ModusToolbox™

Hardware Debugging for CYW207xx and CYW208xx

Document Number: 002-20504 Rev. *D

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
www.cypress.com

Contents

1 Introduction

ModusToolbox™ provides support for source-code-level debugging of applications running on CYW207xx and CYW208xx devices. Though this debugging technique often conflicts with the real-time requirements of IoT, it can provide valuable insight by stopping the CPU at breakpoints or watchpoints and providing the ability to check on the state of code and data using source code symbols.

The ModusToolbox kit hardware supports source-code-level debugging via a Serial Wire Debug (SWD) interface that provides a means for a development PC to control the execution of the CYW207xx and CYW208xx Arm® CPUs. Third-party JTAG/SWD debug probes are used to interface between the development PC and the debug interface hardware. The PC connected to the debug probe uses associated GDB server software. This server provides a socket interface to the GNU debugger (GDB), allowing it to control the debug interface and Arm CPU. ModusToolbox coordinates the interfaces between these entities: the CYW207xx and CYW208xx Arm CPUs, the debug probe and GDB server, GDB, and the symbols GDB needs to translate between source code symbols and hardware memory addresses, registers, and so on.

This document provides a description of hardware debugging support for ModusToolbox software and CYW207xx and CYW208xx devices. This document describes the generic tools, setup, and techniques. Other kit-specific documents are provided to describe those exact implementations.

The hardware interface for the Debug Access Port is provided to the Arm CPU cores within the CYW207xx and CYW208xx devices. These Arm architecture devices use a 2-pin SWD interface. The SWD pins are SWDIO and SWDCK. Many debug probes support SWD.

Current hardware debug support for CYW207xx and CYW208xx devices requires either a Segger J-Link probe or OpenOCD supported probe.

Hardware and Software Requirements

The following items are required to debug an application developed for Cypress WICED SoC.

Item	Description
Reference design board	A hardware reference design board based on a Cypress CYW207xx and CYW208xx SoC.
Segger J-Link	Segger provides several models of JTAG-SWD debug probes that are compatible for hardware debugging. See www.segger.com/downloads/jlink.
Olimex Arm-USB-TINY-H + JTAG-SWD (or similar OpenOCD supported probe)	A debug probe provided by Olimex. See www.olimex.com for debug probe information. The Olimex debugger is based on the FTDI FT2232 device that supports JTAG SWD to USB conversion with a serial engine. The SWD signal must be adapted via Olimex Arm-JTAG-SWD interface. The hardware probe is used with OpenOCD software running on the host PC to provide a GDB server. The Eclipse IDE can use the GDB server to provide source code debugging support.
PC	A computer (Windows, MacOS, or Linux) that hosts the following software items: • ModusToolbox • A GDB server and debug probe interface software. For this document we consider the SEGGER GDB server or OpenOCD.
Debug probe USB cable	Used for PC interface to debug hardware.
OpenOCD software (not needed for J-Link)	OpenOCD is included with ModusToolbox. The Windows USB device driver for the Olimex ARM USB TINY H must be changed in order for OpenOCD software to properly communicate. Do this by using the Zadig application (http://zadig.akeo.ie/) to make sure WinUSB drivers are installed for the two Olimex USB device interfaces. This enables the libusb software interface. Helpful though dated documentation can also be found at http://gnuarmeclipse.github.io/openocd/install/.
Debug probe 10-pin connector/adapter	An adapter may be needed to match from a 20-pin JTAG interface on the debug probe to a 10-pin SWD interface on the development kit.

Table 1. Hardware Reference

2 Debug Probe Software Setup

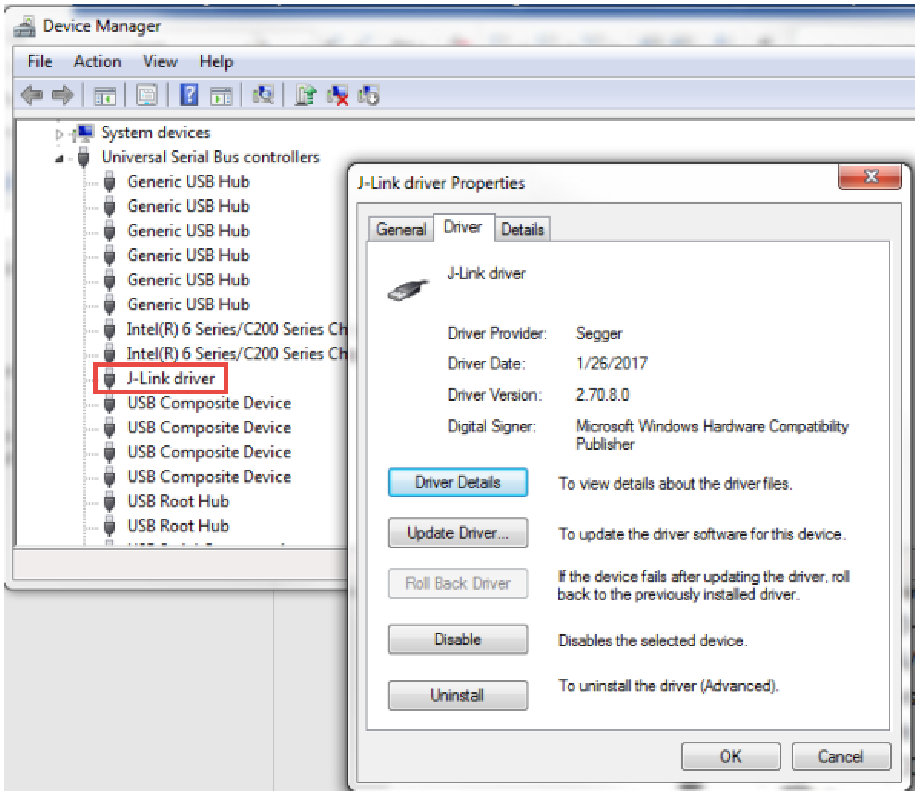
2.1 Debug Probe

Purchase debug probes and software separately. SEGGER provides the J-Link probe that has been demonstrated to work well with CYW207xx and CYW208xx devices. Other debug probes, such as Olimex ARM-USB-TINY-H + JTAG-SWD, are also compatible.

The instructions and screen shots are similar for Windows, Linux, and Mac operating systems unless otherwise noted.

2.2 SEGGER J-Link

1. Install the Segger J-Link GDB Server. Download the software from www.segger.com/downloads/jlink. Look for “J-Link Software and Documentation Pack”.
2. Once the software is installed, connect J-Link to the kit’s debug connector and then plug the J-Link into a USB port on the computer. When USB enumeration completes, the J-Link device should appear on a Windows PC in the Device Manager as Figure 2-1 shows.



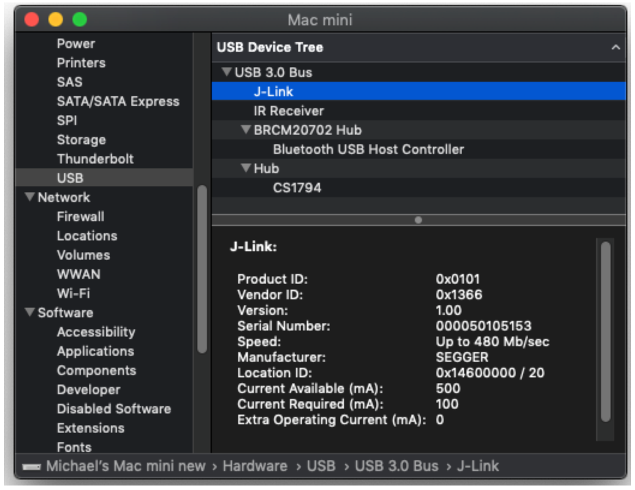
J-Link driver

Driver Provider: Segger
Driver Date: 1/26/2017
Driver Version: 2.70.8.0
Digital Signer: Microsoft Windows Hardware Compatibility Publisher

To view details about the driver files.
To update the driver software for this device.
If the device fails after updating the driver, roll back to the previously installed driver.
Disables the selected device.
To uninstall the driver (Advanced).

Figure 2-1. Device Manager Listing J-Link Device

3. You can check the SEGGER J-Link hardware connection on a Linux PC using `lsusb` on a terminal to list USB devices. On a Mac operating system, you can use the System Information utility.



```
USB Device Tree
▼USB 3.0 Bus
    J-Link
    IR Receiver
  ▼BRCM20702 Hub
      Bluetooth USB Host Controller
  ▼Hub
      CS1794

J-Link:

  Product ID:                    0x0101
  Vendor ID:                     0x1366
  Version:                       1.00
  Serial Number:                 000050105153
  Speed:                         Up to 480 Mb/sec
  Manufacturer:                  SEGGER
  Location ID:                   0x14600000 / 20
  Current Available (mA):        500
  Current Required (mA):         100
  Extra Operating Current (mA):  0
```

Figure 2-2. Using System Information Utility

2.3 Olimex Arm-USB-TINY-H with OpenOCD

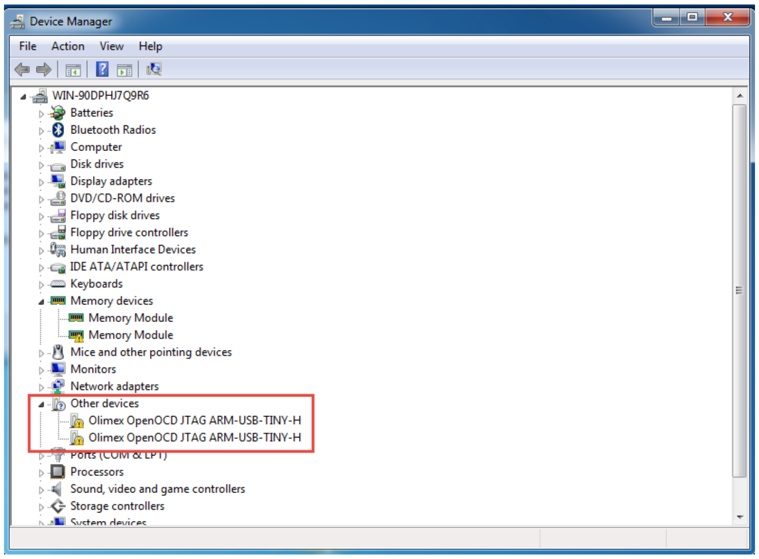
Do the following to use the Olimex debugger:

1. ModusToolbox 1.1 supports OpenOCD tools, found in *<ModusToolbox install dir>/tools/openocd-2.1*.
2. Download driver files for your hardware debugger (see Table 1. Hardware Reference).
3. A driver update from Zadig may be required if the drivers for Olimex Debugger do not install properly on their own. The procedure is described below.

   a. Connect the kit and Olimex Debugger to each other and to USB ports on your computer.

      Note: The following steps are necessary only for a Windows PC.

   b. Open Device Manager and look for the Olimex Debugger. If it shows up under **Other devices** as shown in Figure 2-3, the driver must be updated manually.

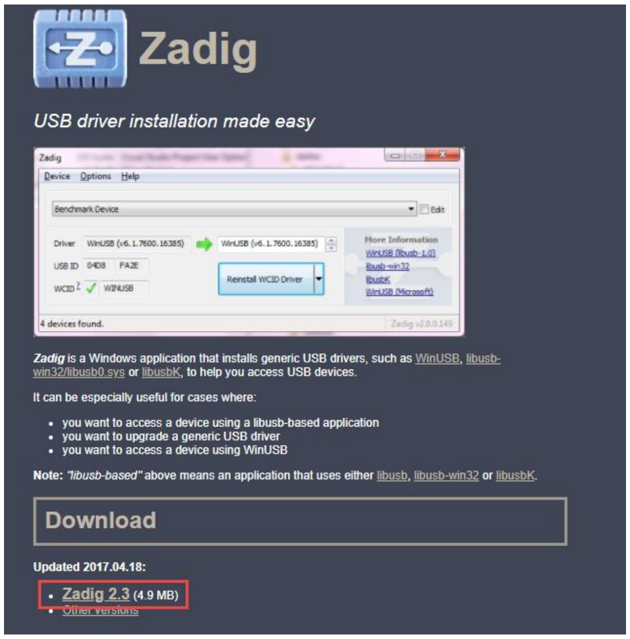


```
WIN-90DPHJ7Q9R6
  Batteries
  Bluetooth Radios
  Computer
  Disk drives
  Display adapters
  DVD/CD-ROM drives
  Floppy disk drives
  Floppy drive controllers
  Human Interface Devices
  IDE ATA/ATAPI controllers
  Keyboards
  Memory devices
    Memory Module
    Memory Module
  Mice and other pointing devices
  Monitors
  Network adapters
  Other devices
    Olimex OpenOCD JTAG ARM-USB-TINY-H
    Olimex OpenOCD JTAG ARM-USB-TINY-H
  Ports (COM & LPT)
  Processors
  Sound, video and game controllers
  Storage controllers
  System devices
```

Figure 2-3. Device Manager Listing Olimex Debugger under Other Devices

If the driver is not installed properly, download the driver update tool from http://zadig.akeo.ie/

4. Go to http://zadig.akeo.ie/ and download Zadig.



Zadig

USB driver installation made easy

Zadig

Device Options Help

Benchmark Device

Driver WinUSB (v6.1.7600.16385) → WinUSB (v6.1.7600.16385)

USB ID 04D8 FA2E

WCID ✓ WINUSB

Reinstall WCID Driver

More Information: WinUSB (libusb-1.0), libusb-win32, libusbK, WinUSB (Microsoft)

4 devices found. Zadig v2.0.0.149

Zadig is a Windows application that installs generic USB drivers, such as WinUSB, libusb-win32/libusb0.sys or libusbK, to help you access USB devices.

It can be especially useful for cases where:

- you want to access a device using a libusb-based application
- you want to upgrade a generic USB driver
- you want to access a device using WinUSB

Note: *"libusb-based"* above means an application that uses either libusb, libusb-win32 or libusbK.

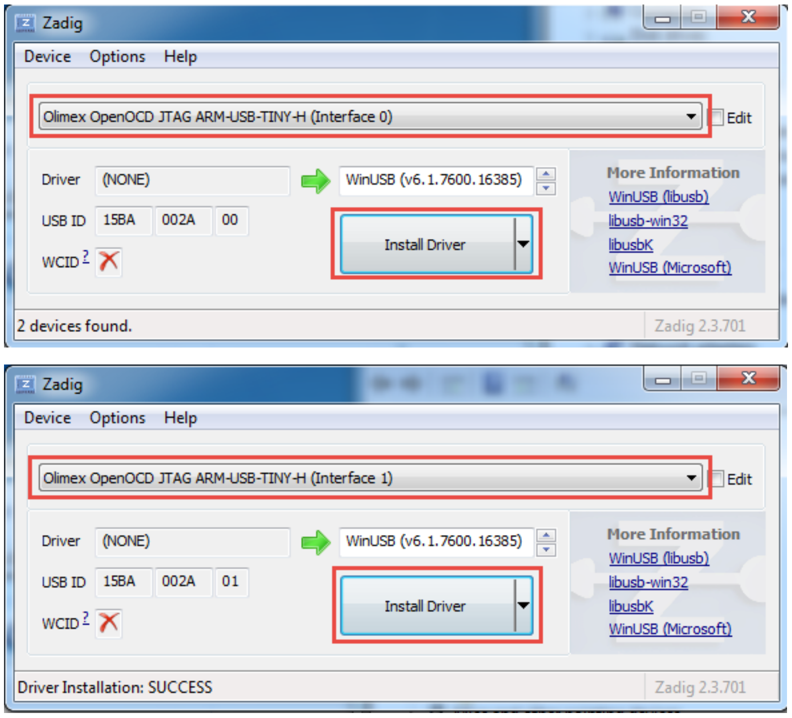
Download

Updated 2017.04.18:

- Zadig 2.3 (4.9 MB)
- Other versions

Figure 2-4. Downloading Zadig

5. After downloading Zadig, run the executable. Select each Olimex channel one at a time and click **Install Driver**.



Zadig

Device Options Help

Olimex OpenOCD JTAG ARM-USB-TINY-H (Interface 0)

Driver (NONE) → WinUSB (v6.1.7600.16385)

USB ID 15BA 002A 00

WCID ✗

Install Driver

More Information: WinUSB (libusb), libusb-win32, libusbK, WinUSB (Microsoft)

2 devices found. Zadig 2.3.701

Zadig

Device Options Help

Olimex OpenOCD JTAG ARM-USB-TINY-H (Interface 1)

Driver (NONE) → WinUSB (v6.1.7600.16385)

USB ID 15BA 002A 01

WCID ✗

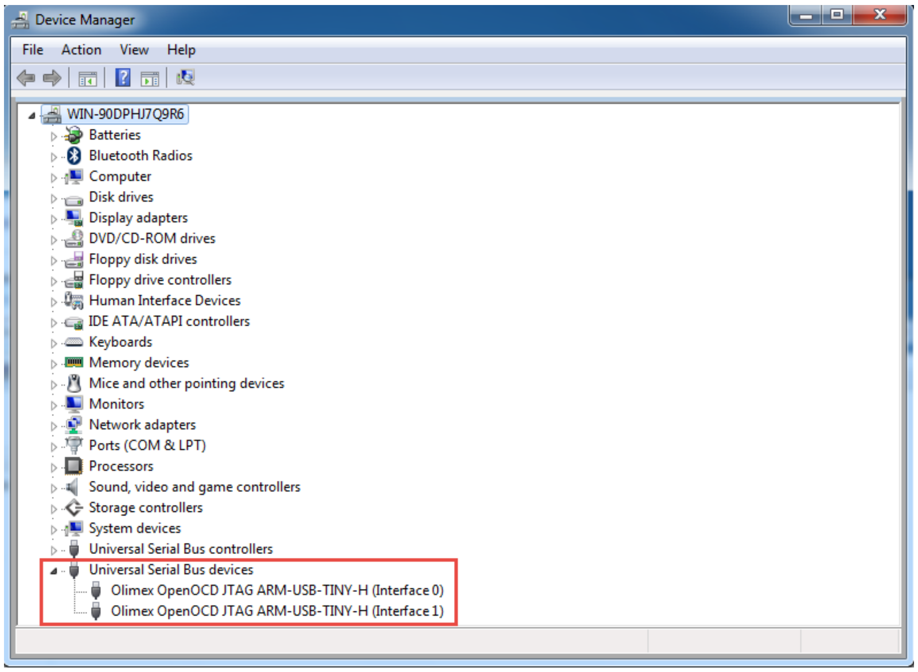
Install Driver

More Information: WinUSB (libusb), libusb-win32, libusbK, WinUSB (Microsoft)

Driver Installation: SUCCESS Zadig 2.3.701

Figure 2-5. Installing Olimex Drivers

After installing the drivers, the Olimex Debugger should show up in the Device Manager under Universal Serial Bus devices as shown in Figure 2-6.



```
WIN-90DPHJ7Q9R6
  Batteries
  Bluetooth Radios
  Computer
  Disk drives
  Display adapters
  DVD/CD-ROM drives
  Floppy disk drives
  Floppy drive controllers
  Human Interface Devices
  IDE ATA/ATAPI controllers
  Keyboards
  Memory devices
  Mice and other pointing devices
  Monitors
  Network adapters
  Ports (COM & LPT)
  Processors
  Sound, video and game controllers
  Storage controllers
  System devices
  Universal Serial Bus controllers
  Universal Serial Bus devices
    Olimex OpenOCD JTAG ARM-USB-TINY-H (Interface 0)
    Olimex OpenOCD JTAG ARM-USB-TINY-H (Interface 1)
```

Figure 2-6. Device Manager Listing Olimex Debugger under Universal Serial Bus Devices

OpenOCD is included with ModusToolbox, so no installation is required. The OpenOCD GDB server can be verified by using the command line to launch it once the Olimex adapter is connected to the kit, the kit is configured to support hardware debugging, and an embedded application that sets to the GPIOs to support hardware debugging has been built and downloaded.

3 Kit Setup

Each Cypress IoT development kit has particular setup requirements for the debug interface. Some have a dedicated debug probe socket, either 10-pin or 20-pin. A few kits do not have a dedicated debug socket, but will have pin headers that can be "fly-wired" to a debug probe connector. Beside the physical connection, each board may have some switch or jumper setting necessary to support hardware debugging. Kit-specific documents are available to describe the exact requirements.

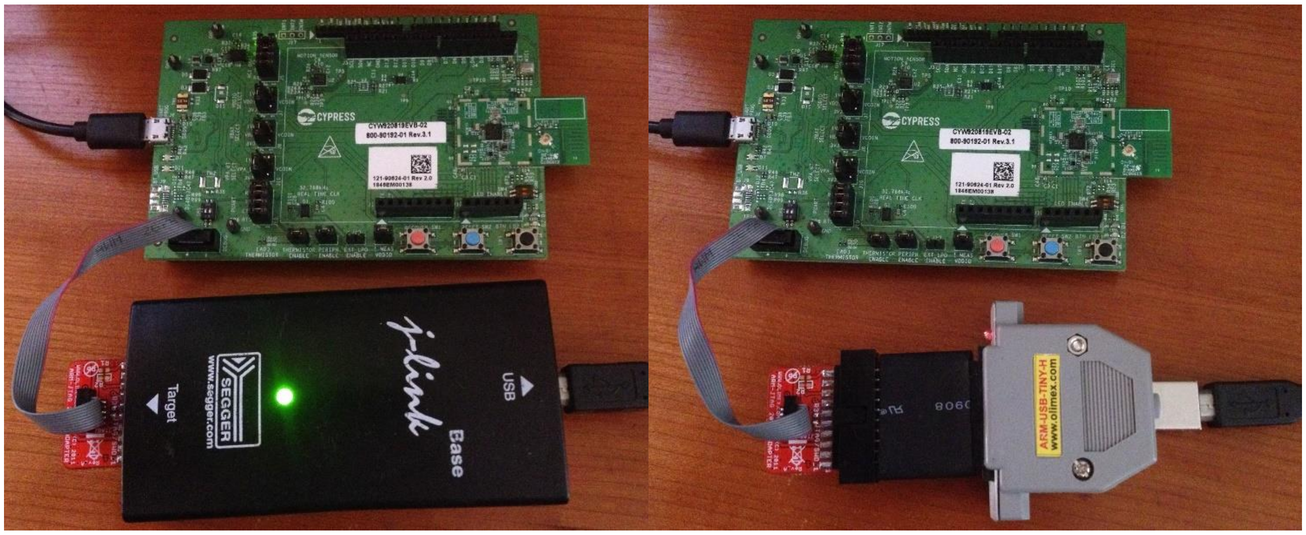


Figure 3-1. Typical Kit with Hardware Debug Probes

4 Preparing the Embedded Application for Debug

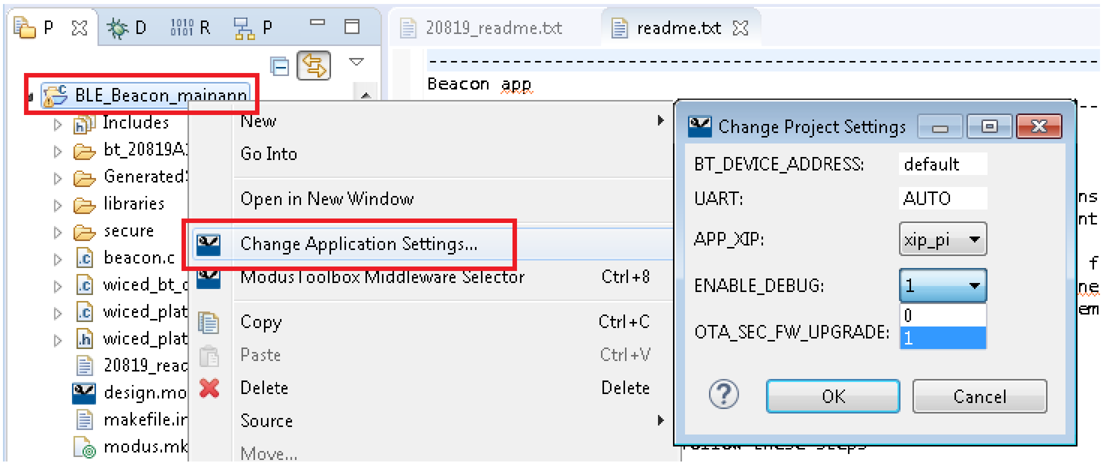
By default, CYW207xx and CYW208xx devices do not enable SWD interface pins. These devices boot from the ROM code and do not have an opportunity to set up SWD pins until the ModusToolbox embedded application is loaded and executed. This means that SWD support must be enabled as a build option so that code will be compiled in to configure pins during the application startup sequence to support SWD. The code necessary to configure GPIOs for debugger support is defined in a macro called `SETUP_APP_FOR_DEBUG_IF_DEBUG_ENABLED()`.

Because the pins must be configured prior to attaching the debugger, the application is built to run in a software loop immediately after configuring the GPIO for debug support. The code for this loop is defined in a macro called `BUSY_WAIT_TILL_MANUAL_CONTINUE_IF_DEBUG_ENABLED()`.

These macros are defined in the source code file *<ModusToolbox install dir>/libraries/bt_sdk-1.1/components/BT-SDK/<platform directory>/WICED/common/spar_utils.h.*

By default, these macros are enabled in an early part of application initialization, just after pins are configured in `wiced_platform_init()`. Depending on the application, it may make sense to move them to another location. Some critical points to consider are whether any GPIO configuration occurs in the application that would be executed subsequent to the debug setup. If the GPIO used for debug have their configuration overwritten by other code, the debug session will not work. Also, if the wait loop is executed in a time-critical portion of the application, then the time-critical functionality could be broken.

To set up the build to enable the desired pins for SWD using the ModusToolbox IDE, right-click on the project name from the workspace explorer, select **Change Application Settings…** and set **ENABLE_DEBUG** to **1**. Note that the SWD pins are defined for each kit and are configured by the macro definitions in *spar_utils.h*. The macro definitions may need to be modified to change the pins that will be used for SWD. The pin configurator is not used because the pin functionality depends on setting or unsetting ENABLE_DEBUG.



BT_DEVICE_ADDRESS: default
UART: AUTO
APP_XIP: xip_pi
ENABLE_DEBUG: 1
OTA_SEC_FW_UPGRADE: 0 / 1

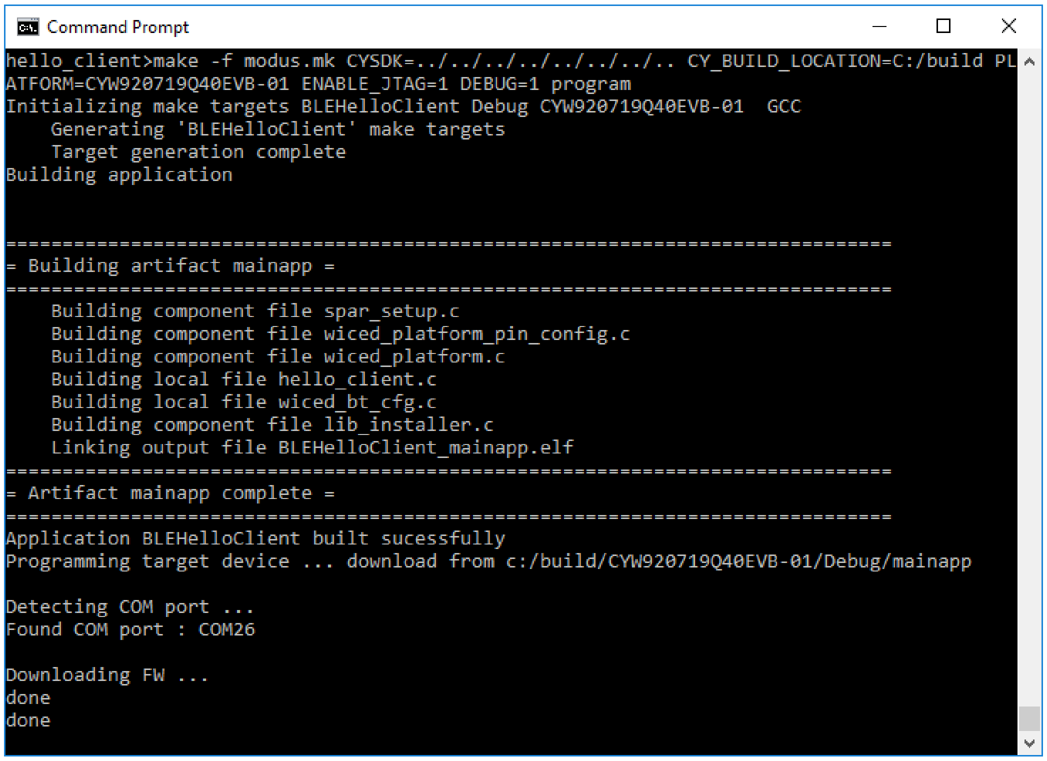
Figure 4-1. Selecting the ENABLE_DEBUG Feature

Now you can build, program, and download the application.

Similarly, to build from the command line, use the make command line option `ENABLE_DEBUG`:

```
make -f modus.mk ENABLE_DEBUG=1 program
```

Additional command line options may be needed depending on the application, CYSDK path, and platform being used.



```
hello_client>make -f modus.mk CYSDK=../../../../../../../.. CY_BUILD_LOCATION=C:/build PL
ATFORM=CYW920719Q40EVB-01 ENABLE_JTAG=1 DEBUG=1 program
Initializing make targets BLEHelloClient Debug CYW920719Q40EVB-01  GCC
    Generating 'BLEHelloClient' make targets
    Target generation complete
Building application


==============================================================================
= Building artifact mainapp =
==============================================================================
    Building component file spar_setup.c
    Building component file wiced_platform_pin_config.c
    Building component file wiced_platform.c
    Building local file hello_client.c
    Building local file wiced_bt_cfg.c
    Building component file lib_installer.c
    Linking output file BLEHelloClient_mainapp.elf
==============================================================================
= Artifact mainapp complete =
==============================================================================
Application BLEHelloClient built sucessfully
Programming target device ... download from c:/build/CYW920719Q40EVB-01/Debug/mainapp

Detecting COM port ...
Found COM port : COM26

Downloading FW ...
done
done
```

Figure 4-2. Command Line Build

5 Using the Hardware Debugger

Before using the debugger, do the following:

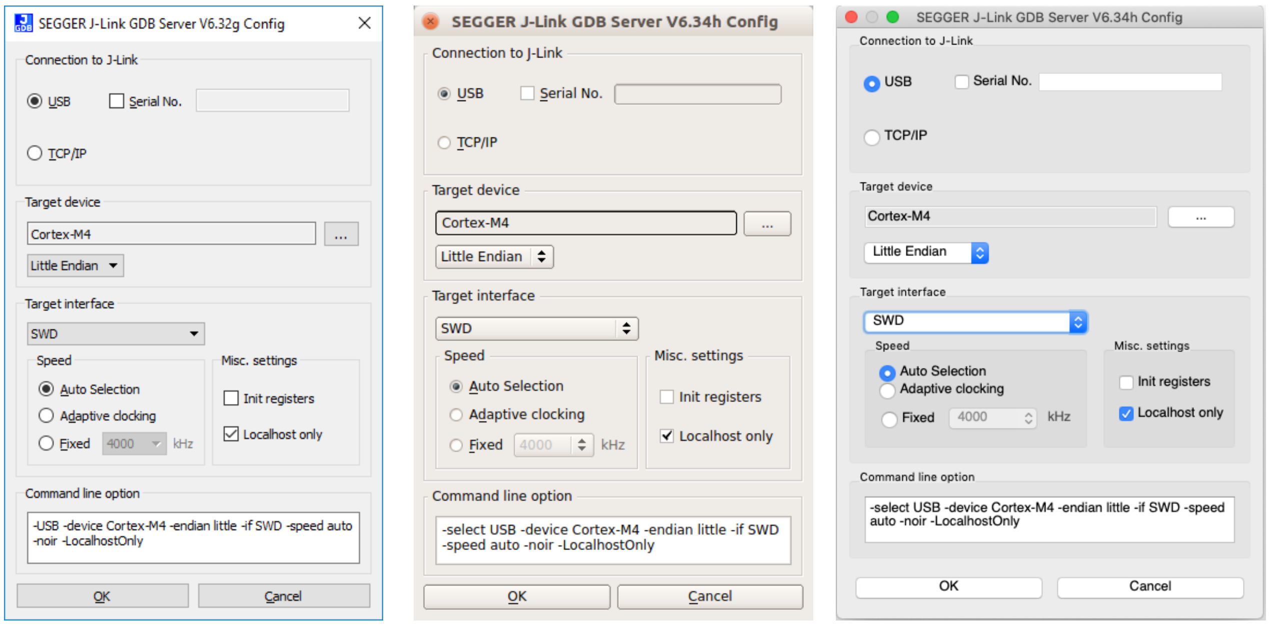
- Connect the hardware debugger to the kit.
- Configure the kit for debugging.
- Build, program, and run an application that has debugging enabled on the kit.

Program execution stops progressing at the wait macro `BUSY_WAIT_TILL_MANUAL_CONTINUE_IF_DEBUG_ENABLED()`. See the kit-specific documents for details on the full setup.

5.1 Validate the Hardware Setup

The SWD interface can be tested independently from the ModusToolbox environment. This does require that an embedded application configured with ENABLE_DEBUG is first built, downloaded, and running on the kit. Normally, this application will be run in the background automatically while performing hardware debugging with the ModusToolbox IDE, so you don't need to start it separately. The GUI version of the GDB Server described below can be used to troubleshoot or support command-line debugging, if required.

1. Connect the probe to the kit and both to the computer.
2. Configure the kit and application for hardware debugging.
3. Program the kit with the application.
4. Run the Segger J-Link GDB Server program and set up as shown in Figure 5-1.
5. Once setup is done, click OK to get the window shown in Figure 5-2.
6. Close the GDB server after the test is completed.



SEGGER J-Link GDB Server V6.32g Config

Connection to J-Link: [✓ USB] Serial No. TCP/IP
Target device: Cortex-M4, Little Endian
Target interface: SWD
Speed: [✓ Auto Selection] Adaptive clocking, Fixed 4000 kHz
Misc. settings: Init registers [✓ Localhost only]
Command line option:

```
-USB -device Cortex-M4 -endian little -if SWD -speed auto -noir -LocalhostOnly
```

SEGGER J-Link GDB Server V6.34h Config

Connection to J-Link: [✓ USB] Serial No. TCP/IP
Target device: Cortex-M4, Little Endian
Target interface: SWD
Speed: [✓ Auto Selection] Adaptive clocking, Fixed 4000 kHz
Misc. settings: Init registers [✓ Localhost only]
Command line option:

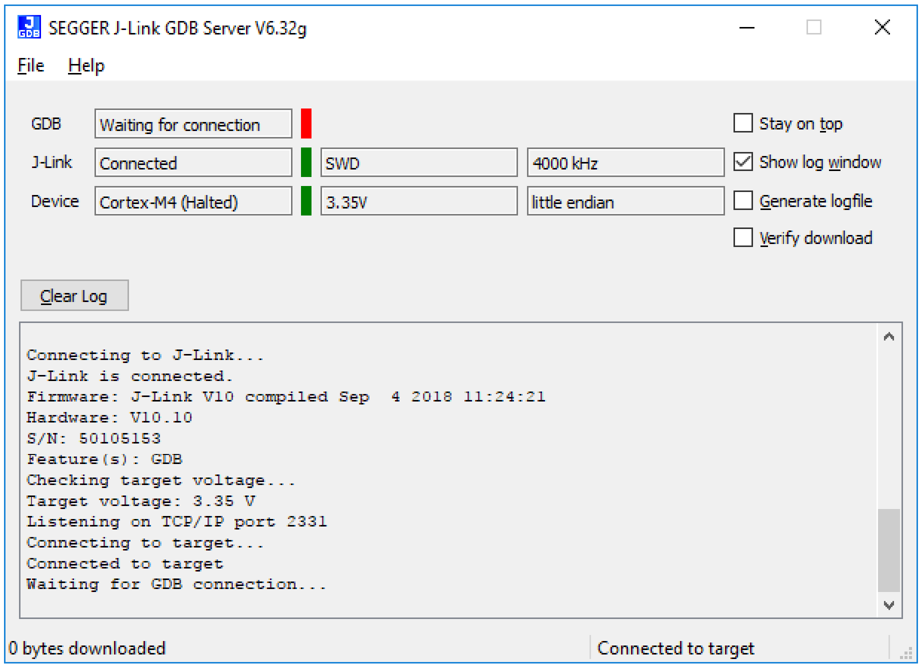
```
-select USB -device Cortex-M4 -endian little -if SWD -speed auto -noir -LocalhostOnly
```

SEGGER J-Link GDB Server V6.34h Config

Connection to J-Link: [✓ USB] Serial No. TCP/IP
Target device: Cortex-M4, Little Endian
Target interface: SWD
Speed: [✓ Auto Selection] Adaptive clocking, Fixed 4000 kHz
Misc. settings: Init registers [✓ Localhost only]
Command line option:

```
-select USB -device Cortex-M4 -endian little -if SWD -speed auto -noir -LocalhostOnly
```

Figure 5-1. SEGGER GUI for Windows, Linux, and mac OS



GDB: Waiting for connection 🟥
J-Link: Connected 🟩 SWD 4000 kHz
Device: Cortex-M4 (Halted) 🟩 3.35V little endian

[ ] Stay on top
[✓] Show log window
[ ] Generate logfile
[ ] Verify download

```
Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V10 compiled Sep  4 2018 11:24:21
Hardware: V10.10
S/N: 50105153
Feature(s): GDB
Checking target voltage...
Target voltage: 3.35 V
Listening on TCP/IP port 2331
Connecting to target...
Connected to target
Waiting for GDB connection...
```

0 bytes downloaded   Connected to target

Figure 5-2. Waiting for GDB Connection

The process is similar to test the OpenOCD setup. Perform steps 1-3 as above.

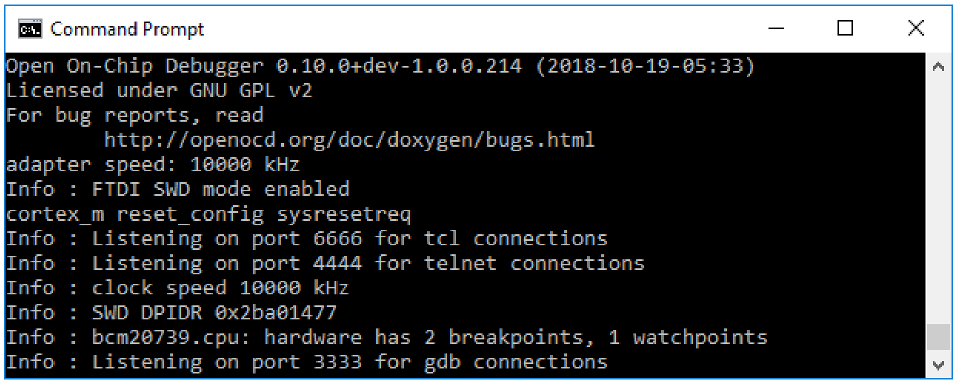
4. Open a terminal, change directory to <ModusToolbox install dir>/tools/openocd-2.1/bin and enter the following command:

```
.\openocd -s ..\scripts -s ..\..\..\libraries\bt_sdk-1.1\components\BT-SDK\<platform
path>\platforms -f <device name>_openocd.cfg
```

   Replace <platform path> with a valid directory name such as 208XX-A1_Bluetooth. Replace <device name> with a valid device such as CYW20819A1.

5. You should see a result as shown in Figure 5-3.

6. Be sure to close the OpenOCD program when the test is complete.



```
Open On-Chip Debugger 0.10.0+dev-1.0.0.214 (2018-10-19-05:33)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 10000 kHz
Info : FTDI SWD mode enabled
cortex_m reset_config sysresetreq
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : clock speed 10000 kHz
Info : SWD DPIDR 0x2ba01477
Info : bcm20739.cpu: hardware has 2 breakpoints, 1 watchpoints
Info : Listening on port 3333 for gdb connections
```

Figure 5-3. Verifying Installation

5.2 Using the IDE

ModusToolbox has launch configurations for each platform. These are the Launch items listed in the Quick Panel. Note that this list will change depending on the application project and the platform settings.

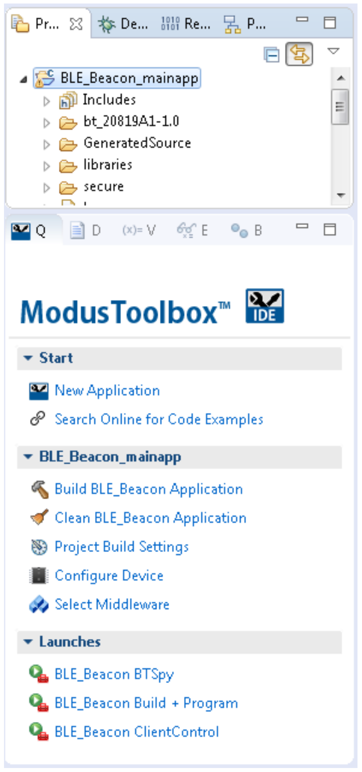


Figure 5-4. Quick Panel, Hardware Debug Not Enabled

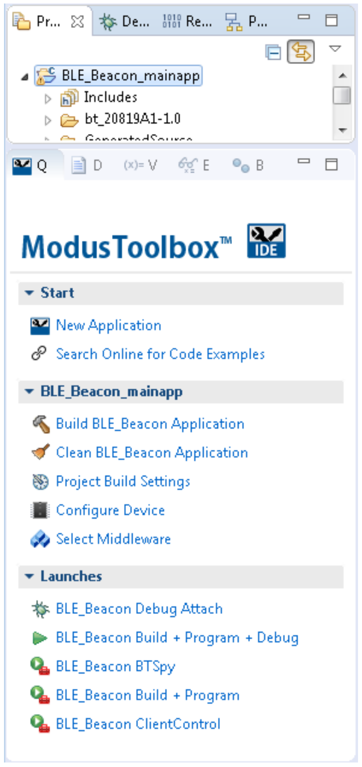
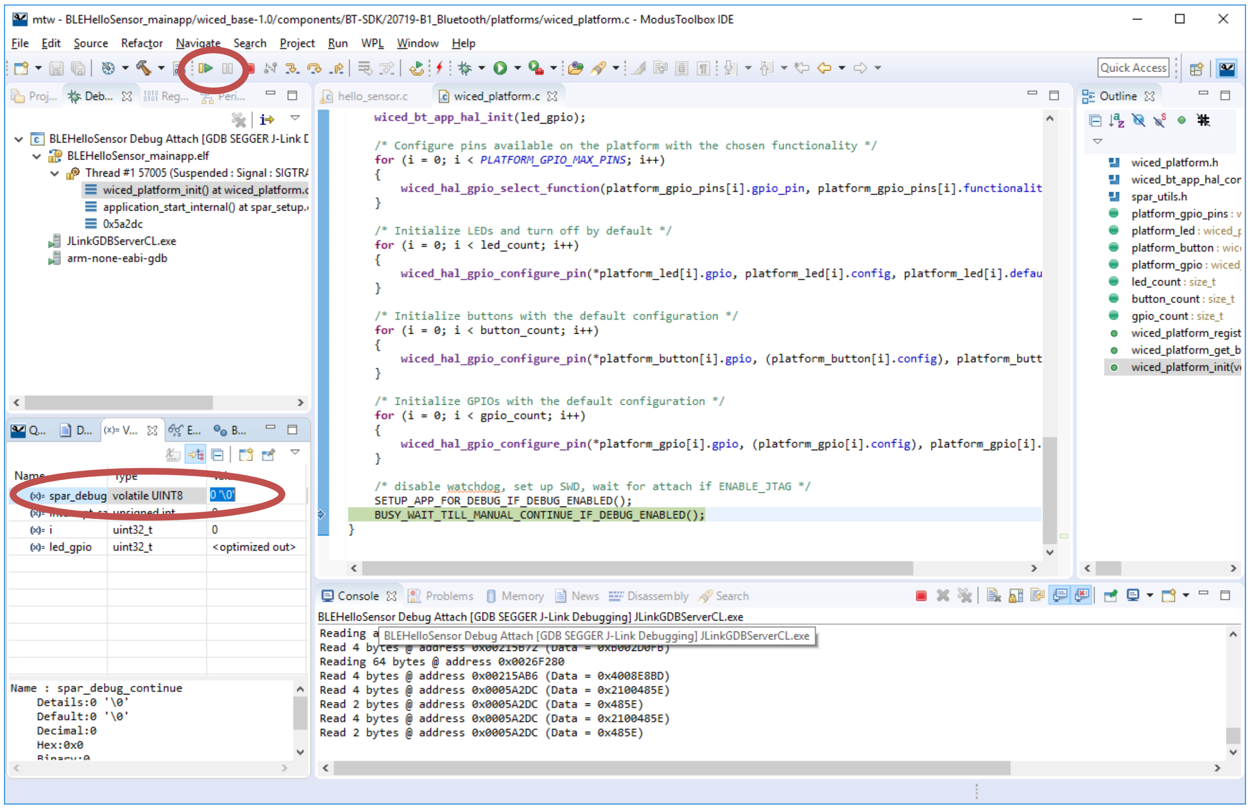


Figure 5-5. Quick Panel, Hardware Debug Enabled

Launch configurations are generated for each new project using the information from .xml files located in the <*ModusToolbox install dir*>*/libraries/bt_sdk-1.1/makefiles/platforms/*<*device*> directories. Existing debug launch items are configured by default for J-Link. An OpenOCD launch configuration can be set up by editing a makefile as described below. If a hardware debugger is attached and a debug-enabled application is running on the board, debugging starts when you click **Debug Attach**. Once launched, clicking the "suspend" debug control button will halt the program execution at the `BUSY_WAIT_TILL_MANUAL_CONTINUE_IF_DEBUG_ENABLED()` loop. This loop can be exited by setting the loop control variable `spar_debug_continue` to non-zero and clicking the debug resume button. At that point, your user application will begin running.



mtw - BLEHelloSensor_mainapp/wiced_base-1.0/components/BT-SDK/20719-B1_Bluetooth/platforms/wiced_platform.c - ModusToolbox IDE

BLEHelloSensor Debug Attach [GDB SEGGER J-Link D
BLEHelloSensor_mainapp.elf
Thread #1 57005 (Suspended : Signal : SIGTRA
wiced_platform_init() at wiced_platform.c
application_start_internal() at spar_setup.
0x5a2dc
JLinkGDBServerCL.exe
arm-none-eabi-gdb

Name	Type	Value
spar_debug	volatile UINT8	0 '\0'
i	uint32_t	0
led_gpio	uint32_t	<optimized out>

```
Name : spar_debug_continue
    Details:0 '\0'
    Default:0 '\0'
    Decimal:0
    Hex:0x0
    Binary:0
```

```
    wiced_bt_app_hal_init(led_gpio);

    /* Configure pins available on the platform with the chosen functionality */
    for (i = 0; i < PLATFORM_GPIO_MAX_PINS; i++)
    {
        wiced_hal_gpio_select_function(platform_gpio_pins[i].gpio_pin, platform_gpio_pins[i].functionalit
    }

    /* Initialize LEDs and turn off by default */
    for (i = 0; i < led_count; i++)
    {
        wiced_hal_gpio_configure_pin(*platform_led[i].gpio, platform_led[i].config, platform_led[i].defau
    }

    /* Initialize buttons with the default configuration */
    for (i = 0; i < button_count; i++)
    {
        wiced_hal_gpio_configure_pin(*platform_button[i].gpio, (platform_button[i].config), platform_butt
    }

    /* Initialize GPIOs with the default configuration */
    for (i = 0; i < gpio_count; i++)
    {
        wiced_hal_gpio_configure_pin(*platform_gpio[i].gpio, (platform_gpio[i].config), platform_gpio[i].
    }

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

BLEHelloSensor Debug Attach [GDB SEGGER J-Link Debugging] JLinkGDBServerCL.exe

```
Reading a
Read 4 bytes
Reading 64 bytes @ address 0x0026F280
Read 4 bytes @ address 0x00215AB6 (Data = 0x4008E8BD)
Read 4 bytes @ address 0x0005A2DC (Data = 0x2100485E)
Read 2 bytes @ address 0x0005A2DC (Data = 0x485E)
Read 4 bytes @ address 0x0005A2DC (Data = 0x2100485E)
Read 2 bytes @ address 0x0005A2DC (Data = 0x485E)
```

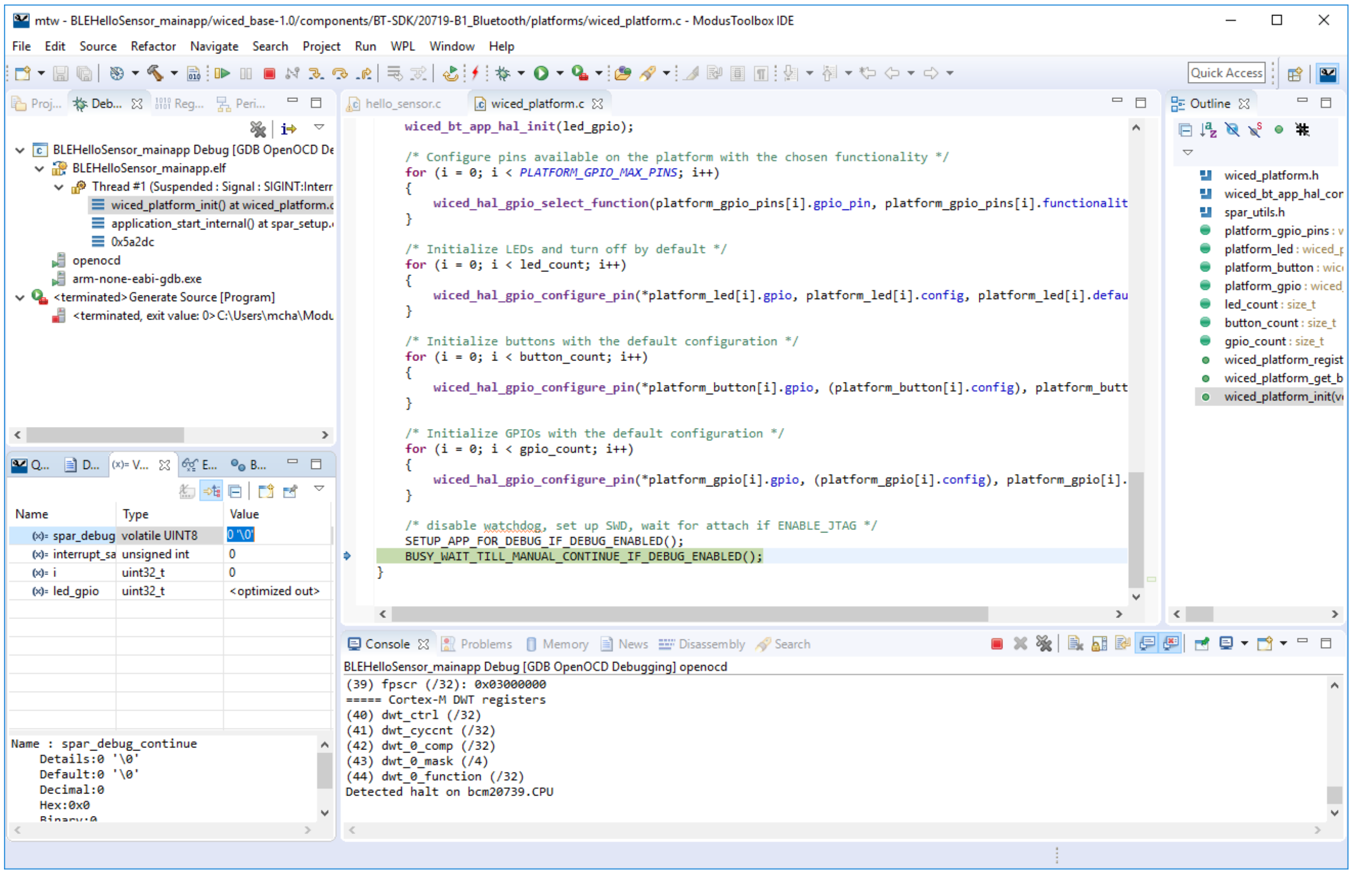
Figure 5-6. Debug View after Pause Using Segger J-Link

An alternative way to access the debug launch configurations is by accessing the menu item **Run** > **Debug Configurations…**. This method also launches a dialog that can be used to modify or define new debug launch configurations.

The OpenOCD launch configuration can be enabled by modifying *<ModusToolbox install dir>/libraries/bt_sdk-1.1/makefiles/platforms/<device>/mainapp.mk.* Within the makefie, the following lines define the launch configuration:

```
ifeq ($(ENABLE_DEBUG),1)
CY_MAINAPP_DEFINES += -DDEBUG -DENABLE_DEBUG=1
CY_MAINAPP_LAUNCH_CFGS=$(CY_RECIPE_DIR)/platforms/$(PLATFORM)/bluetooth.jlink.launch.xml
#CY_MAINAPP_LAUNCH_CFGS=$(CY_RECIPE_DIR)/platforms/$(PLATFORM)/bluetooth.openocd.launch.xml
else
CY_MAINAPP_LAUNCH_CFGS=$(CY_RECIPE_DIR)/platforms/$(PLATFORM)/nojtag.launch.xml
endif
```

Move the “#” comment character from the line with “bluetooth.openocd.launch.xml” to the line with “bluetooth.jlink.launch.xml” to switch from J-Link to OpenOCD launch configuration.


mtw - BLEHelloSensor_mainapp/wiced_base-1.0/components/BT-SDK/20719-B1_Bluetooth/platforms/wiced_platform.c - ModusToolbox IDE
File Edit Source Refactor Navigate Search Project Run WPL Window Help
Quick Access
Proj... Deb... Reg... Peri...
hello_sensor.c wiced_platform.c
Outline
Console Problems Memory News Disassembly Search

```
BLEHelloSensor_mainapp Debug [GDB OpenOCD De
  BLEHelloSensor_mainapp.elf
    Thread #1 (Suspended : Signal : SIGINT:Interr
      wiced_platform_init() at wiced_platform.c
      application_start_internal() at spar_setup.
      0x5a2dc
  openocd
  arm-none-eabi-gdb.exe
<terminated>Generate Source [Program]
  <terminated, exit value: 0>C:\Users\mcha\Modu
```

Name	Type	Value
spar_debug	volatile UINT8	0 '\0'
interrupt_sa	unsigned int	0
i	uint32_t	0
led_gpio	uint32_t	<optimized out>

```
Name : spar_debug_continue
    Details:0 '\0'
    Default:0 '\0'
    Decimal:0
    Hex:0x0
    Binary:0
```

```c
    wiced_bt_app_hal_init(led_gpio);

    /* Configure pins available on the platform with the chosen functionality */
    for (i = 0; i < PLATFORM_GPIO_MAX_PINS; i++)
    {
        wiced_hal_gpio_select_function(platform_gpio_pins[i].gpio_pin, platform_gpio_pins[i].functionalit
    }

    /* Initialize LEDs and turn off by default */
    for (i = 0; i < led_count; i++)
    {
        wiced_hal_gpio_configure_pin(*platform_led[i].gpio, platform_led[i].config, platform_led[i].defau
    }

    /* Initialize buttons with the default configuration */
    for (i = 0; i < button_count; i++)
    {
        wiced_hal_gpio_configure_pin(*platform_button[i].gpio, (platform_button[i].config), platform_butt
    }

    /* Initialize GPIOs with the default configuration */
    for (i = 0; i < gpio_count; i++)
    {
        wiced_hal_gpio_configure_pin(*platform_gpio[i].gpio, (platform_gpio[i].config), platform_gpio[i].
    }

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

```
wiced_platform.h
wiced_bt_app_hal_cor
spar_utils.h
platform_gpio_pins : v
platform_led : wiced_
platform_button : wic
platform_gpio : wiced
led_count : size_t
button_count : size_t
gpio_count : size_t
wiced_platform_regist
wiced_platform_get_b
wiced_platform_init(v
```

```
BLEHelloSensor_mainapp Debug [GDB OpenOCD Debugging] openocd
(39) fpscr (/32): 0x03000000
===== Cortex-M DWT registers
(40) dwt_ctrl (/32)
(41) dwt_cyccnt (/32)
(42) dwt_0_comp (/32)
(43) dwt_0_mask (/4)
(44) dwt_0_function (/32)
Detected halt on bcm20739.CPU
```

Figure 5-7. Debug View after Pause Using OpenOCD

5.3 Hardware Debugging from Command Line

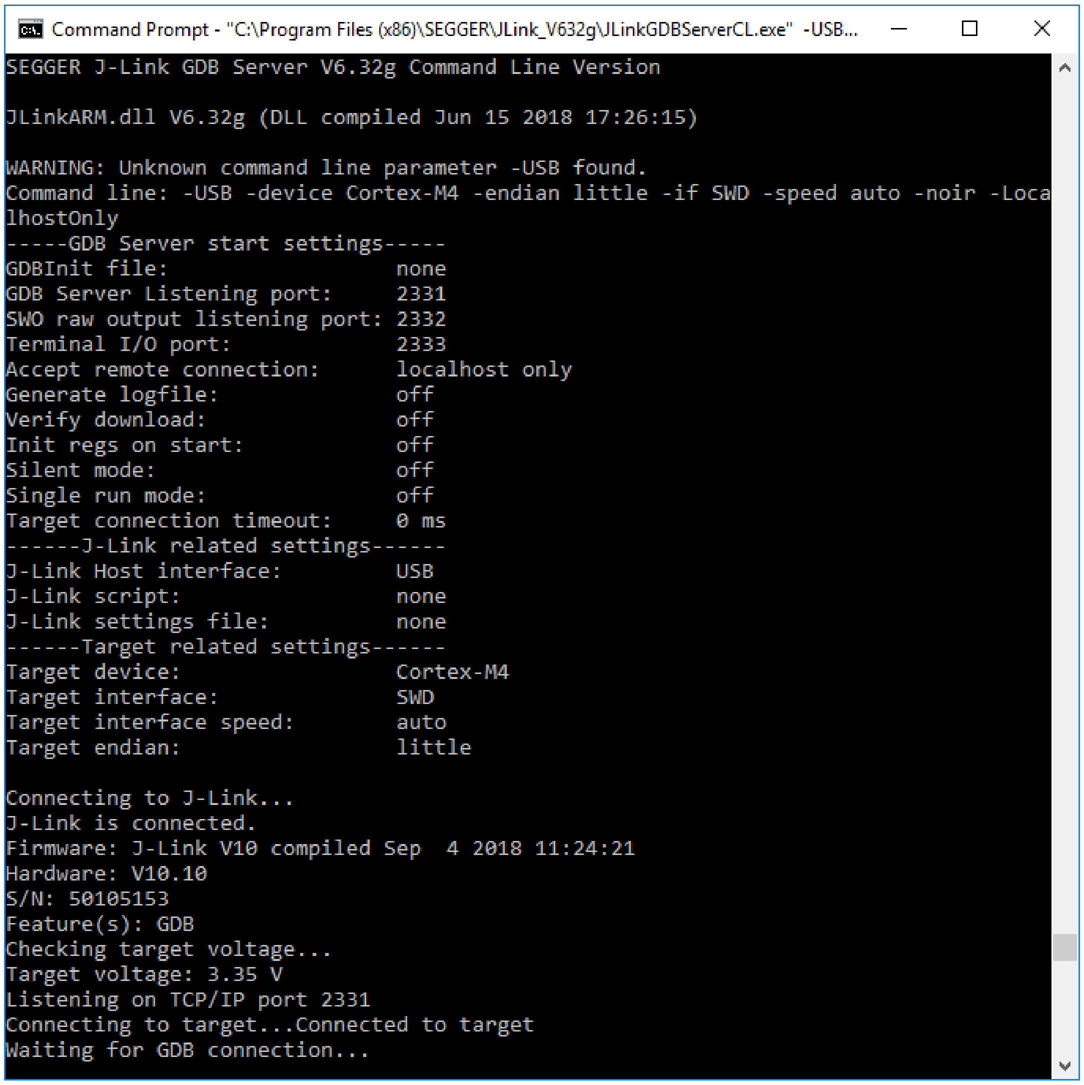
The command line interface can also be used to build applications, download them, launch GDB servers, and run GDB with symbols to perform hardware debugging.

1. Do the following to launch the GDB Server from the command line.

   OpenOCD GDB Server command line from <ModusToolbox install>/tools/openocd-2.1/bin: “.\openocd -s ..\scripts -s ..\..\..\libraries\bt_sdk-1.1\components\BT-SDK\<platform path>\platforms -f <device name>_openocd.cfg

   Or enter the following SEGGER GDB Server command line:

   ```
   C:\Program Files (x86)\SEGGER\JLink_V632g\JLinkGDBServerCL.exe" -USB -device Cortex-M4 -endian little -if SWD -speed auto -noir -LocalhostOnly
   ```


Command Prompt - "C:\Program Files (x86)\SEGGER\JLink_V632g\JLinkGDBServerCL.exe" -USB...

```
SEGGER J-Link GDB Server V6.32g Command Line Version

JLinkARM.dll V6.32g (DLL compiled Jun 15 2018 17:26:15)

WARNING: Unknown command line parameter -USB found.
Command line: -USB -device Cortex-M4 -endian little -if SWD -speed auto -noir -Loca
lhostOnly
-----GDB Server start settings-----
GDBInit file:                  none
GDB Server Listening port:     2331
SWO raw output listening port: 2332
Terminal I/O port:             2333
Accept remote connection:      localhost only
Generate logfile:              off
Verify download:               off
Init regs on start:            off
Silent mode:                   off
Single run mode:               off
Target connection timeout:     0 ms
------J-Link related settings------
J-Link Host interface:         USB
J-Link script:                 none
J-Link settings file:          none
------Target related settings------
Target device:                 Cortex-M4
Target interface:              SWD
Target interface speed:        auto
Target endian:                 little

Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V10 compiled Sep  4 2018 11:24:21
Hardware: V10.10
S/N: 50105153
Feature(s): GDB
Checking target voltage...
Target voltage: 3.35 V
Listening on TCP/IP port 2331
Connecting to target...Connected to target
Waiting for GDB connection...
```

2. Run GNU GDB command-line application, *<ModusToolbox install>/tools/ gcc-7.2.1-1.0/bin/arm-none-eabi-gdb.exe*.

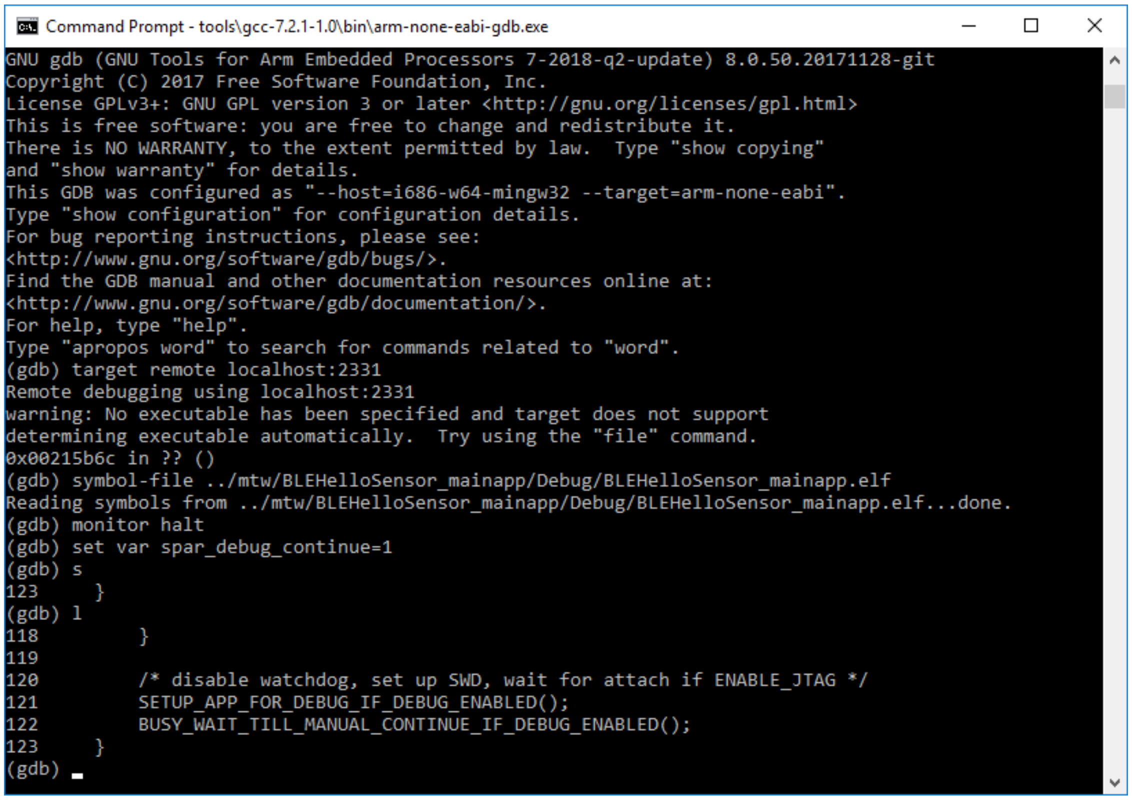
   GNU gdb is an interactive command-line application.

3. Connect gdb with the gdb proper server port with the command `target remote localhost:<port>`.

   Replace <port> with 3333 when using OpenOCD defaults, or 2331 for J-Link.

4. Load the symbol file for the application with *symbol-file <application *.elf>*. Use `monitor halt` to halt the embedded application and `l` (as in *list*) to list the source code at that location.

To break out of the `BUSY_WAIT_TILL_MANUAL_CONTINUE_IF_DEBUG_ENABLED()` loop, set the variable `spar_debug_continue` to non-zero with `set var spar_debug_continue=1`.



Command Prompt - tools\gcc-7.2.1-1.0\bin\arm-none-eabi-gdb.exe

```
GNU gdb (GNU Tools for Arm Embedded Processors 7-2018-q2-update) 8.0.50.20171128-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0x00215b6c in ?? ()
(gdb) symbol-file ../mtw/BLEHelloSensor_mainapp/Debug/BLEHelloSensor_mainapp.elf
Reading symbols from ../mtw/BLEHelloSensor_mainapp/Debug/BLEHelloSensor_mainapp.elf...done.
(gdb) monitor halt
(gdb) set var spar_debug_continue=1
(gdb) s
123     }
(gdb) l
118         }
119
120         /* disable watchdog, set up SWD, wait for attach if ENABLE_JTAG */
121         SETUP_APP_FOR_DEBUG_IF_DEBUG_ENABLED();
122         BUSY_WAIT_TILL_MANUAL_CONTINUE_IF_DEBUG_ENABLED();
123     }
(gdb) _
```

6 Troubleshooting

Here are some issues faced during hardware debugging sessions:

- Failure to connect after “Build + Program + Debug”. When the hardware and software are properly installed, there can be instances of connection failure between the SWD hardware interface and the SWD probe GDB server. Usually a second attempt with “Debug Attach” will be successful. The problem is due to timing between the launching of GDB server and enabling of the SWD hardware.
- When single-stepping through sources, progress is stopped, and a message is displayed like "Break at address "0x7ecba"with no debug information available, or outside of program code”. This is typical when stepping into a library or ROM code where no debug symbols are available. The way out is to select the function further up the call stack and work with a subsequent breakpoint at that source code level.
- If the GDB server fails to launch, look for and close any other instances of the GDB client application that may have been left running (*arm-none-eabi-gdb.exe*).

Document History

Document Title: Hardware Debugging For CYW207xx And CYW208xx

Document Number: 002-20504

Revision	ECN	Submission Date	Description of Change
**	5861331	08/23/2017	Initial release
*A	6373682	11/02/2018	Updated for ModusToolbox
*B	6486075	02/15/2019	Updated title Changed ENABLE_JTAG setting to ENABLE_DEBUG Added references to Linux and Mac Added description for CYW208xx Added troubleshooting section
*C	6554890	04/23/2019	Removed Associated Part Family
*D	6592701	06/11/2019	Updates throughout the document

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709