



Project Configurator & Gecko Platform - Lab

This lab procedure walks through the steps to create a new project utilizing Simplicity Studio 5's Gecko Platform and Project Configurator Wizard. The lab highlights the new process action programming model, a few of the software components available within Project Configurator, and goes into detail about integrating application code available through our peripheral examples repository on GitHub.

There are four project stages throughout the lab signifying a different software addition to the demonstration, with each stage building upon the last.

The first part shows how to build a new project which uses the CLI interface and onboard sensor of the Thunderboard Sense 2, starting with an Empty C project generated by the Project Configurator Wizard. The second part demonstrates how to integrate the ADC peripheral with the use of example code provided in our GitHub repository. The third part outlines the addition of a real-time operating system and the changes in project model with this addition. The fourth and final portion of this lab adds Bluetooth to the project.

KEY POINTS

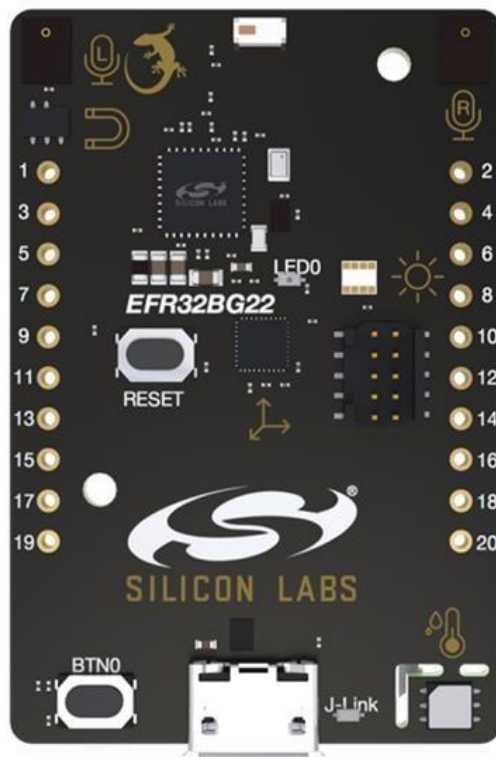
- Gecko Platform and Project Configurator Wizard
- Software component installation and configuration
- Addition of application code (ADC example)
- RTOS and Bluetooth project integration

Prerequisites

1 Prerequisites

For this lab you will need the following:

- EFR32MG12 Thunderboard Sense 2 (SLTB004A)
- Micro-USB to USB Type-A cable (Not included with Thunderboard)
- Simplicity Studio v5
 - Bluetooth SDK 3.1.0.0 or later
 - Gecko SDK Suite 3.1.0 or later
- [BLE Scanner](#) mobile app (or other generic Bluetooth scanning application for Bluetooth demo)
- Accompanying lab source code provided in compressed zip - https://www.dropbox.com/s/4xbz4uzxq9l63nc/v5_ProjectConfigurator_Lab.zip?dl=0



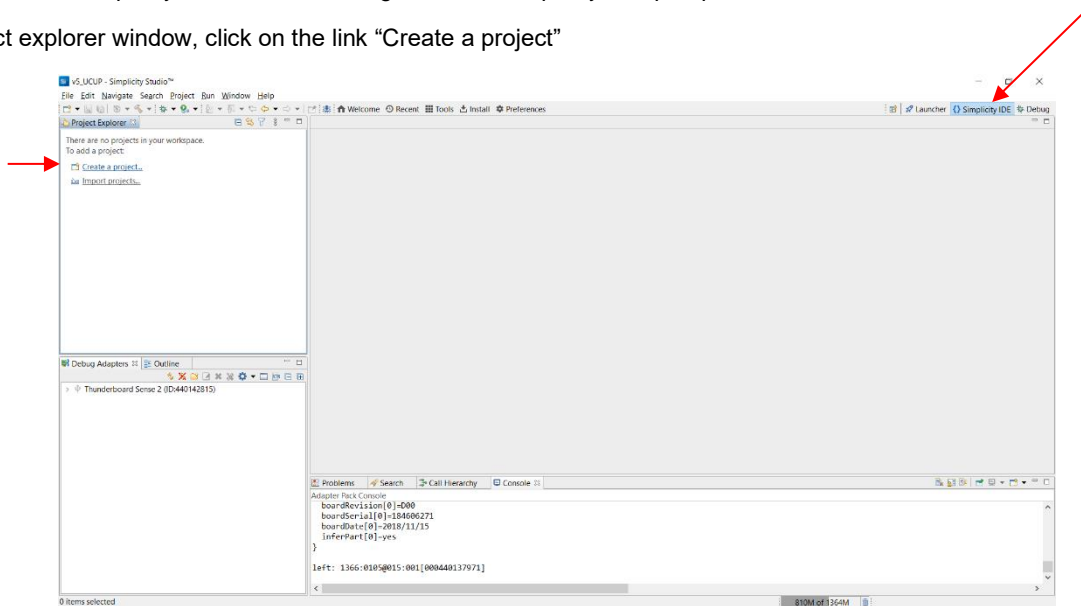
Creating a bare-metal project

2 Creating a bare-metal project

2.1 Create an empty project using Silicon Labs Project Wizard

Run the installation of Simplicity Studio v5 and navigate to the Simplicity IDE perspective.

From the project explorer window, click on the link “Create a project”

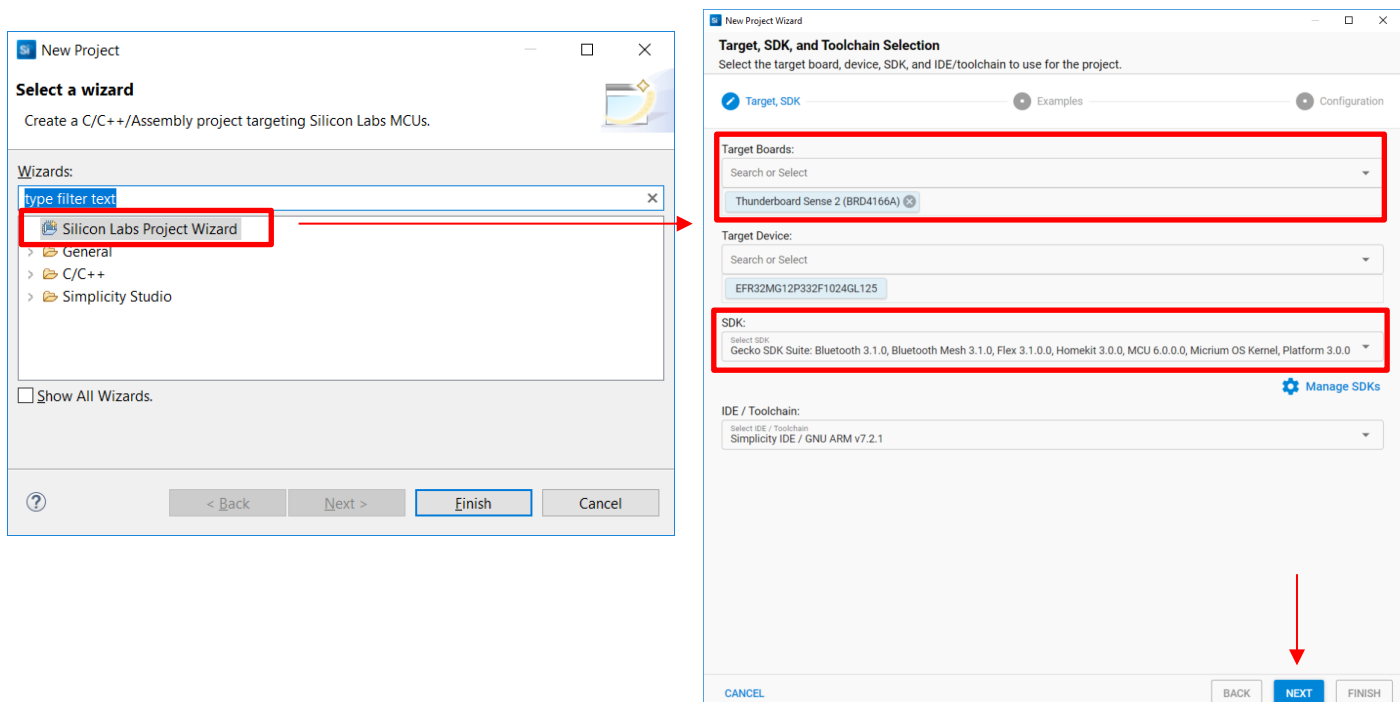


A “New Project” window will pop up. Select the Silicon Labs Project Wizard and click “Finish”.

A “New Project Wizard” window will now display. In the “Target Boards” text entry field, enter the board used for this demo – BRD4166A (If the Thunderboard Sense 2 is already connected to the PC and is the only development kit detected by Simplicity Studio, this field will auto-populate).

Verify that the latest Gecko SDK 3.1.0 or later is selected in the SDK drop-down menu.

Click “Next” to move on to the next configuration window.



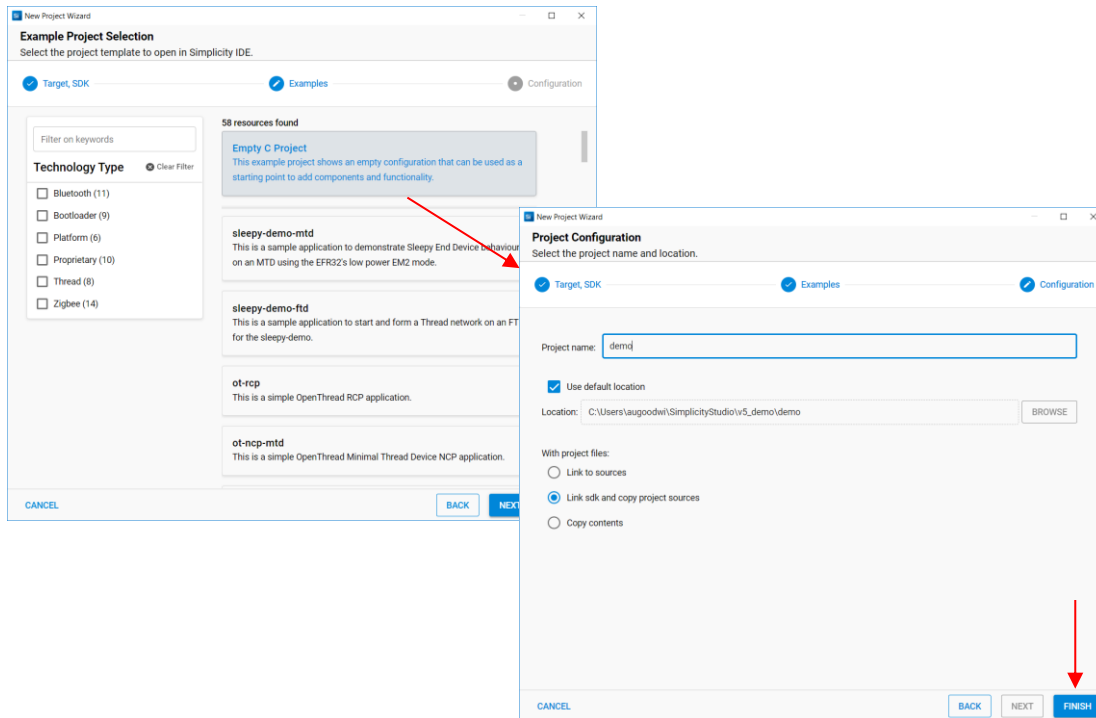
Creating a bare-metal project

Under “Example Project Selection”, click “Empty C Project”. This example project is typically located at the top of the list; filtering by the “Platform” Technology Type will narrow the selection options. Depending on the various packages installed with Simplicity Studio, there may be a different list of available examples than captured in the provided screenshot.

In the “Project Configuration” window:

- Enter a project name – for the purposes of this lab, “demo” is used as the project name
- Use the default project location, or specify a new location
- Make sure the “Link SDK and copy project sources” radio button is selected
- Click “Finish”

The Simplicity Studio Project Wizard will begin to auto-generate the necessary project structure and files



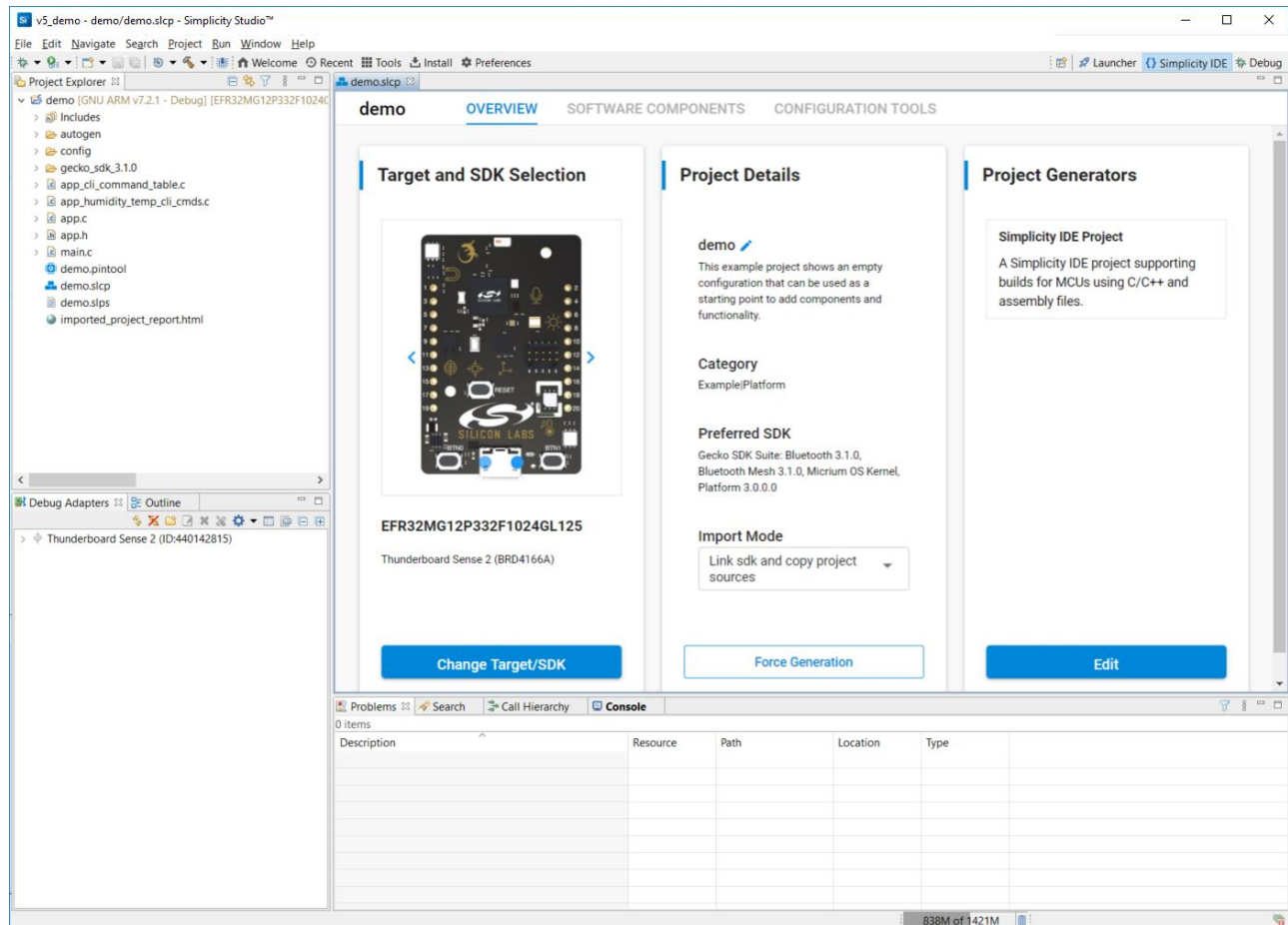
2.2 Explore the new project structure

After the Simplicity Studio Project Wizard completes, the new project directory structure and source will display in the “Project Explorer” window located on the left side in the “Simplicity IDE” perspective.

The *.slcp file will automatically open in the main window; three tabs are now available when viewing the project *.slcp file:

- Overview – Displays information about the target device, SDK selection, and some project details
- Software Components – Project Configurator for simple integration of software tools/drivers for easy intergration of peripherals
- Configuration Tools – Quick access to additional configuration tools, depending on installed components (Pin Tool, e.g.)

Creating a bare-metal project



Exploring main.c gives a glimpse of the new high-level project model utilized with Simplicity Studio v5's Gecko Platform and Project Configurator.

```

int main(void)
{
    // Initialize Silicon Labs device, system, service(s) and protocol stack(s).
    // Note that if the kernel is present, processing task(s) will be created by
    // this call.
    sl_system_init();

    // Initialize the application. For example, create periodic timer(s) or
    // task(s) if the kernel is present.
    app_init();

    #if defined(SL_CATALOG_KERNEL_PRESENT)
    // Start the kernel. Task(s) created in app_init() will start running.
    sl_system_kernel_start();
    #else // SL_CATALOG_KERNEL_PRESENT
    while (1) {
        // Do not remove this call: Silicon Labs components process action routine
        // must be called from the super loop.
        sl_system_process_action();

        // Application process.
        app_process_action();
    }
    #if defined(SL_CATALOG_POWER_MANAGER_PRESENT)
    // Let the CPU go to sleep if the system allows it.
    sl_power_manager_sleep();
    #endif
}
#endif // SL_CATALOG_KERNEL_PRESENT
}

```

Creating a bare-metal project

The main() function is simple: top-level initialization routines and a main while loop or “super loop” which utilizes process action routines for a bare-metal application. With the presence of a real-time operating system, the kernel is started and a task based action model is used.

In the high-level initialization there are two initialization function calls:

- sl_system_init() - within this function, Silicon Labs software components are initialized in a particular order (discussed in training)
 - Silicon Labs (sl_*) initialization functions are located in sl_event_handler.c, and include errata fixes, default pin states, initializing interrupt vector, board specific settings, DCDC/energy mode settings, and clock settings, etc.
 - sl_internal_app_init() will also hold initialization code for software components. Currently blank, as we have not yet installed any software components
- app_init() – located in app.c – this is where the user can add initialization routines specific to their particular project and application. This function will be modified periodically throughout the demonstration.

In the main while loop (super loop), two process action routines are called:

- sl_system_process_action() - Silicon Labs components process action routine
 - sl_internal_app_process_action() – located in sl_event_handler.c; again, currently blank but will hold software component specific process actions
- app_process_action() – located in app.c; application specific process action routines

Because this is an empty project, most initialization routines are currently empty. Adding certain components using Project Configurator will autogenerate new code in the appropriate inits and process action routines.

2.3 Adding Software Components

For this next step, the Command Line Interface (CLI) will be added, which in this case will also use the USART and Virtual COM port onboard the Thunderboard Sense 2. The Power Manager software component is also added to manage energy modes for the application.

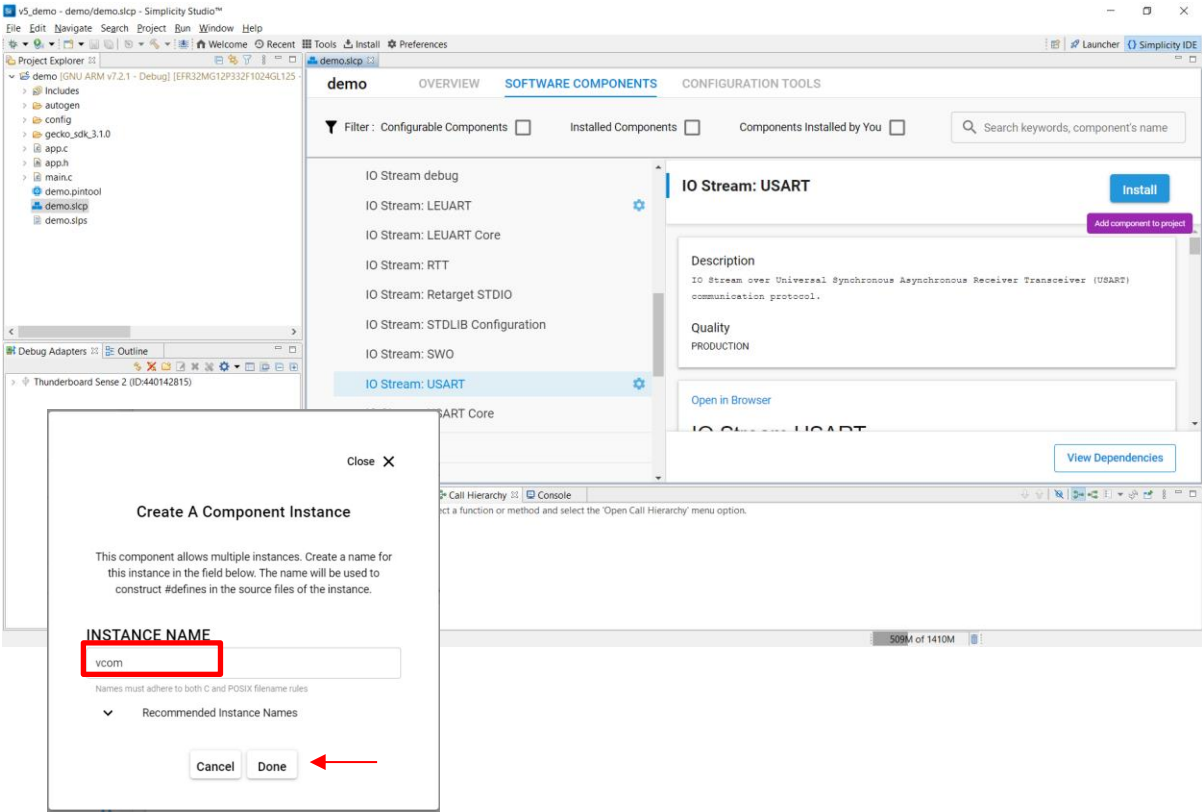
Go back to viewing “demo.slc” in the main window, or re-open this file from the “Project Explorer” window if it was closed.

In the main window, navigate to the “Software Components” tab.

From here, install the following Software Components:

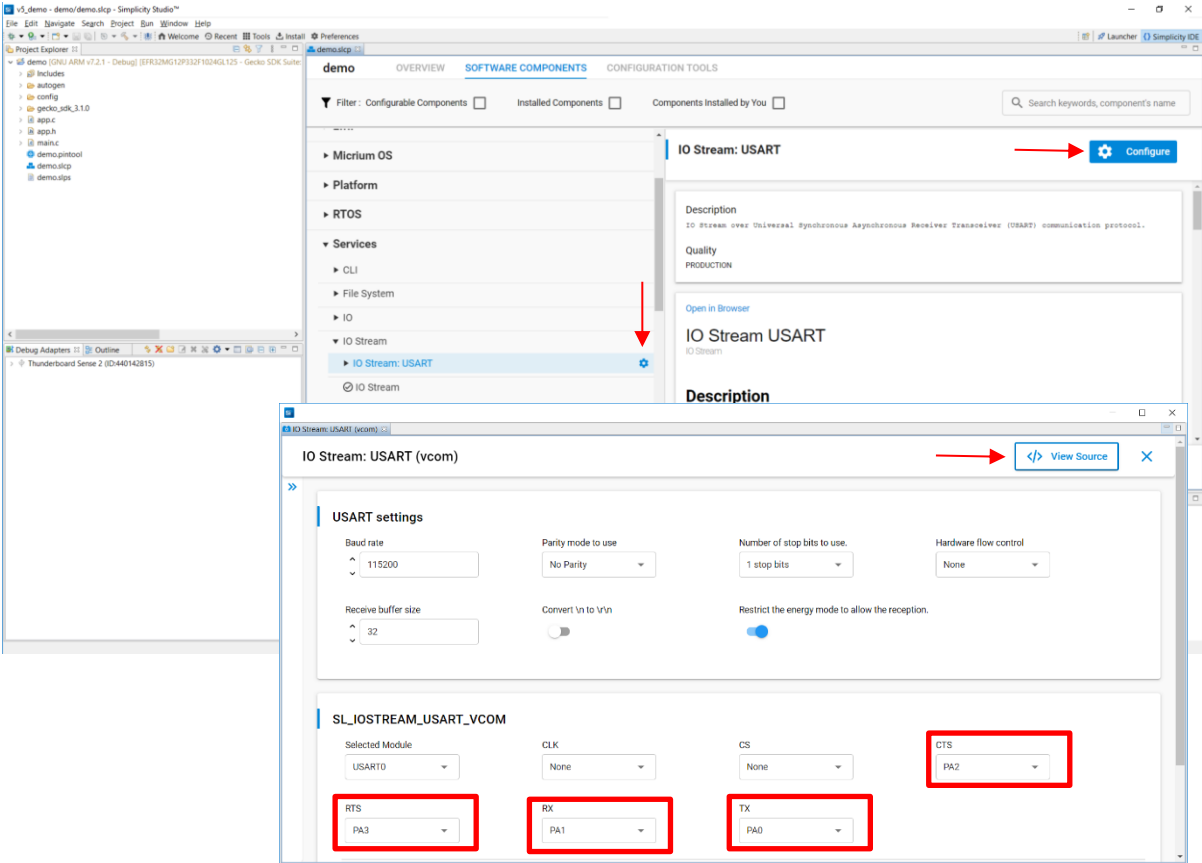
- Services > IO Stream > IO Stream: USART
 - Create Component Instance name: vcom
- Services > CLI > CLI: Command Line Interface
 - Create Component Instance name: vcom
- Services > Power Manager

Creating a bare-metal project



While still viewing “demo.slcip” in the main menu, navigate again to the “Software Components” tab and to the IO Stream: USART (Services > IO Stream > IO Stream: USART).

Click on the “Configure” button in the right section of the main window, or the gear icon in the Software Components tree view



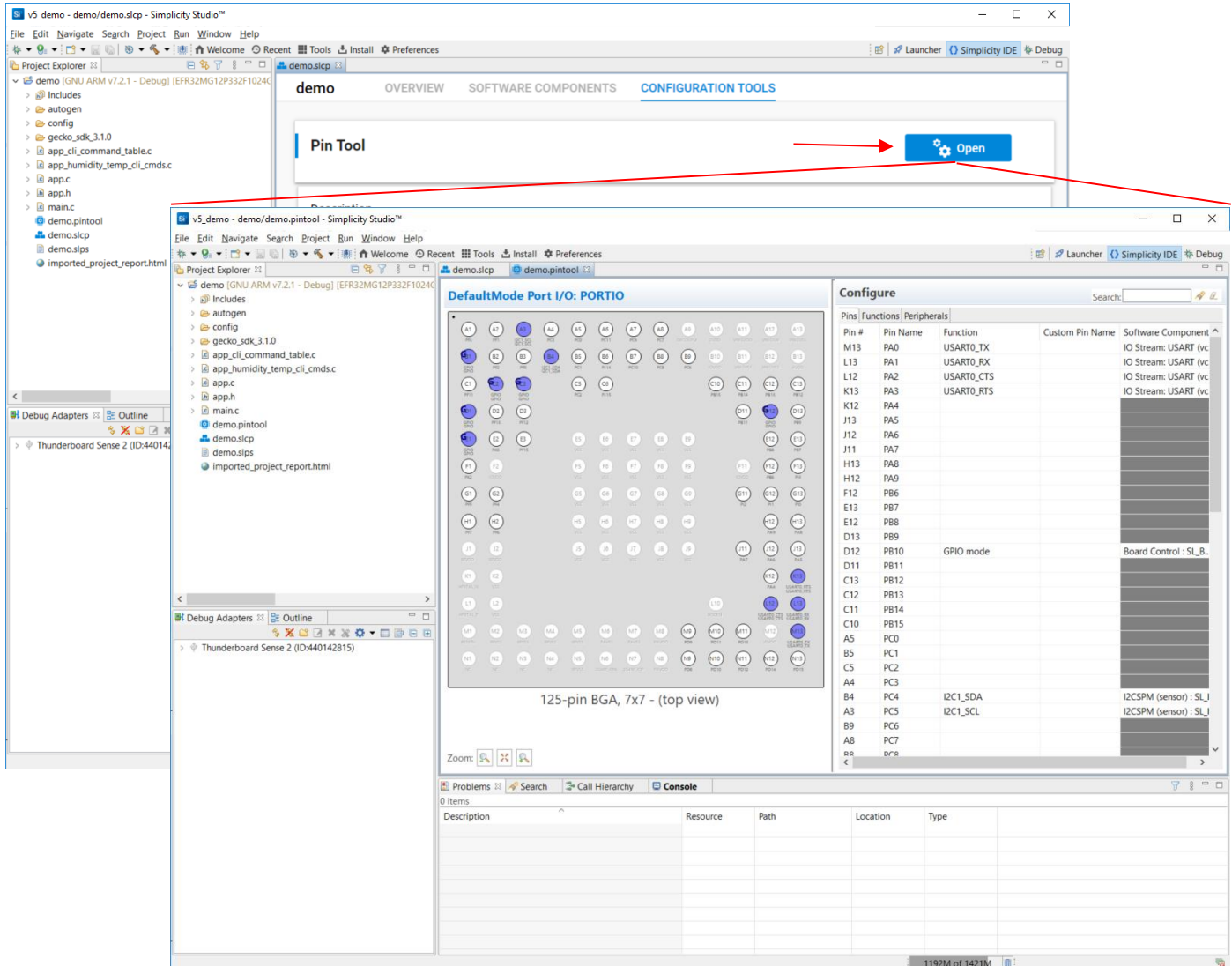
Creating a bare-metal project

Within the USART configuration window USART settings and pin configurations can be modified. The pin configurations default to the pins specified by the hardware selected when creating the new project (refer to the schematic for Thunderboard Sense 2).

For custom configurations, users will want to modify these settings based on their custom PCB to reflect specific pin routing.

From the USART configuration window, the generated source code can also be viewed by clicking the “View Source” button in the upper right corner.

Changes to pin allocation/routing can also be made in the Pin Tool, available under the “Configuration Tools” tab when viewing “demo.slcp”.



Changes made in the USART configuration GUI, source code, or Pin Tool propagate to the other tools when saved. Thus all three configuration tools are linked. Similar configuration tools/options are available for many of the other software components. The same GUI/source/pin tool relationship applies, and configuration changes can be made to any of the three tools.

No changes need to be made in this example, as the default settings apply.

Creating a bare-metal project

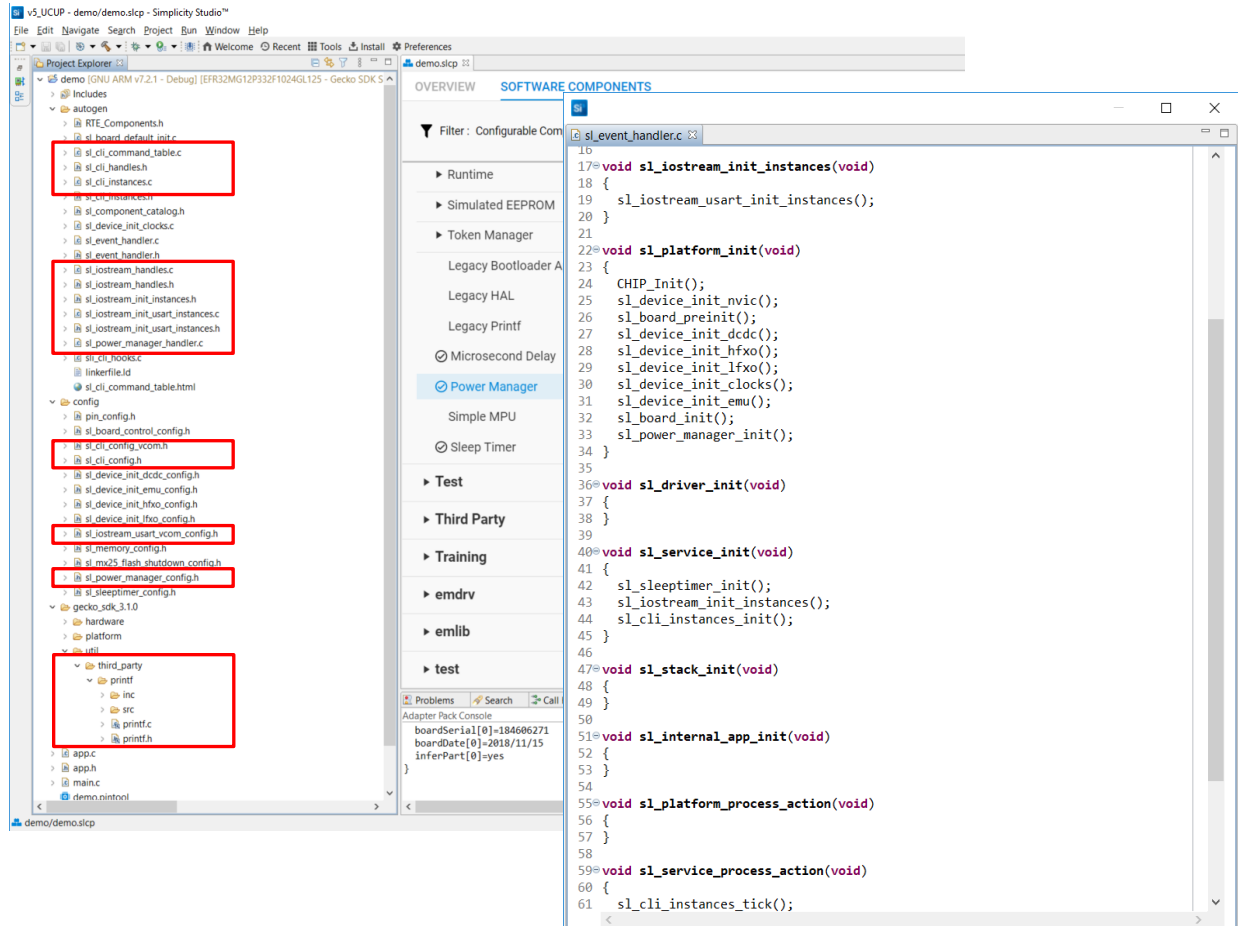
2.3.1 Explore project additions

In the “Project Explorer” window, expand the autogen and config folders. New source files have been added to the project which correspond to the additional software components added through Project Configurator.

Additional initializations are also added within `sl_event_handler.c`.

Third party `printf` functions have also been added (a dependency for the CLI interface).

There is also a new process in `sl_service_process_action` for the CLI interface.



Power Manager has now been added to the project, and the source file `sl_power_manager_handler.c` is now added to the project under the autogen directory.

`sl_power_manager_is_ok_to_sleep()` and `sl_power_manager_sleep_on_isr_exit()` callback functions are implemented which allow software to cancel going to sleep in case of a last-minute event and validate whether the MCU can return to sleep after processing an interrupt when the system was sleeping. The CLI interface and IO Stream software instances have hooks into these functions to properly manage the lowest possible energy mode.

Application code can also be used to overwrite weak functions `app_is_ok_to_sleep()` and `app_sleep_on_isr_exit()` so that application code can also perform these tasks.



Power manager manages energy modes for the application:

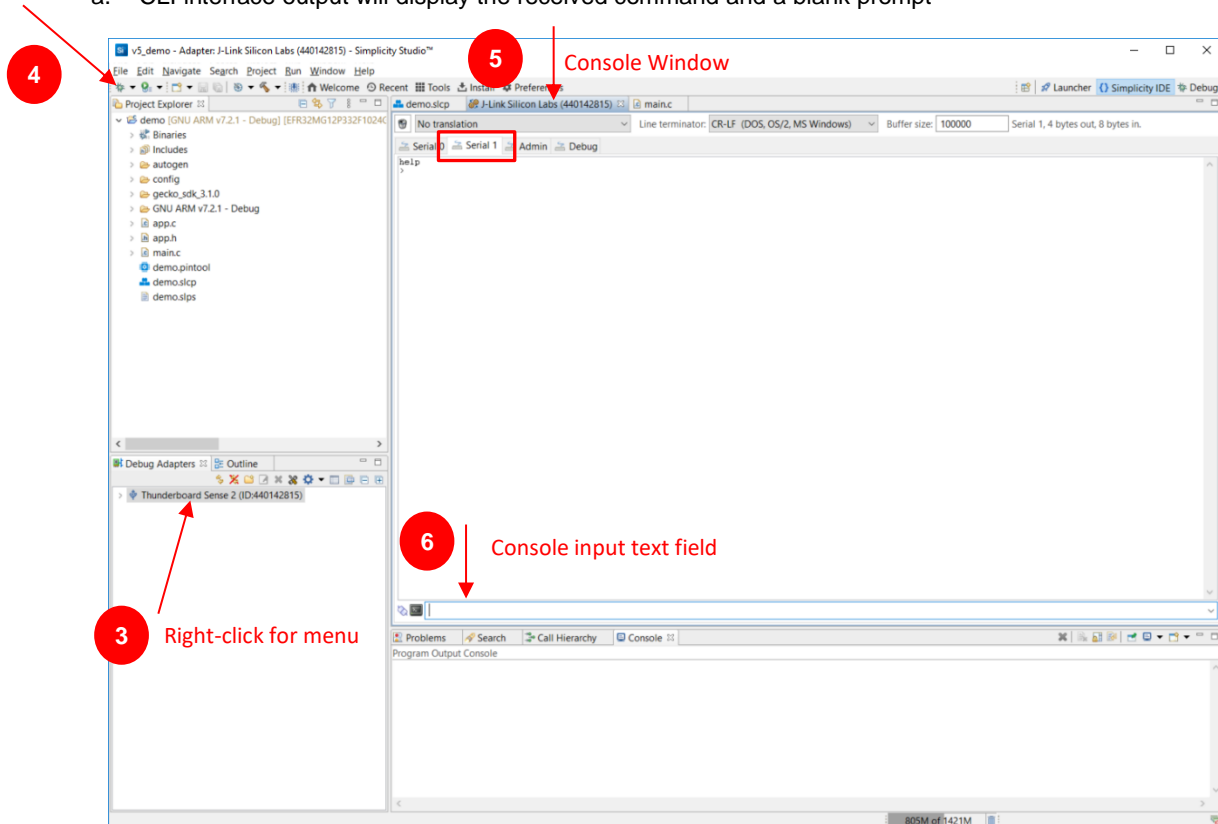
- Ensures that when the system enters sleep, it does so at the lowest energy mode possible. (For this example, because we are using the UART, the lowest energy mode is set to EM1)
- Lowest energy mode allowed is selected based on requirements set by different modules (drivers, stacks, application, etc.)
- Also applies some control on the HFXO (SYXO)
- No support for EM4.

Creating a bare-metal project

2.4 Empty project with CLI interface

At this point, the project can be built and run, but won't really do anything besides show the default CLI interface.

1. Connect the Thunderboard Sense 2 to your PC using a micro-USB cable (if you haven't done so already)
2. When the device is connected to your PC, you should see it listed in the **Debug Adapters** window in Simplicity Studio
3. In the **Debug Adapters** window, right-click on the device to display the options window and select "Launch Console..."
4. Build the project and flash the project to connected Thunderboard Sense 2 using the "Debug" button  in the top menu.
 - a. Simplicity Studio will automatically shift views to the Debug perspective and halt at the MCU at the start of main()
 - b. Press F8 or click on the "Resume" button  to resume program execution
5. In the main window, navigate to the console window and select the "Serial 1" tab; this corresponds to the VCOM port
6. In the console input text field type "help" and press Enter
 - a. CLI interface output will display the received command and a blank prompt



Build the project and flash to connected Thunderboard Sense 2 to display the bare CLI interface (help is the only command currently available).

2.5 Add Sensor Components

In this step the onboard relative humidity and temperature sensor is added to the project. The onboard sensor uses I2C for communication, therefore the addition of an I2C driver is also required. Disconnect from the Thunderboard Sense 2 and return to the Simplicity IDE perspective. In the main window, navigate back to the "demo.slc" or re-open the file by double-clicking in Project Explorer

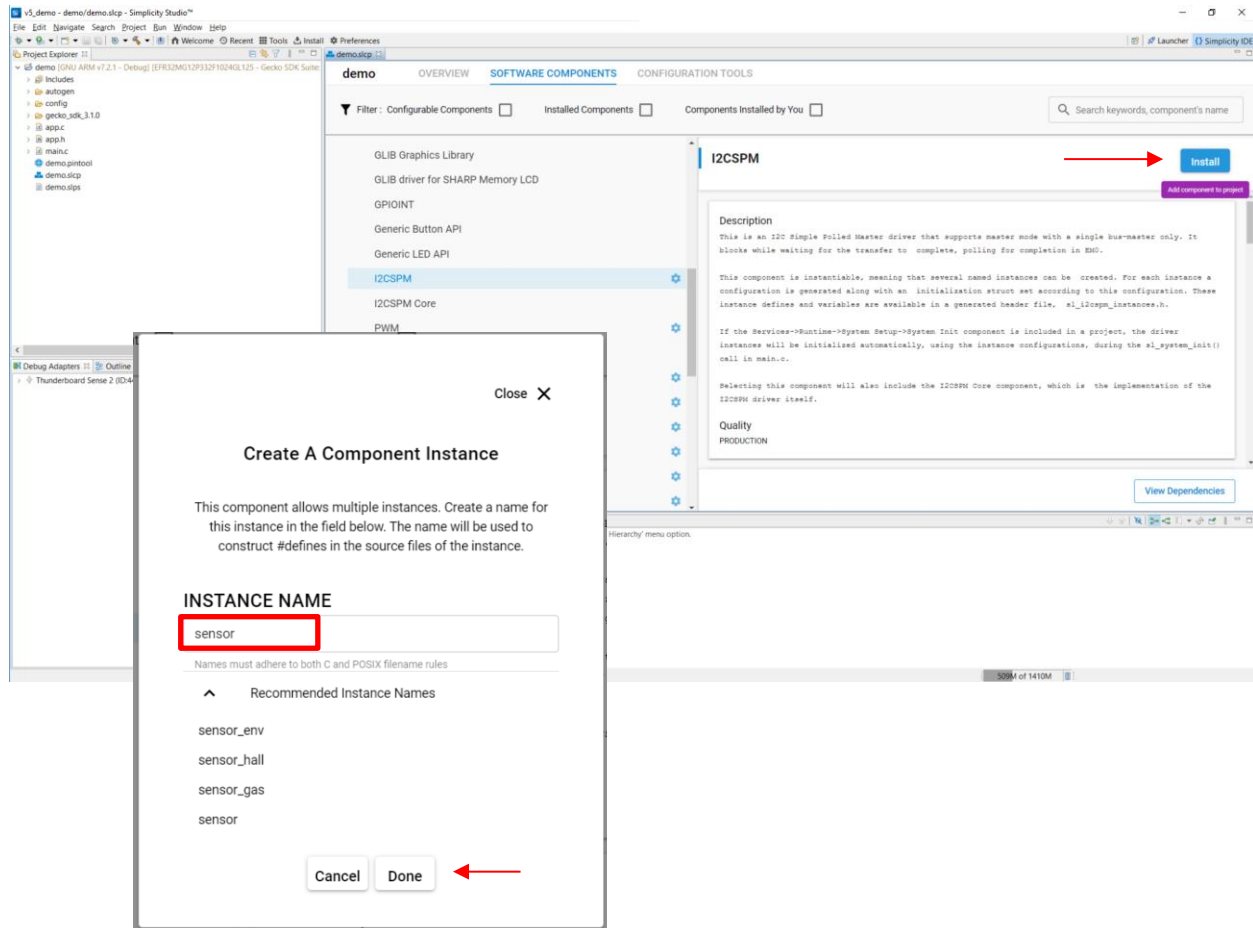
Again, navigate to the "Software Components" tab.

From here, install the following Software Components:

- Platform > Driver > I2CSPM
 - Create Component Instance name: sensor
- Bluetooth > Sensor > Relative Humidity and Temperature sensor

Creating a bare-metal project

NOTE: occasionally the **Assert** software component does not get pulled in, though it is a dependency of this component; Navigate to Bluetooth > Utility > Assert in the Software Component tree view and install if this is not installed with the last step.



2.6 Enable onboard sensor

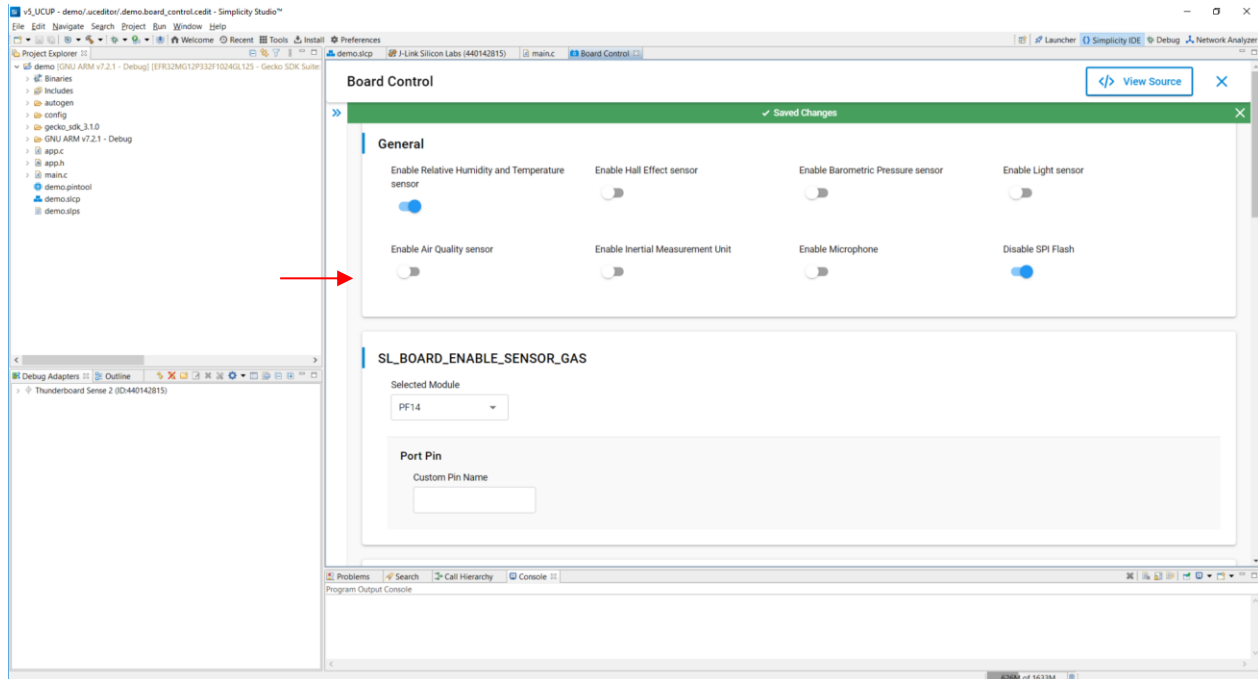
An additional step is required to toggle the enable pin for the onboard sensor.

Again, from the “Software Components” tab, navigate to Platform > Board > Board Control and click the gear to open the Board Control configuration window.

In the “General” section, click the slider below “Enable Relative Humidity and Temperature sensor”.

The slider will change color to blue and a notification will pop up (green) that the changes have been saved.

Creating a bare-metal project



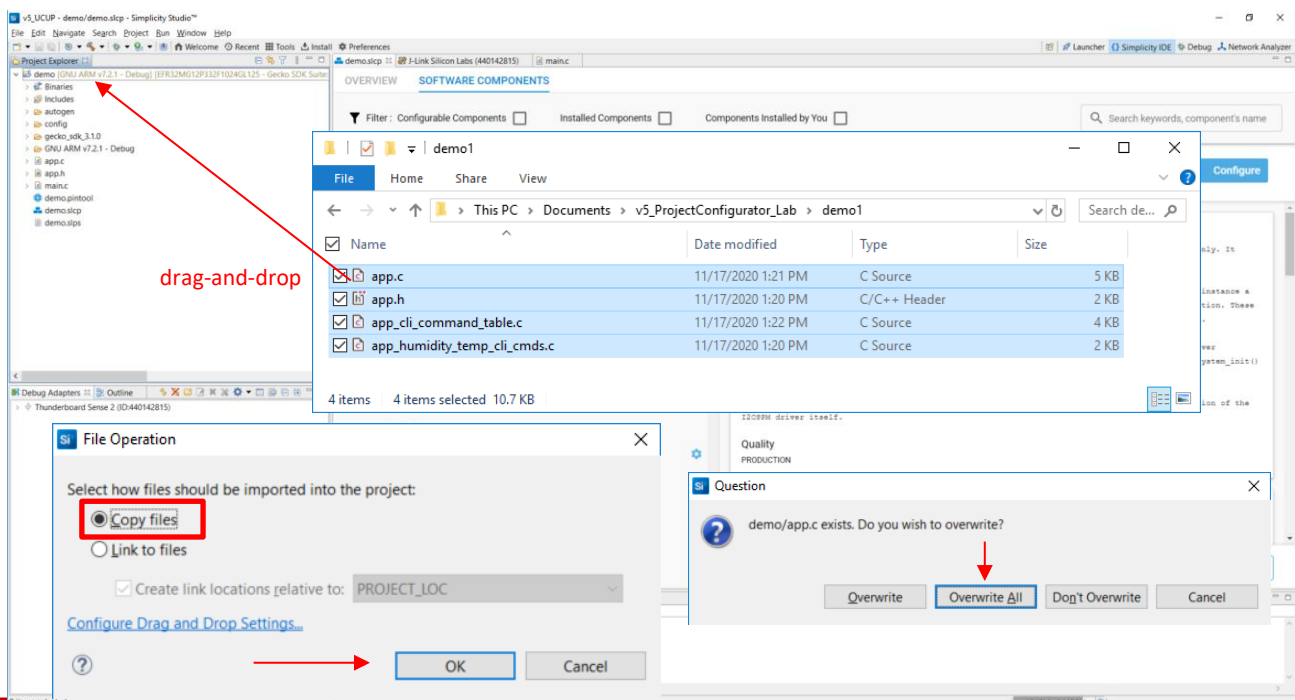
The default GPIO configuration associated with the Thunderboard Sense 2 hardware selection will route the appropriate pin for enabling the sensor.

2.7 Adding Application Specific Code

Download the accompanying lab source code provided in compressed zip – https://www.dropbox.com/s/4xbz4uzxq9l63nc/v5_ProjectConfigurator_Lab.zip?dl=0

Extract the files in the compressed zip to a known location and open the “demo1” folder. Take note of the extraction location. The remaining files will be used through the lab.

Drag-and-drop the files within this folder into the demo project's top directory in Simplicity Studio's “Project Explorer” window.



Creating a bare-metal project

In the “File Operation” window, select the “Copy Files” radio button and press OK.

When prompted about overwriting `app.c/app.h`, select “Overwrite All”.

The following files should be added to the project:



- Updated `app.c/app.h`
 - Initialization
 - Initializes RHT sensor
 - Configures sleeptimer to wake every 2 seconds; ISR sets measurement flag
 - Adds CLI commands to CLI instance
 - On wake, `main()`'s super loop calls `app_process_action()`:
 - Checks for measurement flag
 - Reads relative humidity and temperature
 - Stores values in global variable
- NEW `app_cli_command_table.c`
 - CLI command structs for sensor readout over VCOM
- NEW `app_humidity_temp_cli_cmds.c`
 - Function calls by the CLI interface to print command responses

The provided `app.c` file utilizes `#if !defined(SL_CATALOG_KERNEL_PRESENT)` in order to include different portions of the code when running with/without an RTOS kernel.

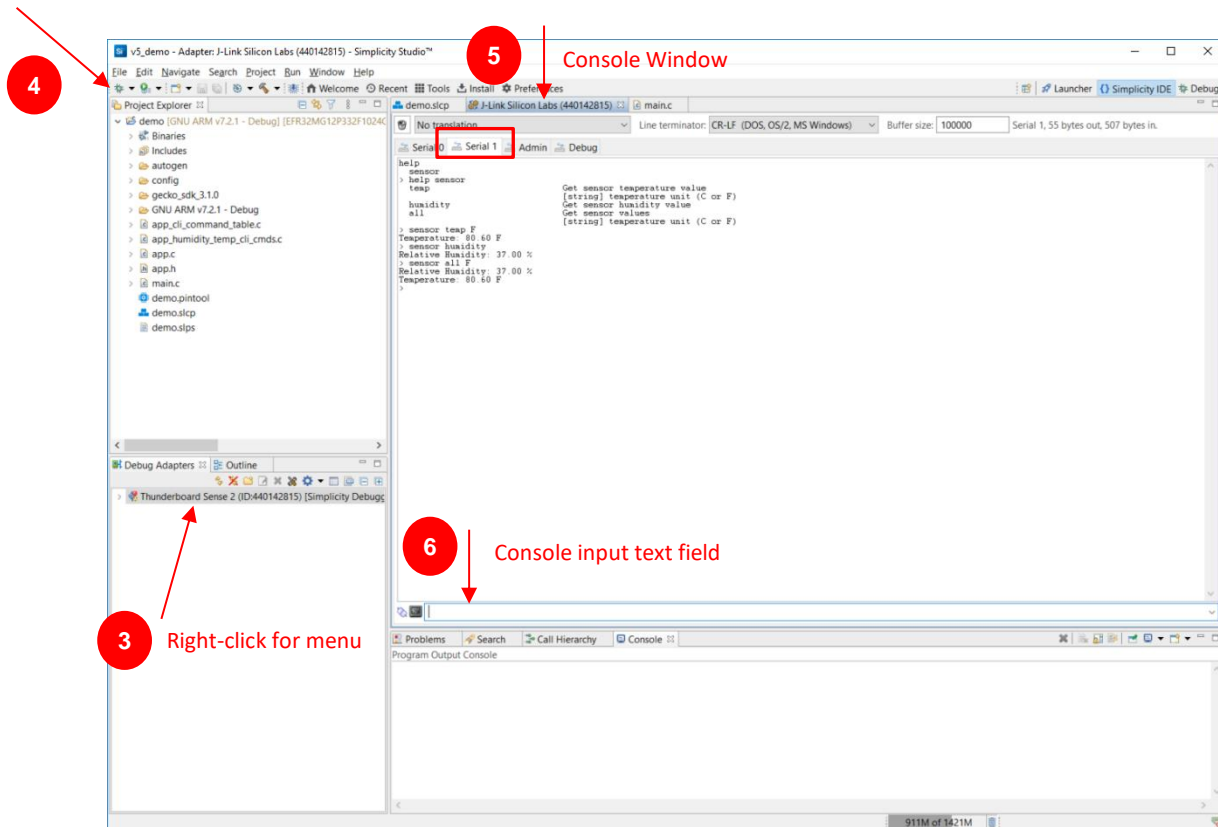
For this first portion, the code is utilizing the programming model with process actions. For the third part where software component FreeRTOS is added to the project, code will utilize the RTOS kernel and task actions.

2.8 Build and test

The project can now be built and run to demonstrate the relative humidity and temperature sensor readout. Additional commands are now available through the CLI interface.

1. Connect the Thunderboard Sense 2 to your PC using a micro-USB cable (if you haven't done so already)
2. When the device is connected to your PC, you should see it listed in the **Debug Adapters** window in Simplicity Studio
3. In the **Debug Adapters** window, right-click on the device to display the options window and select “Launch Console...”
4. Build the project and flash the project to connected Thunderboard Sense 2 using the “Debug” button  in the top menu.
 - a. Simplicity Studio will automatically shift views to the Debug perspective and halt at the MCU at the start of `main()`
 - b. Press F8 or click on the “Resume” button  to resume program execution
5. In the main window, navigate to the console window and select the “Serial 1” tab; this corresponds to the VCOM port
6. The following commands are now available in the console window:
 - a. **help** – general help; returns other command syntax
 - b. **help sensor** – help using the sensor command
 - c. **sensor temp [unit] (C or F)** – returns temperature reading
 - d. **sensor humidity** – returns humidity reading
 - e. **sensor all [unit] (C or F)** – returns temperature and humidity
7. In the console input text field type “help” and press Enter
 - a. CLI interface output will display the received command and “sensor”
 - b. Type “help sensor” to display CLI command information pertaining to the onboard sensor readout
 - c. Type “sensor all F” to display the temperature in degrees Fahrenheit and humidity as a percentage
 - d. Try any of the other commands

Creating a bare-metal project



Disconnect from the Thunderboard Sense 2 debug connection and return to the Simplicity IDE perspective before moving on to the next part of the lab.

Adding the ADC peripheral and application code

3 Adding the ADC peripheral and application code

For this portion of the lab we will be adding the ADC peripheral and using example code provided on our GitHub repository to populate the necessary functions within our application to take a software triggered measurement of the DCDC output voltage.

3.1 Add the ADC software component via Project Configurator

In Simplicity Studio, back in the Simplicity IDE perspective, navigate to the demo.slc in the main viewing window and open the “Software Components” tab.

From here, install the following Software Components:

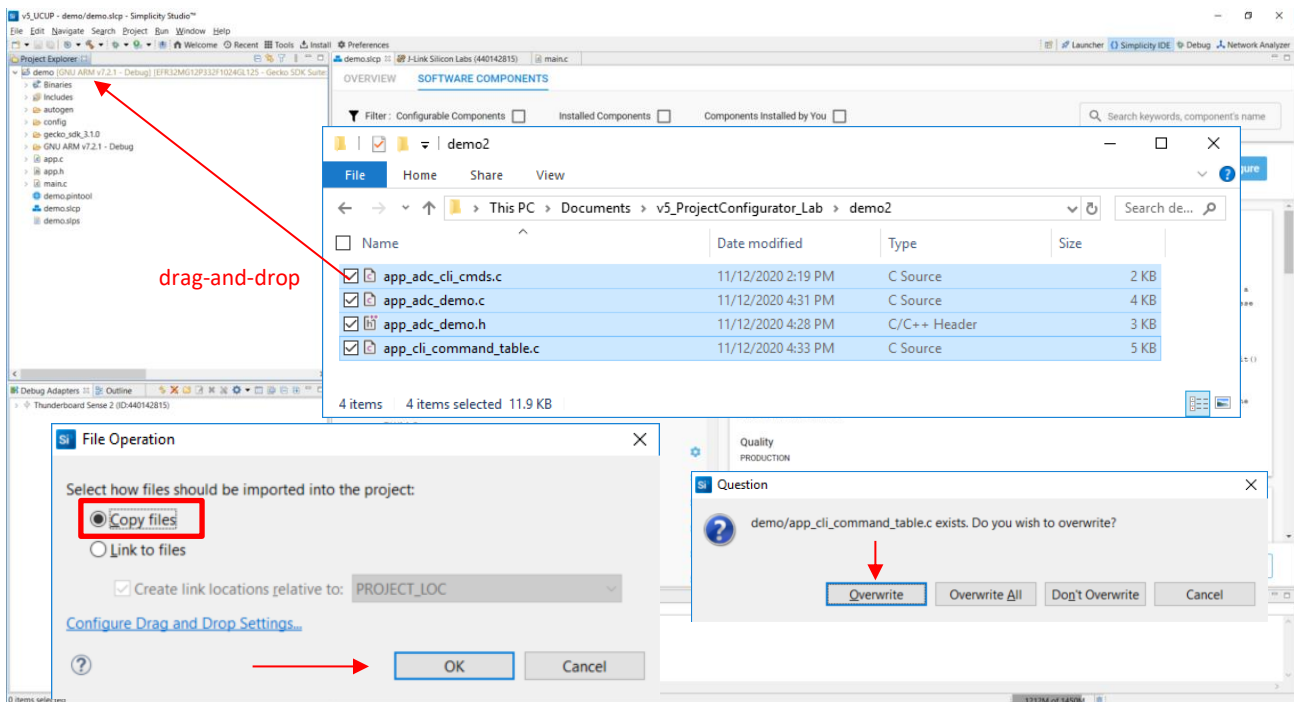
- Platform > Peripheral > ADC

This will add the necessary source files from the emlib library which is required for running code provided through our Peripheral Examples repository on GitHub.

3.2 Adding application specific code

From the previously extracted compressed zip in step 2.7, open the “demo2” folder.

Drag-and-drop the files within this folder into the demo project’s top directory in Simplicity Studio’s “Project Explorer” window.



In the “File Operation” window, select the “Copy Files” radio button and press OK.

When prompted about overwriting app_cli_command_table.c select “Overwrite”.

The following files should be added to the project:

- UPDATED app_cli_command_table.c
 - Adds CLI command structs for ADC readout over VCOM
- app_adc_demo.c/.h
 - Skeleton functions for ADC -> will be utilizing peripheral example code from GitHub to populate routines

Adding the ADC peripheral and application code

- NEW app_adc_cli_cmds.c
 - Function calls by the CLI interface to print command responses

3.3 Implement source for ADC functions

In the "Project Explorer" window, open app_adc_demo.c. There are three function definitions: sl_adc_init(), sl_adc_start_measurement(), and sl_adc_measurement_get().

The source from GitHub ADC Single Polled example is used to populate the three function definitions, with some slight modification. This source is located at https://github.com/SiliconLabs/peripheral_examples/blob/master/series1/adc/adc_single_polled/src/main_s1.c and is also provided in the previously downloaded zip file under the folder "GitHub_ADC_Single_Polled_Src".

Copy the initialization code from the GitHub source's initADC() function into sl_adc_init() in app_adc_demo.c. Modify the positive input selection (initSingle.posSel) from APORT 2 channel 9 to DVDD (adcPosSelDVDD) to measure the DCDC output, which is regulating at 1.8V.

Copy or add the ADC single conversion start function to sl_adc_start_measurement().

Copy or add the code polling the ADC status and then retrieving the single conversion to sl_adc_measurement_get().

```

40 #include "em_adc.h"
41
42 #include "app_adc_demo.h"
43
44 #define adcFreq 1600000
45
46 /**
47  * @brief Initialize ADC function
48  */
49 void sl_adc_init (void)
50 {
51     // Add ADC init code
52     // Enable ADC0 clock
53     CMU_ClockEnable(cmuClock_ADC0, true);
54
55     // Declare init structs
56     ADC_Init_TypeDef init = ADC_INIT_DEFAULT;
57     ADC_InitSingle_TypeDef initSingle = ADC_INITSINGLE_DEFAULT;
58
59     // Modify init structs and initialize
60     init.prescale = ADC_PrescaleCalc(adcFreq, 0); // Init to max ADC clock for Series 1
61
62     initSingle.diff      = false; // single ended
63     initSingle.reference = adcRef2V5; // internal 2.5V reference
64     initSingle.resolution = adcRes12Bit; // 12-bit resolution
65     initSingle.acqTime   = adcAcqTime4; // set acquisition time to meet minimum requirement
66
67     // Select ADC input. See README for corresponding EXP header pin.
68     initSingle.posSel = adcPosSelDVDD;
69     init.timebase = ADC_TimebaseCalc(0);
70
71     ADC_Init(ADC0, &init);
72     ADC_InitSingle(ADC0, &initSingle);
73 }
74
75 void sl_adc_start_measurement(void)
76 {
77     // Start ADC conversion
78     ADC_Start(ADC0, adcStartSingle);
79 }
80
81 void sl_adc_measurement_get(uint32_t *adcData)
82 {
83     // Wait for conversion to be complete
84     while(!(ADC0->STATUS & _ADC_STATUS_SINGLEDV_MASK));
85
86     // Get ADC result
87     *adcData = ADC_DataSingleGet(ADC0);
88 }
89
  
```

Init Code from GitHub source

Start Single conversion

Wait for conversion to complete; store value

Modify to adcPosSelDVDD to monitor DCDC output

Adding the ADC peripheral and application code

The ADC measurement is split in this example into two functions, one to kick off a conversion, and another to retrieve the conversion data, however the two could be combined. The benefit of splitting this up is while the ADC conversion is taking place, other code can be executing (relative humidity and temperature sensor measurement over I2c, e.g.).

3.4 Modify app.c init and process action calls

The ADC measurement and implementation need to be added to the application. In the “Project Explorer” window, open app.c.

At the top of the file, include the new app_adc_demo.h header file – suggested insertion on line 33 -> #include "app_adc_demo.h"

Add global uint32 variable “adcData” to store ADC result – suggested insertion on line 62 -> uint32_t adcData = 0;

In app_init(), add the ADC initialization function sl_adc_init() – suggested insertion on line 83.

Add ADC measurement calls to the process action

- sl_adc_start_measurement() prior to RHT sensor readout to start ADC conversion - insertion on line 122
- sl_adc_measurement_get(&adcData) after RHT sensor readout to retrieve ADC conversion - insertion on line 124

```

76# *****
77 * Initialize application.
78 *****
79# void app_init(void)
80 {
81     sl_sensor_rht_init();
82
83     sl_adc_init();
84 #if defined(SL_CATALOG_KERNEL_PRESENT)
85     /* Set up periodic measurement timer. */
86     sl_sleepimer_start_periodic_timer_ms(&measurement_timer, MEASUREMENT_INTERVAL_MS, measurement_callback, NULL, 0, 0);
87 #else
88     osThreadAttr_t attr;
89
90     attr.name = "sl7013 app task";
91     attr.priority = 25;
92     attr.stack_mem = task_stack;
93     attr.stack_size = STACK_SIZE;
94     attr.cb_mem = thread_cb;
95     attr.cb_size = osThreadCbSize;
96     attr.attr_bits = 0u;
97     attr.tz_module = 0u;
98
99     thread_id = osThreadNew(&task, NULL, &attr);
100     EFM_ASSERT(thread_id != NULL);
101 #endif
102
103 // Remove unused autogen CLI command group
104 sl_cli_command_remove_command_group(sl_cli_vcom_handle, &sl_cli_vcom_command_group);
105
106 // Add application CLI commands
107 sl_cli_command_add_command_group(sl_cli_vcom_handle, &sl_cli_app_command_group);
108 }
109
110 #if defined(SL_CATALOG_KERNEL_PRESENT)
111# *****
112 * App ticking function.
113 *****
114# void app_process_action(void)
115 {
116     CORE_DECLARE_IRQ_STATE;
117
118     CORE_ENTER_CRITICAL();
119     if (measurement_flag) {
120         measurement_flag = false;
121         CORE_EXIT_CRITICAL();
122         sl_adc_start_measurement();
123         sl_sensor_rht_get(&rhtData, &tempData);
124         sl_adc_measurement_get(&adcData);
125     } else {
126         CORE_EXIT_CRITICAL();



```

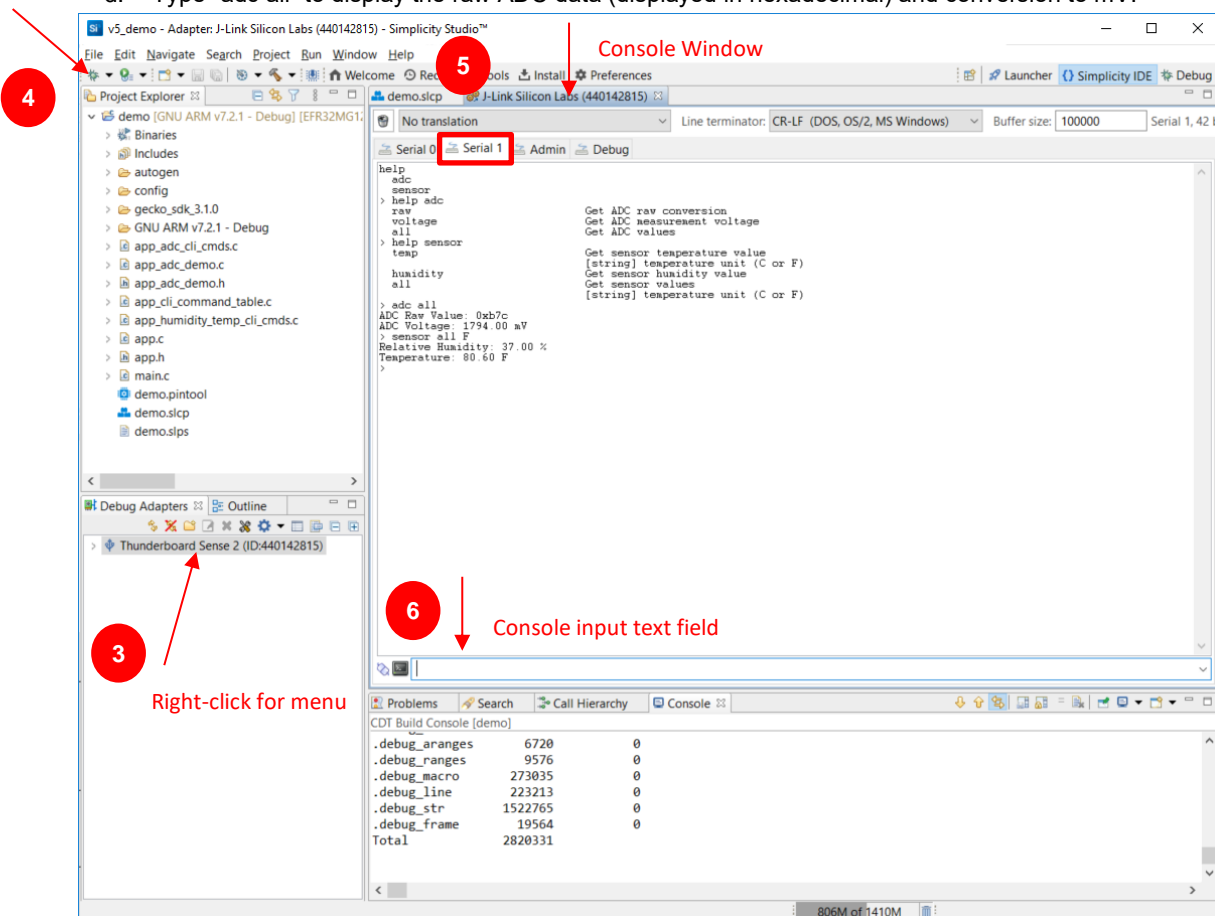
While adding the ADC measurement calls, this is an opportunity to also add the calls to the kernel task, which currently is not compiled with the code, but will be necessary in the next demo. Use the same steps to straddle the RHT sensor readout function in the kernel task with the two ADC measurement calls (insertions on lines 144 and 146).

Adding the ADC peripheral and application code

3.5 Build and test

The project can now be built and run to demonstrate the relative humidity and temperature sensor readout along with DCDC output voltage readout via the ADC peripheral. Additional commands are now available through the CLI interface.

1. Connect the Thunderboard Sense 2 to your PC using a micro-USB cable (the Thunderboard Sense 2 may require the USB cable to be unplugged and reconnected...sometimes when the USB bus puts the device to sleep it does not wake the VCOM port)
2. When the device is connected to your PC, you should see it listed in the **Debug Adapters** window in Simplicity Studio
3. In the **Debug Adapters** window, right-click on the device to display the options window and select “Launch Console...”
4. Build the project and flash the project to connected Thunderboard Sense 2 using the “Debug” button  in the top menu.
 - a. Simplicity Studio will automatically shift views to the Debug perspective and halt at the MCU at the start of main()
 - b. Press F8 or click on the “Resume” button  to resume program execution
5. In the main window, navigate to the console window and select the “Serial 1” tab; this corresponds to the VCOM port
6. The following commands are now available in the console window:
 - a. **help** – general help; returns other command syntax
 - b. **help sensor / help adc** – help using the group command
 - c. **sensor temp [unit] (C or F)** – returns temperature reading
 - d. **sensor humidity** – returns humidity reading
 - e. **sensor all [unit] (C or F)** – returns temperature and humidity
 - f. **adc raw** – returns temperature and humidity
 - g. **adc voltage** – returns temperature and humidity
 - h. **adc all** – returns temperature and humidity
7. In the console input text field type “help” and press Enter
 - a. CLI interface output will display the received command along with “adc” and “sensor”
 - b. Type “help sensor” or “help adc” to display CLI command information for each command group
 - c. Type “sensor all F” to display the temperature in degrees Farenheit and humidity as a percentage
 - d. Type “adc all” to display the raw ADC data (displayed in hexadecimal) and conversion to mV.



Disconnect from the Thunderboard Sense 2 debug connection and return to the Simplicity IDE perspective before moving on to the next part of the lab.

Adding RTOS

4 Adding RTOS

With Project Configurator and the application code already provided, adding a real-time operating system is straight forward.

4.1 Add the RTOS Kernel software component via Project Configurator

From the Simplicity IDE perspective and viewing `demo.slc` in the main window, navigate to the “Software Components” tab.

From here, install the following Software Components:

- RTOS > FreeRTOS > FreeRTOS

In `sl_event_handler.c`, the process action has been replaced with function `sl_kernel_start()`.

In the project folder, `CMSIS>RTOS2` is included automatically

The “super loop” in `main()` has been replaced by the RTOS kernel start, and `app_init()` now creates a kernel task to handle timed measurements every 2 seconds. Sleep timer is no longer utilized.

```

demo.slc | main.c |
18 #include "sl_system_init.h"
19 #include "app.h"
20 #if defined(SL_CATALOG_POWER_MANAGER_PRESENT)
21 #include "sl_power_manager.h"
22 #endif
23 #if defined(SL_CATALOG_KERNEL_PRESENT)
24 #include "sl_system_kernel.h"
25 #else // SL_CATALOG_KERNEL_PRESENT
26 #include "sl_system_process_action.h"
27 #endif // SL_CATALOG_KERNEL_PRESENT
28
29 int main(void)
30 {
31 // Initialize Silicon Labs device, system, service(s) and protocol stack(s).
32 // Note that if the kernel is present, processing task(s) will be created by
33 // this call.
34 sl_system_init();
35
36 // Initialize the application. For example, create periodic timer(s) or
37 // task(s) if the kernel is present.
38 app_init();
39
40 #if defined(SL_CATALOG_KERNEL_PRESENT)
41 // Start the kernel. Task(s) created in app_init() will start running.
42 sl_system_kernel_start();
43 #else // SL_CATALOG_KERNEL_PRESENT
44 while (1) {
45 // Do not remove this call: Silicon Labs components process action routine
46 // must be called from the super loop.
47 sl_system_process_action();
48
49 // Application process.
50 app_process_action();
51
52 #if defined(SL_CATALOG_POWER_MANAGER_PRESENT)
53 // Let the CPU go to sleep if the system allows it.
54 sl_power_manager_sleep();
55 #endif
56 }
57 #endif // SL_CATALOG_KERNEL_PRESENT
58 }

```

```

app.c |
111 #if defined(SL_CATALOG_KERNEL_PRESENT)
112 //*****
113 * App ticking function.
114 *****/
115 void app_process_action(void)
116 {
117 CORE_DECLARE_IRQ_STATE;
118
119 CORE_ENTER_CRITICAL();
120 if (measurement_flag) {
121 measurement_flag = false;
122 CORE_EXIT_CRITICAL();
123 sl_adc_start_measurement();
124 sl_sensor_rht_get(&rhData, &tempData);
125 sl_adc_measurement_get(&adcData);
126 } else {
127 CORE_EXIT_CRITICAL();
128 }
129 }
130
131 static void measurement_callback(sl_sleeptimer_timer_handle_t *handle, void *data)
132 {
133 (void)handle;
134 (void)data;
135 sl_atomic_store(&measurement_flag, true);
136 }
137
138 #else // SL_CATALOG_KERNEL_PRESENT
139
140 static void task(void *arg)
141 {
142 (void)arg;
143
144 while (1) {
145 sl_adc_start_measurement();
146 sl_sensor_rht_get(&rhData, &tempData);
147 sl_adc_measurement_get(&adcData);
148 osDelay(((uint64_t)osKernelGetTickFreq() * MEASUREMENT_INTERVAL_MS) / 1000);
149 }
150 }
151 #endif

```

CDT Build Console [demo]

.debug_aranges	6720	0
.debug_ranges	9576	0
.debug_macro	270035	0
.debug_line	223213	0
.debug_str	1522765	0
.debug_frame	19564	0
Total	2820331	

If the ADC functions were not added to the kernel task as suggested in step 3.4, the ADC functions `sl_adc_start_measurement()` and `sl_adc_measurement_get(&adcData)` must be added surrounding `sl_sensor_rht_get()` in `app.c`

NOTE: For this demo, we use FreeRTOS, but either the FreeRTOS or Micrium Kernel will work with the provided application code. To switch to the Micrium OS, first uninstall FreeRTOS in the Software Components, then install RTOS > Micrium OS > Kernel > Micrium OS Kernel.

4.2 Build and test

The project can now be built using an RTOS kernel and tasked based programming model. The CLI interface will have the same command list and appear no different functionally from the CLI console. Build and run using the same steps outlined in 3.5.

Again, be sure to disconnect from the Thunderboard Sense 2 debug connection and return to the Simplicity IDE perspective before progressing to the final part of the lab.

Adding Bluetooth to the project

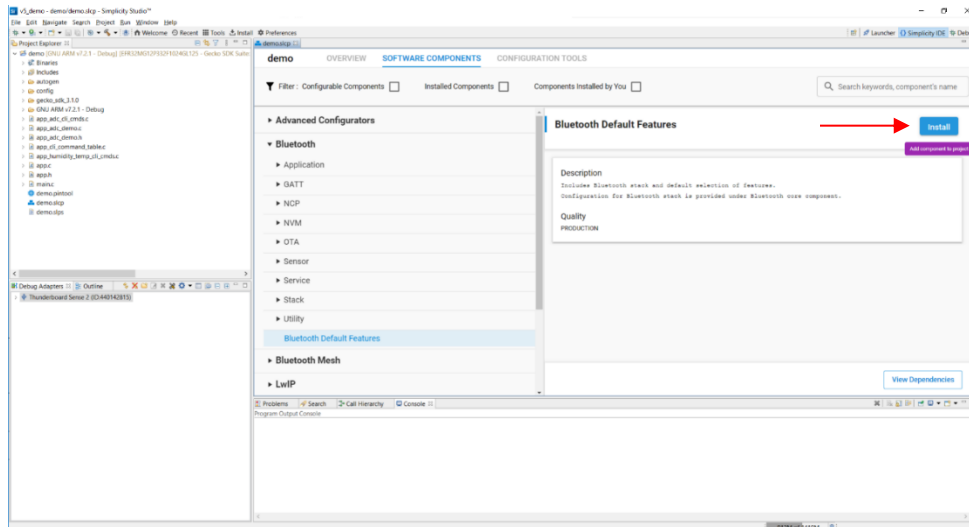
5 Adding Bluetooth to the project

5.1 Add the Bluetooth software component via Project Configurator

In Simplicity Studio, back in the Simplicity IDE perspective, navigate to the demo.slcip in the main viewing window and open the “Software Components” tab.

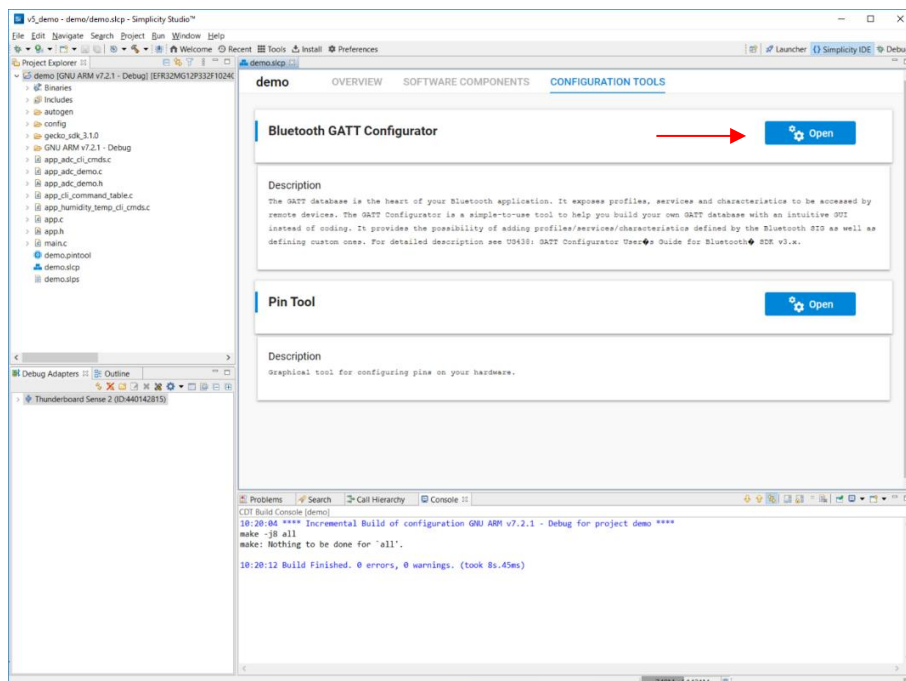
From here, install the following Software Components:

- Bluetooth > Bluetooth Default Features
- RTOS > FreeRTOS > FreeRTOS Heap 4
- Bluetooth > Service > Simple timer service for FreeRTOS



5.2 Update Device Name via GATT configurator

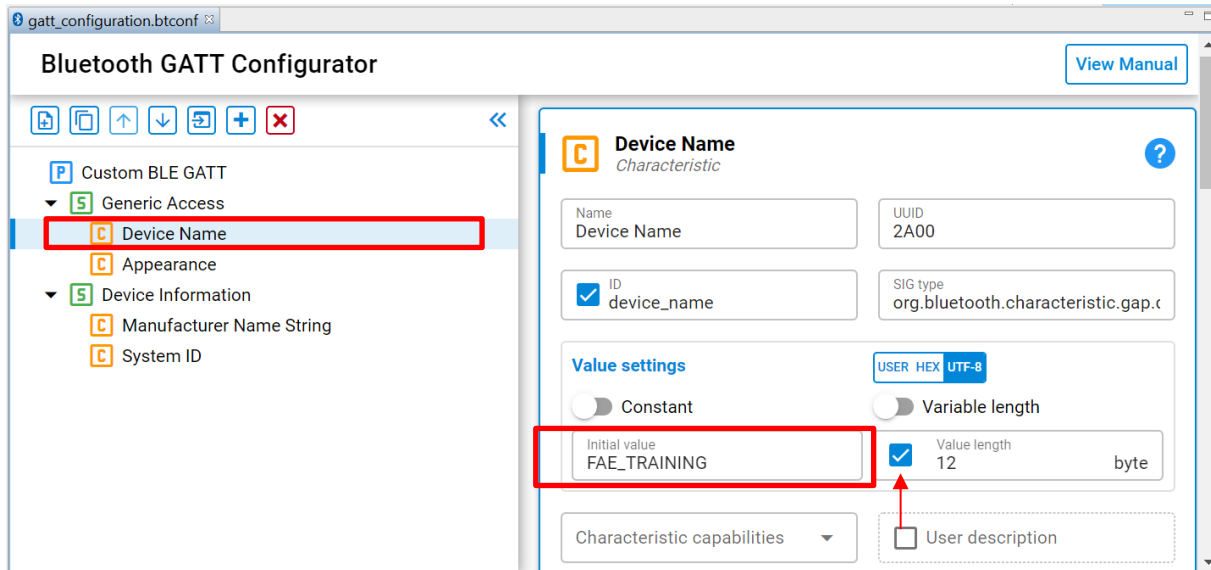
While still viewing demo.slcip in the main window, navigate to the “Configuration Tools” tab and open the Bluetooth GATT Configurator tool.



Adding Bluetooth to the project

Within the Bluetooth GATT Configurator tool window, select from the GATT tree Custom BLE GATT > Generic Access > Device Name.

In the configuration window to the right, modify the **Device Name** characteristic from “Silabs Example” to “FAE_TRAINING” and uncheck the overwrite auto calculated length checkbox. Also consider adding a unique numeral or any unique name in order to identify with mobile device.

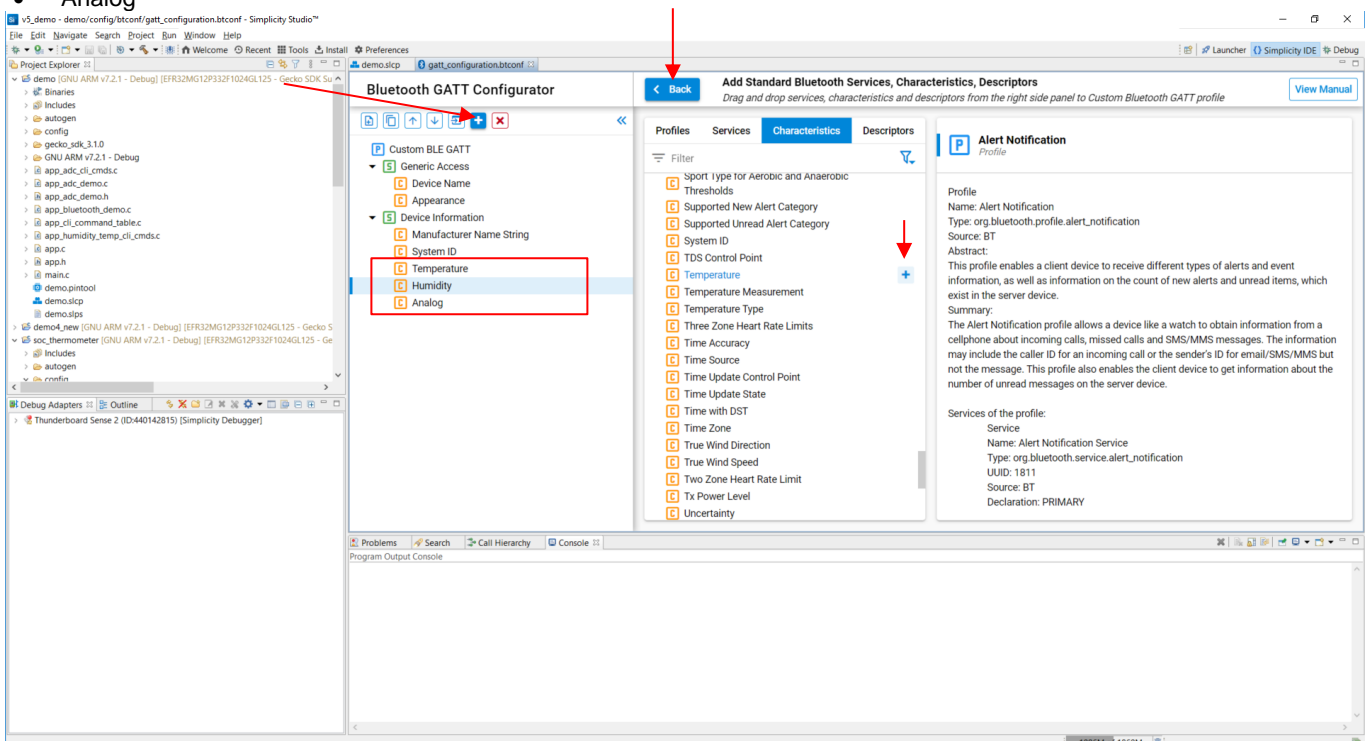


5.3 Add new GATT characteristics to Device Info service

Select the “Device Information” Service in the GATT tree view and then click on the “+” icon to add some standard GATT items to display the sensor information

Select “Characteristics” tab in the main GATT configuration window; add the following characteristics:

- Temperature
- Humidity
- Analog



Adding Bluetooth to the project

Click “Back” to view the new characteristics in the main view

For the Temperature Characteristic:

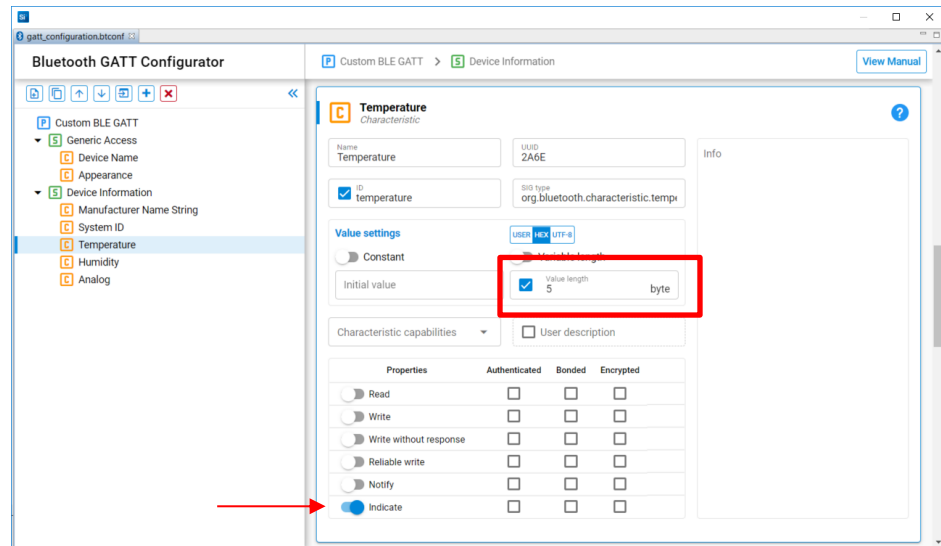
- change the value length value to 5 bytes
- turn on the “indicate” property

For the Humidity Characteristic:

- change the value length value to 4 bytes
- turn on the “indicate” property

For the Analog Characteristic:

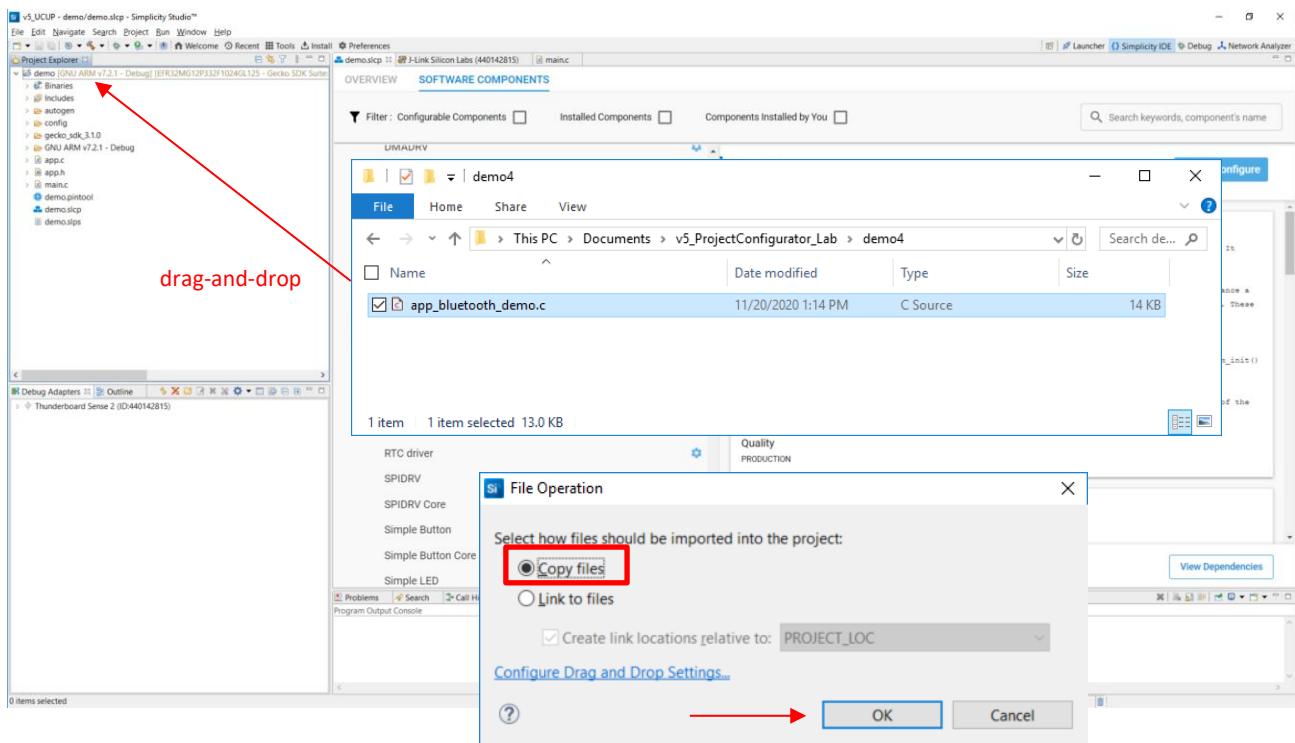
- change the value length value to 7 bytes
- turn on the “indicate” property



5.4 Add application specific code

From the previously extracted compressed zip in step 2.7, open the “demo4” folder.

Drag-and-drop the file `app_bluetooth_demo.c` within this folder into the demo project's top directory in Simplicity Studio's “Project Explorer” window.



In the “File Operation” window, select the “Copy Files” radio button and press OK.

Adding Bluetooth to the project

5.5 Build and flash to device; Use BLE Scanner to locate Thunderboard Sense 2 and readout sensor data

1. Build and flash the latest project configuration using the same methods as used previously in steps 2.8
2. Open BLE Scanner on your mobile device.
3. Locate your demo device by device name (FAE_TRAINING) programmed in step 5.2 and click connect
4. Expand the "Device Information" service and navigate to the "Temperature" characteristic
5. Click the "Indicator" button next to the characteristic to enable indications and observe the sensor value
6. Follow step 5 above for "Humidity" and "Analog" to view these sensor value

