

## Chapter 5: Debugging

This chapter introduces you to using the SWD interface for debugging the AIROC™ MCUs.

### Table of contents

<b>5.1</b>	<b>Introduction .....</b>	<b>2</b>
<b>5.2</b>	<b>Documentation .....</b>	<b>2</b>
<b>5.3</b>	<b>Hardware Setup .....</b>	<b>3</b>
<b>5.4</b>	<b>Firmware Setup .....</b>	<b>4</b>
<b>5.5</b>	<b>Programming.....</b>	<b>4</b>
<b>5.6</b>	<b>Launching the Debugger .....</b>	<b>5</b>
<b>5.7</b>	<b>Using the Debugger .....</b>	<b>6</b>
<b>5.8</b>	<b>Troubleshooting .....</b>	<b>8</b>
<b>5.9</b>	<b>Exercises .....</b>	<b>9</b>
Exercise 1: Setup and Run the Debugger .....		9

### Document conventions

Convention	Usage	Example
Courier New	Displays code and text commands	CY_ISR_PROTO(MyISR) ; make build
<i>Italics</i>	Displays file names and paths	<i>sourcefile.hex</i>
[bracketed, bold]	Displays keyboard commands in procedures	[Enter] or [Ctrl] [C]
Menu > Selection	Represents menu paths	File > New Project > Clone
<b>Bold</b>	Displays GUI commands, menu paths and selections, and icon names in procedures	Click the <b>Debugger</b> icon, and then click <b>Next</b> .

## 5.1 Introduction

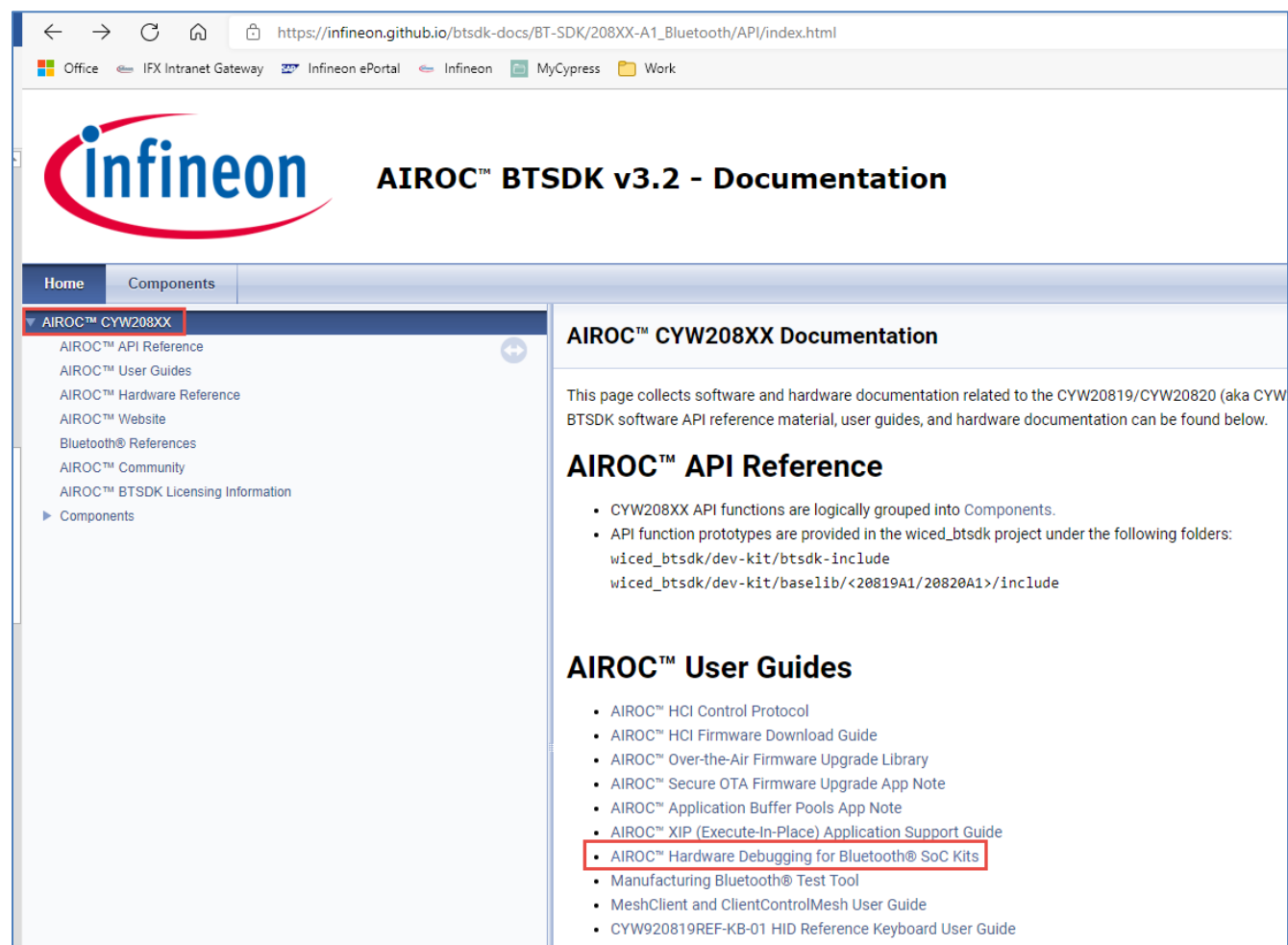
Debugging is a critical aspect of developing complex MCU applications. For AIROC™ Bluetooth® SDK applications, there are several debugging techniques available. These include:

- Debug print messages
- SWD debugging
- HCI commands and monitoring
- BTSPy

You have already used debug print messages extensively in previous chapters. SWD debugging will be covered in this chapter. The last two debug methods (HCI commands/monitoring and BTSPy) are related directly to Bluetooth® operation and will therefore be covered in the Level-3 Bluetooth® class.

## 5.2 Documentation

The AIROC™ BTSDK documentation contains a link to a hardware debugging guide. Much of the information from that guide is contained in this chapter, but we will focus on using the integrated KitProg3 debugger on the kit with the Eclipse IDE for ModusToolbox™. Additional details such as software setup for J-Link and using the debugger in Visual Studio Code can be found in the hardware debugging guide. You can access the guide from the AIROC™ User Guides section of the main documentation page:



The screenshot displays the Infineon AIROC™ BTSDK v3.2 - Documentation page. The page is structured with a sidebar on the left and a main content area on the right. The sidebar contains a navigation menu with the following items: Home, Components, AIROC™ CYW208XX (highlighted with a red box), AIROC™ API Reference, AIROC™ User Guides, AIROC™ Hardware Reference, AIROC™ Website, Bluetooth® References, AIROC™ Community, AIROC™ BTSDK Licensing Information, and Components. The main content area is titled "AIROC™ CYW208XX Documentation" and contains the following sections:

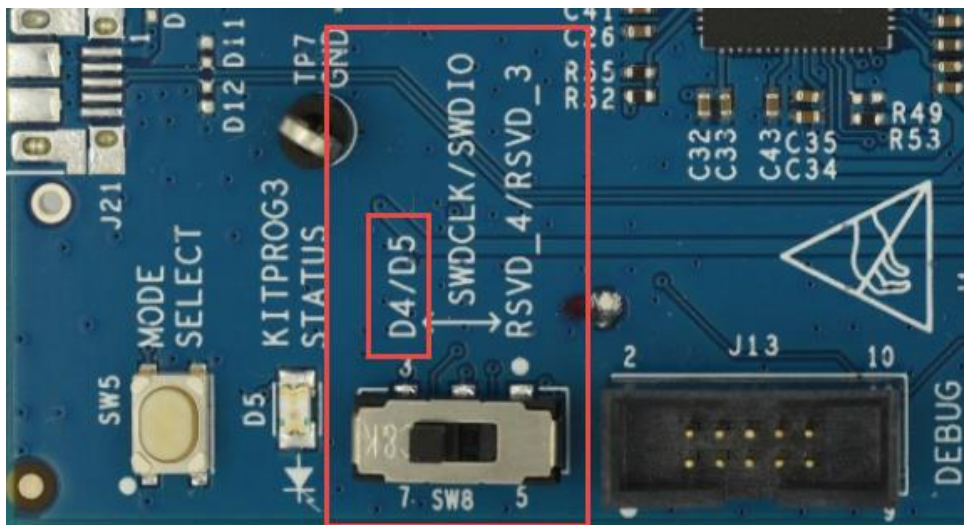
- AIROC™ CYW208XX Documentation**

This page collects software and hardware documentation related to the CYW20819/CYW20820 (aka CYW BTSDK) software API reference material, user guides, and hardware documentation can be found below.
- AIROC™ API Reference**
  - CYW208XX API functions are logically grouped into Components.
  - API function prototypes are provided in the wiced\_btSDK project under the following folders:
    - wiced\_btSDK/dev-kit/btSDK-include
    - wiced\_btSDK/dev-kit/baseLib/<20819A1/20820A1>/include
- AIROC™ User Guides**
  - AIROC™ HCI Control Protocol
  - AIROC™ HCI Firmware Download Guide
  - AIROC™ Over-the-Air Firmware Upgrade Library
  - AIROC™ Secure OTA Firmware Upgrade App Note
  - AIROC™ Application Buffer Pools App Note
  - AIROC™ XIP (Execute-In-Place) Application Support Guide
  - **AIROC™ Hardware Debugging for Bluetooth® SoC Kits** (highlighted with a red box)
  - Manufacturing Bluetooth® Test Tool
  - MeshClient and ClientControlMesh User Guide
  - CYW920819REF-KB-01 HID Reference Keyboard User Guide

## 5.3 Hardware Setup

The CYW920835M2EVB-01 kit that we are using for this class contains an integrated programmer/debugger device called a KitProg3. You have been using it thus far for programming and for transmitting UART messages, but it also supports debugging over the SWD port. The kit also supports using an external debug probe such as a MiniProg4 or a Segger J-Link probe via the 10-pin debug header.

When using the integrated KitProg3 debugger, the only hardware setup required is to ensure that the SWDCLK/SWDIO switch (SW8) is in the default **D4/D5** position. That position connects the KitProg3 SWD lines to the corresponding SWD pins of the CYW20835 device.



When using the 10-pin debug header with an external probe such as a J-Link, the SWDCLK/SWDIO switch (SW8) must be in the **RSVD\_4/RSVD\_3** and BT KP3 switch on the back of the board (SW15) must be in the **DBG HDR** position. The first switch disconnects the KitProg3 SWD lines while the second switch connects the debug header SWD lines to the corresponding SWD pins of the CYW20835.

## 5.4 Firmware Setup

The only change required in the firmware is to set the variable `ENABLE_DEBUG` to 1 in the *makefile*. This does three things:

1. Configures the SWD pins

The CYW920835M2EVB-01 kit uses P2 for SWD\_CLK and P3 for SWD\_IO. These pins are connected to Arduino header pins D4 and D5 and are normally available as GPIOs. In debug mode, these pins are used for SWD and must therefore not be used for anything else in your application.

2. Disables the watchdog timer

The watchdog timer will reset the device if it isn't cleared often enough. This is normally done automatically by the idle thread, but that may not happen during debug so it is necessary to disable the timer.

3. Enables a busy-wait loop at start up

As you learned previously, programming of the MCU is done via the HCI-UART interface. Therefore, it isn't possible to program over SWD and then immediately launch the debugger. Instead, the firmware is configured to wait after the initial boot sequence before launching the user's application. This allows you to attach to the running device and start debugging right at the start of the user's application.

Because the device is in a busy-wait loop at power up, the board will not show any activity after programming or reset until the user breaks out of the loop from inside the debugger. That means that normal operation of the user's application is not possible without using the debugger.

*Note: The settings described above are implemented in the file `wiced_btSDK/dev-kit/baselib/<device>/<version>/COMPONENT_<device>/WICED/common/spar_utils.h`. If your debug requirements are different, you can modify that file or replace it with your own settings.*

## 5.5 Programming

Once you have edited the *makefile* to set `ENABLE_DEBUG` to 1, you must program the device as normal. This step must be done separately from starting the debugger since the debugger will attach to the running target device as described above.

Once the firmware has been programmed with the debug enabled firmware, it may not be possible to re-acquire the HCI-UART to re-program the device. In order to get around this limitation, the device has a "recovery" mode that forces it into a mode where the HCI-UART remains active. In order to enter this mode, do the following:

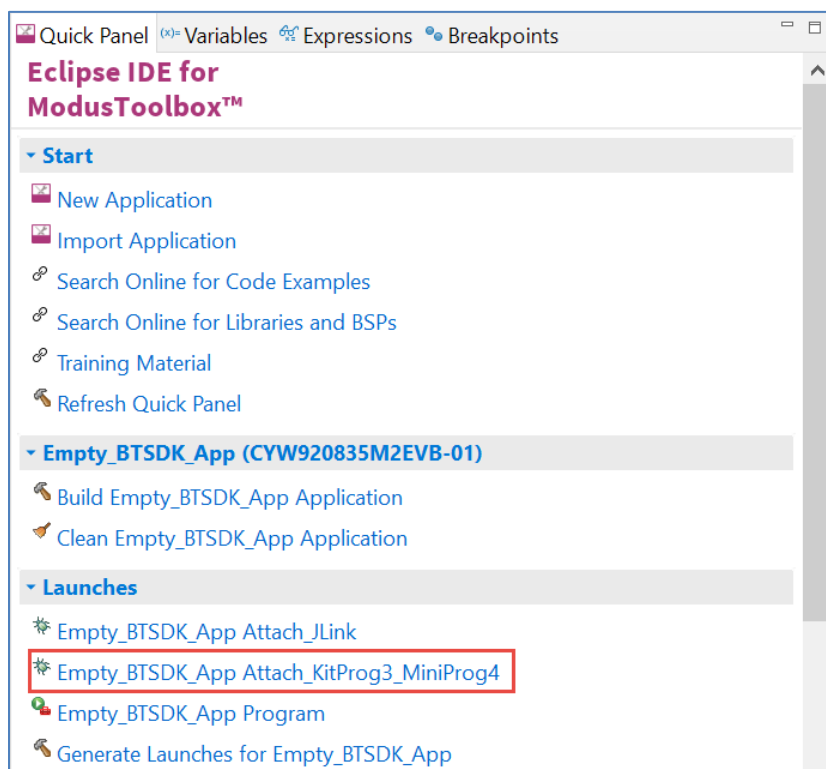
1. Hold down the "Recover" button on the kit
2. Press and release the "Reset" button
3. Release the "Recover" button

Once you follow that procedure, you should be able to program normally.

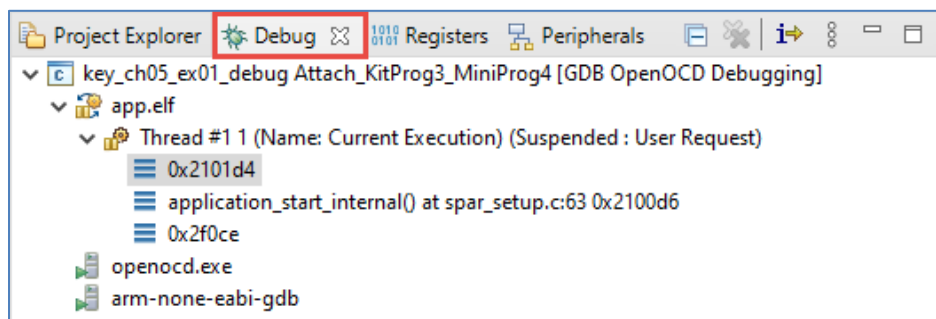
## 5.6 Launching the Debugger

Conversely to programming, when launching the debugger, you must NOT be in recovery mode. Therefore, press and release the "Reset" button on the kit prior to launching the debugger.

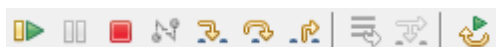
Launch configurations are provided for attaching to the target with the debugger from KitProg3/MiniProg4 and from J-Link. In the Eclipse IDE for ModusToolbox™, you will find them in the Launches section of the Quick Panel. If you are using the built-in debugger on the kit, use the **<Appname> Attach\_KitProg3\_MiniProg4** selection from the quick panel.





When the debugger launches, the upper-left window in Eclipse will switch from the **Project Explorer** tab to the **Debug** tab showing the threads in your application, and their status.



You will see debugging controls in the top menu bar:



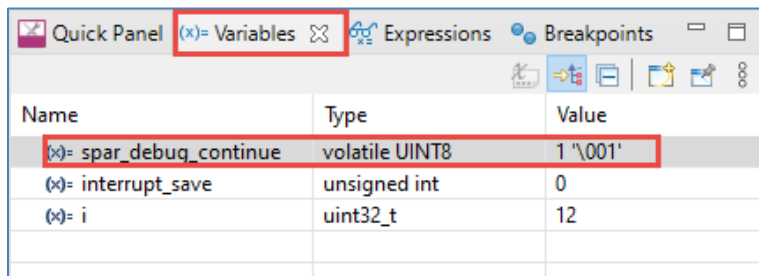
## 5.7 Using the Debugger

Once you have configured the firmware, programmed the board, and started the debugger, the application will sit in the `BUSY_WAIT_TILL_MANUAL_CONTINUE_IF_DEBUG_ENABLED` loop. Resume execution (  ) and then suspend (  ) to make sure the firmware is inside the loop.

```
/* any other personality-based initialization */
init_cycfg_all();

/* disable watchdog, set up SWD, wait for attach if ENABLE_DEBUG */
SETUP_APP_FOR_DEBUG_IF_DEBUG_ENABLED();
BUSY_WAIT_TILL_MANUAL_CONTINUE_IF_DEBUG_ENABLED();
}
```

In the Quick Panel window, switch over to the **Variables** tab. Erase the entire value for `spar_debug_contine`, enter a new value of 1 and press Enter.



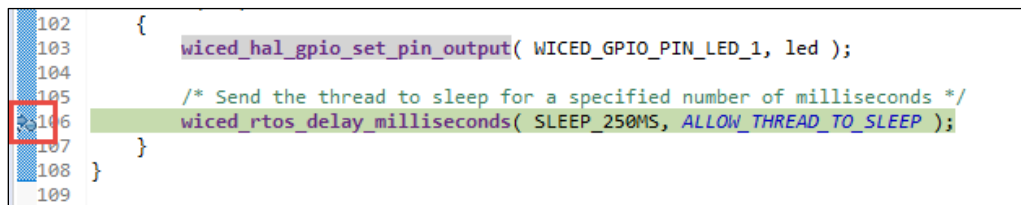
Name	Type	Value
(x)= spar_debug_continue	volatile UINT8	1 '\001'
(x)= interrupt_save	unsigned int	0
(x)= i	uint32_t	12

*Note:* Once you press enter, the value displayed will change to: 1 '\001'.

After you have changed the value of `spar_debug_continue`, you can resume execution and the program will go beyond the busy wait loop.

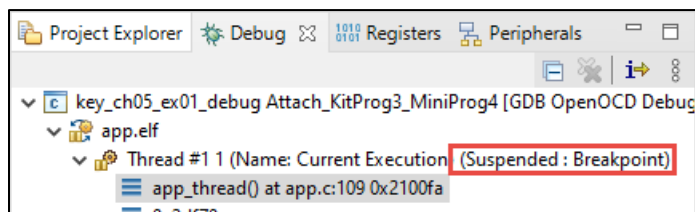
When program execution is paused, you can add breakpoints to halt at specific points in the code. To add a breakpoint, open the source file (switch to the **Project Explorer** tab in the upper-left window if necessary to find the file), click on the line where you want a breakpoint and double-click in the bar to the left of the code window or right-click in that area and select "Toggle Breakpoint". An enabled is a circle with a checkmark next to it.

Click the **Resume** button (shown in the figure above) to resume execution. The program will halt once it reaches the breakpoint.



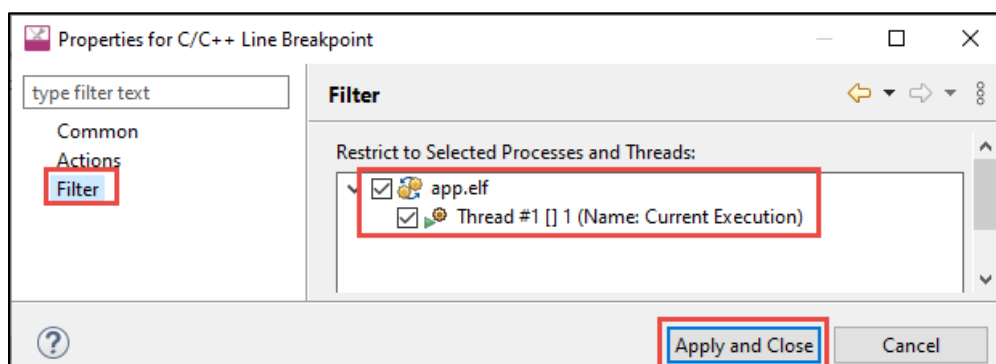
```
102 {
103     wiced_hal_gpio_set_pin_output( WICED_GPIO_PIN_LED_1, led );
104
105     /* Send the thread to sleep for a specified number of milliseconds */
106     wiced_rtos_delay_milliseconds( SLEEP_250MS, ALLOW_THREAD_TO_SLEEP );
107 }
108 }
109
```

Once a thread suspends due to a breakpoint you will see that line of code highlighted in green as shown above and you will see that the thread is suspended due to the breakpoint in the debug window as shown below.

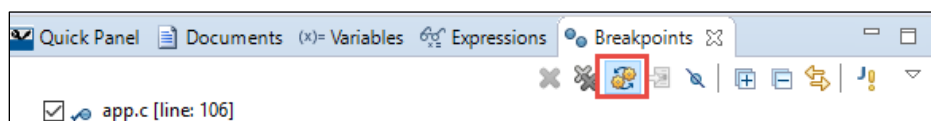


You can enable or disable breakpoints by double clicking on the green circle next to the line in the source code or from the **Breakpoints** tab in the Quick Panel.

If breakpoints are created prior to starting the current debug session, they may not be associated with the current thread and will be indicated by a circle without a check mark. To enable the breakpoints in the current thread, right-click the desired breakpoint and select **Breakpoint Properties...** Click on **Filter** and make sure the boxes are checked as shown below.



If you do not see any breakpoints in the **Breakpoints** tab, turn off the **Show Breakpoints Supported by Selected Target** button as shown below.



Click the red **Terminate** button (■) to stop debugging. Once debugging stops, you will likely want to switch back to the **Project Explorer** tab instead of the **Debug** tab. Alternately, you can use the menu item **Window > Perspective > Reset Perspective** to reset the ModusToolbox™ perspective to the default window sizes and placement.

A few additional notes on debugging:

- Single stepping through code requires a free breakpoint.
- There are two hardware breakpoints available.
- Only hardware breakpoints can be used for ROM opcodes.
- The device uses both ROM and RAM for firmware. For example, many WICED API functions will call into ROM code.
- Source code and symbols are not provided for ROM and some patch library areas. Source code debugging will be limited to the code for which you have source available.

Typically, the step command is ignored by GDB if a breakpoint is not available.

## 5.8 Troubleshooting

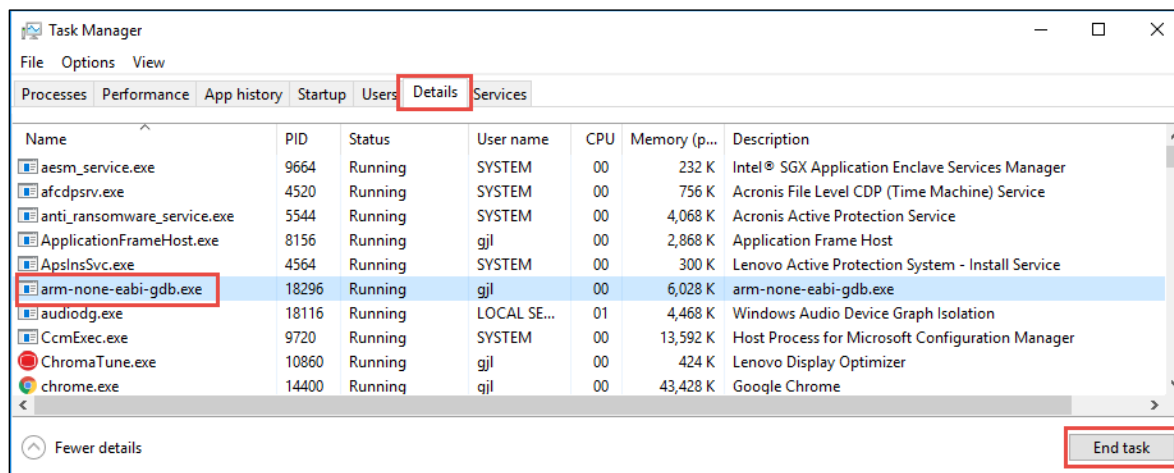
If you get an error during programming that says "Serial port not detected...", put the kit into recovery mode by following the steps below and then try programming again:

1. Hold down the "Recover" button on the kit
2. Press and release the "Reset" button
3. Release the "Recover" button

If the debugger does not connect, verify that you have done the following:

1. Set the switches on the kit as required for your debug hardware
2. Updated the *makefile* to enable debugging
3. Programmed the debug-enabled firmware on the kit
4. Pressed and released the "Reset" button on the kit

If you get an error message when launching the debugger about another instance already running, open the task manager ([**Ctrl**]+[**Alt**]+[**Del**]) and end any tasks named *arm-none-eabi-gdb.exe*.





## 5.9 Exercises

### Exercise 1: Setup and Run the Debugger

- ☐ 1. Create a new application using the template **ch05\_ex01\_debug**.
- ☐ 2. Open the *makefile* and set the value for `ENABLE_DEBUG` to 1.
- ☐ 3. Verify that the SWDCLK/SWDIO switch is set to **D4/D5**.
- ☐ 4. Program the application to the kit (**ch05\_ex01\_debug Program**).

*Note:* If you get an error saying the serial port cannot be detected, put the kit into recovery mode and then try programming again.

- ☐ 5. When programming completes, press and release the **Reset** button on the kit.
- ☐ 6. Launch the debugger and attach to the device (**ch05\_ex01\_debug Attach\_KitProg3\_MiniProg4**).
- ☐ 7. Place a break point at the `wiced_rtos_delay_milliseconds` call in the `app_task` thread in *app.c*.
- ☐ 8. Click **Resume** and then **Pause** execution. You should be in the `BUSY_WAIT_TILL_MANUAL_CONTINUE_IF_DEBUG_ENABLED` loop.
- ☐ 9. Set the value of `spar_debug_contine` to 1 to break out of the loop.

*Note:* Erase the entire value and then enter a new value of 1.

*Note:* The **Variables** tab is next to the **Quick Panel** tab in the lower-left window.

- ☐ 10. Execute a few more times and notice that execution suspends at the breakpoint you added.
- ☐ 11. Observe that the LED does not turn on because the variable `led` never changes.
- ☐ 12. Change the value of the variable `led`.

*Note:* The LED is active low.

*Note:* Type in a new value and press **Enter** for it to take effect.

- ☐ 13. Restart execution and observe that the LED turns on.
- ☐ 14. Stop the debugger by clicking the **Stop** button.
- ☐ 15. Switch the tabs back to Project Explorer (upper-left) and Quick Panel (lower-left).

☐ *Note:* You can alternately use the menu item **Window > Perspective > Reset Perspective** to reset the tabs.

- ☐ 16. Put the kit into recovery mode so that it is ready to re-program later.

#### Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

**Published by**  
**Infineon Technologies AG**  
**81726 Munich, Germany**

**© 2022 Infineon Technologies AG.**  
**All Rights Reserved.**

#### IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffheitsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office ([www.infineon.com](http://www.infineon.com)).

#### WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.