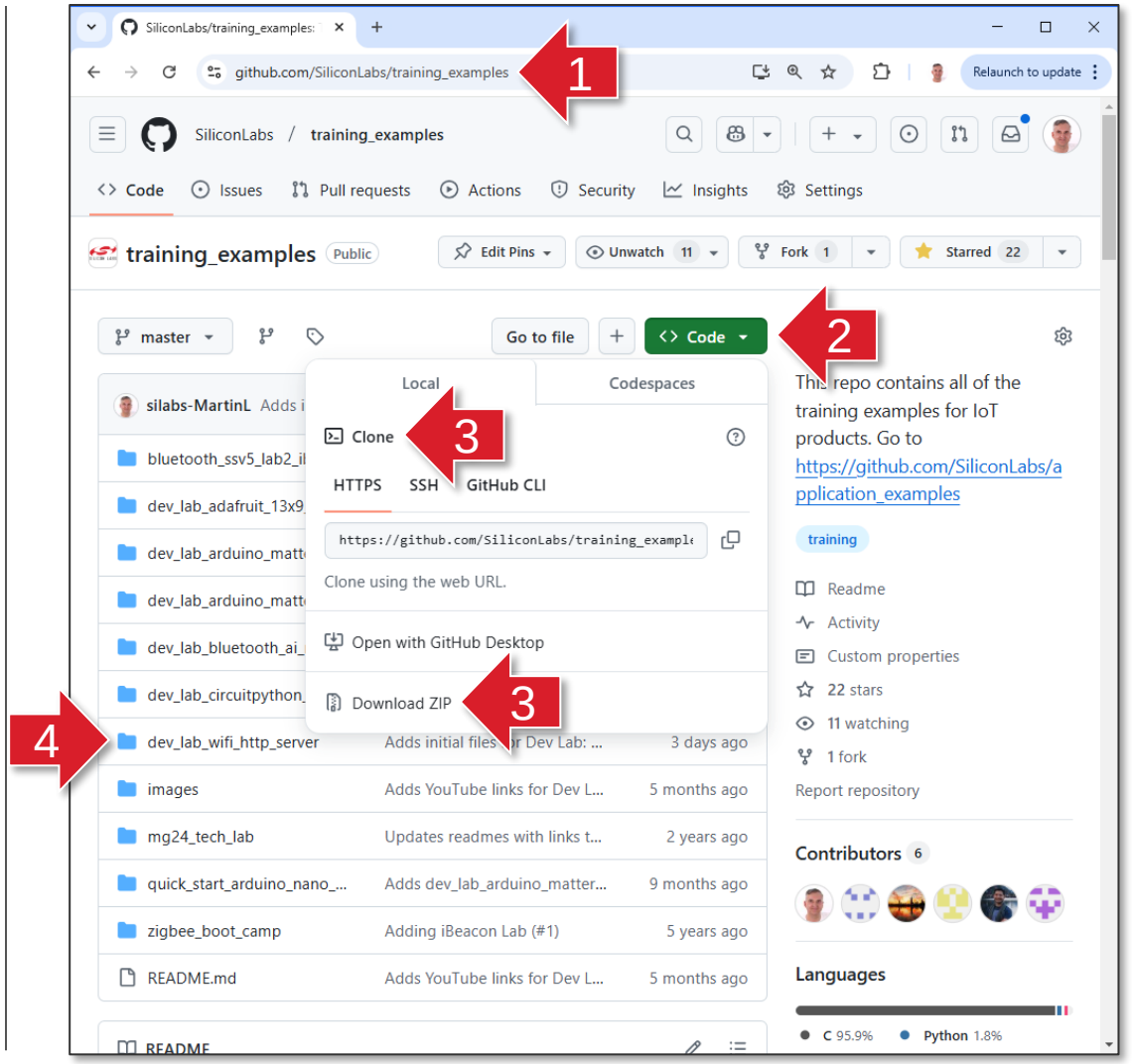

SILICON LABS
SiWG917 Sleeping Wi-Fi HTTP Server
Martin Looker (Silicon Labs)
7th May 2025
v3.5
82%
Humidity
18°
Temperature

In this workshop:

- Connect and setup SiWG917 boards in the Simplicity Studio v5 IDE
- Create, build and run the Wi-Fi HTTP Server example application
- Adapt the Wi-Fi HTTP Server application to display button states in a browser
- Make the joining and rejoining process robust
- How to measure power use with Simplicity Studio’s Energy Profiler
- Apply sleep modes to reduce power consumption
- How to use software APIs to construct Wi-Fi applications for the SiWG917

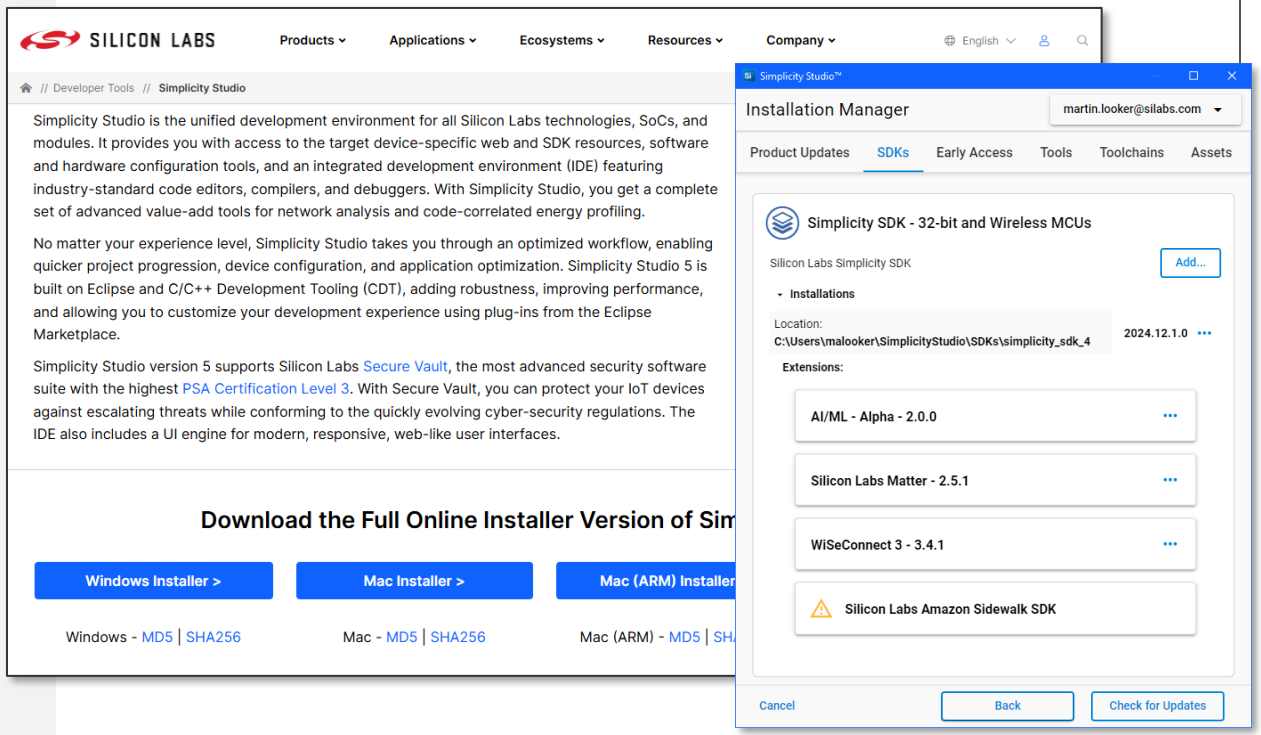
GitHub Silicon Labs Training Examples

- The Silicon Labs Training Examples repository on GitHub contains source files for training workshops and videos
- To download source files for this workshop:
  1. Visit the Silicon Labs Training Examples repository on GitHub https://github.com/SiliconLabs/training_examples
  2. From the Code dropdown
  3. Clone the repository with your favorite Git client
     or
     Click the Download ZIP option (unzip the files on the local PC)
  4. Source files for this workshop are located in the dev_lab_wifi_http_server/source folders
  5. A PDF version of this presentation is in the dev_lab_wifi_http_server/presentation folder

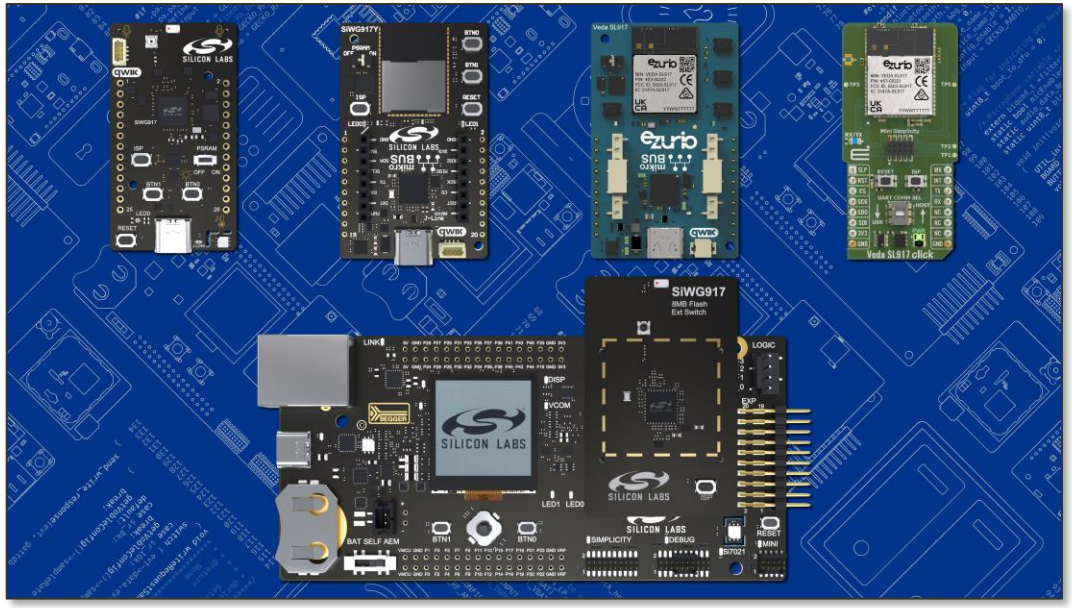


Prerequisites

- Simplicity Studio v5 installed:
  - Download from: https://www.silabs.com/developers/simplicity-studio
  - Ensure the Simplicity SDK is installed including WiSeConnect 3.4.1 or later
- TeraTerm or similar serial terminal application



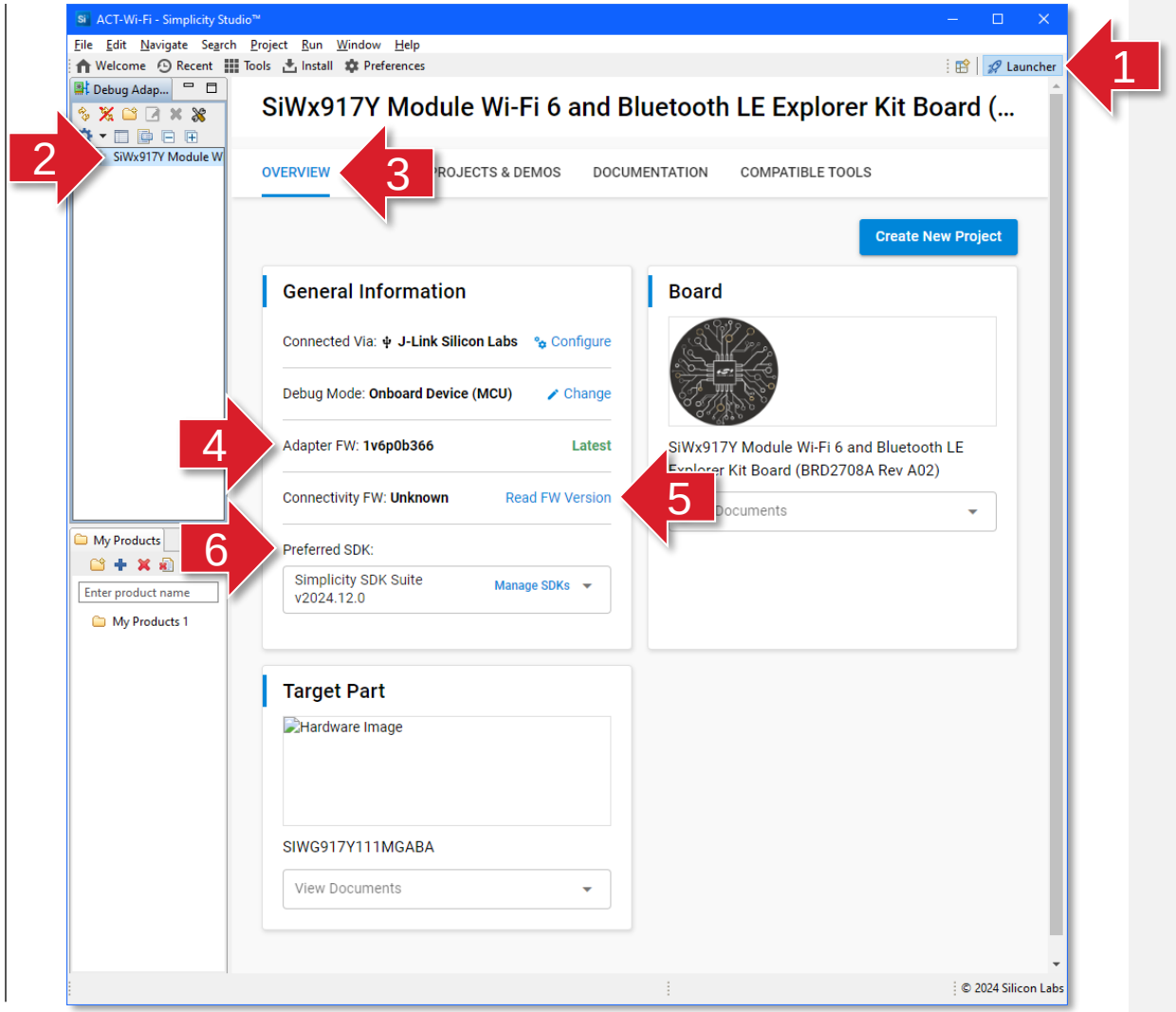
- SiWG917 Wi-Fi kits and boards
  - Silicon Labs Dev Kit
  - Silicon Labs Explorer Kit
  - Ezurio SL917 Veda Explorer Kit
  - Ezurio SL917 Click
  - Silicon Labs Pro Kit
  - More information at https://www.silabs.com/wireless/wi-fi?tab=kits



HTTP Server Project

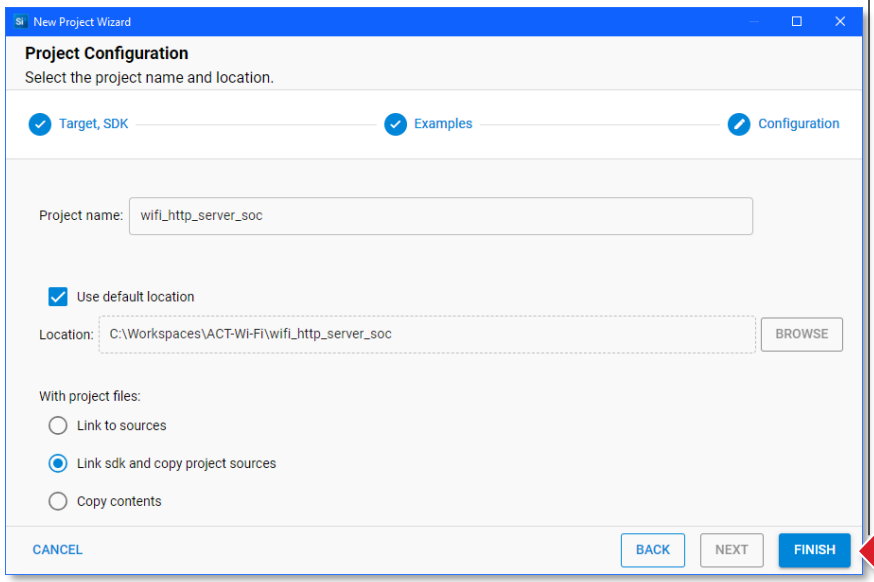
Connect and Update Board

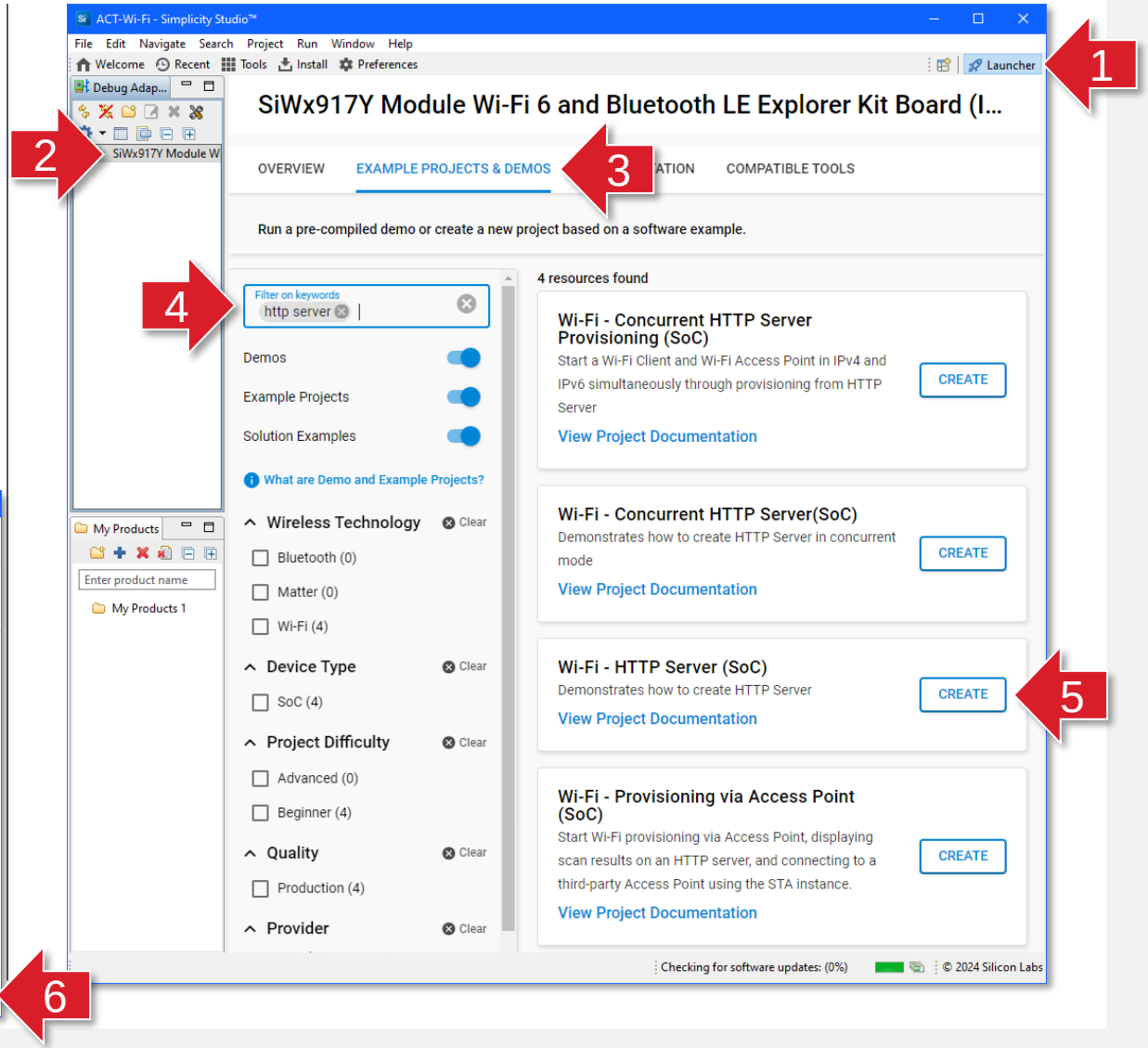
- To connect and update board firmware:
  1. Go to the Launcher perspective
  2. Connect the board using USB and select in the Debug Adapters panel
  3. Make sure the Overview page is selected
  4. In the General Information box, update Adapter FW if the latest is not installed
  5. Read the Connectivity FW version and update if the latest is not installed
  6. Check the Preferred SDK is set to Simplicity SDK Suite



Create Example Application

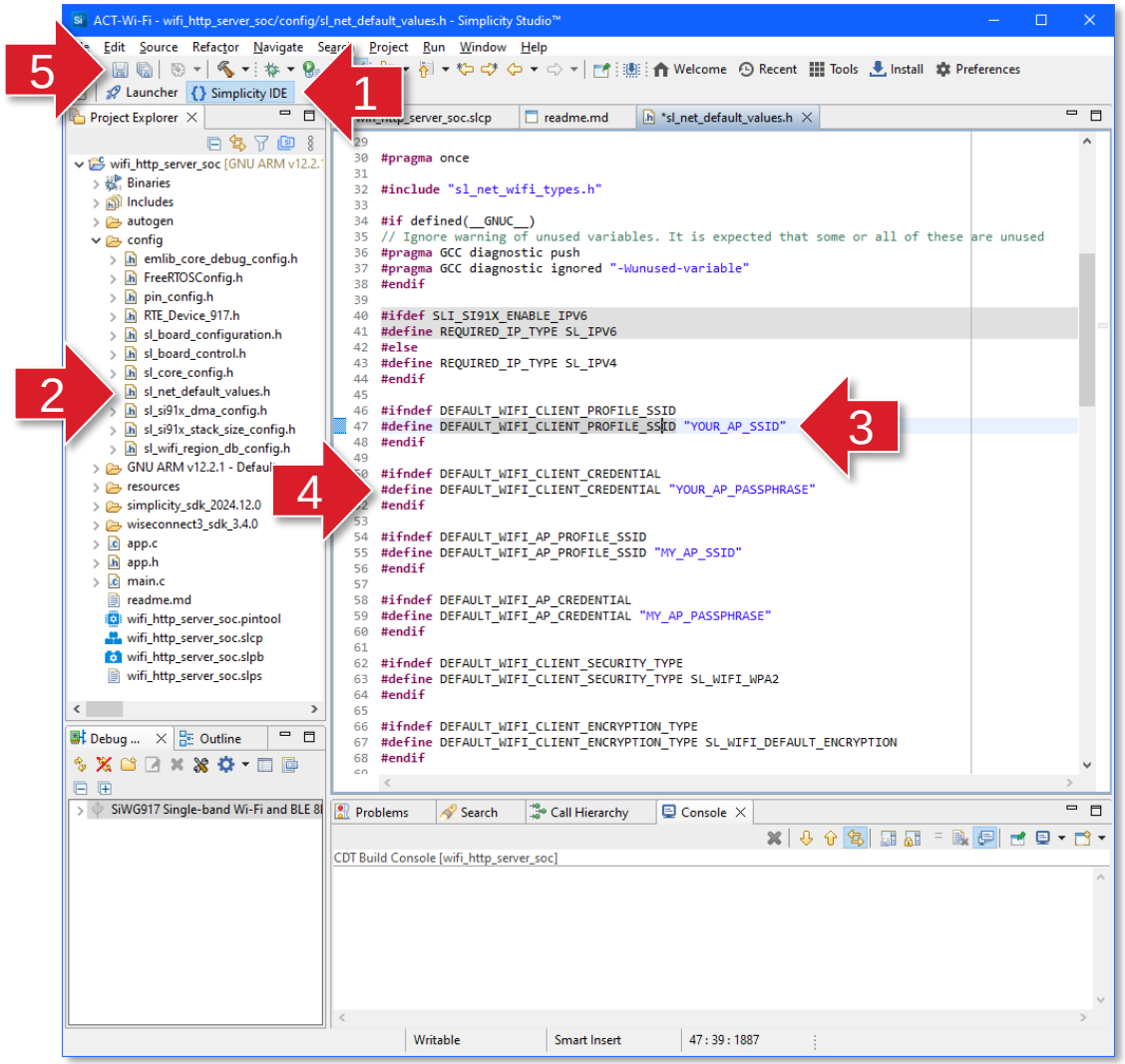
- To create the example application:
  1. Go to the Launcher perspective
  2. Make sure the board is selected in the Debug Adapters panel
  3. Select the Example Projects & Demos page is selected
  4. Enter HTTP Server into the filter editbox
  5. In the Wi-Fi – HTTP Server (SoC) box, click the Create button
  6. In the New Project Wizard window, click the Finish button





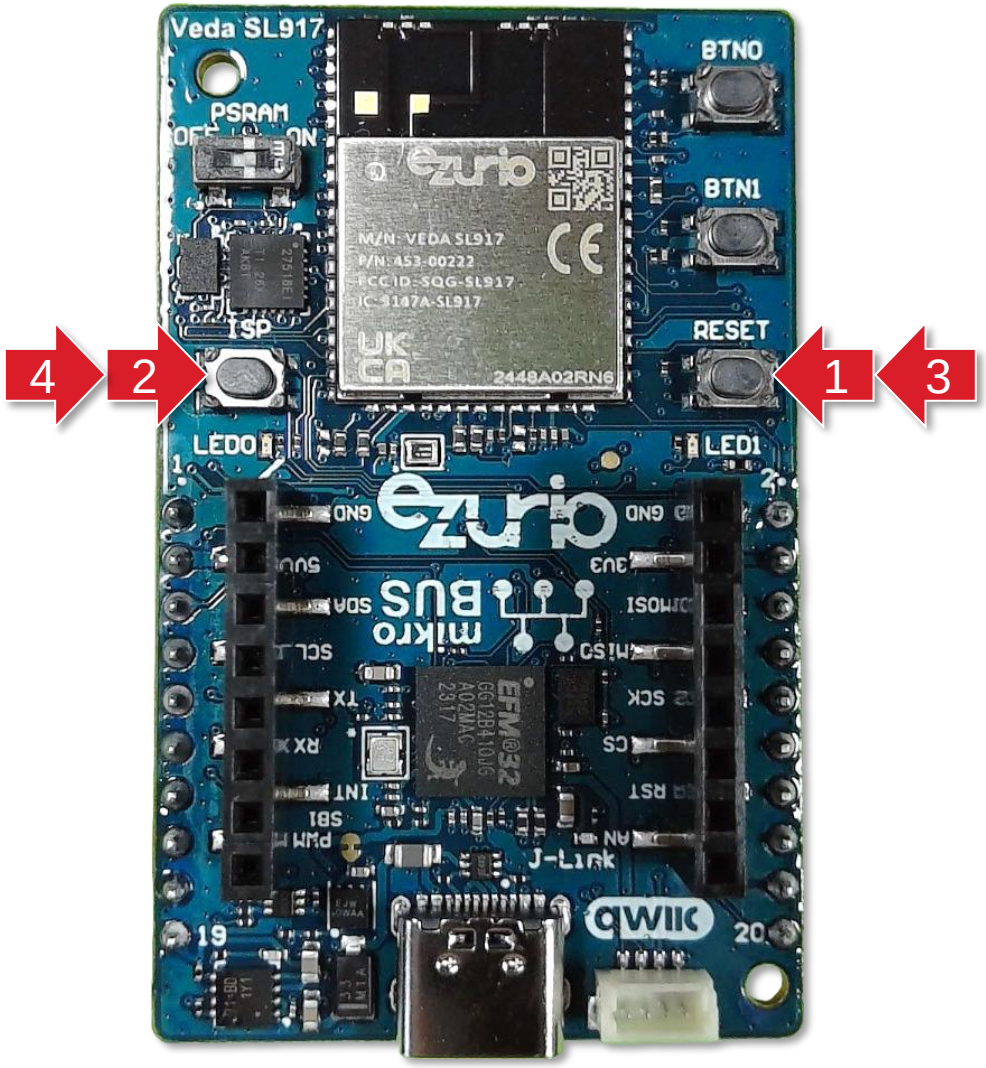
Configure Wi-Fi Access Point

- To configure the Wi-Fi access point credentials:
  1. Make sure the Simplicity IDE perspective is selected
  2. In the Project Explorer panel, open the config/sl_net_default_values.h file
  3. Update the DEFAULT_WIFI_CLIENT_PROFILE_SSID define with the SSID of the Wi-Fi network to join
  4. Update the DEFAULT_WIFI_CLIENT_CREDENTIAL define with the password of the Wi-Fi network to join
  5. Save the changes to the file



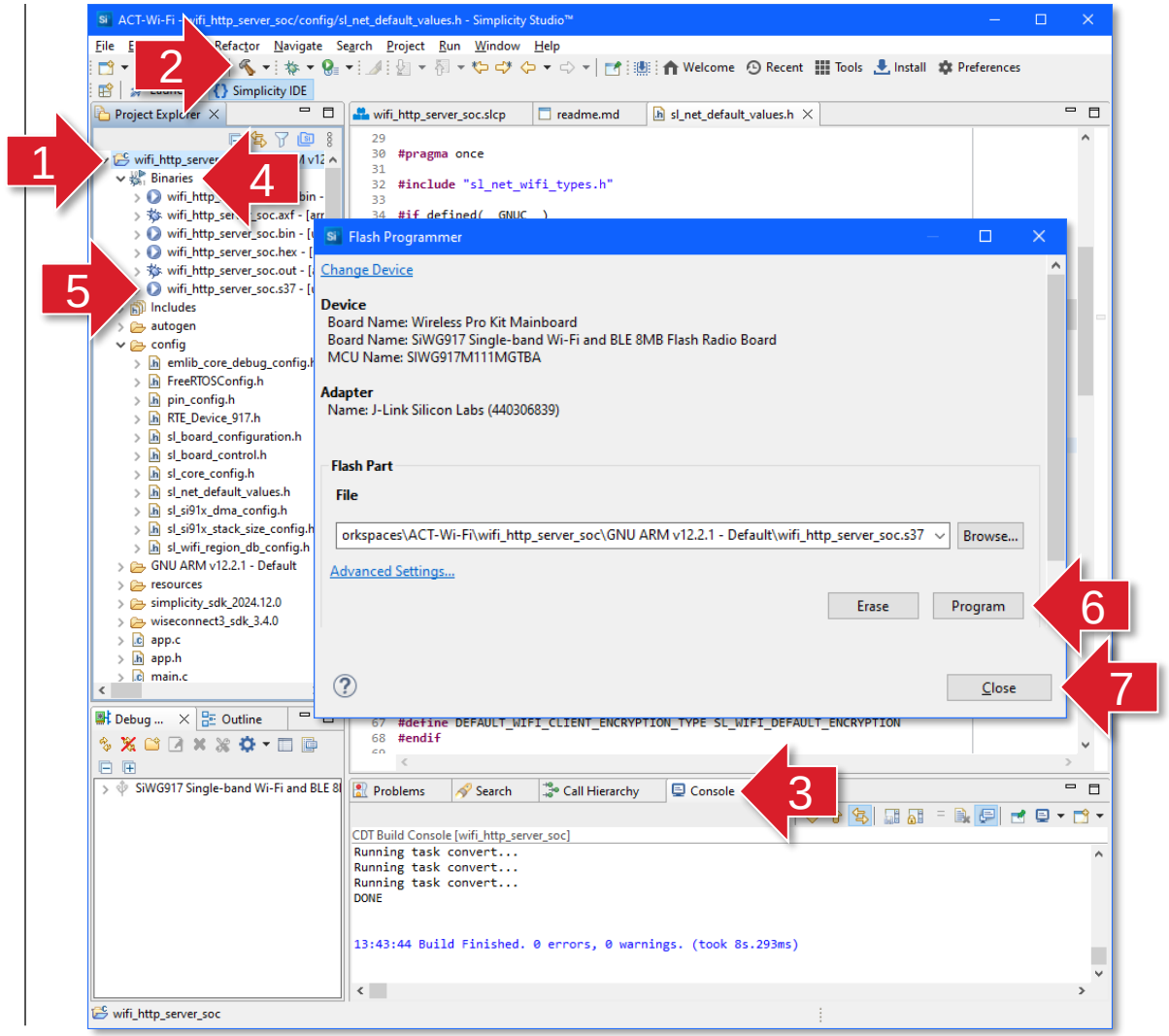
ISP Mode

- Once we allow the application processor to sleep the board needs to be placed into ISP mode to allow programming
  - Programming is not possible while the M4 is in a sleep state
- To put the board into ISP mode:
  1. Press and hold the RESET button
  2. Press and hold the ISP button
  3. Release the RESET button
  4. Release the ISP button



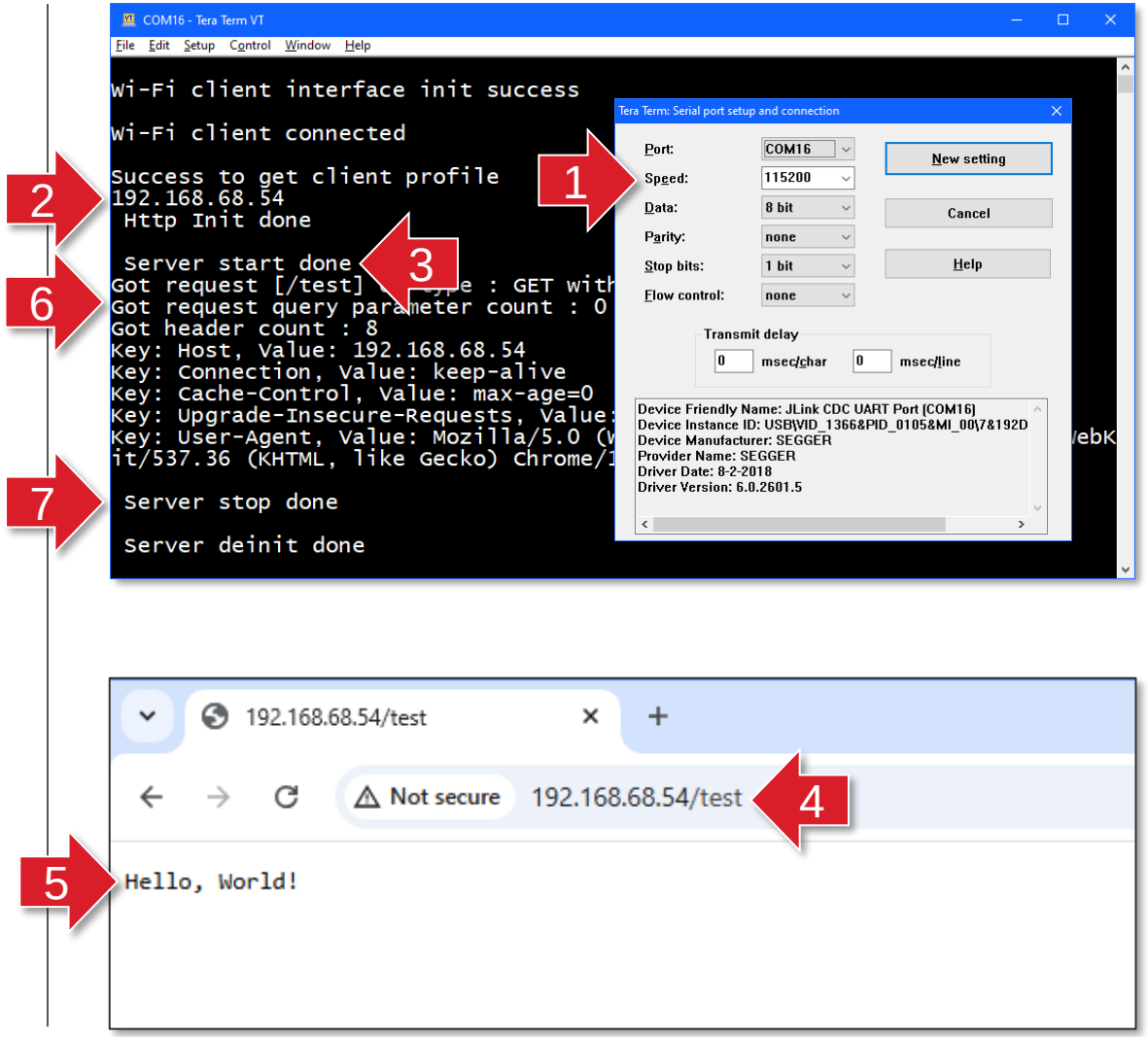
Compile and Flash

- To compile the software:
  1. In the Project Explorer panel, select the top-level project
  2. Click the Build (hammer) button on the toolbar
  3. Compilation progress is shown in the Console panel
- To flash the software:
  4. In the Project Explorer panel, open the Binaries folder
  5. Locate the .s37 file, right click and select Flash to device…
  6. In the Flash Programmer window, click the Program button
  7. When complete, click the Close button

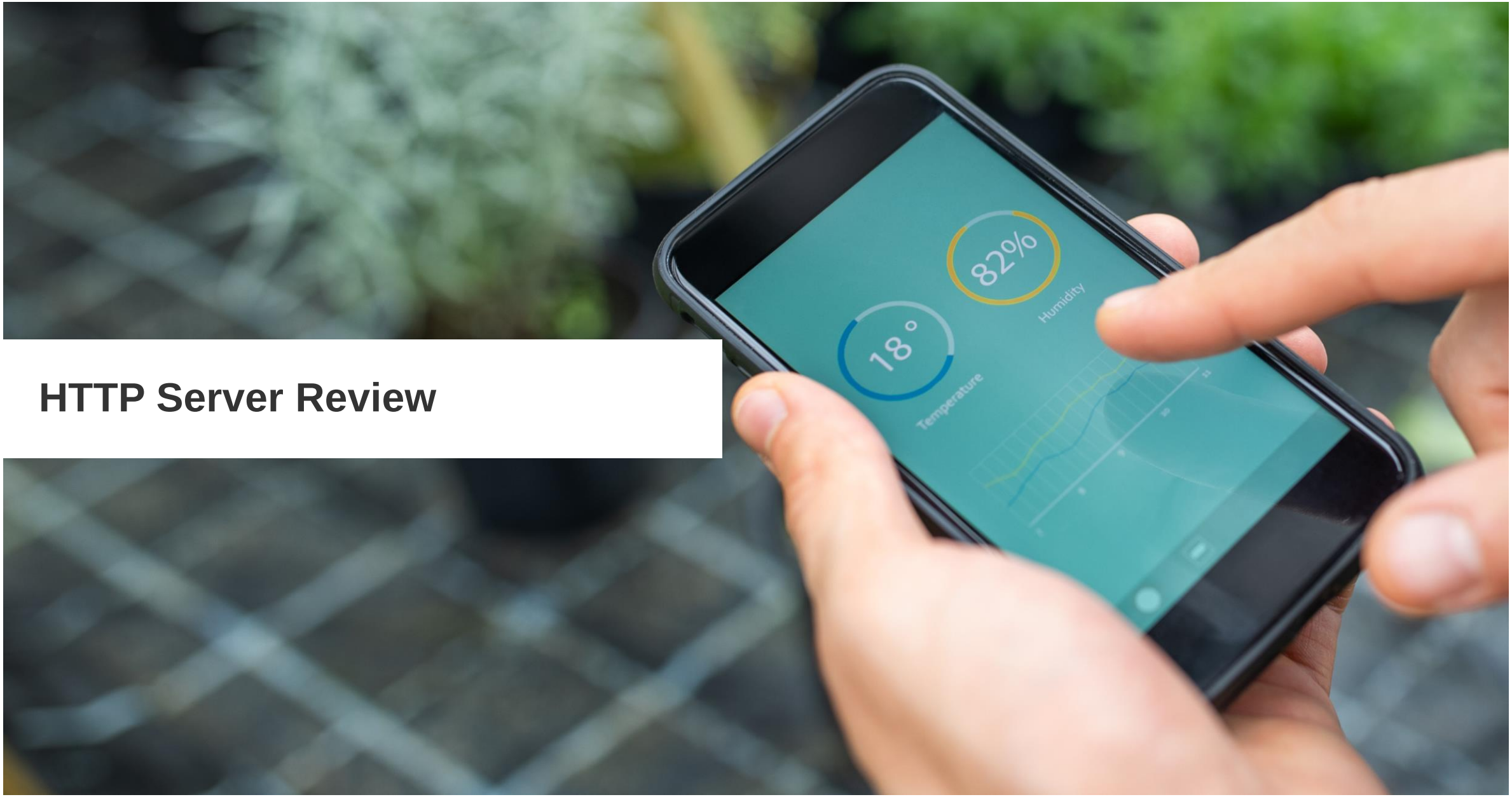


Operation

- To operate the HTTP Server:
  1. Connect a serial terminal to the board using: 115200 baud, 8 data bits, no parity, 1 stop bit, no flow control
  2. Reset the board, when the device joins the network its IP address will be output to the terminal
  3. Server start done will be output when the HTTP server is running
  4. In a web browser enter the IP address followed by /test
  5. Hello, World! Will be displayed in the browser window
  6. Data on the HTTP request is displayed in the serial terminal
  7. After serving the request, the HTTP server is stopped and deinitialised
  8. If the browser is refreshed, connection refused will be returned as the server is no longer running (not shown)



HTTP Server Review



Application Startup

- The application code is structured around FreeRTOS with the application parts of the code in app.c:
- The app_init() function is called at startup from main.c
- The app_init() function creates a new thread that runs in the application_start() function

```
335 void app_init(const void *unused)
336 {
337   UNUSED_PARAMETER(unused);
338   osThreadNew((osThreadFunc_t)application_start, NULL, &thread_attributes);
339 }
340
341 static void application_start(void *argument)
342 {
```

Wi-Fi Startup

- The application_start() function handles the setup and joining of the Wi-Fi network:
- The sl_net_init() function initializes the network interface
  - The interface is initialized as a client device
  - The Wi-Fi configuration passed in using a sl_wifi_device_configuration_t structure
  - Note, despite the variable name this configures the Wi-Fi *not* the HTTP Server
- The sl_net_up() function brings up the network interface, for a client:
  - Uses the default profile which includes the SSID and password set in the sl_net_default_values.h file
  - Scans and connects to the network specified in the profile
- The sl_net_get_profile() function reads back the profile when it has joined the network
  - The IP address is extracted from this profile and output to the serial port

```
341 static void application_start(void *argument)
342 {
343   UNUSED_PARAMETER(argument);
344   sl_status_t status                    = 0;
345   sl_http_server_config_t server_config = { 0 };
346
347   status = sl_net_init(SL_NET_WIFI_CLIENT_INTERFACE, &http_server_configuration, NULL, NULL);
348   if (status != SL_STATUS_OK) {
349     printf("\r\nFailed to start Wi-Fi Client interface: 0x%lx\r\n", status);
350     return;
351   }
352   printf("\r\nWi-Fi client interface init success\r\n");
353
354   status = sl_net_up(SL_NET_WIFI_CLIENT_INTERFACE, SL_NET_DEFAULT_WIFI_CLIENT_PROFILE_ID);
355   if (status != SL_STATUS_OK) {
356     printf("\r\nFailed to bring Wi-Fi client interface up: 0x%lX\r\n", status);
357     return;
358   }
359   printf("\r\nWi-Fi client connected\r\n");
360
361   status = sl_net_get_profile(SL_NET_WIFI_CLIENT_INTERFACE, SL_NET_DEFAULT_WIFI_CLIENT_PROFILE_
362   if (status != SL_STATUS_OK) {
363     printf("Failed to get client profile: 0x%lx\r\n", status);
364     return;
365   }
366   printf("\r\nSuccess to get client profile\r\n");
367
368   ip_address.type = SL_IPV4;
369   memcpy(&ip_address.ip.v4.bytes, &profile.ip.ip.v4.ip_address.bytes, sizeof(sl_ipv4_address_t)
370   print_sl_ip_address(&ip_address);
```

HTTP Server Startup

- Near the start of app.c, an array of sl_http_server_handler_t structures are initialized:
  - Each element pairs a URI with a function to be called when the URI is requested
  - The /test URI causes the buffered_request_handler() function to be called
- The application_start() function also starts the HTTP Server:
  - A sl_http_server_config_t structure is initialized with settings for the HTTP Server, including the request handlers
  - The sl_http_server_init() function initializes the server using the configuration structure
  - The sl_http_server_start() function begins running the server
  - The thread is kept in a loop while the is_server_running variable is true

```
101 static sl_http_server_handler_t request_handlers[4] =
102     { { .uri = "/test",      .handler = buffered_request_handler },
103       { .uri = "/data",      .handler = large_data_handler },
104       { .uri = "/cert1.pem", .handler = large_response_handler },
105       { .uri = "/cert2.pem", .handler = chunked_large_response_handler } };
```

```
372   server_config.port            = HTTP_SERVER_PORT;
373   server_config.default_handler  = NULL;
374   server_config.handlers_list    = request_handlers;
375   server_config.handlers_count   = 4;
376   server_config.client_idle_time = 1;
377
378   status = sl_http_server_init(&server_handle, &server_config);
379   if (status != SL_STATUS_OK) {
380     printf("\r\nHTTP server init failed:%lx\r\n", status);
381     return;
382   }
383   printf("\r\n Http Init done\r\n");
384
385   status = sl_http_server_start(&server_handle);
386   if (status != SL_STATUS_OK) {
387     printf("\r\n Server start fail:%lx\r\n", status);
388     return;
389   }
390   printf("\r\n Server start done\r\n");
391
392   is_server_running = true;
393   while (is_server_running) {
394     osDelay(100);
395   }
```

HTTP Request Handler (1)

- The buffered_request_handler() function is called when the /test URI is requested:
- Local variables are initialized including:
  - http_response for the response data
  - header for the response header
- The next part of the function outputs data in the HTTP request to the serial port

```
238 sl_status_t buffered_request_handler(sl_http_server_t *handle, sl_http_server_request_t *req)
239 {
240   sl_http_recv_req_data_t recvData        = { 0 };
241   sl_http_server_response_t http_response = { 0 };
242   sl_http_header_t request_headers[5]     = { 0 };
243   sl_http_header_t header                 = { .key = "Server", .value = "SI917-HTTPServer" };
244
245   printf("Got request [%s] of type : %s with data length : %lu\n",
246          req->uri.path,
247          request_type[req->type],
248          req->request_data_length);
249   if (req->request_data_length > 0) {
250     recvData.request       = req;
251     recvData.buffer        = (uint8_t *)response;
252     recvData.buffer_length = 1024;
253
254     sl_http_server_read_request_data(handle, &recvData);
255     response[recvData.received_data_length] = 0;
256     printf("Got request data as : %s\n", response);
257   }
258
259   printf("Got request query parameter count : %u\n", req->uri.query_parameter_count);
260   if (req->uri.query_parameter_count > 0) {
261     for (int i = 0; i < req->uri.query_parameter_count; i++) {
262       printf("query: %s, value: %s\n", req->uri.query_parameters[i].query, req->uri.query_param
263     }
264   }
265
266   printf("Got header count : %u\n", req->request_header_count);
267   sl_http_server_get_request_headers(handle, req, request_headers, 5);
268
269   int length = (req->request_header_count > 5) ? 5 : req->request_header_count;
270   for (int i = 0; i < length; i++) {
271     printf("Key: %s, Value: %s\n", request_headers[i].key, request_headers[i].value);
272   }
```

HTTP Request Handler (2)

- The final part of the buffered_request_handler() function constructs and sends the response:
- The response code is set to OK
- The content type is set to plain text
- The headers are added
- The response data is set to “Hello, World!” and data lengths set appropriately
- The sl_http_server_send_response() function is then used to transmit the HTTP response
- Finally, the is_server_running variable is set to false
  - This allows the loop in the application_start() function to end

```
274   // Set the response code to 200 (OK)
275   http_response.response_code = SL_HTTP_RESPONSE_OK;
276
277   // Set the content type to plain text
278   http_response.content_type = SL_HTTP_CONTENT_TYPE_TEXT_PLAIN;
279   http_response.headers      = &header;
280   http_response.header_count = 1;
281
282   // Set the response data to "Hello, World!"
283   char *response_data               = "Hello, World!";
284   http_response.data                = (uint8_t *)response_data;
285   http_response.current_data_length  = strlen(response_data);
286   http_response.expected_data_length = http_response.current_data_length;
287   sl_http_server_send_response(handle, &http_response);
288   is_server_running = false;
289   return SL_STATUS_OK;
290 }
```

HTTP Server Shutdown

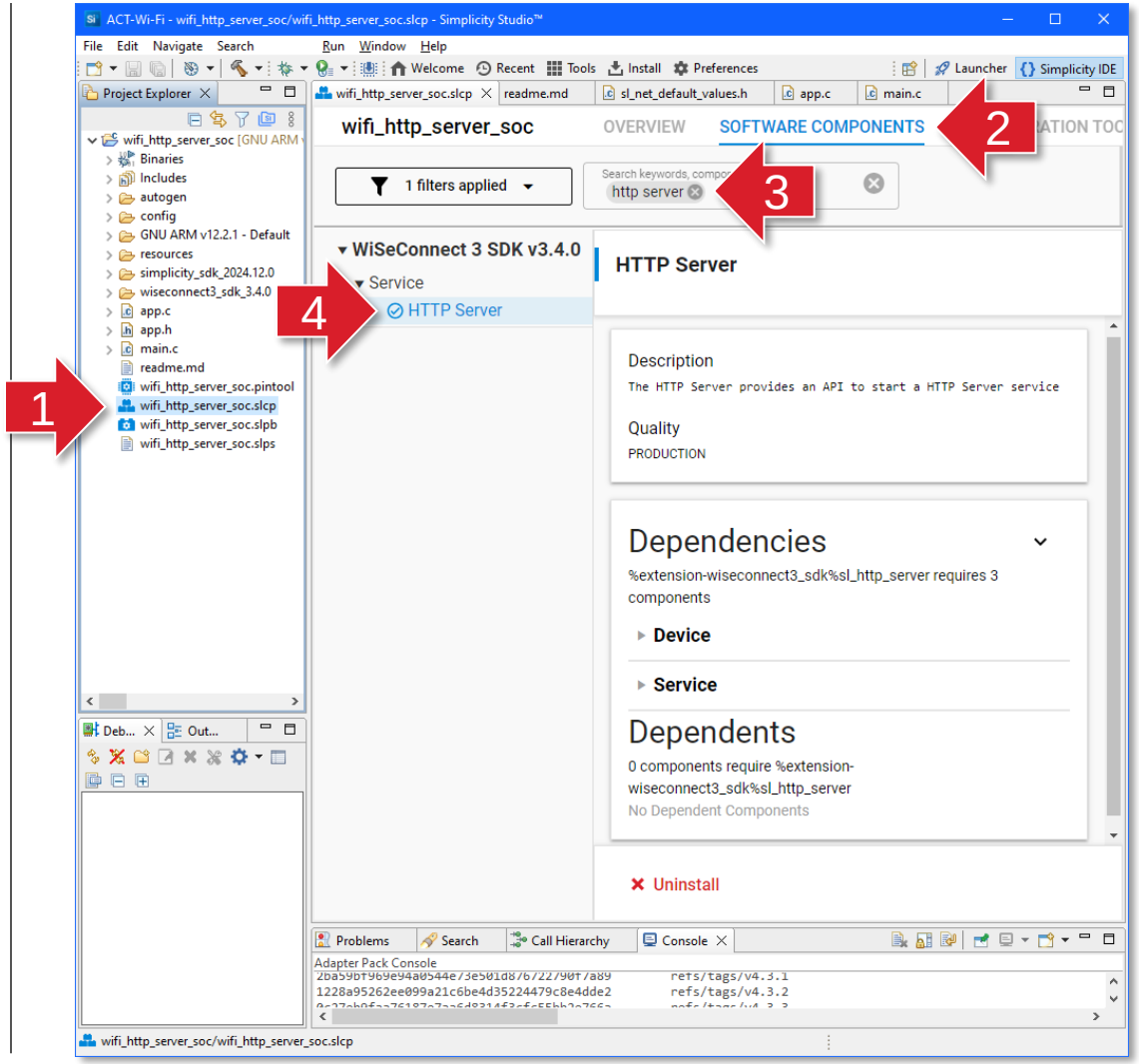
- When the while loop exits in the application_start() function:
  - The sl_http_server_stop() function is called to stop the server
  - The sl_http_server_deinit() is called to deinitialize the server

```
392   is_server_running = true;
393   while (is_server_running) {
394     osDelay(100);
395   }
396
397   status = sl_http_server_stop(&server_handle);
398   if (status != SL_STATUS_OK) {
399     printf("\r\n Server stop fail:%lx\r\n", status);
400     return;
401   }
402   printf("\r\n Server stop done\r\n");
403
404   status = sl_http_server_deinit(&server_handle);
405   if (status != SL_STATUS_OK) {
406     printf("\r\n Server deinit fail:%lx\r\n", status);
407     return;
408   }
409   printf("\r\n Server deinit done\r\n");
410 }
```

HTTP Server Project Enhancements

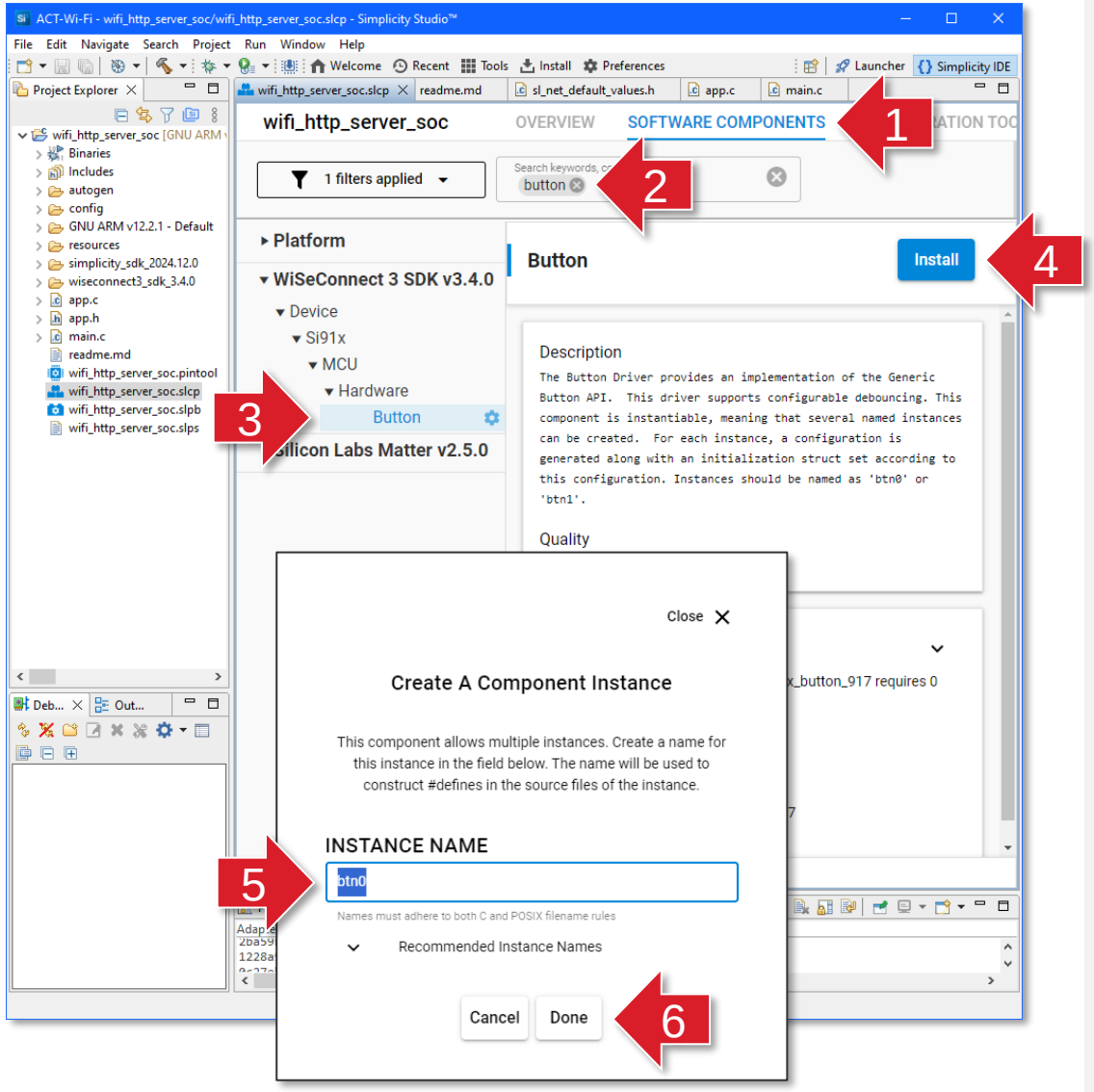
HTTP Server Software Component

- Software Components are a convenient way to add functionality to a project
- The HTTP Server Software Component is pre-installed in the example project
  - This provides the structures and functions used to operate the HTTP Server
- To view it:
  1. Open the .slcp file in the main project
  2. Go to the Software Components page
  3. Search for HTTP Server
  4. This component is already installed in the example project, as indicated by the tickbox
- The .slcp file also provides other options to manage the project and access relevant tools



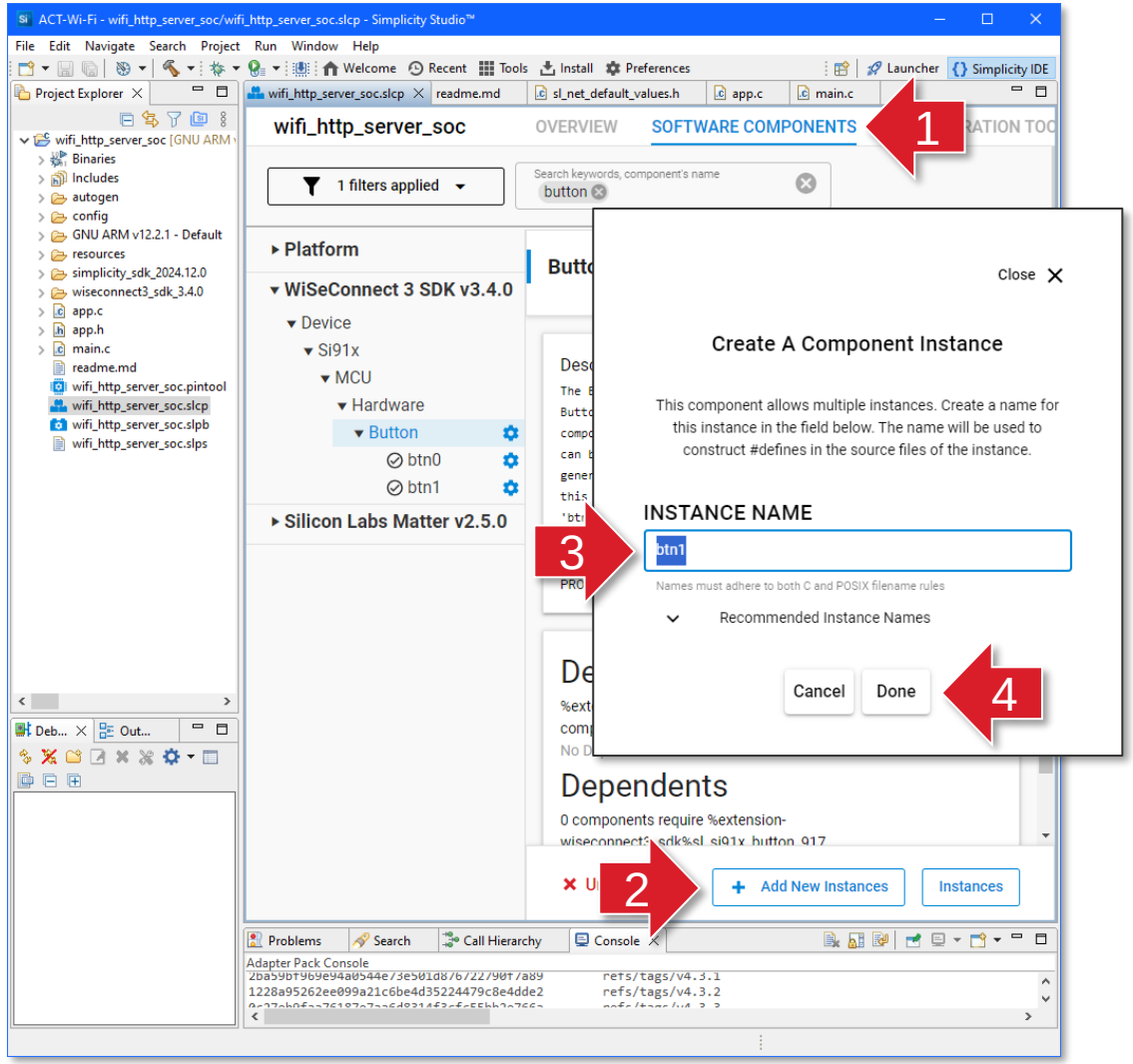
Button 0 Software Component

- We will add Software Components for the buttons we want to monitor:
  1. Make sure you are on the Software Components page
  2. Search for Button
  3. In the tree on the left, unfold: WiSeConnect 3 SDK > Device > Si91x > MCU > Hardware > Button
  4. Click the Install button
  5. In the Create a Component Instance window check the instance name is set to btn0
  6. Click the Done button
- This adds the button APIs to the project and also code to use it with Button 0 on the board with the correct pin configuration



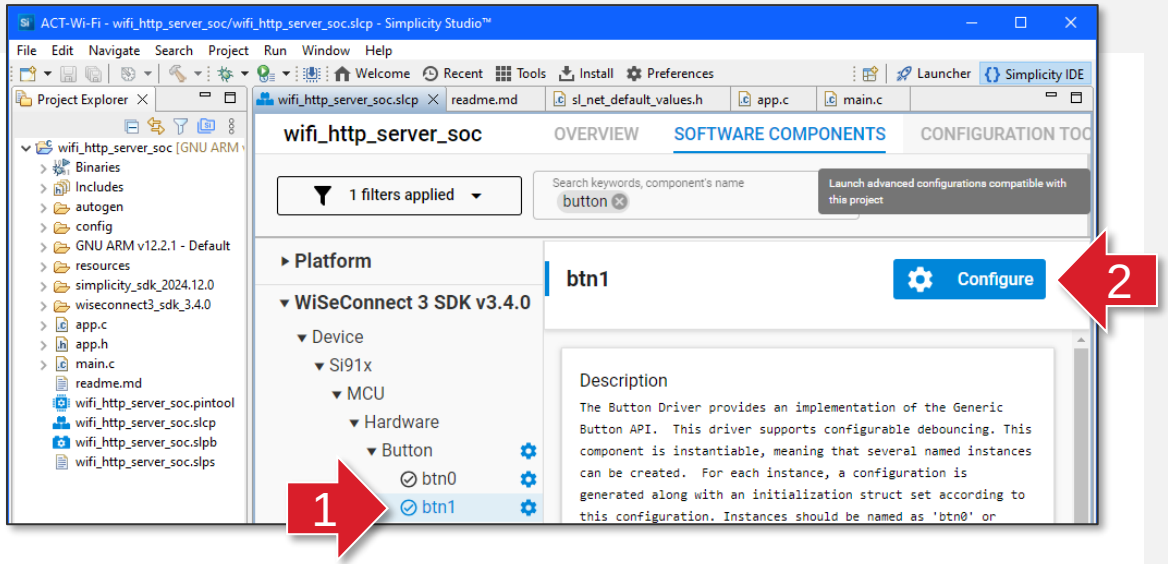
Button 1 Software Component

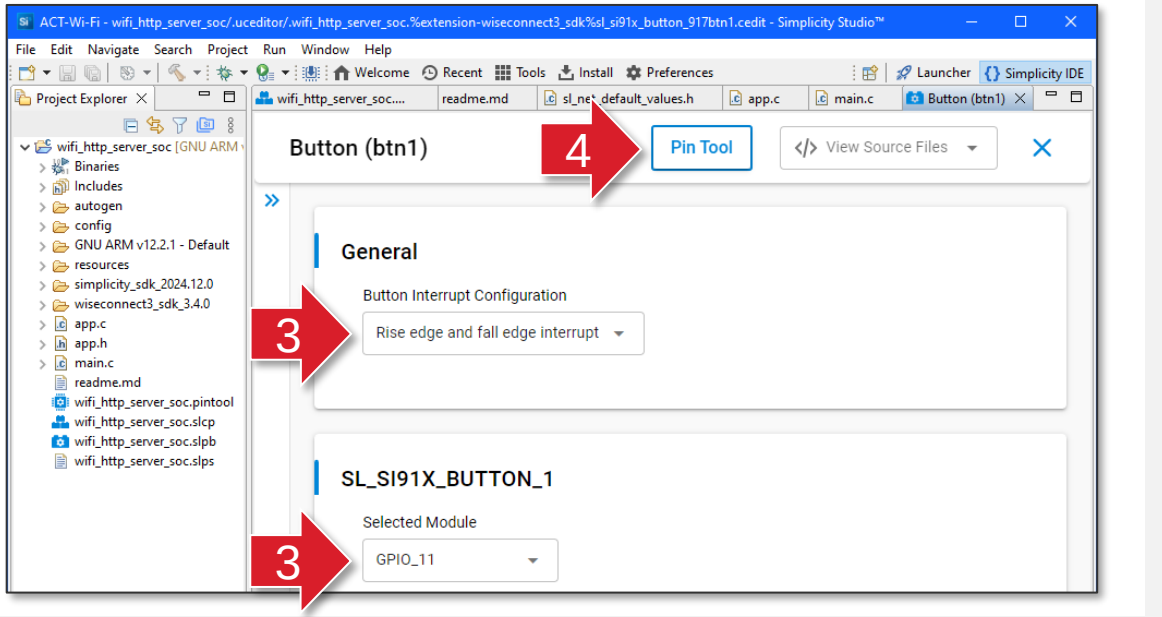
- To add the instance for Button 1:
  1. Make sure you are on the Software Components page
  2. Click the Add New Instances button
  3. In the Create a Component Instance window check the instance name is set to btn1
  4. Click the Done button

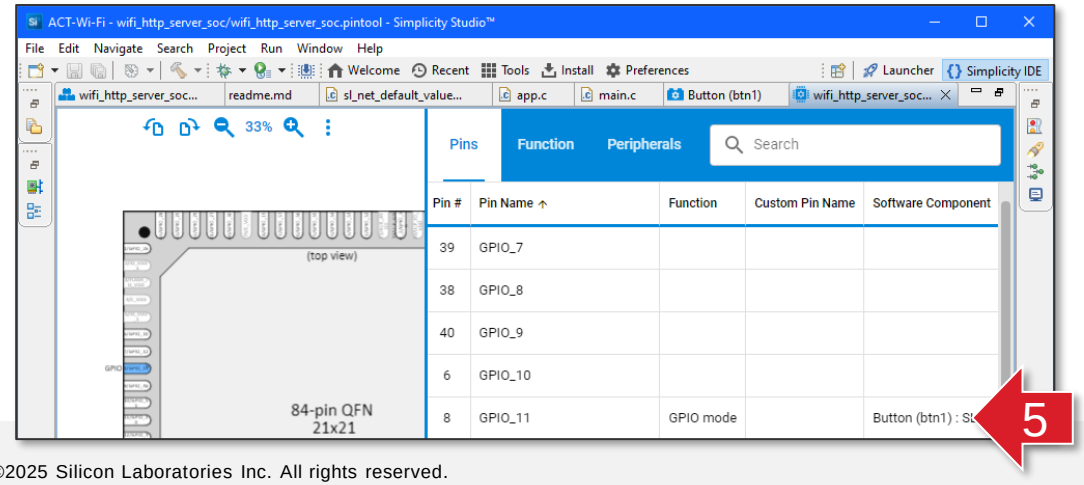


Button Configuration

- We used recommended instance names which are pre-mapped onto the GPIO for the buttons on the board
- We can view this configuration:
  1. Click one of the instances under Button in the treeview
  2. Click the Configure button
  3. The Interrupt Configuration and assigned GPIO pin can be viewed and edited in this window
  4. Click the Pin Tool button
  5. This tool shows the low-level assignment of pins to peripherals including the GPIO assignment for the Button instance

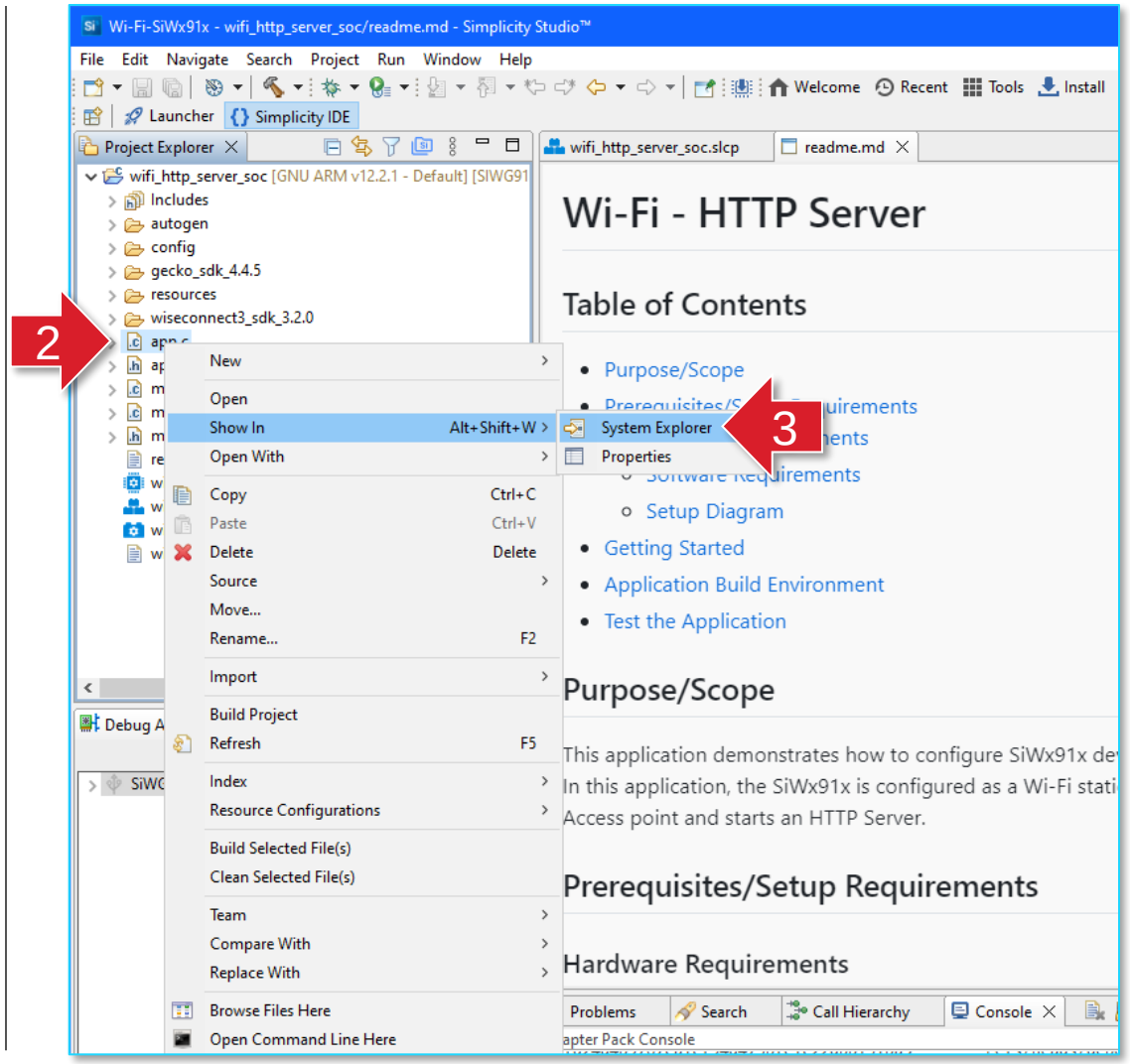




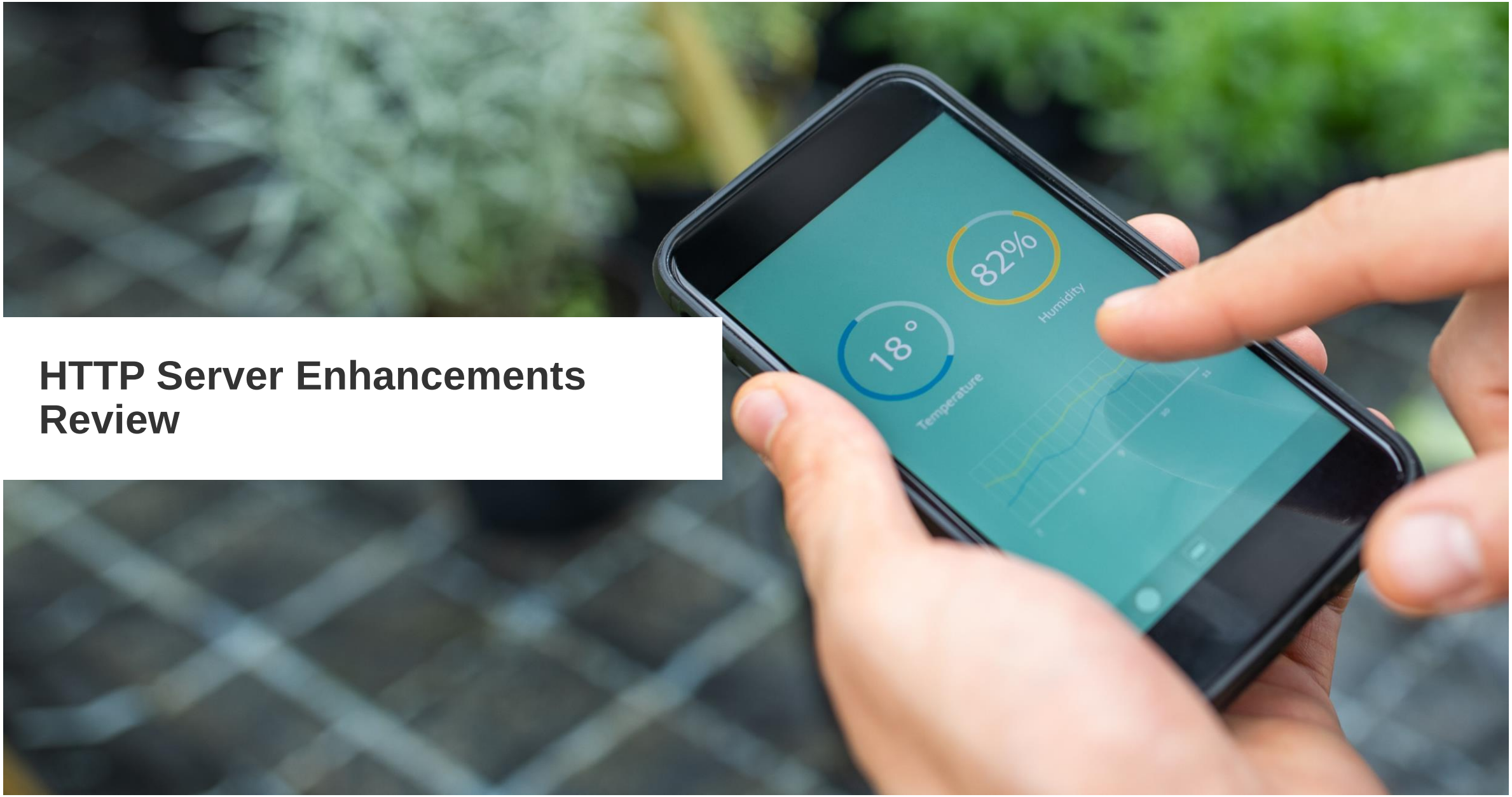


Update app.c – HTTP Server Enhancements

- To update app.c in the project folder:
  1. Locate the app.c file downloaded from GitHub (located in the dev_lab_wifi_http_server/source_1_server folder) and copy to the clipboard
  2. In Simplicity Studio, locate the app.c file in the Project Explorer panel
  3. Right-click and select Show in > System Explorer
  4. Paste the app.c file from the clipboard into the opened folder (replacing the existing file)



HTTP Server Enhancements Review



app.c – Includes, Defines and Global Variables

- New includes, defines and global variables are added to app.c:
- Includes to access the button APIs
- An APP_VERSION define
- A HTML_RESPONSE define containing a formatting string for the HTML response:
  1. The meta tag contains browser instructions to request a refresh every 5 seconds
  2. Tokens are in place to include the seconds the application has been running and states of button 0 and button 1
- A HTML_RESPONSE define with a maximum size for the HTML response buffer
- Global variables for:
  - Number of seconds the application has been running
  - States for buttons
  - A buffer in which to build the HTML response

```
43 #include "sl_si91x_button.h"
44 #include "sl_si91x_button_pin_config.h"
45 #include "sl_si91x_button_instances.h"
```

```
47 /*******************************************************
48  *                      Macros
49  ******************************************************/
50 #define APP_VERSION "v0.0.6"
```

```
52 #define HTML_RESPONSE "<!DOCTYPE html>\r\n" \
53   "<html>\r\n" \
54   "  <head>\r\n" \
55   "    <title>SiWG917 HTTP Server</title>\r\n" \
56   "    <meta http-equiv=\"refresh\" content=\"5\">" \
57   "  </head>\r\n" \
58   "  <body>\r\n" \
       <p>SiWG917 HTTP Server " APP_VERSION "</p>\r\n" \
       <pre>seconds = %ld</pre>\r\n" \
       <pre>button0 = %d</pre>\r\n" \
62     <pre>button1 = %d</pre>\r\n" \
63   "  </body>\r\n" \
64   "</html>"
65 #define HTML_RESPONSE_SIZE 768
```

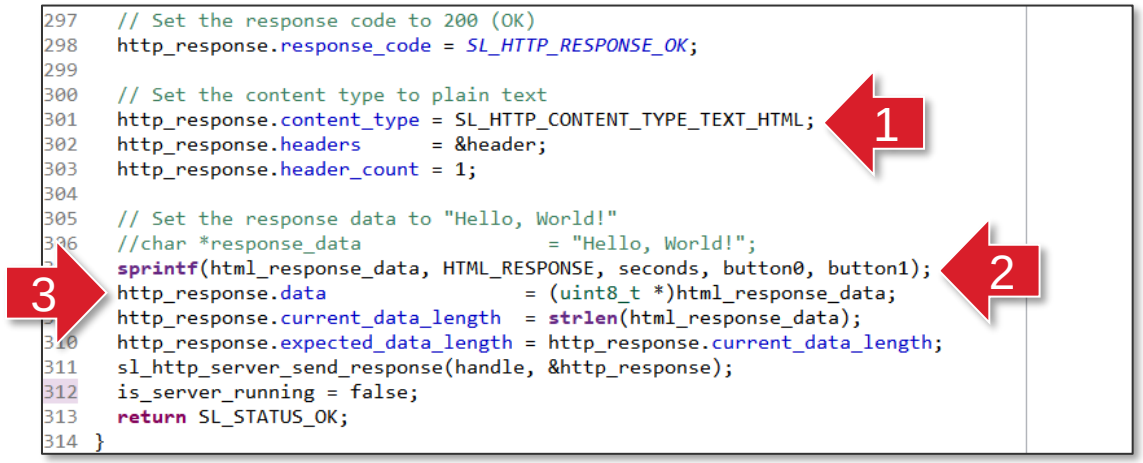
1

2

```
132 uint32_t seconds = 0;
133 int8_t   button0 = BUTTON_STATE_INVALID;
134 int8_t   button1 = BUTTON_STATE_INVALID;
135 char html_response_data[HTML_RESPONSE_SIZE] = "";
```

app.c – buffered_request_handler()

- Changes are made to the response generated in the buffered_request_handler() function:
  1. The content_type is changed to TEXT_HTML
  2. The current seconds, button0 and button1 states are formatted into the html_response_data buffer
  3. The html_response_data buffer is added to the response, along with an updated length



app.c – application_start()

- Changes are made to the application_start() function which runs the application's thread:
  1. A debug message, with the version number is output to the serial port
  2. The while loop is allowed to run forever
  3. In the while loop, the delay is increased to 1 second (from 100ms)
  4. The seconds, button0 and button1 variables are updated
     - For the buttons, the pins are read directly

```
367  static void application_start(void *argument)
368  {
369    UNUSED_PARAMETER(argument);
370    sl_status_t status                    = 0;
371    sl_http_server_config_t server_config = { 0 };
372
373    printf("\r\nSiWG917 HTTP Server %s\r\n", APP_VERSION);
```

1

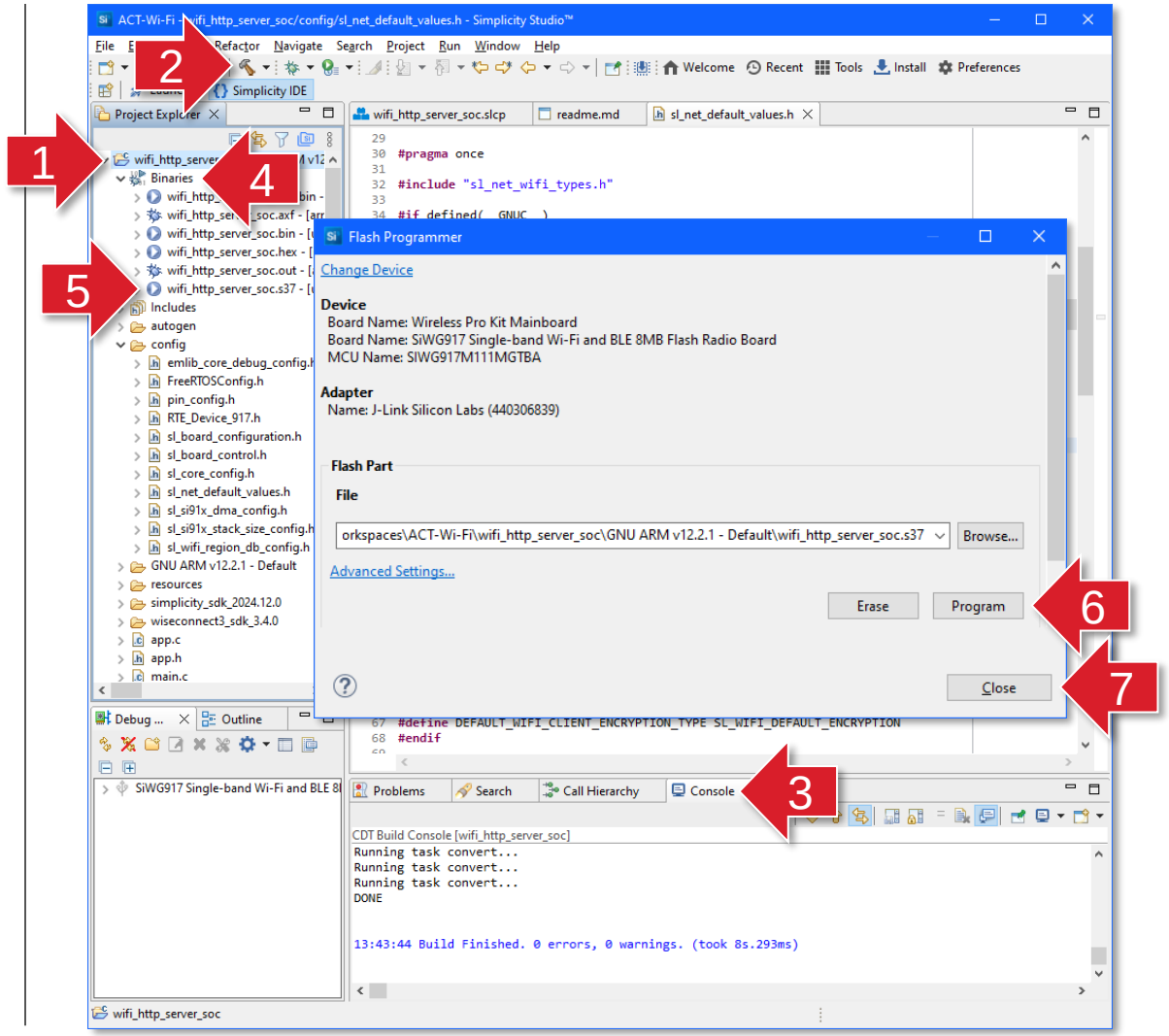
```
    is_server_running = true;
    while (1) {
      osDelay(1000);
      seconds++;
      button0 = sl_si91x_button_pin_state(SL_BUTTON_BTN0_PIN);
      button1 = sl_si91x_button_pin_state(SL_BUTTON_BTN1_PIN);
    }
```

2 3 4

HTTP Server Enhancements Operation

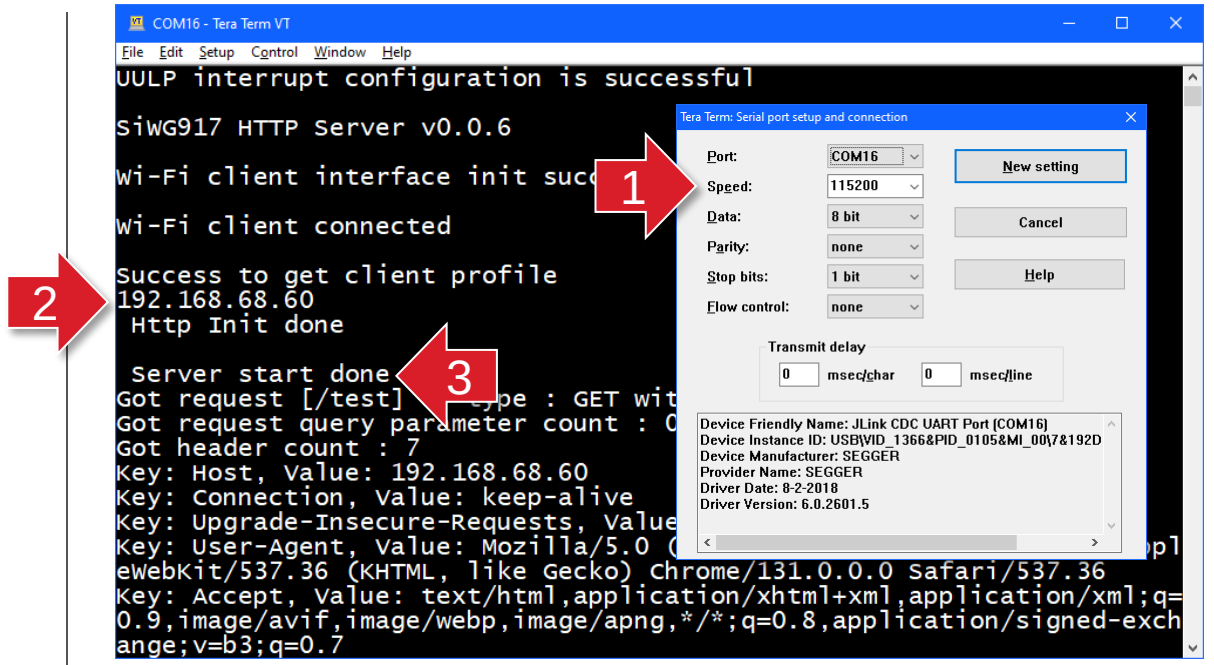
Compile and Flash

- To compile the software:
  1. In the Project Explorer panel, select the top-level project
  2. Click the Build (hammer) button on the toolbar
  3. Compilation progress is shown in the Console panel
- To flash the software:
  4. In the Project Explorer panel, open the Binaries folder
  5. Locate the .s37 file, right click and select Flash to device…
  6. In the Flash Programmer window, click the Program button
  7. When complete, click the Close button



Operation

- To operate the HTTP Server:
  1. Connect a serial terminal to the board using: 115200 baud, 8 data bits, no parity, 1 stop bit, no flow control
  2. Reset the board, when the device joins the network its IP address will be output to the terminal
  3. Server start done will be output when the HTTP server is running
  4. In a web browser enter the IP address followed by /test
  5. The HTML will be displayed in the browser window, including the seconds, button0 and button 1 values
  6. The browser will automatically refresh every 5 seconds (not shown)



Network Join and Rejoin

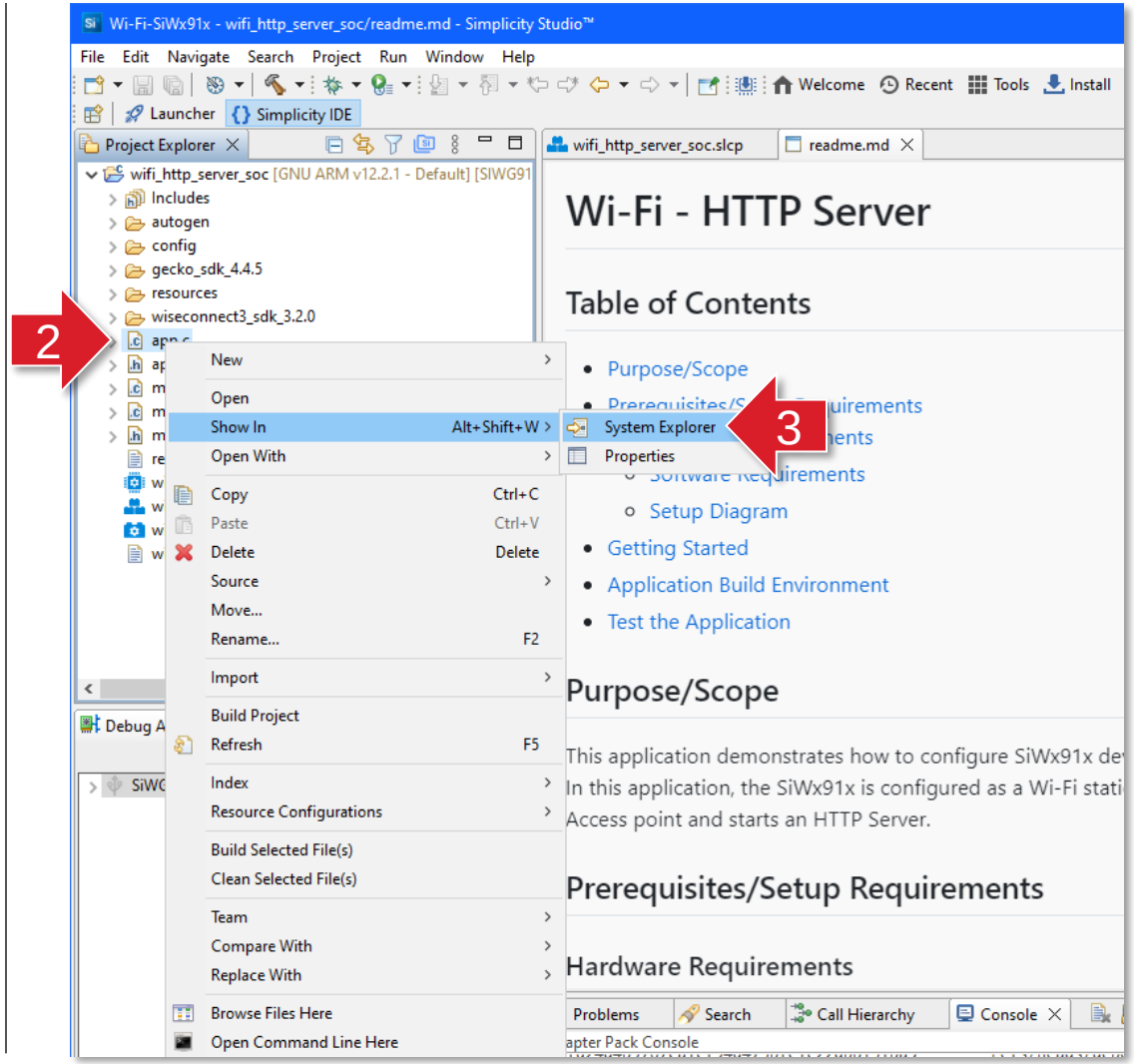
Joining and Rejoining

- The sl_net_up() function provides a lot of functionality when joining the network
  - It both scans for and joins the configured access point
- If it fails to join, it will eventually return with a failure status code
  - Currently app.c will end processing when this happens
- Similarly, if a joined network is lost:
  - The use of sl_net_up() to join will trigger some rejoin attempts
    - The attempts can be configured by calling the sl_wifi_set_advanced_client_configuration() function
- If it fails to rejoin, the device will remain unconnected to the network unless further actions are taken
  - Currently app.c does not detect or react to entry into this state
- In this section we will address these potential issues in the application

```
380   status = sl_net_up(SL_NET_WIFI_CLIENT_INTERFACE, SL_NET_DEFAULT_WIFI_CLIENT_PROFILE_ID);
381   if (status != SL_STATUS_OK) {
382     printf("\r\nFailed to bring Wi-Fi client interface up: 0x%lX\r\n", status);
383     return;
384   }
385   printf("\r\nWi-Fi client connected\r\n");
```

Update app.c – Join and Rejoin

- To update app.c in the project folder:
  1. Locate the app.c file downloaded from GitHub (located in the dev_lab_wifi_http_server/source_2_rejoin folder) and copy to the clipboard
  2. In Simplicity Studio, locate the app.c file in the Project Explorer panel
  3. Right-click and select Show in > System Explorer
  4. Paste the app.c file from the clipboard into the opened folder (replacing the existing file)



Joining Updates

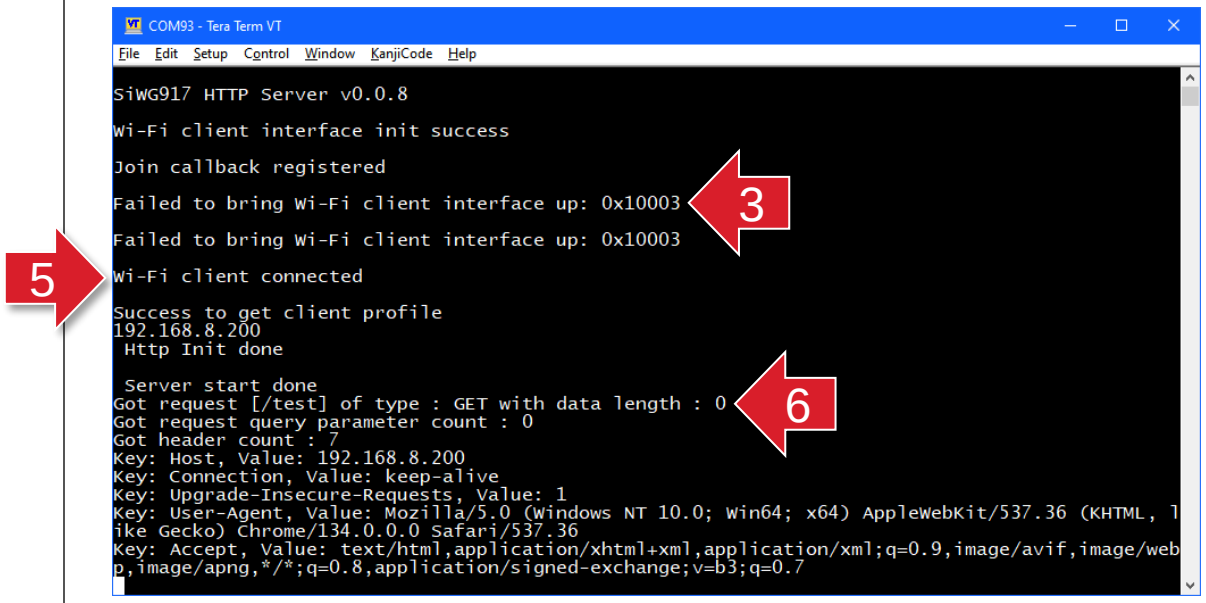
- A new global variable status_net_up is added to track the network state
- In the application_start() function:
  - The call to sl_net_up() is moved inside a loop
  - The code remains in the loop until the network is joined
  - A fixed 10 second delay is applied each time sl_net_up() fails to join the network
    - These delays could be increased over time if required
  - The status_net_up global variable is used to track the network state

```
137 sl_status_t status_net_up                  = SL_STATUS_INVALID_STATE;
```

```
393   while (status_net_up != SL_STATUS_OK) {
394     status_net_up = sl_net_up(SL_NET_WIFI_CLIENT_INTERFACE, SL_NET_DEFAULT_WIFI_CLIENT_PROFILE_ID);
395     if (status_net_up != SL_STATUS_OK) {
396       printf("\r\nFailed to bring Wi-Fi client interface up: 0x%lX\r\n", status_net_up);
397       osDelay(10000);
398     }
399   }
400   printf("\r\nWi-Fi client connected\r\n");
```

Joining Operation

- To see these changes in action:
  1. Turn off the router being used or disable the Wi-Fi through its interface
  2. Compile and flash the updated application
  3. The Failed to bring Wi-Fi client interface up: 0x10003 debug message will be repeated whilst the device in unable to join the network
  4. Turn on the router or enable the Wi-Fi
  5. The Wi-Fi client connected debug message will be displayed when the device joins the network
  6. The webpage can be loaded in a browser to check the HTTP server operation



Rejoining Updates

- In the application_start() function:
  - The sl_wifi_set_join_callback() function is called to configure a callback to join_callback_function() when a join event occurs
- In the join_callback_function():
  - The global status_net_up variable is set to FAIL if the event indicates the automatic rejoin attempts have failed
- In the main loop section of the application_start() function
  - If the status_net_up variable is not set to OK, looping code that calls the sl_net_up() is added to attempt to rejoin the network
  - Each time sl_net_up() fails a fixed 10 second delay is applied
    - In this case the seconds counter is also updated
  - After a successful rejoin the IP Address of the device is output as it may change
- Note there is no need to restart the HTTP Server when the device rejoins, it continues to run

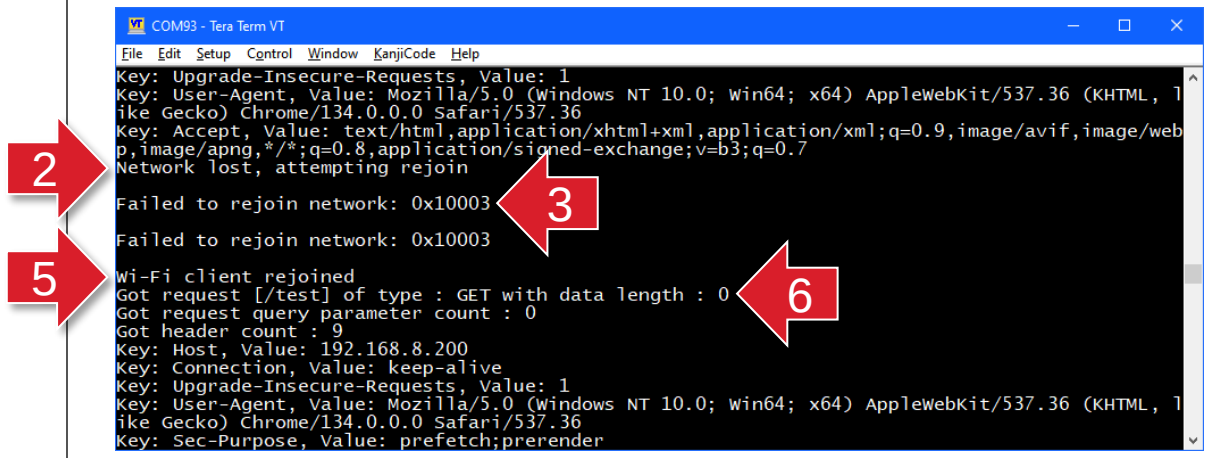
```
385   // Register the join callback
386   status = sl_wifi_set_join_callback(join_callback_function, NULL);
387   if (status != SL_STATUS_OK) {
388       printf("\r\nFailed to register join callback: 0x%lx\r\n", status);
389       return;
390   }
391   printf("\r\nJoin callback registered\r\n");
```

```
470 sl_status_t join_callback_function(sl_wifi_event_t event, char *data, uint32_t data_length, void *optional_arg
471 {
472     UNUSED_PARAMETER(data);
473     UNUSED_PARAMETER(data_length);
474     UNUSED_PARAMETER(optional_arg);
475
476     // Network join fail ?
477     if ((SL_WIFI_EVENT_FAIL_INDICATION | SL_WIFI_JOIN_EVENT) == event) {
478         // Flag network is down
479         status_net_up = SL_STATUS_FAIL;
480     }
481
482     return SL_STATUS_OK;
483 }
```

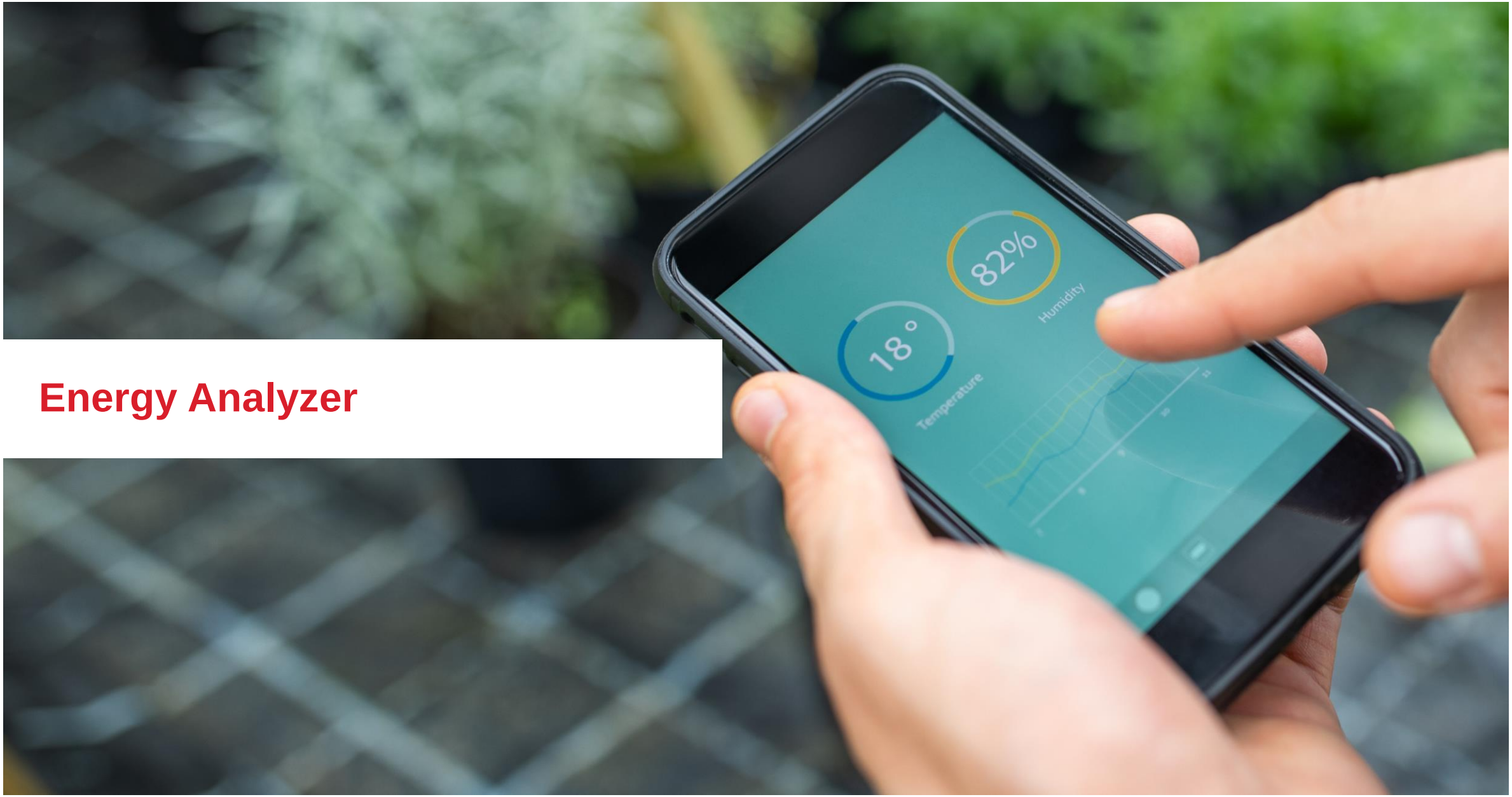
```
441     if (status_net_up != SL_STATUS_OK) {
442       printf("Network lost, attempting rejoin\r\n");
443       // Attempt rejoin ?
444       while (status_net_up != SL_STATUS_OK) {
445         status_net_up = sl_net_up(SL_NET_WIFI_CLIENT_INTERFACE, SL_NET_DEFAULT_WIFI_CLIENT_PROFILE_ID);
446         if (status_net_up != SL_STATUS_OK) {
447           printf("\r\nFailed to rejoin network: 0x%lX\r\n", status_net_up);
448           osDelay(10000);
449           seconds += 10;
450         }
451       }
452       printf("\r\nWi-Fi client rejoined\r\n");
453
454       status = sl_net_get_profile(SL_NET_WIFI_CLIENT_INTERFACE, SL_NET_DEFAULT_WIFI_CLIENT_PROFILE_ID, &profile
455       if (status != SL_STATUS_OK) {
456         printf("Failed to get client profile: 0x%lx\r\n", status);
457         return;
458       }
459       printf("\r\nSuccess to get client profile\r\n");
460
461       ip_address.type = SL_IPV4;
462       memcpy(&ip_address.ip.v4.bytes, &profile.ip.ip.v4.ip_address.bytes, sizeof(sl_ipv4_address_t));
463       print_sl_ip_address(&ip_address);
464       printf("\r\n");
465     }
466   }
```

Rejoining Operation

- To see these changes in action:
  1. Turn off the router or disable the Wi-Fi
  2. The Network lost, attempting rejoin debug message will be output when the device is unable to rejoin the network automatically
  3. The Failed to rejoin network: 0x10003 debug message will be repeated whilst the application code attempts to rejoin
  4. Turn on the router or enable the Wi-Fi
  5. The Wi-Fi client rejoined debug message will be output when the device rejoins
  6. The webpage can be reloaded in a browser to check the HTTP Server is still operating

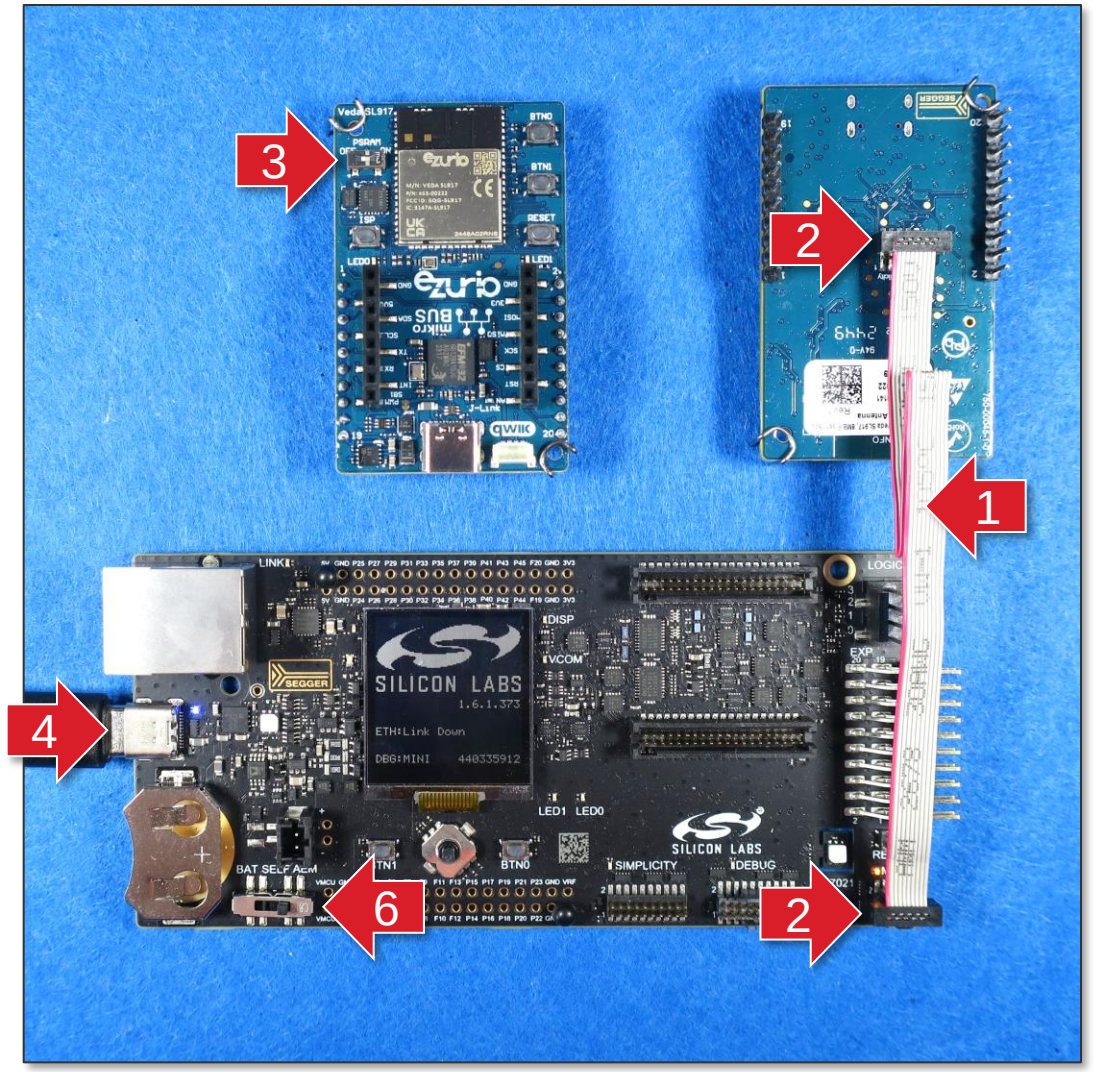


Energy Analyzer



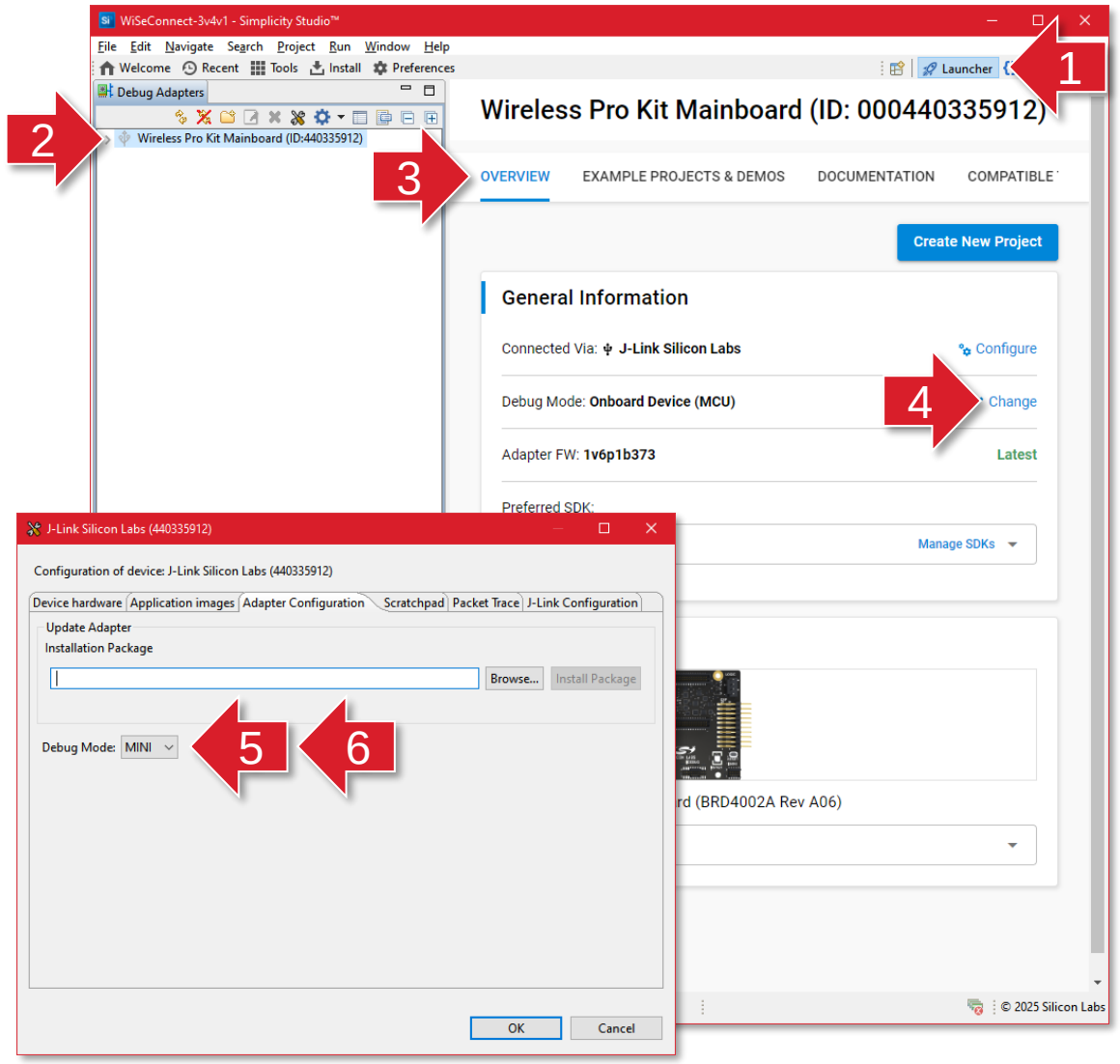
Energy Analyzer – Hardware Setup

- The Simplicity Studio IDE includes an Energy Analyzer that can be used to view the power consumption of a running application using a Pro Kit board
- When working with a Pro Kit and Explorer or Dev Kit:
  1. Connect the boards with a Mini Simplicity cable between the two Mini Simplicity Connectors
  2. Ensure that pin 1 on both boards are connected to each other
  3. On an Explorer Kit ensure the PSRAM switch is in the OFF position
  4. Ensure that only the Pro Kit board is powered via USB
- When working with a Pro Kit and Radio Board:
  5. No additional wiring is required
- In both scenarios:
  6. Ensure the switch on the Pro Kit is set to AEM (Advanced Energy Monitor)



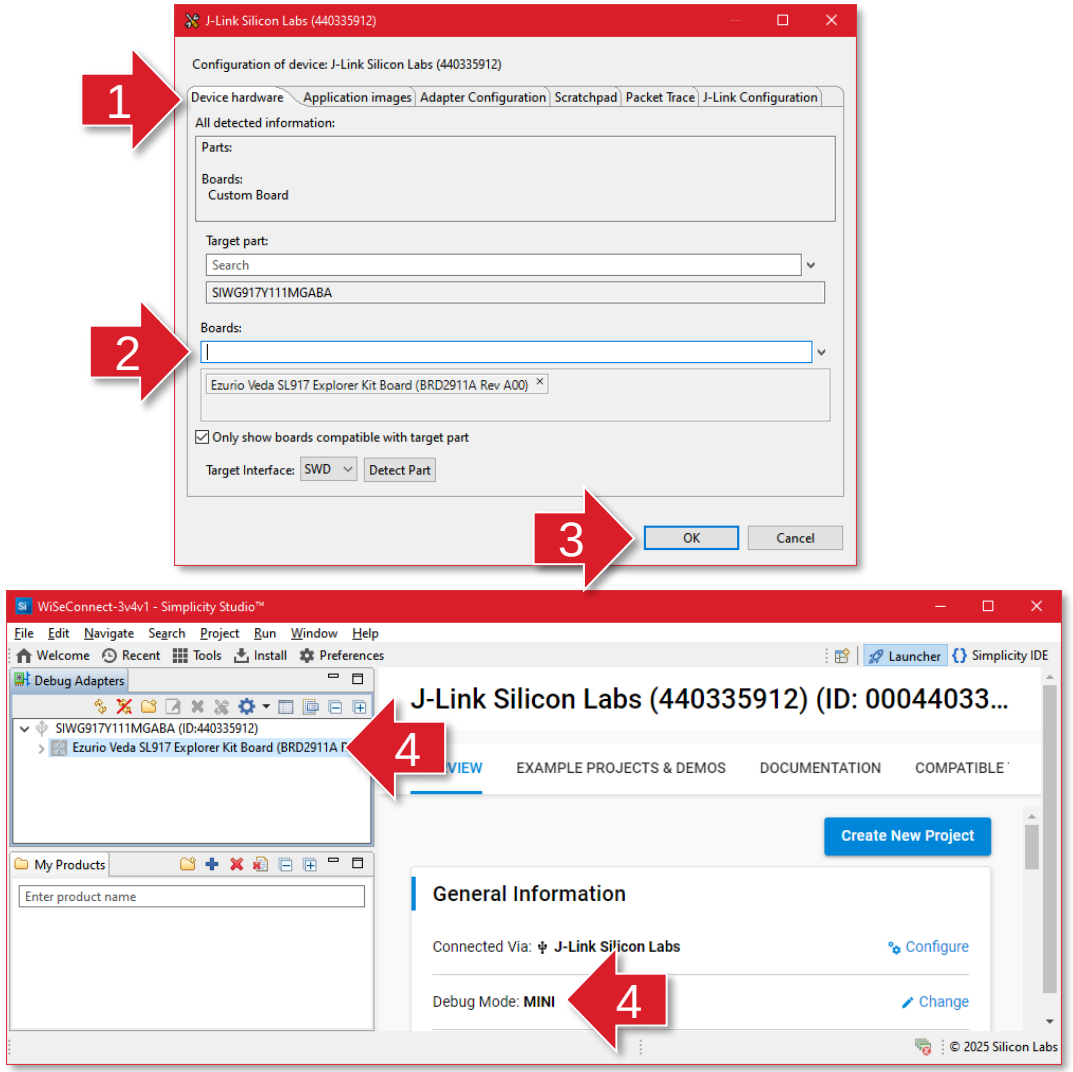
Energy Analyzer – Software Setup (1)

- To check and update the software setup in Simplicity Studio:
  1. Open the Launcher perspective
  2. Select the board in the Debug Adapters panel
  3. Select the Overview tab
- When working with a Pro Kit and Explorer or Dev Kit:
  4. In the General Information box, next to Debug Mode, click the Change link
  5. In the Configuration window, on the Adapter Configuration tab, set Debug Mode to MINI
- When working with a Pro Kit and Radio Board:
  6. Ensure the Debug Mode is set to MCU



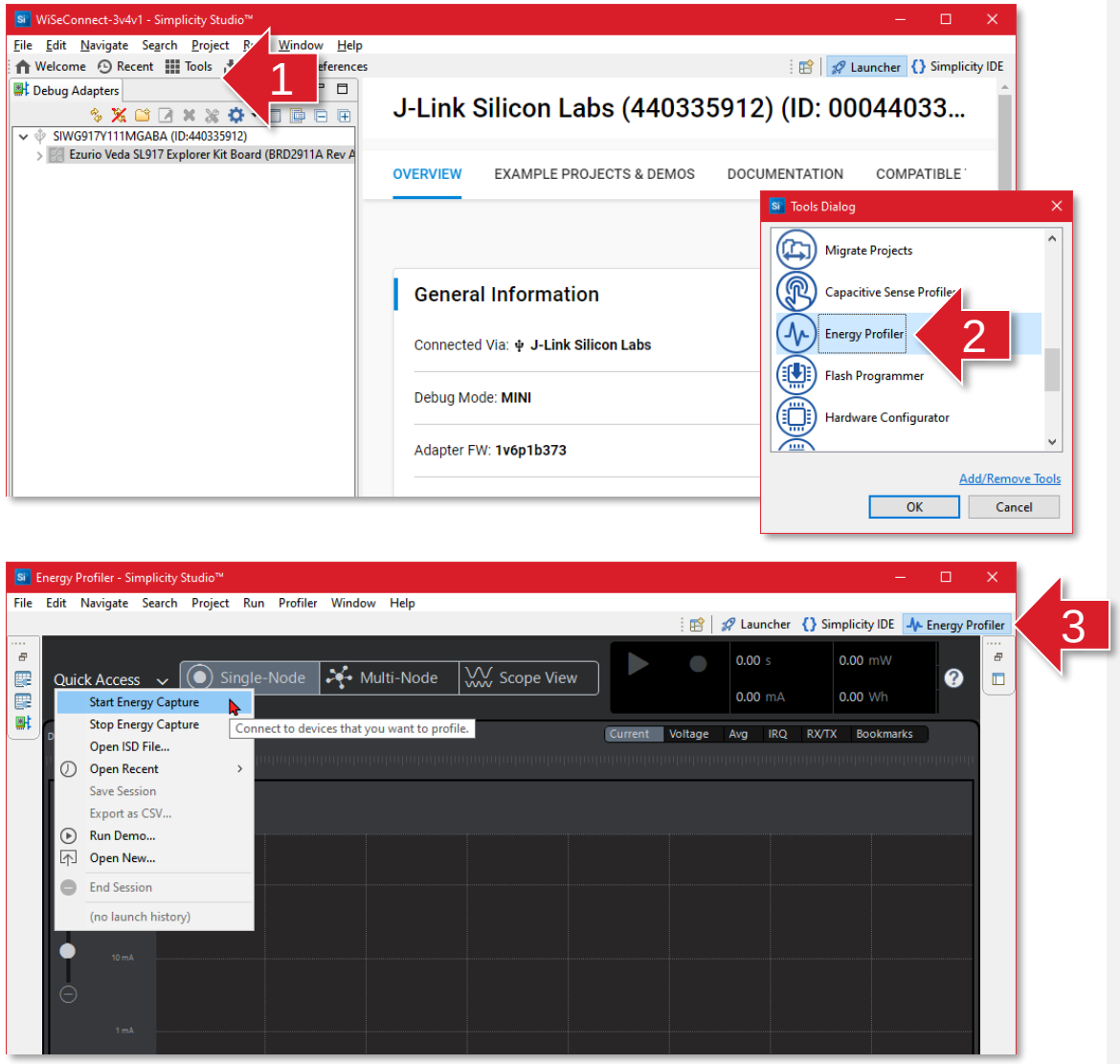
Energy Analyzer – Software Setup (2)

- When working with a Pro Kit and Explorer or Dev Kit:
  1. In the Configuration window, select the Device Hardware tab
  2. In the Boards editbox search for and select the board number for the Explorer or Dev Kit
     - BRD2911A for Ezurio Veda SL917 Explorer Kit
     - BRD2708A for Silicon Labs SiWG917Y Explorer Kit
     - BRD2605A for Silicon Labs SiWG917 Dev Kit
  3. Click the OK button
  4. Check the Debug Mode and external board are set correctly in the Launcher
- When working with a Pro Kit and Radio Board:
  5. The Radio Board is detected automatically



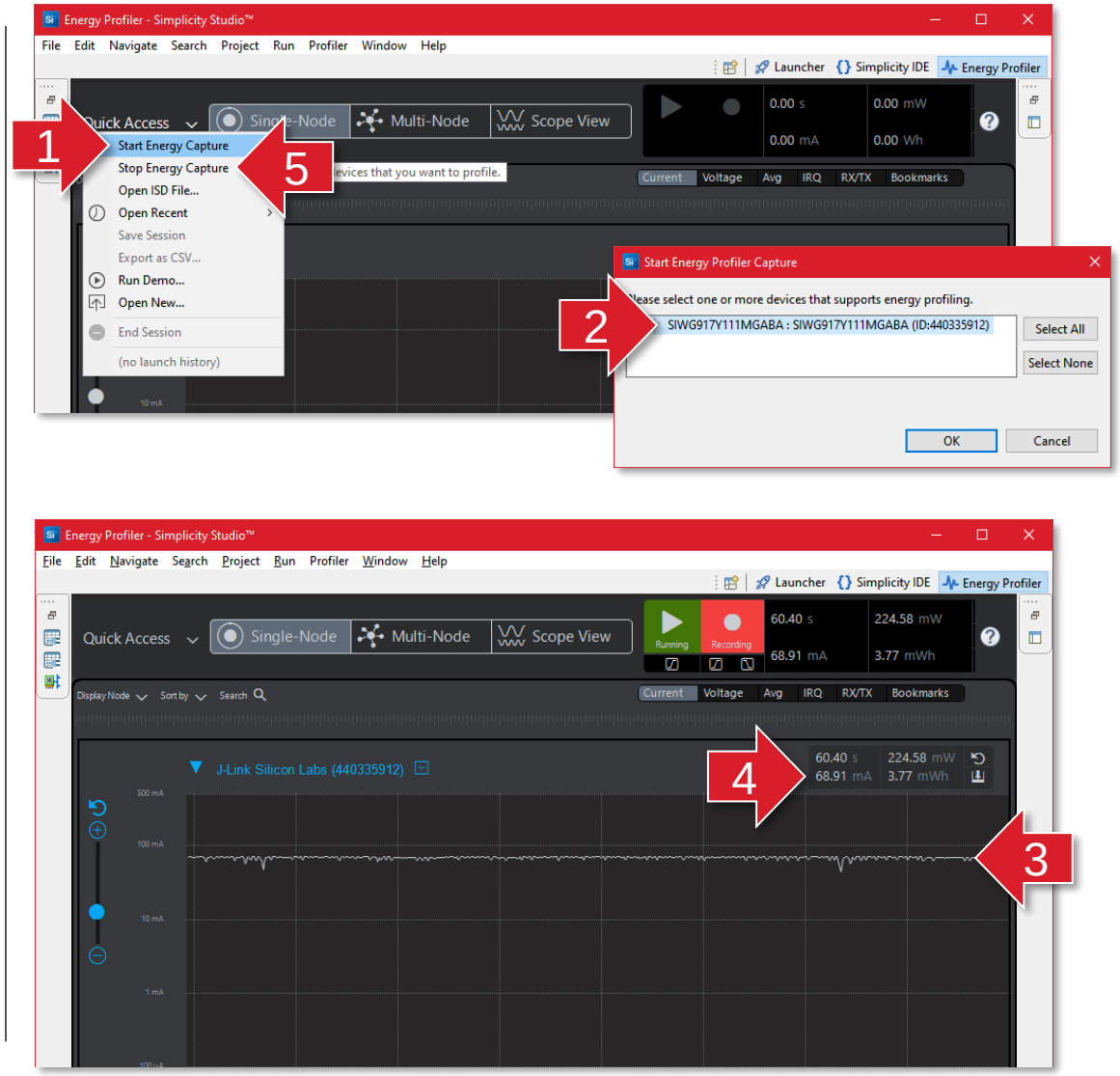
Energy Analyzer – Operation (1)

- To open the Energy Analyzer:
  1. On the toolbar, click the Tools button
  2. In the Tools dialog, select Energy Profiler then click the OK button
  3. Simplicity Studio will switch to the Energy Profiler perspective



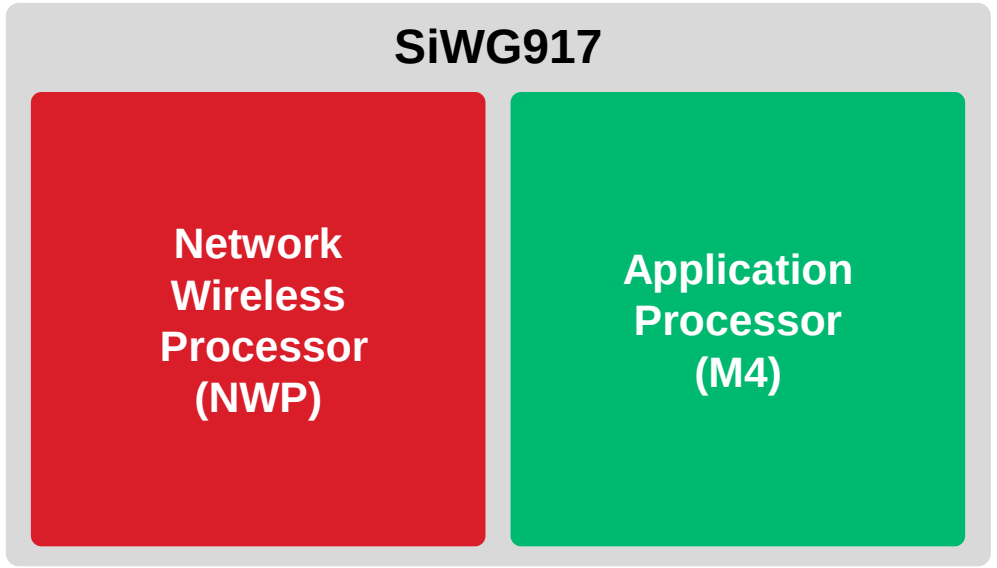
Energy Analyzer – Operation (2)

- To view the energy consumption of the running application:
  1. In the Quick Access dropdown, select Start Energy Capture
  2. In the Start Energy Profiler Capture dialog, select the connected board then click the OK button
  3. The current being used by the board is displayed in the graph
  4. The running application consumes around 70mA on average
  5. In the Quick Access dropdown, select Stop Energy Capture to end processing (you will need to do this before flashing the board)



SiWG917 Processors

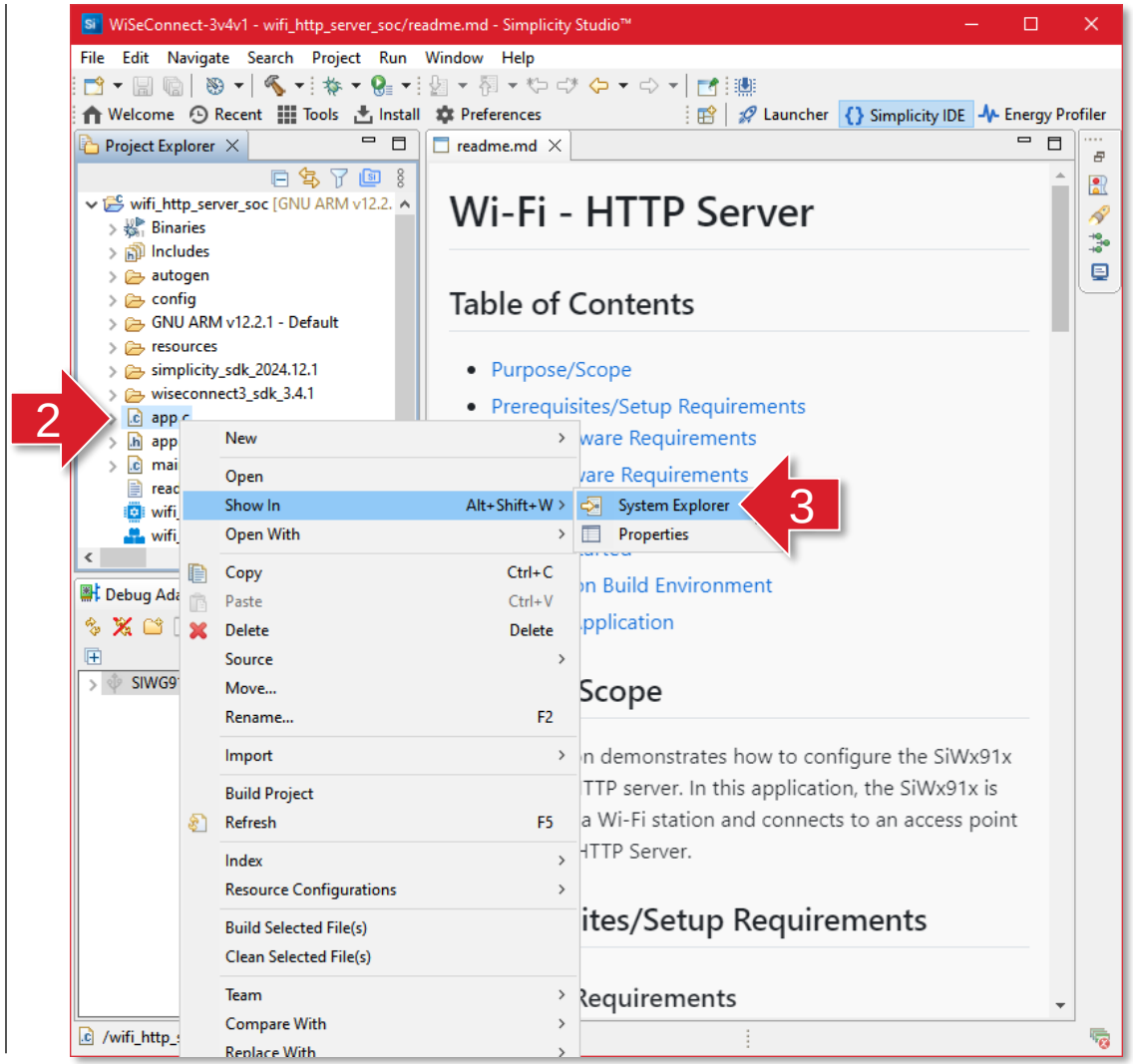
- The SiWG917 contains two processors:
  - The Network Wireless Processor (NWP) handles radio communications
  - The Application Processor (M4) runs the user application
- These processors can be placed into sleep modes when not in use to reduce power consumption



Network Wireless Processor Sleep

Update app.c – NWP Sleep

- To update app.c in the project folder:
  1. Locate the app.c file downloaded from GitHub (located in the dev_lab_wifi_http_server/source_3_nwp_sleep folder) and copy to the clipboard
  2. In Simplicity Studio's Simplicity IDE perspective, locate the app.c file in the Project Explorer panel
  3. Right-click and select Show in > System Explorer
  4. Paste the app.c file from the clipboard into the opened folder (replacing the existing file)



app.c – application_start()

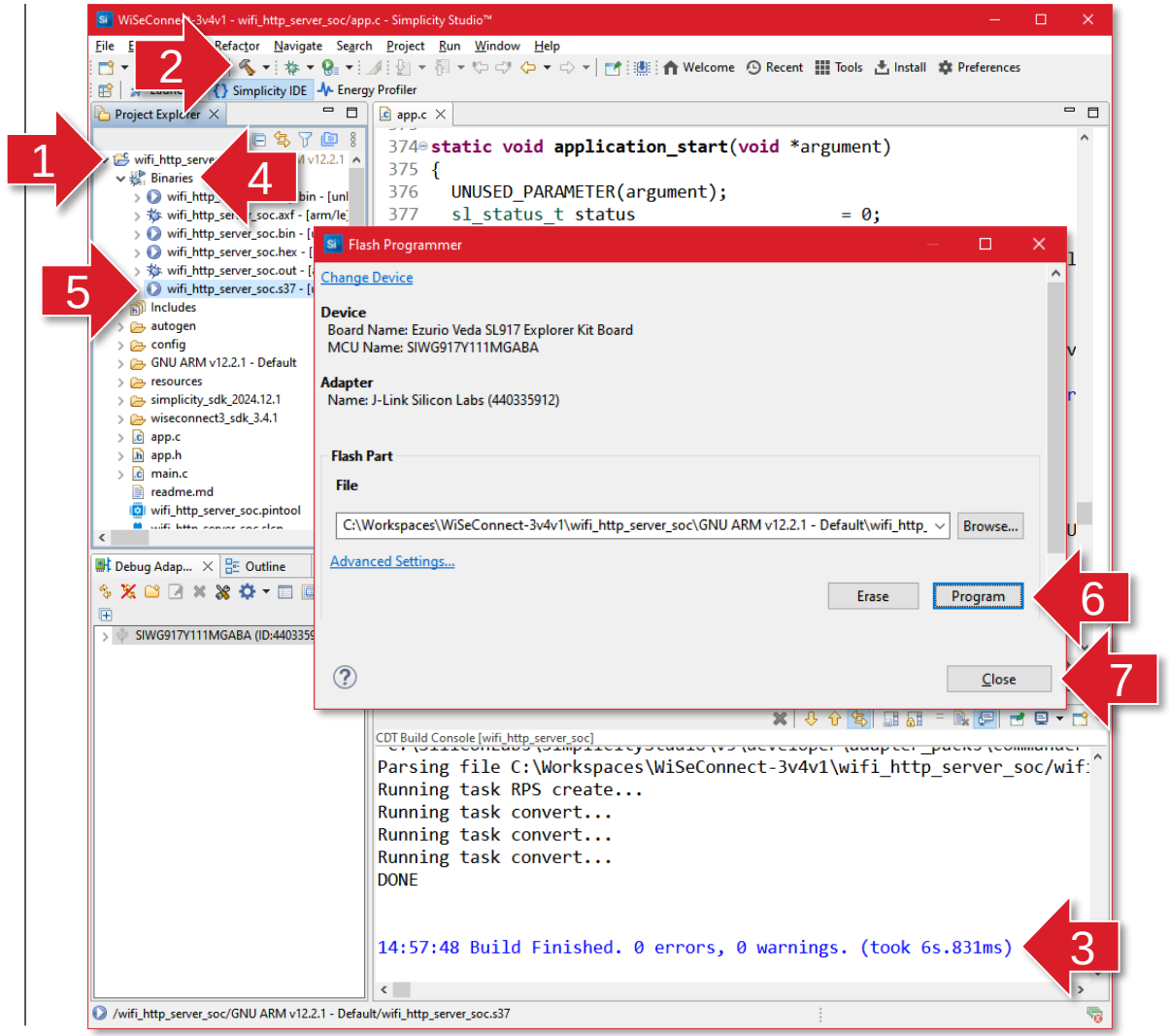
- Changes are made to the application_start() function which runs the application's thread:
  1. The sl_wifi_filter_broadcast() function is called to reduce the number of broadcast messages the NWP needs to wake up and process
     - Defines, set near the start of app.c are used for parameters
  2. The sl_wifi_set_performance() is called which allows the NWP to sleep when idle
  3. The performance_profile structure configures controls the sleep mode, here it allows power saving when associated with a Wi-Fi AP
     - The structure is configured near the start of the application_start() function
- These functions are not called until after the HTTP server is started
  - The NWP should be kept awake until ready to run the main part of the application

```
status = sl_http_server_start(&server_handle);
if (status != SL_STATUS_OK) {
  printf("\r\n Server start fail:%lx\r\n", status);
  return;
}
printf("\r\n Server start done\r\n");

// reduces packets nwp needs to wake for
status = sl_wifi_filter_broadcast(BROADCAST_DROP_THRESHOLD, BROADCAST_IN_TIM, BROADCAST_TIM_TILL_NEXT_COMMAND);
if (status != SL_STATUS_OK) {
  printf("\r\nsl_wifi_filter_broadcast Failed, Error Code : 0x%lX\r\n", status);
  return;
}
// set performance profile
status = sl_wifi_set_performance_profile(&performance_profile);
if (status != SL_STATUS_OK) {
  printf("\r\nPower save configuration Failed, Error Code : 0x%lX\r\n", status);
  return;
}

is_server_running = true;
while (1) {
  osDelay(1000);
  seconds++;
  button0 = sl_si91x_button_pin_state(SL_BUTTON_BTN0_PIN);
  button1 = sl_si91x_button_pin_state(SL_BUTTON_BTN1_PIN);
```

```
static void application_start(void *argument)
{
  UNUSED_PARAMETER(argument);
  sl_status_t status                    = 0;
  sl_http_server_config_t server_config = { 0 };
  sl_wifi_performance_profile_t performance_profile = { .profile = ASSOCIATED_POWER_SAVE_LOW_LATENCY };
```

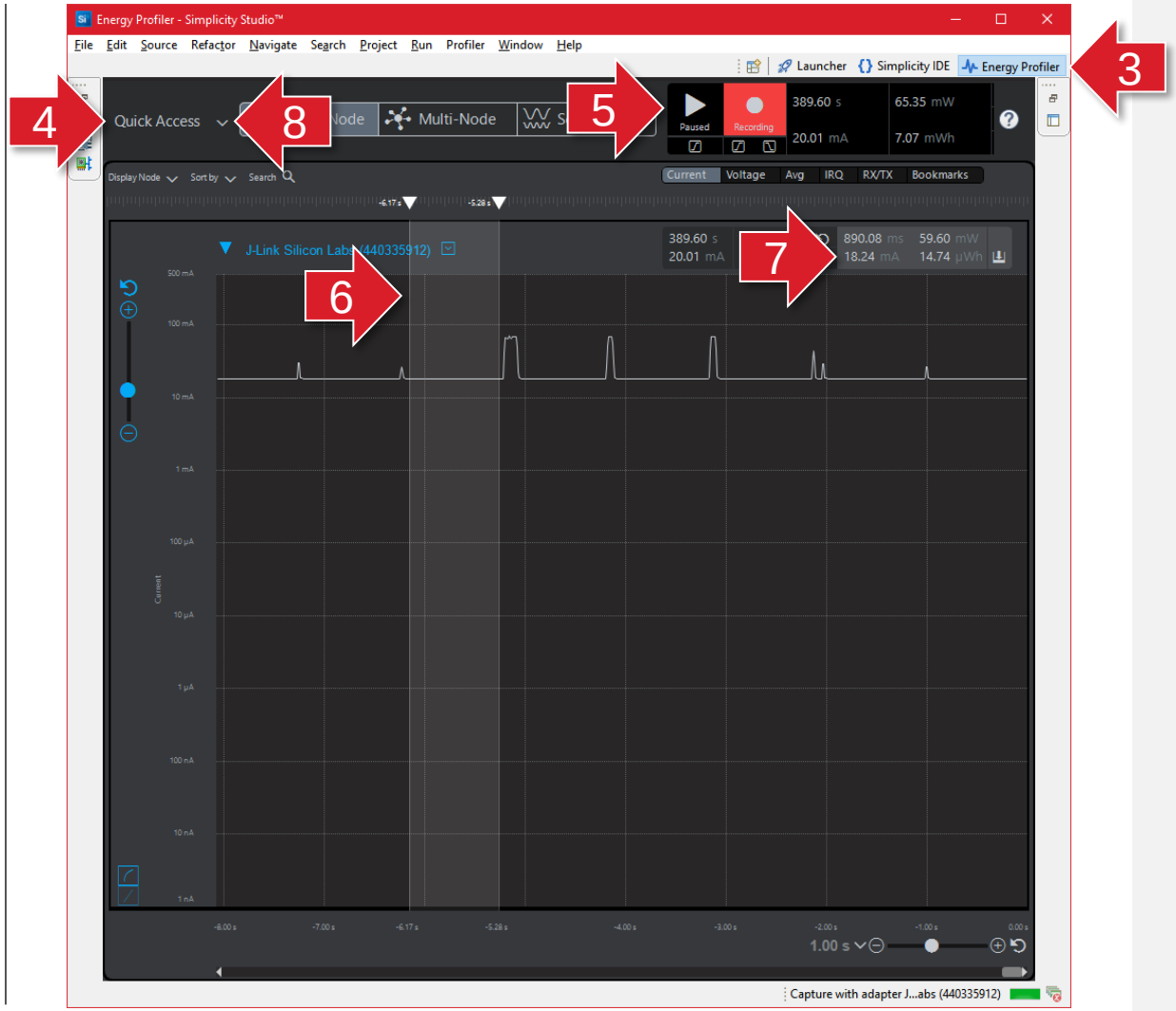
Compile and Flash

- To compile the software:
  1. In the Project Explorer panel, select the top-level project
  2. Click the Build (hammer) button on the toolbar
  3. Compilation progress is shown in the Console panel
- To flash the software:
  4. In the Project Explorer panel, open the Binaries folder
  5. Locate the .s37 file, right click and select Flash to device…
  6. In the Flash Programmer window, click the Program button
  7. When complete, click the Close button



NWP Sleep Energy Analyzer

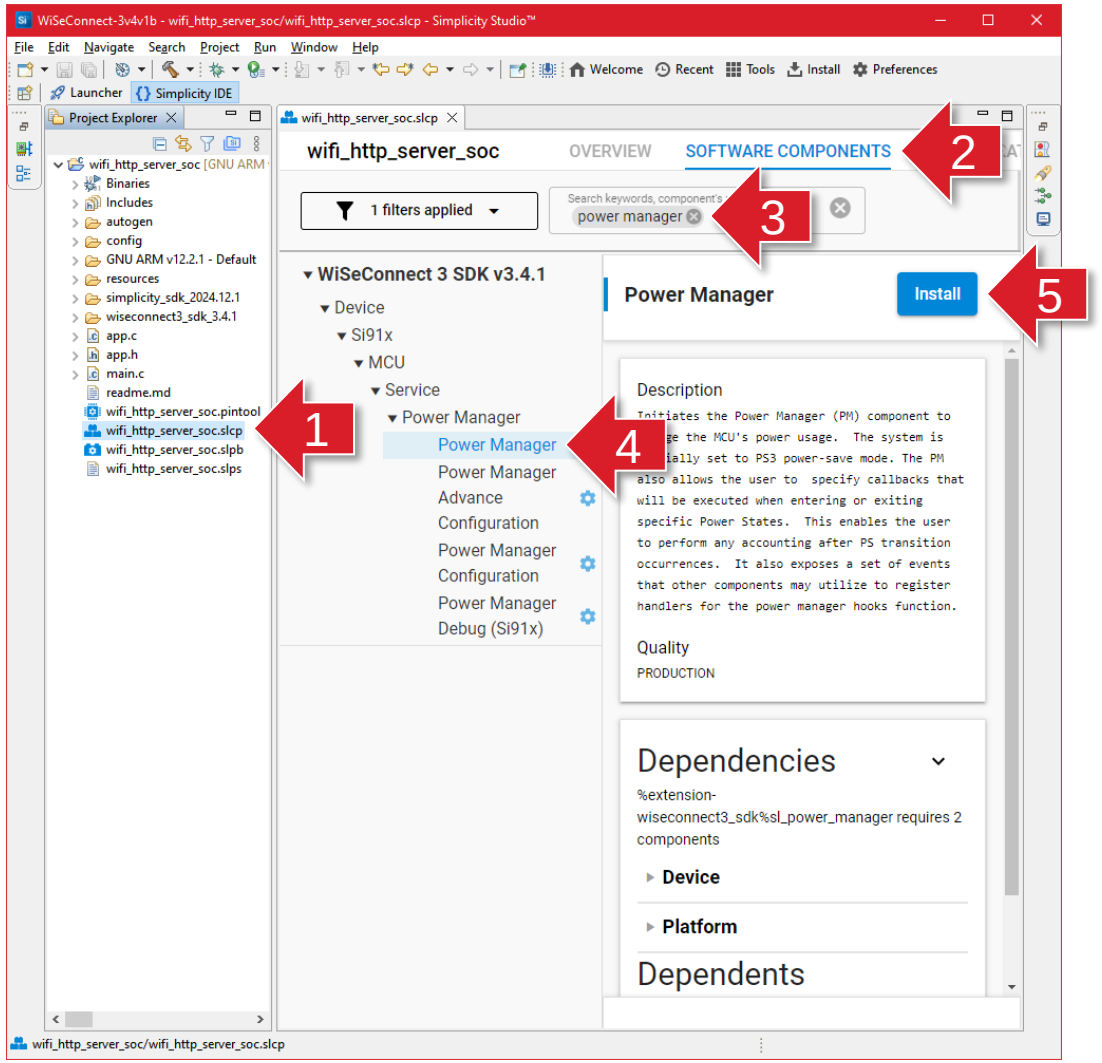
- To measure the energy when the NWP is sleeping:
  1. Reset the board and observe the debug to be sure the HTTP Server is started
  2. Open the browser so the web page is being requested
  3. Open the Energy Profiler perspective
  4. In the Quick Access dropdown, select Start Energy Capture then select the connected board
  5. Allow the collection of some data then Pause the live playback
  6. Select a portion of the graph with the mouse where the device is idle
  7. We can see that when the NWP is sleeping the running application is using around 18 mA
  8. In the Quick Access dropdown, select Stop Energy Capture to allow the board to be flashed later



Application Processor Sleep

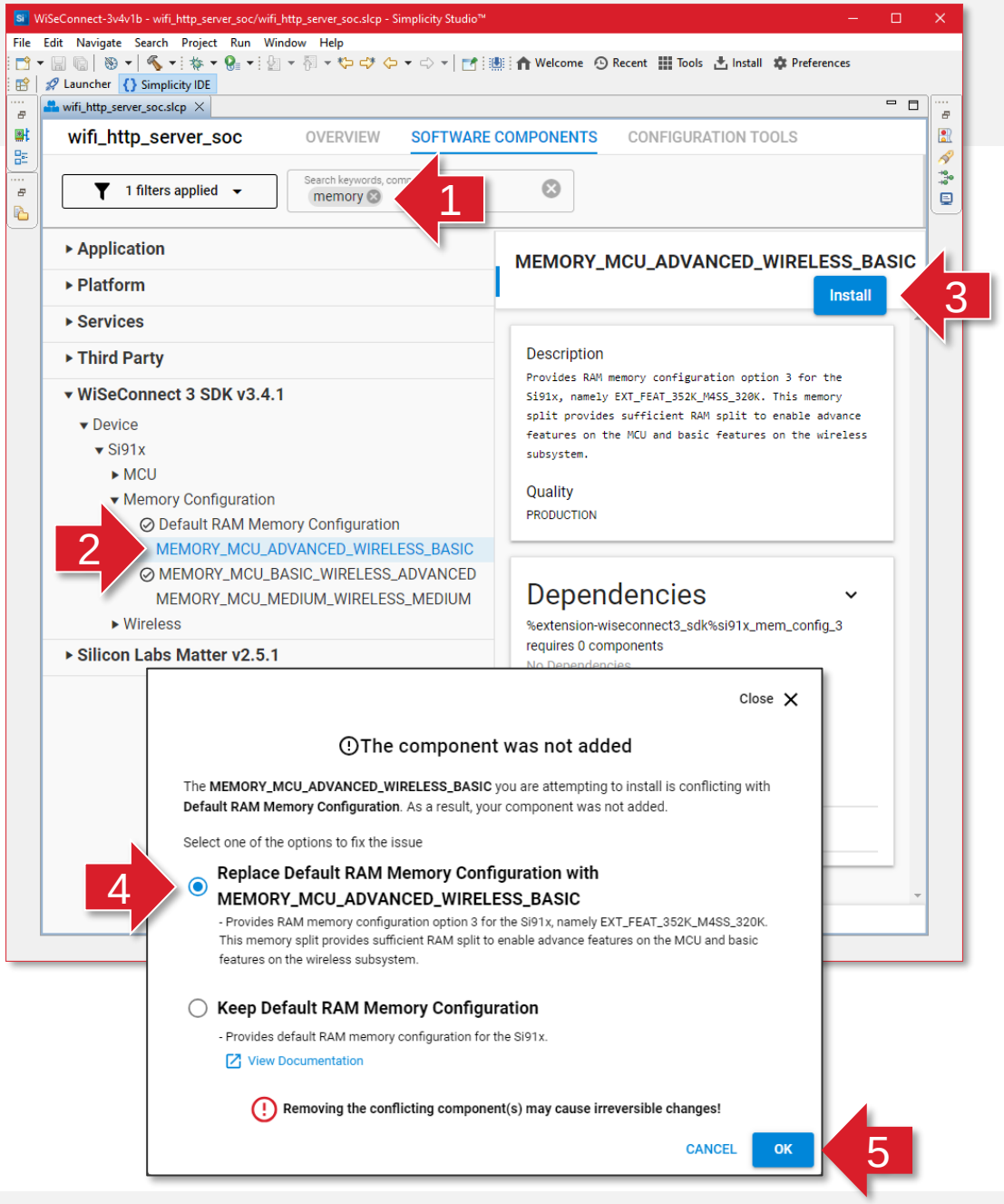
Power Manager Software Component

- The Power Manager Software Component allows the application processor to sleep when idle, to enable it:
  1. From the Project Explorer panel, open the .slcp file
  2. Select the Software Components tab
  3. Search for Power Manager
  4. Select WiSeConnect 3 SDK > Device > Si91x > MCU > Service > Power Manager > Power Manager
  5. Click the Install button
- In this default configuration the NWP is allocated 480 kb of RAM and the M4 is allocated 192 kb
  - Note that the RAM allocated to the NWP must always be retained during sleep



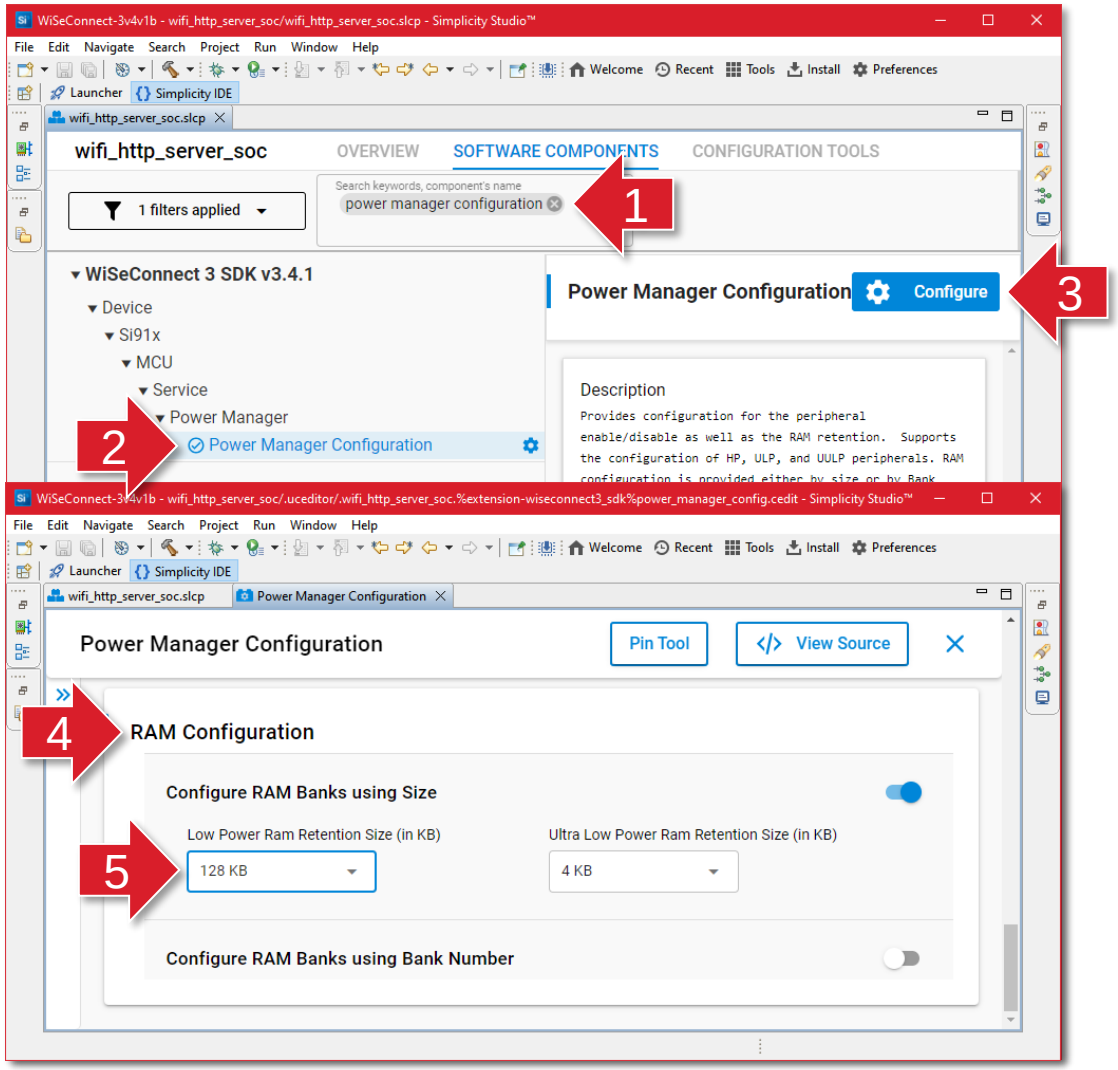
Memory Configuration

- We can change the default RAM allocation:
  1. From the Software Components tab, search for Memory
  2. Select WiSeConnect 3 SDK > Device > Si91x > Memory Configuration > MEMORY_MCU_ADVANCED_WIRELESS_BASIC
  3. Click the Install button
  4. When prompted, select Replace Default RAM Memory Configuration with MEMORY_MCU_ADVANCED_WIRELESS_BASIC
  5. In the prompt box, click the OK button
- This change assigns 352 kb to the NWP and 320 kb is to the M4
  - This reduces the RAM retained during sleep



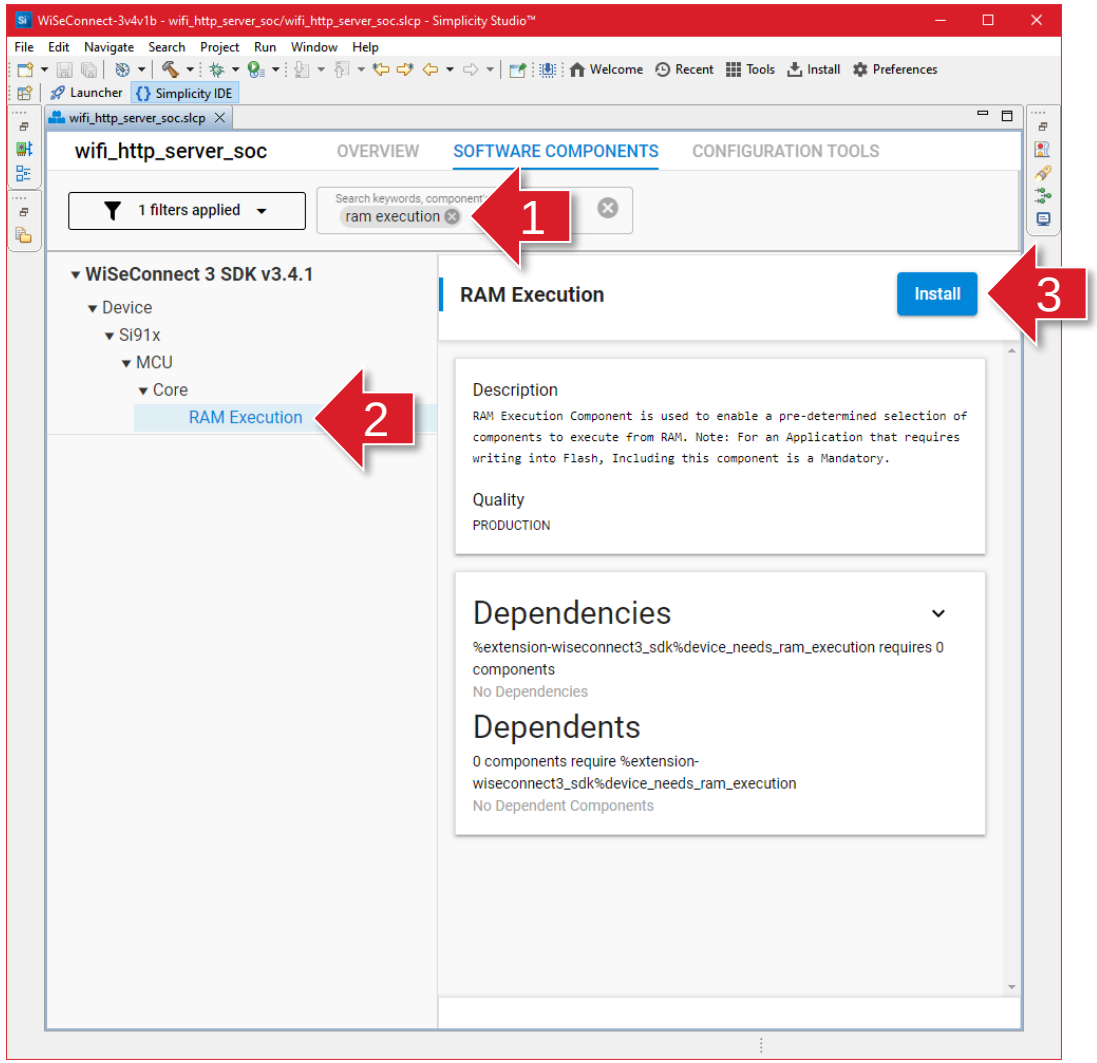
Power Manager Configuration

- We can further configure how much M4 RAM is retained during sleep:
  1. From the Software Components tab, search for Power Manager Configuration
  2. Select WiSeConnect 3 SDK > Device > Si91x > MCU > Service > Power Manager > Power Manager Configuration
  3. Click the Configure button
  4. In the Power Manager Configuration window, scroll down to RAM Configuration
  5. Change the Low Power Ram Retention Size from 320 KB to 128 KB
- Only 128 kb of the M4's 320 kb of RAM will now be retained during sleep
  - The amount of RAM that needs to be retained for the M4 is dependent on the complexity of the application



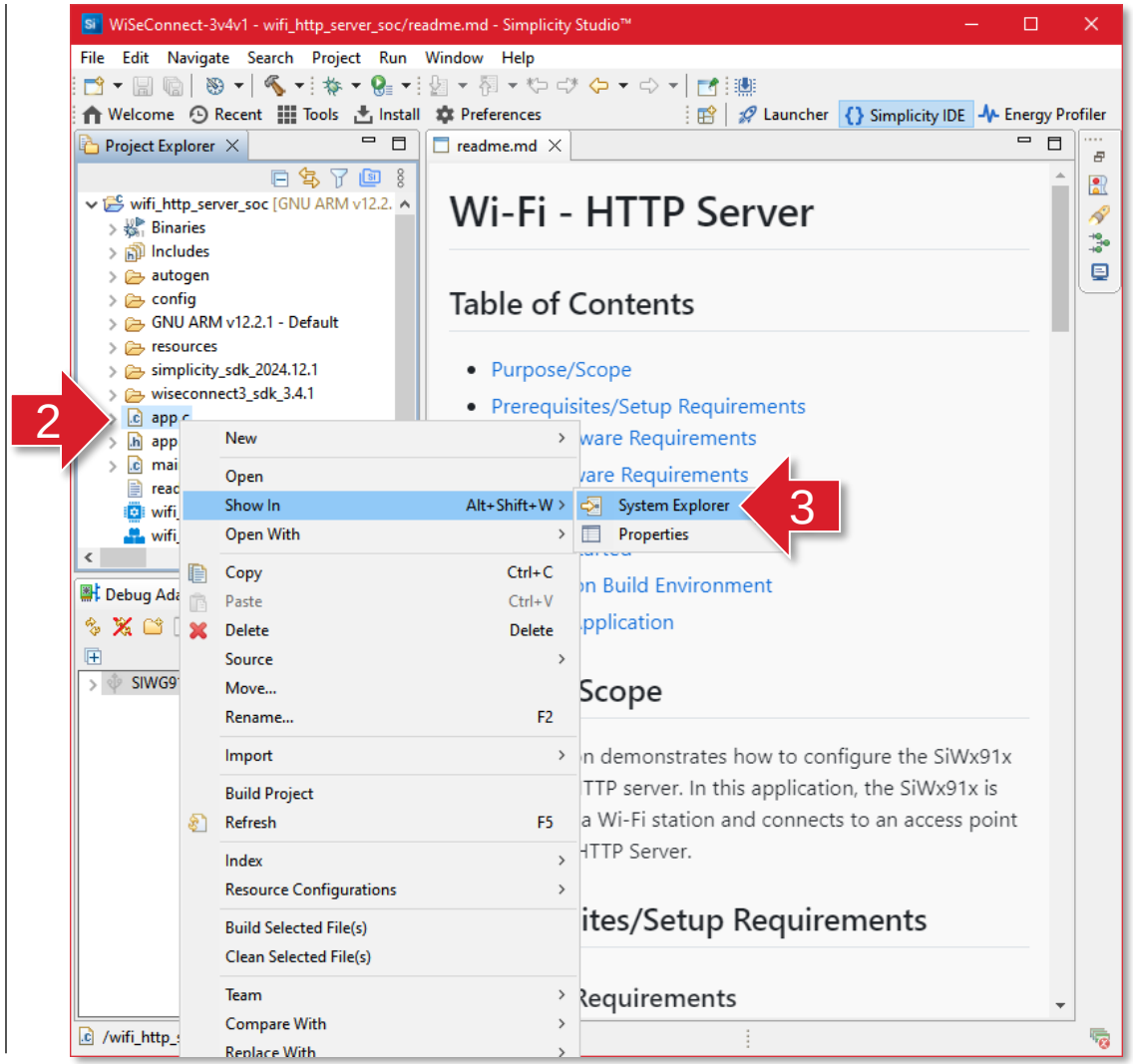
RAM Execution Software Component

- The RAM Execution Software component allows a pre-determined selection of components to execute from RAM, enable it:
  1. From the Software Components tab, search for RAM Execution
  2. Select WiSeConnect 3 SDK > Device > Si91x > MCU > Core > RAM Execution
  3. Click the Install button
- Flash is turned off during sleep, this allows code to run from RAM to turn on flash and restore context for the application to continue running



Update app.c – M4 Sleep

- To update app.c in the project folder:
  1. Locate the app.c file downloaded from GitHub (located in the dev_lab_wifi_http_server/source_4_nwp_m4_sleep folder) and copy to the clipboard
  2. In Simplicity Studio's Simplicity IDE perspective, locate the app.c file in the Project Explorer panel
  3. Right-click and select Show in > System Explorer
  4. Paste the app.c file from the clipboard into the opened folder (replacing the existing file)



app.c – application_start() – Button 1 Reinitialization

- The GPIO configuration for button 1 is not retained through sleep and needs to be reinitialized after waking
  - Button 0 uses a UULP GPIO and its configuration is retained through sleep
- Near the start of app.c:
  1. sl_driver_gpio.h and sl_si91x_driver_gpio.h are included to provide access to GPIO APIs
- At the beginning of the application_start() function:
  2. A sl_si91x_gpio_pin_config_t structure is created to re-configure the Button 1 GPIO
- In the main loop of the application_start() function:
  3. After exiting the delay, the sl_gpio_driver_init() and sl_spio_set_configuration() functions are called to re-configure Button 1

1

```
#include "sl_driver_gpio.h"
#include "sl_si91x_driver_gpio.h"

#include "sl_si91x_power_manager.h"
```

2

```
static void application_start(void *argument)
{
  UNUSED_PARAMETER(argument);
  sl_status_t status                    = 0;
  sl_http_server_config_t server_config = { 0 };
  sl_wifi_performance_profile_t performance_profile
    = { .profile = ASSOCIATED_POWER_SAVE_LOW_LATENCY };
  sl_si91x_gpio_pin_config_t sl_button1_pin_config
    = { { button_btn1.port, button_btn1.pin }, GPIO_INPUT };

  sl_si91x_power_manager_add_ps_requirement(SL_SI91X_POWER_MANAGER_PS3);
```

3

```
  is_server_running = true;
  while (1) {
    sl_si91x_power_manager_remove_ps_requirement(SL_SI91X_POWER_MANAGER_PS3);
    osDelay(1000);
    sl_si91x_power_manager_add_ps_requirement(SL_SI91X_POWER_MANAGER_PS3);
    // reinitialise button 1 after sleep
    sl_gpio_driver_init();
    sl_gpio_set_configuration(sl_button1_pin_config);
    // update data
    seconds++;
    button0 = sl_si91x_button_pin_state(SL_BUTTON_BTN0_PIN);
    button1 = sl_si91x_button_pin_state(SL_BUTTON_BTN1_PIN);
```

app.c – application_start() – Power Manager

- When running with multiple application tasks it is important to ensure the application processor remains awake whilst running application code
  - This application uses a single task, so these changes are not strictly necessary here but represent best practice
- Near the start of app.c:
  1. sl_si91x_power_manager.h is included to provide access to Power Manager APIs
- At the beginning of the application_start() function:
  2. A call to sl_si91x_power_manager_add_ps_requirement() is made to keep the M4 awake during processing
- In the main loop of the application_start() function:
  3. Calls to sl_si91x_power_manager_remove_ps_requirement() and sl_si91x_power_manager_add_ps_requirement() are placed before and after the osDelay() call to allow the M4 to sleep

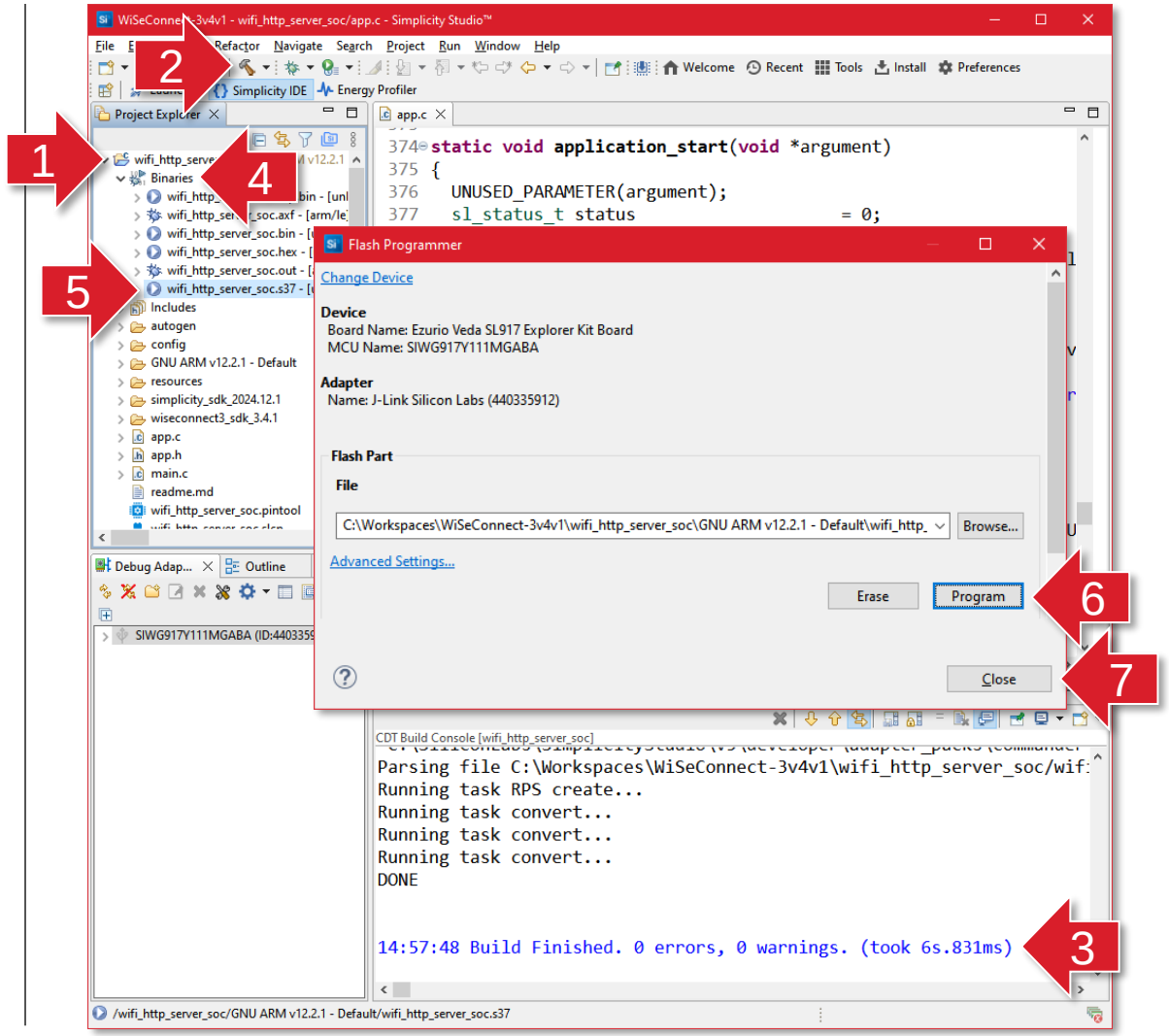
```
49 #include "sl_driver_gpio.h"
50 #include "sl_si91x_driver_gpio.h"
51
1  #include "sl_si91x_power_manager.h"
```

```
379 static void application_start(void *argument)
380 {
381   UNUSED_PARAMETER(argument);
382   sl_status_t status                    = 0;
383   sl_http_server_config_t server_config = { 0 };
384   sl_wifi_performance_profile_t performance_profile
385     = { .profile = ASSOCIATED_POWER_SAVE_LOW_LATENCY };
386   sl_si91x_gpio_pin_config_t sl_button1_pin_config
387     = { { button_btn1.port, button_btn1.pin }, GPIO_INPUT };
388
2  sl_si91x_power_manager_add_ps_requirement(SL_SI91X_POWER_MANAGER_PS3);
```

```
462   is_server_running = true;
463   while (1) {
3  sl_si91x_power_manager_remove_ps_requirement(SL_SI91X_POWER_MANAGER_PS3);
465     osDelay(1000);
466     sl_si91x_power_manager_add_ps_requirement(SL_SI91X_POWER_MANAGER_PS3);
467     // reinitialise button 1 after sleep
468     sl_gpio_driver_init();
469     sl_gpio_set_configuration(sl_button1_pin_config);
470     // update data
471     seconds++;
472     button0 = sl_si91x_button_pin_state(SL_BUTTON_BTN0_PIN);
473     button1 = sl_si91x_button_pin_state(SL_BUTTON_BTN1_PIN);
```

Compile and Flash

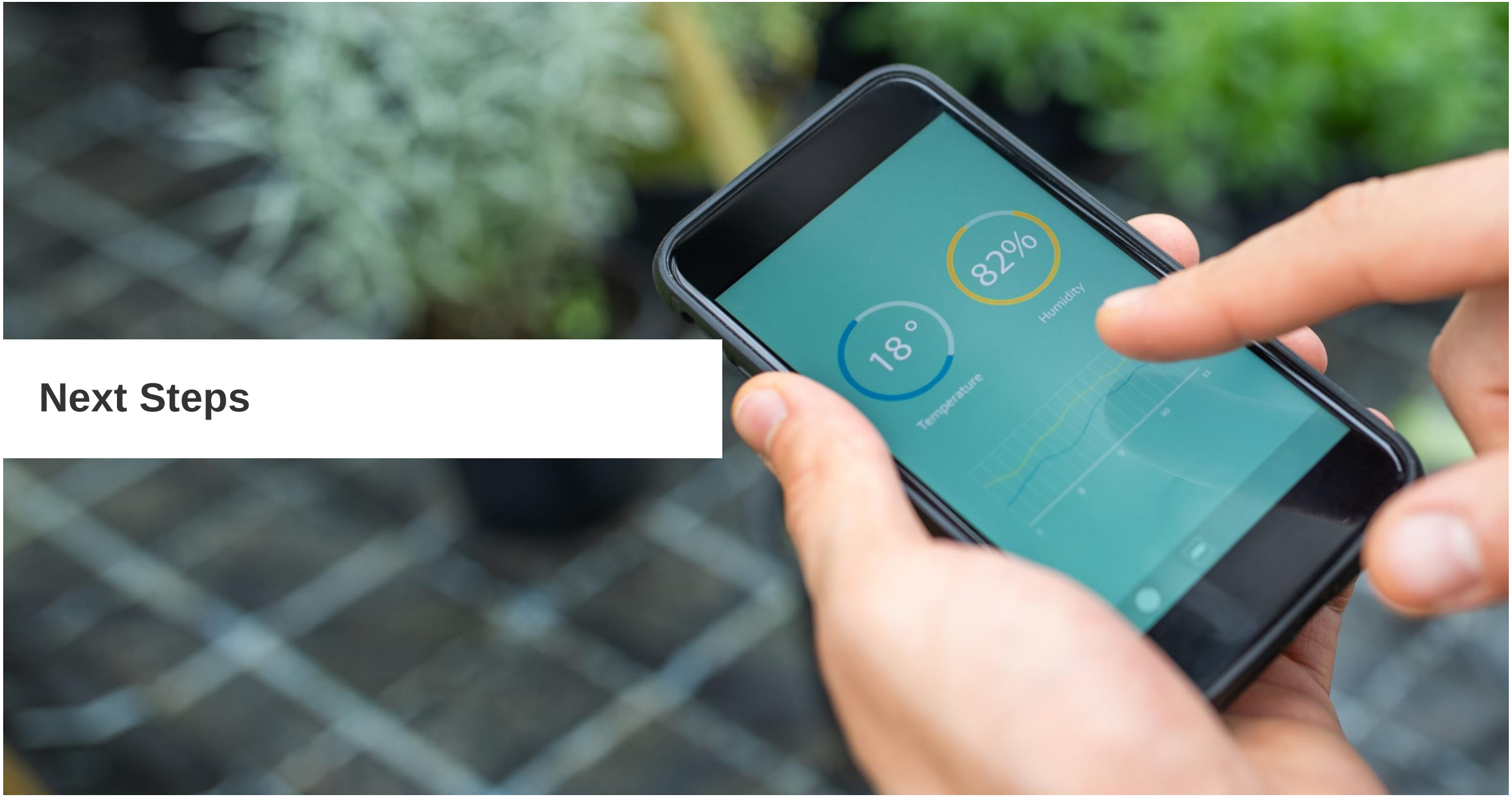
- To compile the software:
  1. In the Project Explorer panel, select the top-level project
  2. Click the Build (hammer) button on the toolbar
  3. Compilation progress is shown in the Console panel
- To flash the software:
  4. In the Project Explorer panel, open the Binaries folder
  5. Locate the .s37 file, right click and select Flash to device…
  6. In the Flash Programmer window, click the Program button
  7. When complete, click the Close button



NWP and M4 Sleep Energy Analyzer

- To measure the energy when the NWP and M4 are sleeping:
  1. Reset the board and observe the debug to be sure the HTTP Server is started
  2. Open the browser so the web page is being requested
  3. Open the Energy Profiler perspective
  4. In the Quick Access dropdown, select Start Energy Capture then select the connected board
  5. Allow the collection of some data then Pause the live playback
  6. Select a portion of the graph with the mouse where the device is idle
  7. We can see that when the NWP and M4 are sleeping the running application is using around 40 uA
  8. In the Quick Access dropdown, select Stop Energy Capture to allow the board to be flashed later

Next Steps



Next Steps – Sleep

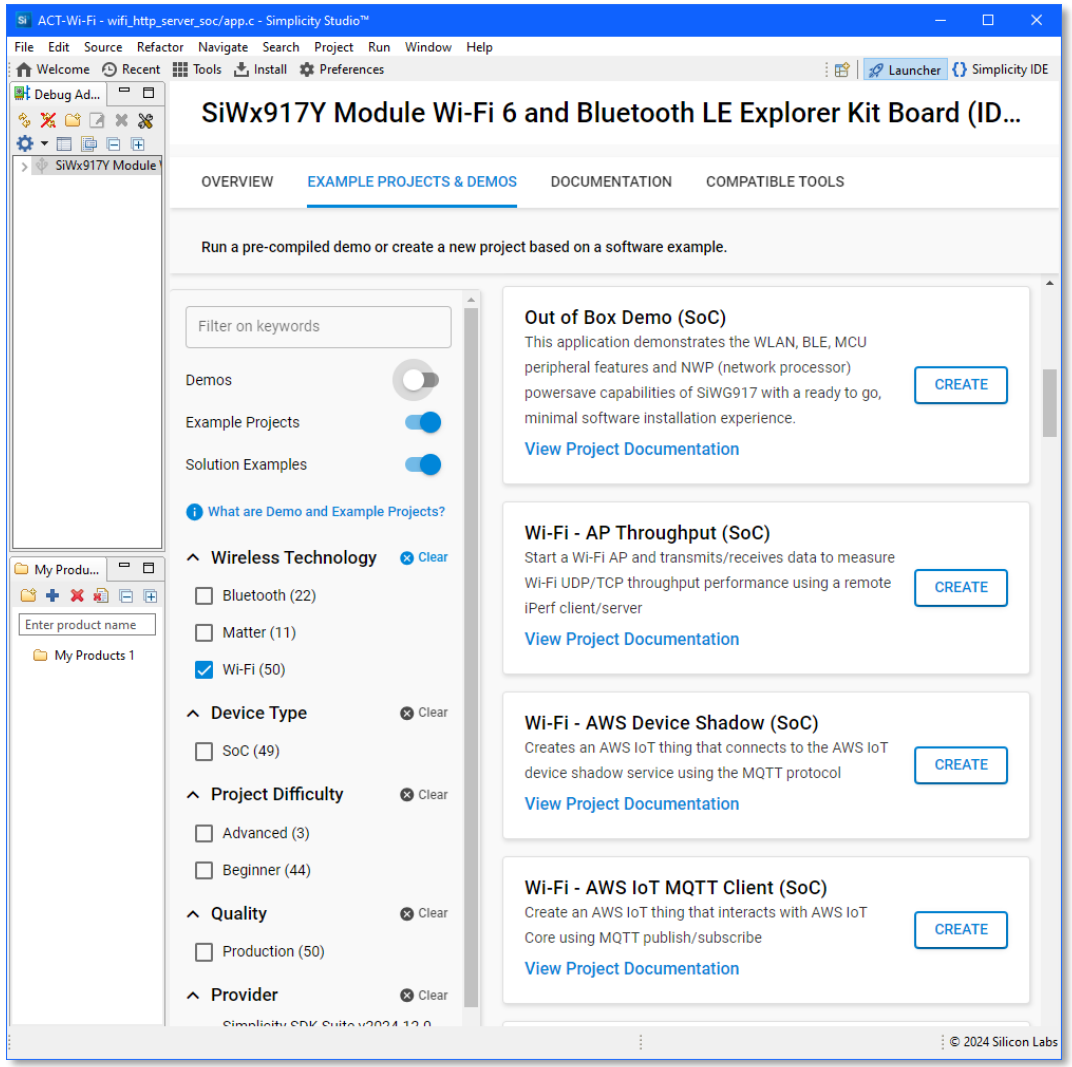
- The HTTP Server now has basic support for sleeping, but we've not fine-tuned its performance:
  - There are extensive debug messages that can be removed for a production build
  - The device wakes once per second to read buttons and update the timer, it would be more efficient to read the buttons only when a HTTP request is made and take the seconds reading from a running timer
    - The application has been structured with regular readings as a device may need to take regular readings from a sensor, regardless of when and how the data is retrieved
  - For a sensor type application, structuring it to send data periodically can be more efficient than having a device that needs to be available to respond to requests at any time
    - This also allows the use of the TWT feature of Wi-Fi 6 to save additional power
- A Powersaving Application Note is being worked on to provide additional guidance

```
Got request [/test] of type : GET with data length : 0
Got request query parameter count : 0
Got header count : 10
Key: Host, Value: 192.168.8.200
Key: User-Agent, Value: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:137.0
efox/137.0
Key: Accept, Value: text/html,application/xhtml+xml,application/xml;q=0.9,
Key: Accept-Language, Value: en-GB,en;q=0.5
Key: Accept-Encoding, Value: gzip, deflate
```

```
459   is_server_running = true;
460   while (1) {
461     sl_si91x_power_manager_remove_ps_requirement(SL_SI91X_POWER_MANAGER_PS3);
462     osDelay(1000);
463     sl_si91x_power_manager_add_ps_requirement(SL_SI91X_POWER_MANAGER_PS3);
464     // reinitialise button 1 after sleep
465     sl_gpio_driver_init();
466     sl_gpio_set_configuration(sl_button1_pin_config);
467     // update data
468     seconds++;
469     button0 = sl_si91x_button_pin_state(SL_BUTTON_BTN0_PIN);
470     button1 = sl_si91x_button_pin_state(SL_BUTTON_BTN1_PIN);
```

Next Steps – Wi-Fi

- There are lots more Wi-Fi examples available in Simplicity Studio
  - Choose the one closest to your final application as your starting point
  - The Out of Box Demo (SoC) provides a good overview showing how to join a Wi-Fi network by providing the SSID and password over Bluetooth, pinging and MQTT data transfer
  - The Powersave examples show how to enter various sleep modes
  - There are many Matter over Wi-Fi examples
- For more fully-featured examples check the Silicon Labs Wi-Fi Examples repository on GitHub: https://github.com/SiliconLabs/wifi_applications
- Documentation on the Wi-Fi APIs is available from the Documentation page in the Launcher and also online: https://docs.silabs.com/wiseconnect/latest/wiseconnect-developing-with-wiseconnect-sdk
- Subscribe to the Silicon Labs YouTube channel, where we will be adding Wi-Fi tutorials in the future: https://www.youtube.com/@ViralSilabs/videos



SILICON LABS

Thank You

Questions?