LED value). This function should be added within the main_task FOR loop (after the IF / ELSE statement)
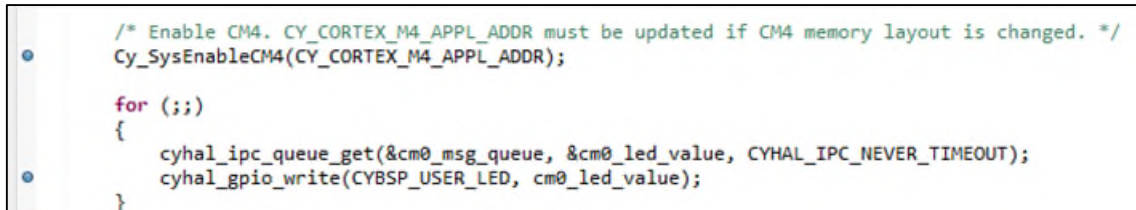
   ```
   cyhal_ipc_queue_put(&cm4_msg_queue, &cm4_led_value, CYHAL_IPC_NEVER_TIMEOUT);
   ```

8. **Build** the application

   - Ensure correct application project is selected (the active project within Eclipse)

   - Select the **Build Application** action from the Quick Panel

   - Confirm no build errors

Section 4 – Debugging a Dual Core application with ModusToolbox v3.0

1. Arrange the source files such that the CM4 and CM0 main.c files are docked side-by-side

2. Add a breakpoint to the **CM0** project (main.c)

   - Double-click the left margin of the file to add a breakpoint to the line of code that Enables the CM4 and the line that writes the GPIO value
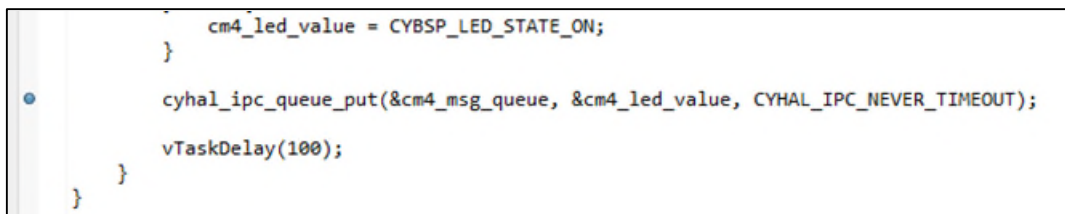
   ```
   /* Enable CM4. CY_CORTEX_M4_APPL_ADDR must be updated if CM4 memory layout is changed. */
   Cy_SysEnableCM4(CY_CORTEX_M4_APPL_ADDR);

   for (;;)
   {
       cyhal_ipc_queue_get(&cm0_msg_queue, &cm0_led_value, CYHAL_IPC_NEVER_TIMEOUT);
       cyhal_gpio_write(CYBSP_USER_LED, cm0_led_value);
   }
   ```

3. Add a breakpoint to the **CM4** project (main.c)

   - Double-click the left margin of the file to add a breakpoint to the line of code that sends ('puts') the message queue

   ```
           cm4_led_value = CYBSP_LED_STATE_ON;
       }

       cyhal_ipc_queue_put(&cm4_msg_queue, &cm4_led_value, CYHAL_IPC_NEVER_TIMEOUT);

       vTaskDelay(100);
       }
   }
   ```

4. Start the MultiCore debug session

   - Ensure that the top level (MyApp) application folder is selected within the Eclipse Project Explorer

   - Within the Quick Panel, launch the **Debug MultiCore (KitProg3_MiniProg4)** configuration

   ```
   ▼ Launches
      ▶ MyApp Program Application (JLink)
      ▶ MyApp Program Application (KitProg3_MiniProg4)
      ⊞ MyApp Debug MultiCore (JLink)
      ⊞ MyApp Debug MultiCore (KitProg3_MiniProg4)
      ⚒ Generate Launches for MyApp
   ```

5. MultiCore Debug (and Attach) process

   - The MultiCore debug session starts by launching the CM0 project (the initial core to boot)

     o Within the Debug Panel, the CM0 breaks at the initial breakpoint at the start of main

     o The CM4 project does not yet have an active (attached) thread

   - After executing the **Cy_SysEnableCM4()** function, the CM4 will be visual within the Debug Panel, and will hit an initial breakpoint at the start of main.

   - **Resuming** the project will hit the next two successive breakpoint on the two cores. The active project can be selected within the Debug Panel.

6. When done, ensure that core sessions are **terminated** (recall the CM0 is the primary debug session). With the sessions terminates, return to the **Project Explorer**.

Section 5 – Creating an Access Point with the Wi-Fi Core Library Bundle

1. Highlight the parent application (MyApp) and open the **Library Manager** from the Quick Panel

2. Add **Wi-Fi Core library bundle** to project
   - Highlight the "**proj_cm4 Libraries**" entry from the central panel and select the **Add Library** button
   - Place a check mark next to **Wi-Fi -> wifi-core-freertos-lwip-mbedtls**
   - Select **OK** to return to the main view and review changes.
   - Select **Update** to clone the requested middleware libraries and any dependencies into project
   - Select **Close** once update is completed

3. Integrate Wi-Fi Core Library bundle into application, following guidance in the Readme
   - Open mtb_shared / wifi-core-freertos-lwip-mbedtls / release-<version> / **README.md**
   - Complete steps **1, 2, 3, & 6** within the README's Quick Start section
     - Copy lwipopts.h and mbedtls_user_config.h files
     - Add **DEFINES+=MBEDTLS_USER_CONFIG_FILE='"mbedtls_user_config.h"'**
     - Add **DEFINES+=CYBSP_WIFI_CAPABLE**
     - Ensure correct COMPONENTS are included: **FREERTOS LWIP MBEDTLS**

4. Add code to configure and start an access point based on code snippets in API docs
   - Open **proj_cm4 API Documentation** -> **Wi-Fi Connection Manager Library: Overview** document
   - Locate and copy all code from **Snippet 8: Clients connected to the Soft AP**
   - Paste copy at bottom of CM4 main.c file
   - Update WIFI_SSID_AP & WIFI_KEY_AP to something unique
   - Correct coding error associated with **cy_wcm_init** status check
     - Should be: **if(result != CY_RSLT_SUCCESS)** *(Critical Step)*
   - Remove last several lines from **snippet_wcm_ap_get_client**
     - Remove line starting at **cy_rtos_delay_milliseconds(…)** through last **printf**

5. Add necessary header files based on functions within the documentation code snippet

   - Code snippet includes functions that start with **cy_wcm_xxxx** and **cy_nw_xxxx**

   - Add the following header files to include these functions
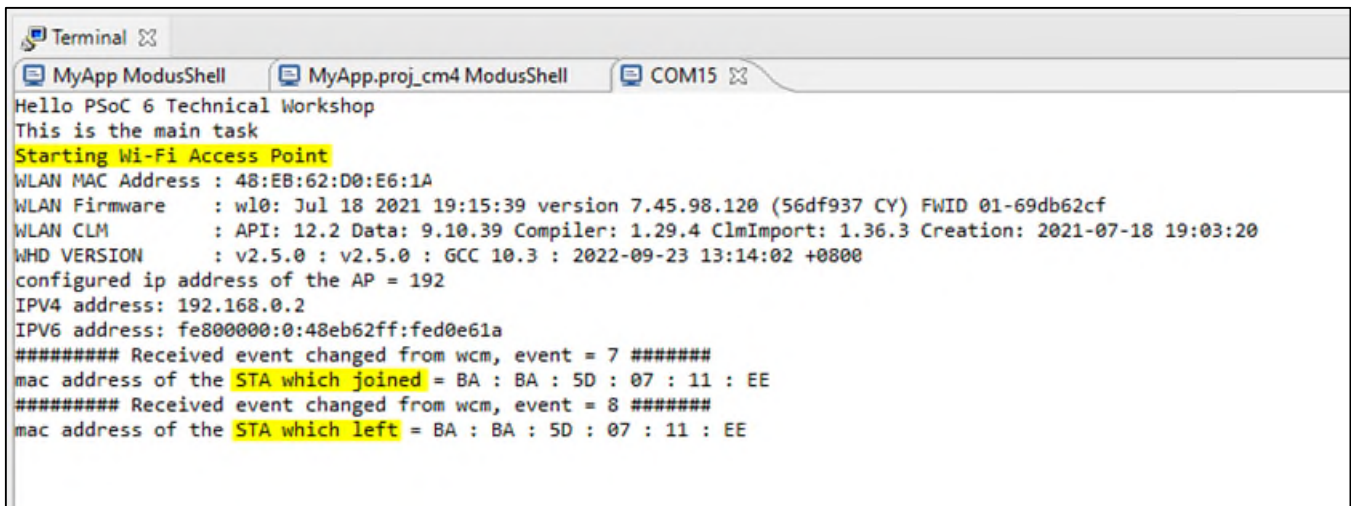
     ```
     #include "cy_wcm.h"
     #include "cy_nw_helper.h"
     ```

6. Call function to start access point from main_task

   - Add function prototype for **snippet_wcm_ap_get_client** function at top of main.c

   - Locate `// TODO Create Access Point` within the main_task of main.c

     o Add **printf("Starting Wi-Fi Access Point \n");**

     o Add **snippet_wcm_ap_get_client();**

7. Compile and Flash application

   - Ensure CM4 project is selected (the active project within Eclipse)

   - Select the appropriate **<Project Name>.cm4 Program (KitProg3_MiniProg4)** launch action from the Quick Panel

   - Observe results in terminal, using laptop or mobile device to connect to Access Point

```
Terminal ⊠
  MyApp ModusShell    MyApp.proj_cm4 ModusShell    COM15 ⊠
Hello PSoC 6 Technical Workshop
This is the main task
Starting Wi-Fi Access Point
WLAN MAC Address : 48:EB:62:D0:E6:1A
WLAN Firmware    : wl0: Jul 18 2021 19:15:39 version 7.45.98.120 (56df937 CY) FWID 01-69db62cf
WLAN CLM         : API: 12.2 Data: 9.10.39 Compiler: 1.29.4 ClmImport: 1.36.3 Creation: 2021-07-18 19:03:20
WHD VERSION      : v2.5.0 : v2.5.0 : GCC 10.3 : 2022-09-23 13:14:02 +0800
configured ip address of the AP = 192
IPV4 address: 192.168.0.2
IPV6 address: fe800000:0:48eb62ff:fed0e61a
######### Received event changed from wcm, event = 7 #######
mac address of the STA which joined = BA : BA : 5D : 07 : 11 : EE
######### Received event changed from wcm, event = 8 #######
mac address of the STA which left = BA : BA : 5D : 07 : 11 : EE
```

Section 6 – Running an embedded HTTP Server supporting RESTful APIs

1. Highlight the parent application (MyApp) and open the **Library Manager** from the Quick Panel

2. Add HTTP Server library to project
   - Highlight the "**proj_cm4 Libraries**" entry from the central panel and select the **Add Library** button
   - Place a check mark next to **Middleware -> http-server**
   - Select **OK** to return to the main view and review changes.
   - Select **Update** to clone the requested middleware libraries and any dependencies into project
   - Select **Close** once update is completed

3. Integrate HTTP Server into application, following guidance in the http-server Readme
   - Open mtb_shared / http-server / release-<version> / **README.md**
   - Follow guidance in Quick Start -> ModusToolbox section
     - Ensure correct COMPONENTS are included:
       FREERTOS LWIP MBEDTLS SECURE_SOCKETS
     - Add **DEFINES+=MAX_NUMBER_OF_HTTP_SERVER_RESOURCES=**
     - Set MAX_NUMBER_OF_HTTP_SERVER_RESOURCES=**10**

4. Import reference header files into project workspace
   - Locate provided **html_web_page.h** / **http_server.h** / **http_server.c** files
   - Drag-and-Drop files into Eclipse IDE's Project Explorer view (into **MyProject** folder)
   - Confirm that files will be "**Copied**" into project

5. Include the required header files into the main.c file

   ```
   #include "http_server.h"
   ```

   Note: the html_web_page.h is included for us within the http_server header file and does not need to be included in the main.c file directly.

   The remaining HTTP-Server content is derived closely from the mtb-example-anycloud-wifi-web-server code example, but has been simplified for this workshop.

6. Add code to configure and start the HTTP Server, code to be copied from PSoC_6_Code_Snippets.c file
   - Locate `// TODO Create HTTP Server` within the main_task of main.c
     - Add **printf("Starting HTTP Server");**
     - Add **start_http_server();**

Section 7 – Responding to RESTful APIs with RTOS aware commands

1. Update the global LED variable to respond to the web page (within **softap_resource_handler** function within **http_server.c**)

   - Locate `// TODO Code when LED Turned On`
     - Add **cm4_led_value = CYBSP_LED_STATE_ON;**

   - Locate `// TODO Code when LED Turned Off`
     - Add **cm4_led_value = CYBSP_LED_STATE_OFF;**

   - Locate `// TODO Notify Main task to update LED`
     - **xTaskNotifyGive(main_task_handle);**

2. Update main_task FOR loop (**main.c**) to process LED value updates

   - Wait for task notification prior to sending the message queue
     - Add **ulTaskNotifyTake(pdTRUE, portMAX_DELAY);**

   - Remove the two functions that set the cm4_led_value within the IF / ELSE statement (keep the IF / ELSE statement itself, just remove the contents within the brackets { }

3. Compile and Flash application

   - Ensure CM4 project is selected (the active project within Eclipse)

   - Select the appropriate **<Project Name>.cm4 Program (KitProg3_MiniProg4)** launch action from the Quick Panel

   - Observe results in terminal, using laptop or mobile device to connect to Access Point

4. Connect to embedded HTTP Server

   - Monitor the serial terminal to ensure that the HTTP Server has started

   - Use a browser on the connected device (laptop or mobile device) and browse to the address **http://192.168.0.2**

   - Selecting the LED On / LED Off should result in the board LED turning on and off
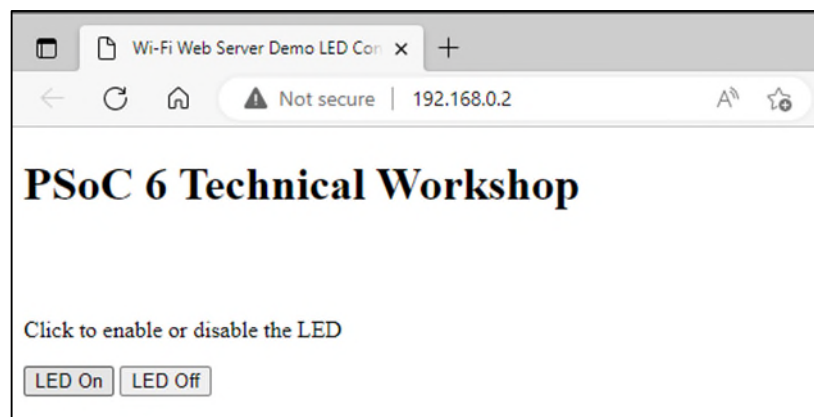
Section 8 – Bonus Content: Extending the embedded HTTP Server with Server-Side Events

1. Add Server-Side Event handler function to **http_server.c**, code to be copied from
   PSoC_6_Code_Snippets.c file
   - Reference code adds a handler function that is called when updating the dynamic event content
   - Copy the following function
     - **process_sse_handler**
   - Locate `// TODO Add Server Side Event Handler function here...` and paste the copied code

2. Register the server-side event handler within the **configure_http_server** function, code to be copied from
   PSoC_6_Code_Snippets.c file
   - Reference code registered the handler function added in the previous step with the HTTP server
   - Copy the code snippet related to event registration (PSoC_6_Socde_Snippets.c file)
   - Locate `// TODO Register the Server Side Event Handler here...` and paste the copied code

3. Stream content to the web client when the LED status is updated.
   - Declare a character array for holding the event update text within the **main_task** function
     ```c
     char http_response[64] = {0};
     ```
   - Print the desired message into the character array within the appropriate condition of the IF-ELSE statement of
     the main_task FOR loop
     ```c
     sprintf( http_response, "The LED is turned ON." );

     sprintf( http_response, "The LED is turned OFF." );
     ```
   - Add function call to stream SSE payload content. Add function call after the queue put function within the
     main_task FOR loop.
     ```c
     send_http_stream(&http_response);
     ```
     Note: send_http_stream function is defined in the http_server.c file for reference.

4. Compile and Flash application as done in previous steps

5. Connect to embedded HTTP Server
   - Monitor the serial terminal to ensure that the HTTP Server has started
   - Use a browser on the connected device (laptop or mobile device) and browse to the address **http://192.168.0.2**
   - Selecting the LED On / LED Off should result the board LED turning on and off and the web page updating with
     the latest status