SimpleLink™ Wi-Fi® and Internet of Things CC3220

Programmer's Guide

Literature Number: SWRU464
February 2017

Contents

List of Figures

List of Tables

Introduction

The CC3220 device is part of the SimpleLink™ microcontroller (MCU) platform, which consists of Wi-Fi®, *Bluetooth*® low energy, Sub-1 GHz and host MCUs, which all share a common, easy-to-use development environment with a single core software development kit (SDK) and rich tool set. A one-time integration of the SimpleLink platform enables you to add any combination of the portfolio's devices into your design, allowing 100 percent code reuse when your design requirements change. For more information, visit www.ti.com/simplelink.

The SimpleLink Wi-Fi CC3220x device is a single-chip microcontroller (MCU) with built-in Wi-Fi connectivity, created for the Internet of Things (IoT). The CC3220 device is a wireless MCU that integrates a high-performance ARM® Cortex®-M4 MCU, letting customers develop an entire application with one device. This document introduces the user to the environment setup for the CC3220x device, along with some reference examples from the software development kit (SDK). This document explains both the platform and the framework available to enable further application development.

1.1 Overview

The royalty-free CC3220 embedded Wi-Fi foundation SDK from Texas Instruments™ is a complete software platform for developing Wi-Fi applications. The SDK is based on the CC3220 device, a complete Wi-Fi System-on-Chip (SoC) solution. The CC3220 family of devices comes in the following three variants:

- CC3220: Base variant
- CC3220S: CC3220 with MCU security
- CC3220SF: CC3220S and internal flash

The CC3220 solution combines a 2.4-GHz Wi-Fi PHY/MAC and TCP/IP networking engine with an MCU, up to 256KB of on-chip RAM, 1MB of internal flash (for CC3220SF), and a comprehensive range of peripherals. Figure 1-1 shows a block diagram of the included peripherals.

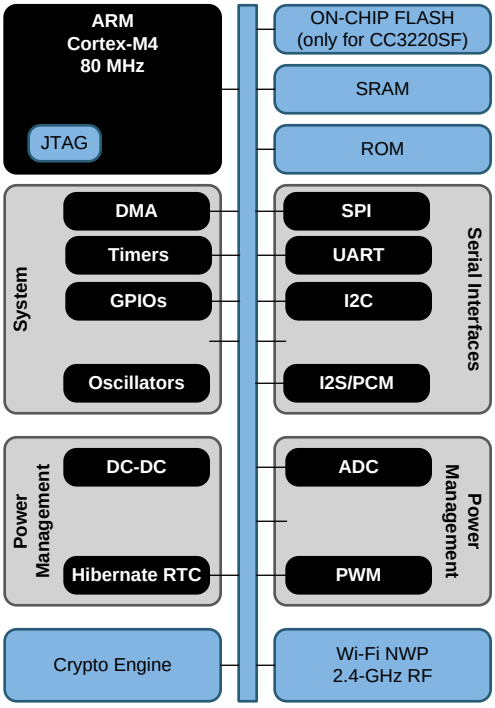

Figure 1-1. Overview of Peripherals

1.2 Software Components

The CC3220 platform includes a user-programmable host and a comprehensive networking solution combined with a Wi-Fi engine, all on one device (see Figure 1-2). The CC3220 SDK is envisaged to provide an easy-to-use framework, hosted in the on-chip microcontroller, to use the WLAN and networking services. The CC3220 device also provides a comprehensive listing of drivers for peripherals interfaced with the MCU, along with the reference code for peripheral use and a few sample applications to highlight use of networking services.

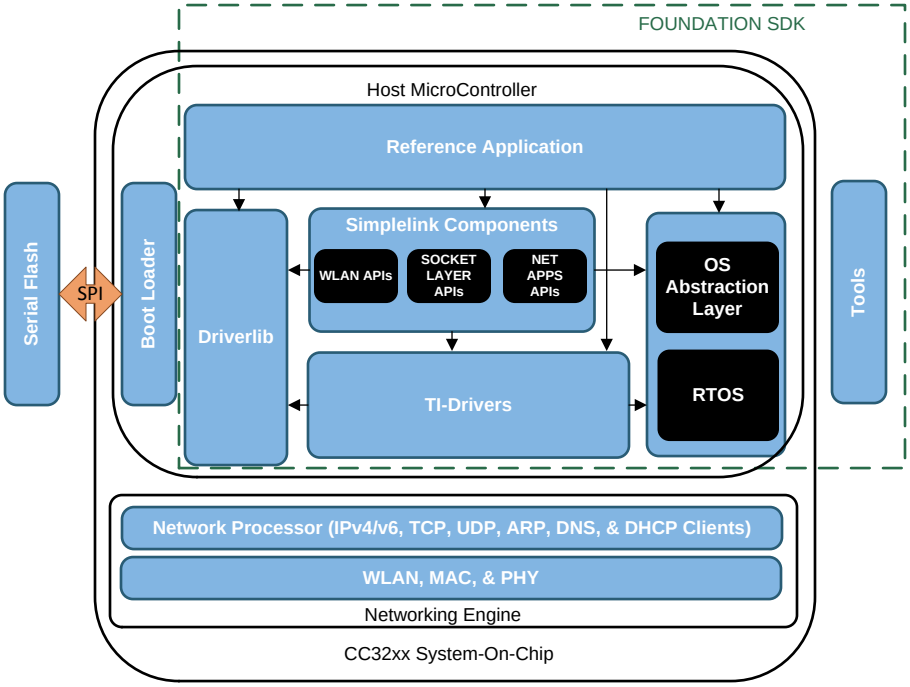

Figure 1-2. SimpleLink™ Wi-Fi® CC3220 Software Components

1.3 CC3220 LaunchPad™ Development Kit Platform

The SimpleLink Wi-Fi CC3220 LaunchPad™ development kit is the default hardware companion for the foundation SDK. There are two variants of this board, which are as follows:

- CC3220S-LAUNCHXL: hosts the CC3220 and CC3220S device variants
- CC3220SF-LAUNCHXL: hosts the CC3220SF device variant

This board hosts any of the CC3220, CC3220S, or CC3220SF devices with an interface suitable for application software development and debugging. The CC3220 LaunchPad development kit also supports the TI BoosterPack™ interface, enabling interfacing with a large and varied collection of peripheral systems.

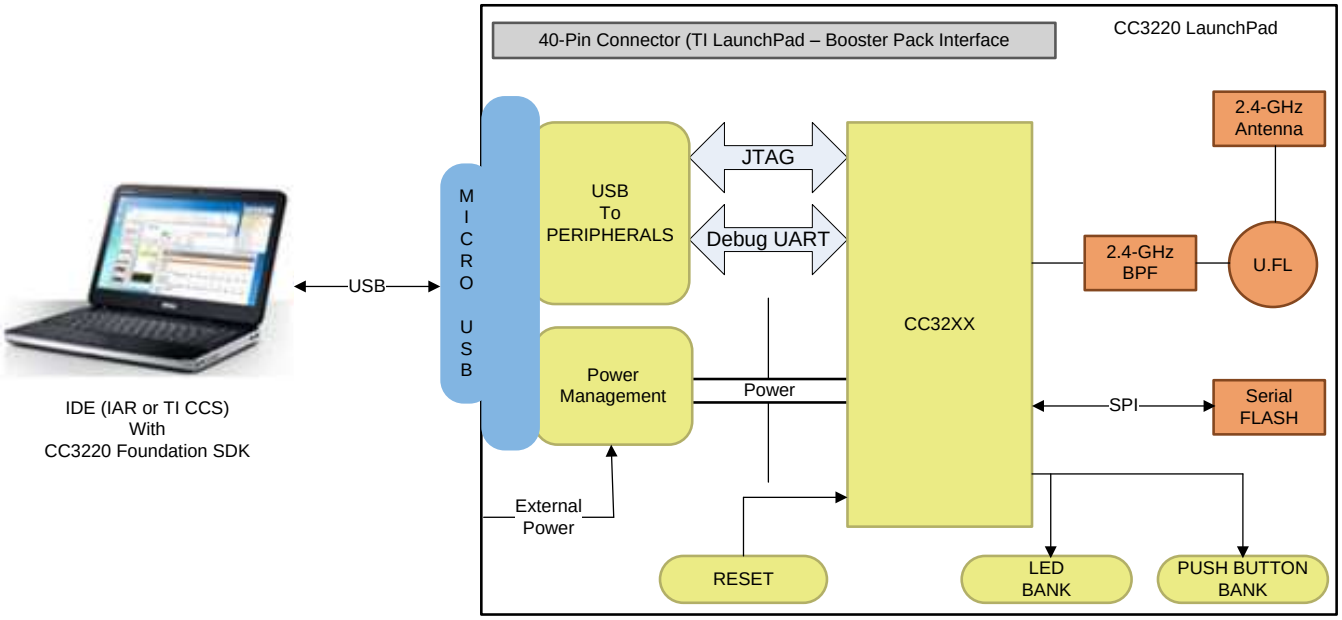

Figure 1-3. CC3220 LaunchPad™ Platform

Foundation SDK – Getting Started

This section includes information on the installation process and the directory structure for CC3220 SDK. To install the necessary tools and environment setup for development, refer to the *CC3220 SimpleLink Wi-Fi and Internet of Things Software Getting Started Guide*.

2.1 Package Components Overview

The top-level structure contains the following components, as shown in Table 2-1.

Table 2-1. Package Components

Directory	Content
docs	• SimpleLink Wi-Fi CC3220 Programmer's Guide • For application development, see *CC3220 SimpleLink Wi-Fi and Internet of Things Software Getting Started Guide* • Documentation for hardware details in Hardware folder • Documentation for SimpleLink host driver in html format under simplelink_api folder • Application notes for all the sample application present in examples directory • Peripheral driver library user's guide • Documentation for netapps libraries • Release Notes
driverlib	• Contains the peripheral driver library source files • The driverlib.a is also provided in ccs, ewarm, and gcc directories
examples	• Contains comprehensive set of examples exercising the peripherals and various capabilities of the networking engine
inc	• Contains the register definition header files
oslib	• Contains the interface file to configure Free-RTOS and TI-RTOS
netapps	• **http**: Contains the HTTP (Hyper Text Transfer Protocol) client and server library • **smtp**: Contains the SMTP (Simple Mail Transfer Protocol) client library • **tftp**: Contains the TFTP (Trivial File Transfer Protocol) client library • **xmpp**: Contains the XMPP (Extensive Messaging and Presence Protocol) client library • **json**: Contains JSON parser library • **mqtt**: Contains the Message Queue Telemetry Transport (MQTT) client and server library
simplelink	• SimpleLink host driver code, which is responsible for communication with the networking engine
tidrivers_simplelink	• Set of interface drivers for various peripherals
osal_tidrivers	• OS abstraction layer for the TI drivers
third_party	• Contains the following third-party components: – FreeRTOS source files
ti_rtos	• Contains the TI-RTOS configuration file and CCS, IAR™, and GCC projects
tools	• **ccs_target_configurations** – XDS110 configuration file for CCS Debugger • **iar_patch** – contains the patch files required for IAR • **ccs_patch** – contains patch files for CCS • **gcc_scripts** – Script files required for the GCC compiler • **certificate_store** – various certificate files

2.2 Prerequisites

Most of the information regarding the required tools, drivers, and environment setup is provided in the *CC3220 SimpleLink Wi-Fi and Internet of Things Software Getting Started Guide*. Table 2-2 lists each with a brief description. For more detail on the prerequisites, refer to the *CC3220 SimpleLink Wi-Fi and Internet of Things Software Getting Started Guide*.

Table 2-2. Prerequisites

Toll	Remarks	Location
Development Environment		
IAR	Refer to the release notes for supported version.	Installation: www.iar.com
Or/and		
CCS	Refer to the release notes for supported version. After the installation, follow the steps in *tools\ccs_target_configuration\readme.txt* and *tools\ccs_patch\readme.txt* to debug.	Installation: http://processors.wiki.ti.com/index.php/Code_Composer_Studio_Beta_Downloads
Latest TI-RTOS in CCSv6.1	Latest TI RTOS should be installed for this SDK.	Refer to Section 5.1.2.1
Or/and		
GCC	To enable CC3220 SDK development on Linux® environment.	Refer to Section 5.1.3.1
Support Tools		
HyperTerminal or TeraTerm	Serial communication tool to communicate over the UART with the CC3220 device.	Hyperterminal TeraTerm
Iperf	Measures TCP and UDP bandwidth performance	
SLImageCreator	Utility for downloading and flashing the applications on the host device	SL-IMAGECREATOR
Drivers Required		
XDS110	XDS110 drivers for debugging, image download, and UART prints on the serial terminal	http://processors.wiki.ti.com/index.php/XDS_Emulation_Software_Package#XDS110_Reset_Download
Service Pack		
CC31xx_CC32xx service pack	Service pack contains the additional network improvements, as well as bug fixes in the form of a patch which is flashed along with the application image.	CC3220 SDK

Foundation SDK – Components

The CC3220 foundation SDK package includes three main building blocks:

- SimpleLink Component Library: This library hosts APIs that serve the connectivity features.
- Peripheral Driver Library: This library hosts APIs to access MCU peripherals.
- TI drivers: Contains easy-to-use interface drivers for various peripherals.

This section also lists the sample and reference applications packaged in the SDK.

3.1 SimpleLink™ Component Library

3.1.1 SimpleLink™ Modular Decomposition

Figure 3-1 shows the modular decomposition.

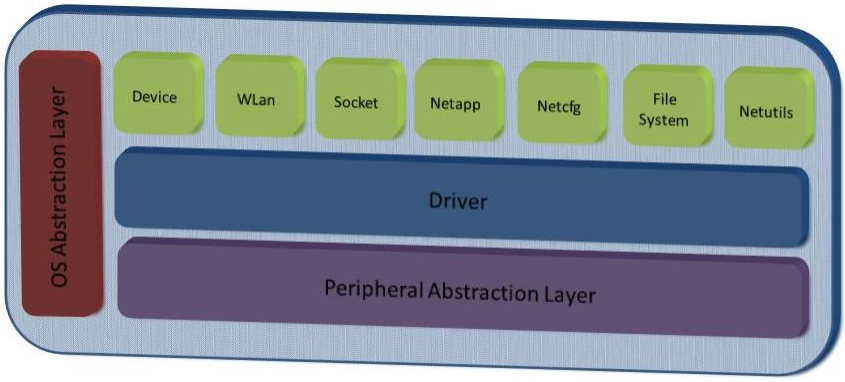

Figure 3-1. Modular Decomposition

TI's SimpleLink framework provides a wide set of capabilities, ranging from basic device management through wireless network configuration, Berkeley Software Distribution (BSD) socket services, and more. For better design granularity, these capabilities are segregated into individual modules. Each module represents different functionality or capability of the SimpleLink framework.

Table 3-1 lists the different components in the SimpleLink framework.

Table 3-1. SimpleLink™ Components

Component	Functionality
device	• Initializes the host • Controls the communication with the network processor
wlan	• Connects to the access point • Scans access points • Removes or adds access point profiles • WLAN security
socket	• UDP/TCP client socket • UDP/TCP server socket • UDP/TCP Rx/Tx

Table 3-1. SimpleLink™ Components (continued)

Component	Functionality
netapp	• DNS resolution • Pings remote device • Address resolution protocol
netcfg	• IP/MAC address configuration
fs	• File system read/write
netutils	• Provides network utility, such as secure content delivery

3.1.2 Using the TI SimpleLink™ Framework

TI's SimpleLink framework provides a rich, yet simple set of APIs. For detailed information on the APIs and their usage, refer to the document “docs\SimpleLink_CC3220_Host_Driver.chm”.

A ready-to-use port of TI SimpleLink framework is available in the CC3220 foundation SDK. The source code is also shared if the developer desires further customization. The following note describes simple possible customization and the associated procedure.

NOTE: All modifications and adjustments to the driver should be made in the user.h header file only. This ensures a smooth transaction to new versions of the driver at the future.

1. Modify the user.h file to include the required configurations and adjustments. The default version is used by the applications present in the SDK.
2. Select the capabilities required for the application: TI built three different predefined sets of capabilities that fit most of the target applications. TI recommends trying and choosing one of these predefined capabilities sets before building a customized set.

   The available sets are:

   - SL_TINY – Provides the best-in-class footprint in terms of code and data consumption
   - SL_SMALL – Compatible to most common networking applications
   - SL_FULL – Provides access to all SimpleLink functionalities

   The default configuration provides all the SimpleLink functionalities.
3. Choose the memory management model. SimpleLink framework supports two memory models:
   - Static (default)
   - Dynamic

   If the dynamic model is chosen, the user must provide the malloc and free routines for the SimpleLink driver. Using the dynamic mode allocates the required resources on sl_Start, and releases these resources on sl_Stop.
4. OS Adaptation. The SimpleLink framework runs on:
   - Non-OS and single-threaded platforms
   - Multithreaded platforms

   If a multithreaded environment is chosen, the user must provide a few basic adaptation routines to allow the driver to protect access to resources for different threads (locking object), and to allow synchronization between threads (sync objects). The required OS abstraction functions are implemented for FreeRTOS and TI-RTOS. For the list of APIs that must be adapted to support other RTOS, refer to *<sdk_installion>\cc3220-sdk\oslib\osi.h*.

In addition, the framework supports running in an exclusive context. If using this mode, a spawn method should be supplied by the application that enables running functions in a temporary context.

5. Set asynchronous event handlers routines. The SimpleLink framework generates asynchronous events in several situations. These asynchronous events could be masked. To catch these events, the application must provide handler routines. If the application does not provide a handler routine and the event is received, the driver will drop this event.

3.1.3 Project Configurations

3.1.3.1 IAR

Choose the configuration options from the *Project->Edit configurations->os/nonos/os_debug/nonos_debug* menu, or as shown in Figure 3-2.

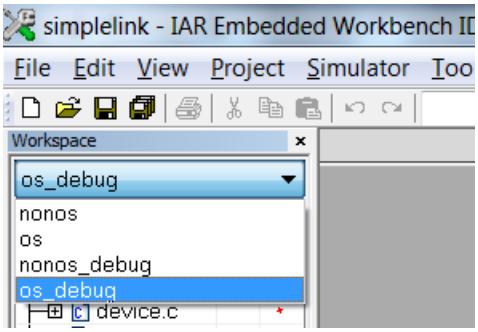

Figure 3-2. SimpleLink™ Configuration Switch (IAR)

3.1.3.2 CCS

Choose the configuration options from the *Project->Build Configuration->Set active* menu, or as shown in Figure 3-3.

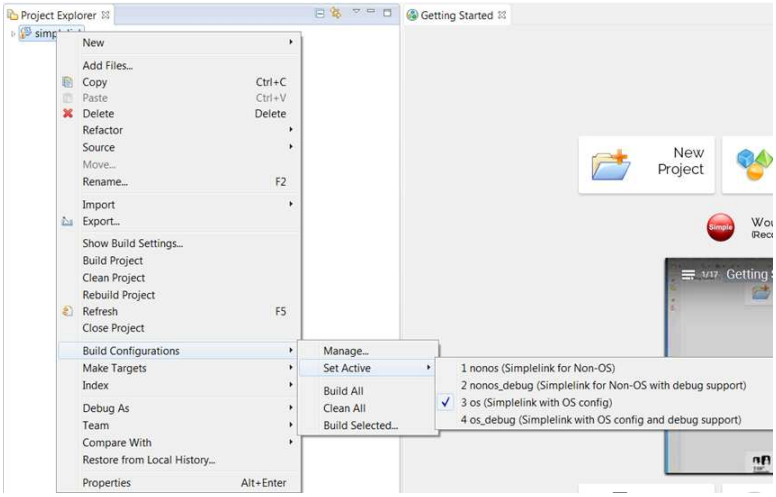

Figure 3-3. SimpleLink™ Device Configuration Switch (CCS)

3.1.3.3 GCC

While compiling the SimpleLink library, specifying the “target = nonos/nonos_debug/os_debug” will compile the SimpleLink for nonos/nonos_debug/os_debug, respectively. The default case compiles the OS configuration, as shown in Figure 3-4.

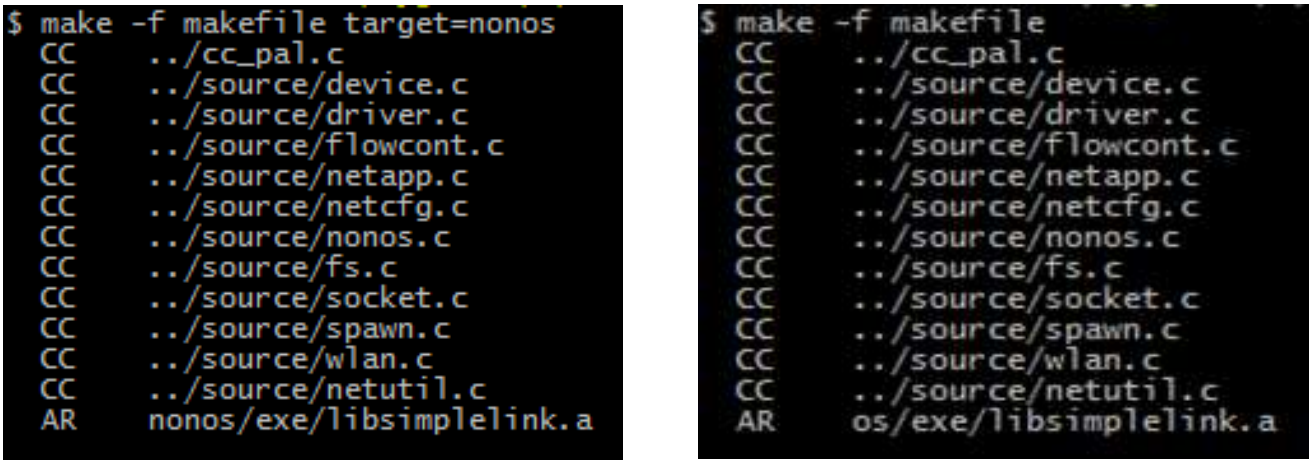

Figure 3-4. SimpleLink™ Device Configuration Switch (GCC)

3.1.4 Development Versus Deployment Scenario

To support the reloading of application images using the debugger, without having to reset the device (LaunchPad development kit), the implementation in the cc_pal.c (simplelink) file requires a NwpPowerOnPreamble routine to stop networking services, and a delay in the NwpPowerOn() function must be introduced for proper operation.

This routine is required because a core reset from the debugger only resets when the APPs processor and the networking engine are still active. Thus, on the next debug session, the networking engine must be stopped and restarted. This results in additional delays and greater overall current consumption. These additional steps and delay are required only for debugging purposes, and they should not be a part of the deployment applications.

For ease of use, the CC3220 SDK package provides separate configurations of the SimpleLink library for the development (debug) and deployment scenarios. Use one of the following two configurations for the deployment scenario (this does not include the additional steps and delay), as per the use-case.

- nonos – SimpleLink for non-OS environment
- os – SimpleLink for OS environment

All of these configurations are prebuilt in the SDK package, along with their generated output library. By default, all of the networking examples link to one of these configurations. The corresponding debug configurations are also present as part of the project.

- nonos_debug - SimpleLink for non-OS environment, with debug support
- os_debug – SimpleLink for OS environment, with debug support

These configurations are not prebuilt; the user must build them first before linking them to their application. This results in overall greater current consumption. Link the networking example to the debug configurations of the SimpleLink library while debugging to enhance the user experience.

Follow these steps to link the application to the debug configuration.

3.1.4.1 Relinking to the SimpleLink™ Library in CCS

1. Compile the relevant debug configuration for the SimpleLink library.
2. Right-click on the application, and navigate to Properties>ARM Linker>File Search Path.
3. Edit the SimpleLink library path, as shown in Figure 3-5 and Figure 3-6.

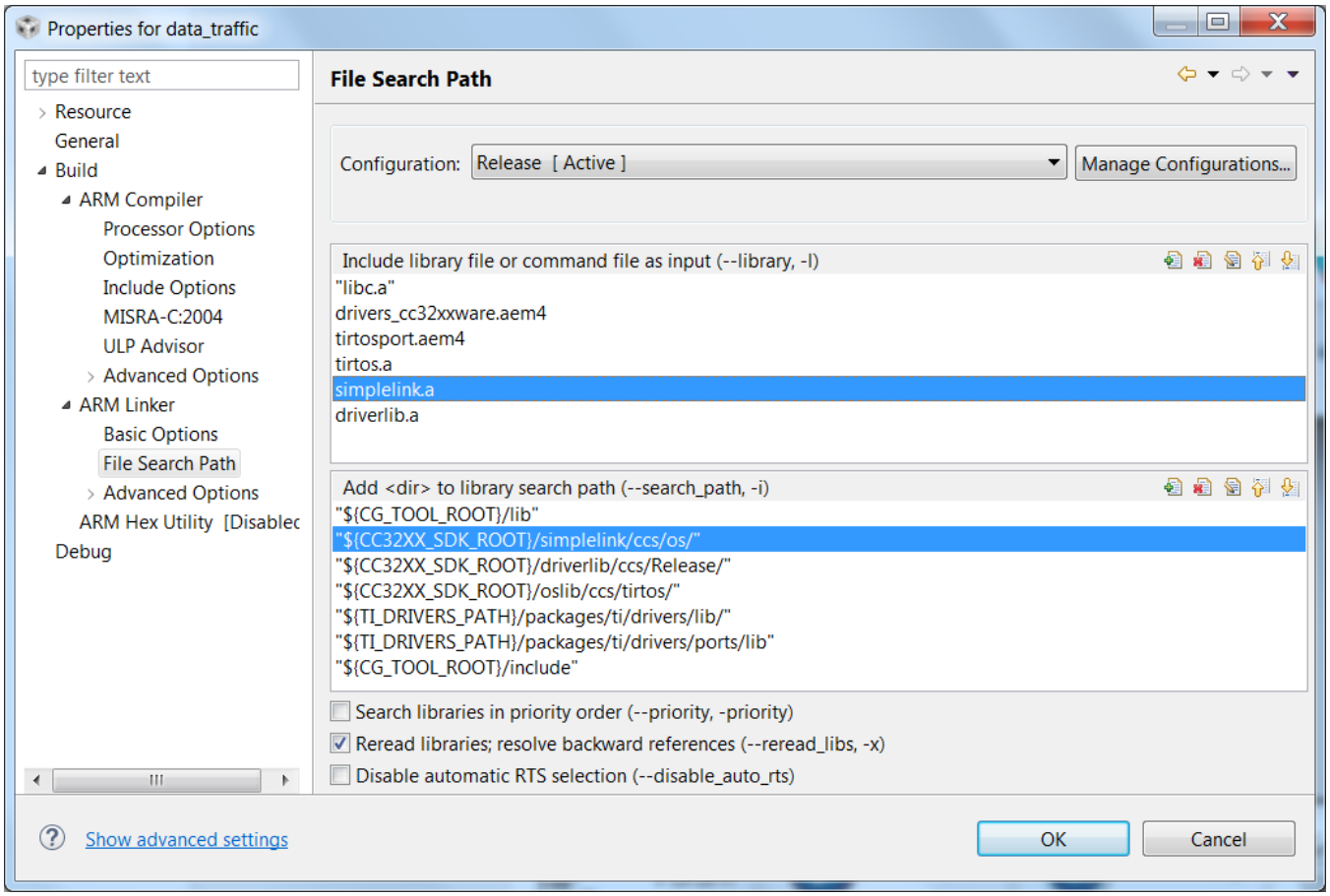

Figure 3-5. Relinking SimpleLink™ Library (CCS) (1 of 2)

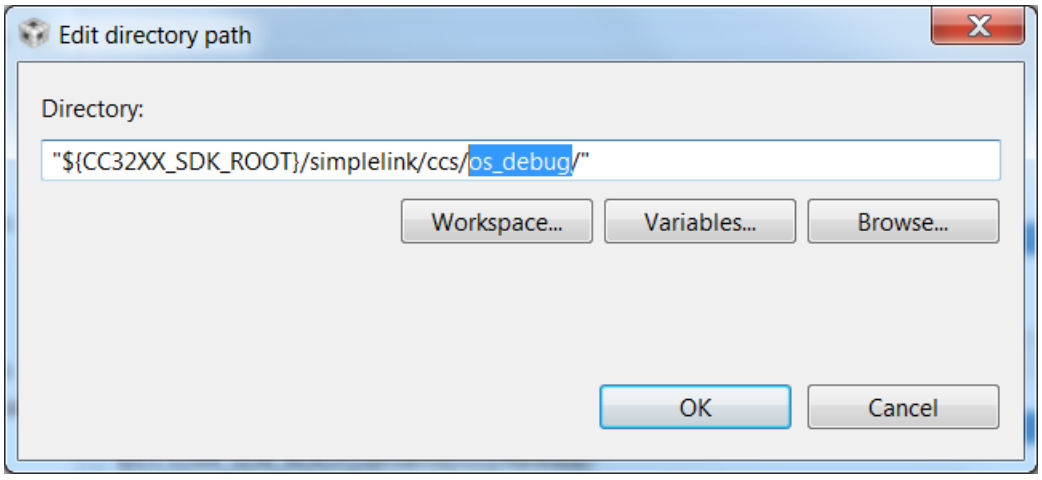

Figure 3-6. Relinking SimpleLink™ Library (CCS) (2 of 2)

3.1.4.2 Relinking to the SimpleLink™ Library in IAR

1. Compile the relevant debug configuration for the SimpleLink library.
2. Right-click on the application, and navigate to Options>Linker>Library.
3. Edit the SimpleLink library path, as shown in Figure 3-7 and Figure 3-8.

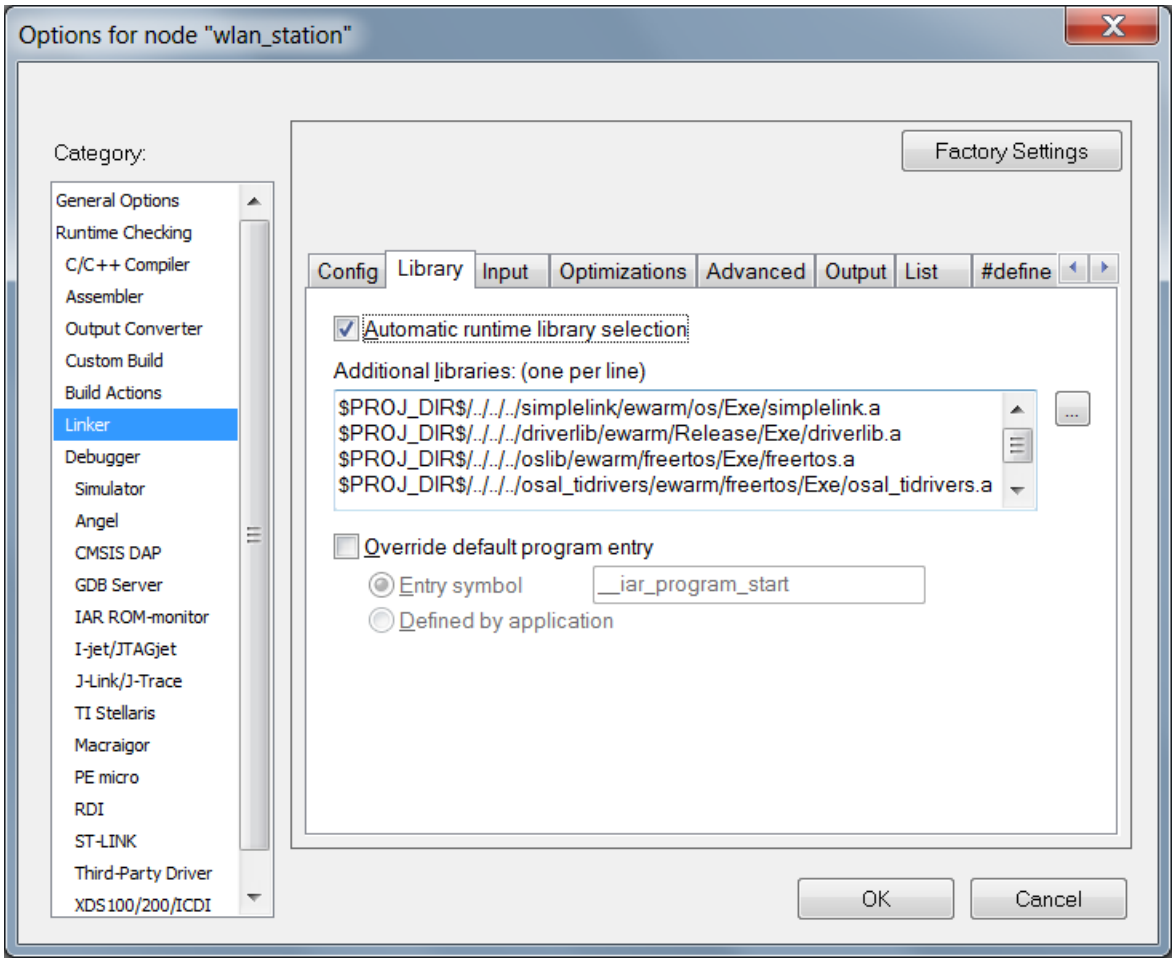

Figure 3-7. Relinking SimpleLink™ Library (IAR) (1 of 2)

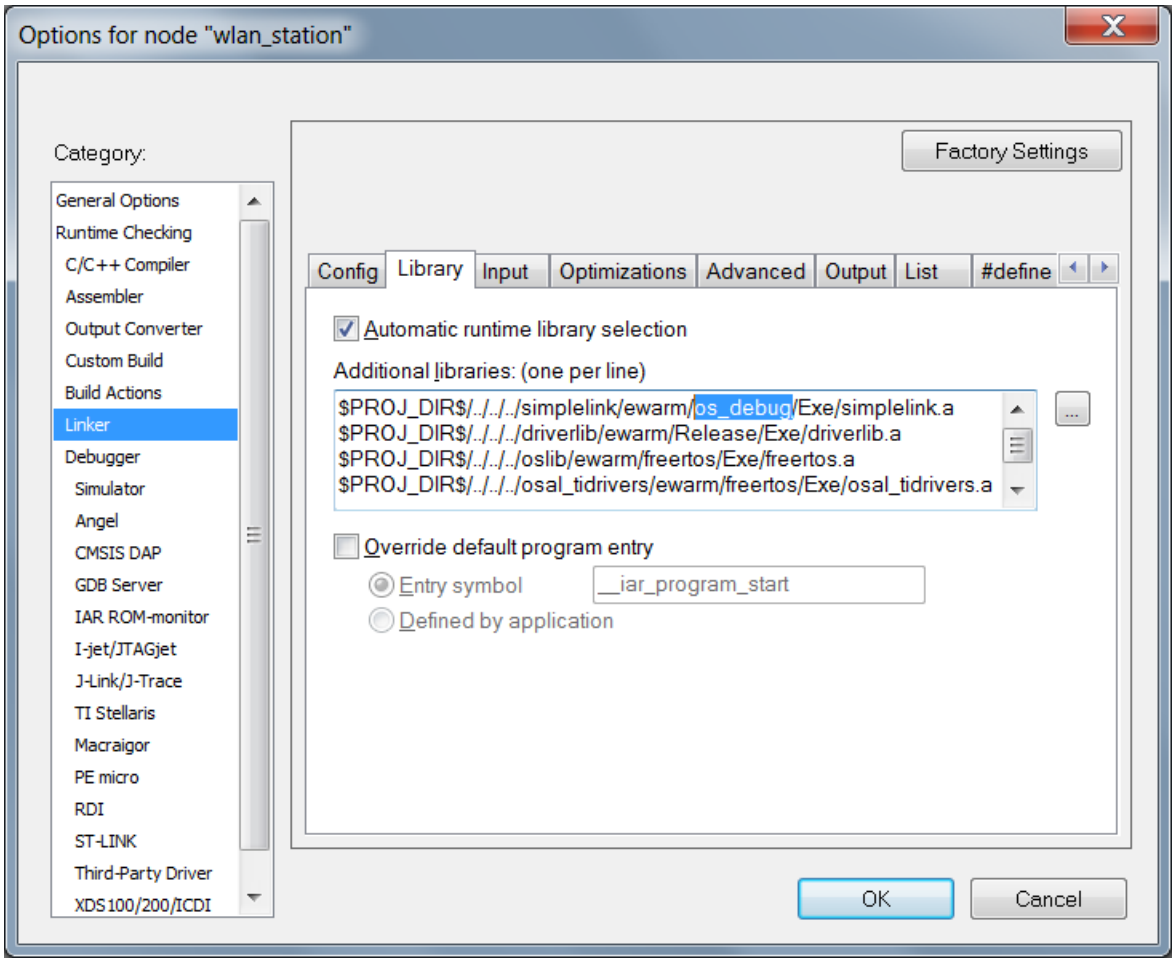

Figure 3-8. Relinking SimpleLink™ Library (IAR) (2 of 2)

3.1.4.3 Relinking to the SimpleLink™ Library in GCC

1. Compile the relevant debug configuration for the SimpleLink library.
2. Open the application Makefile, and navigate to the linking portion.
3. Change the linking code, as shown in Figure 3-9 and Figure 3-10.

```
#
# Rules for building the wlan_station example.
#
${BINDIR}/wlan_station.axf: ${OBJDIR}/main.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/pinmux.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/cc_launchpad.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/uart_if.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/timer_if.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/startup_${COMPILER}.o
${BINDIR}/wlan_station.axf: ${ROOT}/simplelink/${COMPILER}/os/exe/libsimplelink.a
${BINDIR}/wlan_station.axf: ${ROOT}/driverlib/${COMPILER}/${BINDIR}/libdriver.a
${BINDIR}/wlan_station.axf: ${ROOT}/oslib/${COMPILER}/freertos/exe/libfreertos.a
${BINDIR}/wlan_station.axf: ${ROOT}/osal_tidrivers/${COMPILER}/freertos/exe/libosal_t
${BINDIR}/wlan_station.axf: ${ROOT}/tidrivers_cc32xx/packages/ti/drivers/lib/drivers_
SCATTERgcc_wlan_station=${ROOT}/example/common/CC3220.ld
ENTRY_wlan_station=ResetISR
```

Figure 3-9. Relinking SimpleLink™ Library (GCC) (1 of 2)

```
#
# Rules for building the wlan_station example.
#
${BINDIR}/wlan_station.axf: ${OBJDIR}/main.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/pinmux.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/cc_launchpad.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/uart_if.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/timer_if.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/startup_${COMPILER}.o
${BINDIR}/wlan_station.axf: ${ROOT}/simplelink/${COMPILER}/os/exe/libsimplelink_debug.a
${BINDIR}/wlan_station.axf: ${ROOT}/driverlib/${COMPILER}/${BINDIR}/libdriver.a
${BINDIR}/wlan_station.axf: ${ROOT}/oslib/${COMPILER}/freertos/exe/libfreertos.a
${BINDIR}/wlan_station.axf: ${ROOT}/osal_tidrivers/${COMPILER}/freertos/exe/libosal_tidr
${BINDIR}/wlan_station.axf: ${ROOT}/tidrivers_cc32xx/packages/ti/drivers/lib/drivers_cc3
SCATTERgcc_wlan_station=${ROOT}/example/common/CC3220.ld
ENTRY_wlan_station=ResetISR
```

Figure 3-10. Relinking SimpleLink™ Library (GCC) (2 of 2)

3.2 Peripheral Driver Library

The CC3220 ROM contains the peripheral driver library (driverlib) and the bootloader. The ROM driverlib can be used by applications to reduce their flash and SRAM footprint, allowing the flash (or RAM) to be used for other purposes (such as additional features in the application).

The driverlib supports APIs for the modules listed below:

- ADC_Analog_to_Digital_Converter
- AES_Advanced_Encryption_Standard
- Camera
- CRC_Cyclic_Redundancy_Check
- DES_Data_Encryption_Standard
- Flash
- GPIO_General_Purpose_InputOutput
- I2C
- I2S
- Interrupt
- PinPRCM_Power_Reset_Clock_Module
- Secure_Digital_Host
- SHA_Secure_Hash_Algorithm
- SPI_Serial_Peripheral_Interface
- Systick
- GPT_General_Purpose_Timer
- UART
- UDMA_Micro_Direct_Memory_Access
- Utils
- WDT_Watchdog_Timer_

For detailed information on the APIs and their use, refer to the doxyen inside the SDK: *C:/ti/simplelink_cc32xx_sdk_1_02_02_00/docs/tidrivers/doxygen/html/index.html*.

3.3 TI Drivers

TI drivers are for a number of peripherals and have a unified interface across the TI's family of MCUs. These driver interfaces are built on top of the peripheral driver library and optionally use real-time operating system (RTOS). All the drivers are power-aware, and power-management framework library is included as part of this package. It also contains the source and library for FAT file system. All this content is encapsulated in the form of prebuilt libraries. The following are the libraries used in the SDK:

- CCS:
  <cc3220-sdk>\tidrivers_cc32xx\packages\ti\drivers\lib\drivers_cc32xxware.aem4
  <cc3220-sdk>\tidrivers_cc32xx\packages\ti\drivers\lib\power_cc32xx_tirtos.aem4
  <cc3220-sdk>\tidrivers_cc32xx\packages\ti\drivers\ports\lib\tirtosport.aem4
  <cc3220-sdk>\tidrivers_cc32xx\packages\ti\mw\fatfs\lib\release\ti.mw.fatfs.aem4
- IAR:
  <cc3220-sdk>\tidrivers_cc32xx\packages\ti\drivers\lib\drivers_cc32xxware.arm4
  <cc3220-sdk>\tidrivers_cc32xx\packages\ti\drivers\lib\power_cc32xx_tirtos.arm4
  <cc3220-sdk>\tidrivers_cc32xx\packages\ti\drivers\ports\lib\tirtosport.arm4
  <cc3220-sdk>\tidrivers_cc32xx\packages\ti\mw\fatfs\lib\release\ti.mw.fatfs.arm4
- GCC:
  <cc3220-sdk>\tidrivers_cc32xx\packages\ti\drivers\lib\drivers_cc32xxware.am4g
  <cc3220-sdk>\tidrivers_cc32xx\packages\ti\drivers\lib\power_cc32xx_tirtos.am4g
  <cc3220-sdk>\tidrivers_cc32xx\packages\ti\drivers\ports\lib\tirtosport.am4g
  <cc3220-sdk>\tidrivers_cc32xx\packages\ti\mw\fatfs\lib\release\ti.mw.fatfs.am4g

3.3.1 Library Description

drivers_cc32xxware – Contains the device drivers for various peripherals. All of these drivers are power-aware. Below is the list of peripherals supported by TI-Drivers for the CC3220 device:

- Camera
- GPIO
- I2C
- I2S
- Power
- PWM Timer
- SDHost
- SDSPI
- SPI
- UART
- WatchDog

power_cc32xx_tirtos – Power-management framework for TI-RTOS environment. The corresponding library for FreeRTOS and non-OS environment can be found in osal_tidrivers directory.

tirtosport – Porting layer (required by the drivers) for the TI-RTOS. The corresponding library for FreeRTOS and non-OS environment can be found in osal_tidrivers directory.

ti.mw.fatfs – FAT file system library

For more information on these drivers, refer to *<cc3220-sdk>\tidrivers_cc32xx\docs\tidriversAPIs.html*.

3.3.2 Rebuilding TI Drivers

The process to rebuild these drivers is different from other libraries, as there is no CCS or IAR project provided. The user can rebuild these drivers using the drivers.mak (found in *<cc3220-sdk>\tidrivers_cc32xx*). drivers.mak is a general template, with options for building these drivers for other simplelink products. The user must only specify the CC3220-related configuration, and rebuild these drivers using the following command:

From a Windows command prompt

```
c:\ti\xdctools_<version>\gmake.exe –f drivers.mak
```

From Cygwin:

```
c:/ti/xdctools_<veersion>/gmake.exe –f drivers.mak
```

NOTE: By default, the libraries are compiled to use the driverlib APIs from ROM. If the user doesn't want to use the ROM APIs, they can remove the "-DUSE_CC3220_ROM_DRV_API" flag from the *<cc3220-sdk>tidrivers_cc32xx\packages\ti\drivers\package.bld* and recompile.

A sample configuration for the drivers.mak file is given below:

NOTE: The user must only adjust the path variables to match their host environment; the other settings should be left as they are in the default makefile.

```
#
#  ======== drivers.mak ========
#

#
# Update the variables below with paths to point to compilers & products
# installed locally.  If a particular compiler or product will not be used it
# must remain undefined.
#
# NOTE: Paths must not contain spaces.  If using a Windows PC the following
# command can be used to determine a directory's DOS path:
#
#     > for %I in ("path") do echo %~sI
#
# IAR Workbench example:
#     Path:        c:\Program Files (x86)\IAR Systems\Embedded Workbench 7.2
#     DOS Path:    c:\PROGRA~2\IARSYS~1\EMBEDD~1.2
#

ti.targets.arm.elf.M4 ?= C:/TI/ccsv6/tools/compiler/ti-cgt-arm_5.2.7
gnu.targets.arm.M4 ?= C:/cygwin/
iar.targets.arm.M4 ?= C:/PROGRA~2/IARSYS~1/EMBEDD~1.3/arm

XDC_INSTALL_DIR ?= C:/TI/xdctools_3_32_00_06_core
BIOS_INSTALL_DIR ?= C:/ti/tirtos_cc32xx_2_16_01_14/products/bios_6_45_02_31
BIOS_PACKAGES_DIR ?= $(BIOS_INSTALL_DIR)/packages

CC32XXWARE ?= C:/CC3220SDK/cc3220-sdk/

#
# Set XDCARGS to some of the variables above.  XDCARGS are passed
# to the XDC build engine... which will load drivers.bld... which will
# extract these variables and use them to determine what to build and which
# toolchains to use.
#
# Note that not all of these variables need to be set to something valid.
# Unfortunately, since these vars are unconditionally assigned, your build line
# will be longer and more noisy than necessary (e.g., it will include CC_V5T
# assignment even if you're just building for C64P).
#
```

```
# Some background is here:
#     http://rtsc.eclipse.org/docs-tip/Command_-_xdc#Environment_Variables
XDCARGS= \
    ti.targets.arm.elf.M4=$(ti.targets.arm.elf.M4) \
    gnu.targets.arm.M4=$(gnu.targets.arm.M4) \
    iar.targets.arm.M4=$(iar.targets.arm.M4) \
    CC32XXWARE=$(CC32XXWARE) \

#
# Set XDCPATH to contain necessary repositories.
#
XDCPATH = $(BIOS_PACKAGES_DIR);
ifneq ("$(NDK_INSTALL_DIR)", "")
XDCPATH = $(BIOS_PACKAGES_DIR);$(NDK_PACKAGES_DIR);
endif
export XDCPATH

#
# Set XDCOPTIONS.  Use -v for a verbose build.
#
export XDCOPTIONS=v

#
# Set XDC executable command
# Note that XDCBUILDCFG points to the drivers.bld file which uses
# the arguments specified by XDCARGS
#
XDC = $(XDC_INSTALL_DIR)/xdc XDCBUILDCFG=./drivers.bld

#########################################################
## Shouldnt have to modify anything below this line ##
#########################################################

all:
    @ echo building drivers packages ...
    @ $(XDC) XDCARGS="$(XDCARGS)" all -j $(j) -PR ./packages

release:
    @ echo building drivers release ...
    @ $(XDC) XDCARGS="$(XDCARGS)" release -j $(j) -PR ./packages

clean:
    @
```

Getting Started With CC3220 LaunchPad

Follow these steps to get started with the SimpleLink Wi-Fi CC3220 LaunchPad development kit:

1. Ensure the XDS110 drivers are installed on the PC. Refer to the Getting Started guide to install the XDS110 driver.
2. Connect CC3220 LaunchPad to the PC.
3. When the XDS110 driver is installed, the device displays with one com port (XDS110 Class Application/User UART), as in Figure 4-1.

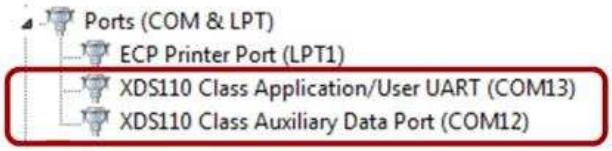

Figure 4-1. Device Manager

To configure the device into SWD/JTAG mode, refer to *cc3220-sdk\docs\hardware\CC3220-LP_User's guide.pdf*.

Foundation SDK – Development Flow

This section describes the typical development flow, using the building blocks hosted in Foundation SDK. The section focuses more on the network aspects of the CC3220 device. For this purpose, a suite of simple Getting Started applications are presented in the SDK. This section starts with a comprehensive description of these applications, along with the build and execute procedure with a few IDEs, and finally with the procedure to burn the application image in the nonvolatile storage.

The SDK contains seven simple network applications to demonstrate the connection and packet-handling functionality. These applications use the SimpleLink APIs to demonstrate the functionality. The source for these applications is modular, and can be referenced or reused by the developer.

Application	Description
Getting started with WLAN Station	Reference application to use the CC3220 device in STA mode
Getting started with WLAN AP	Reference application to use the CC3220 device in AP (access point) mode
Getting started with WLAN Station (NON-OS)	Reference application to use the CC3220 device in STA mode (NON-OS environment)
Getting started with WLAN AP (NON-OS)	Reference application to use the CC3220 device in AP (access point) mode (NON-OS environment)
Data Traffic	Various network capabilities, including UDP and TCP communication
SimpleLink Provisioning	Provisioning using AP provisioning method
File Operations Application	File system for secure and nonsecure files

Details for all these application can be found in the *<cc3220-sdk>\docs\examples* folder of the installation package. These applications use many networking APIs, the details for which are captured in the *<cc3220-sdk>\>\docs\simplelink\SimpleLink_CC3220_Host_Driver.html* document.

5.1 Compilation, Build, and Execution Procedure

Refer to the IAR/CCS help documentation that contains elaborate information on compiling, building, and executing user applications. The following sections highlight the key project options exercised during the development process. ‘Getting started with wlan station’ application is taken up as a reference to demonstrate the development environment. Similar procedures apply to all reference applications, or for new developments. Most steps mentioned here are already performed for all reference applications (present in example/ folder) and captured in the project files.

Key notes:

- The IDEs may need patching for their current versions. Refer to section 2.7.1.1 and 2.7.2.1 of the *CC3220 SimpleLink Wi-Fi and Internet of Things Software Getting Started Guide* before going further.
- The CC3220 SDK contains the prebuilt binaries for all the sample applications based on CC3220SF variant. The user must change the device variant in the project settings and recompile the application to work with other device variants.
- While using the debugger, clean and rebuild the libraries (driverlib, simplelink, oslib) to avoid any source file association problems.
- While creating new project under this SDK, call MAP_PRCMCC3200MCUInit() as the first call in main() for the proper initialization of the device.
- Visit the CC32xx_TI-RTOS page before creating any TI-RTOS-based application.

5.1.1 Development Environment – IAR

This section demonstrates the steps to create a new project in IAR. If the user wants to work with the existing projects in the SDK, refer to the *CC3220 SimpleLink Wi-Fi and Internet of Things Software Getting Started Guide*.

5.1.1.1 Creating a Project

1. Go to *File→New→Workspace*.
2. Go to *Project→Create New Project (Tool Chain = ARM, Project Templates = C)*, and provide a name for the project (the “wlan_station” project name is used for this document).
3. Open *Project Option*, and follow the settings as given in Figure 5-1 and Figure 5-2. Select the correct device variant (CC3220SF by default for all the existing applications in SDK).

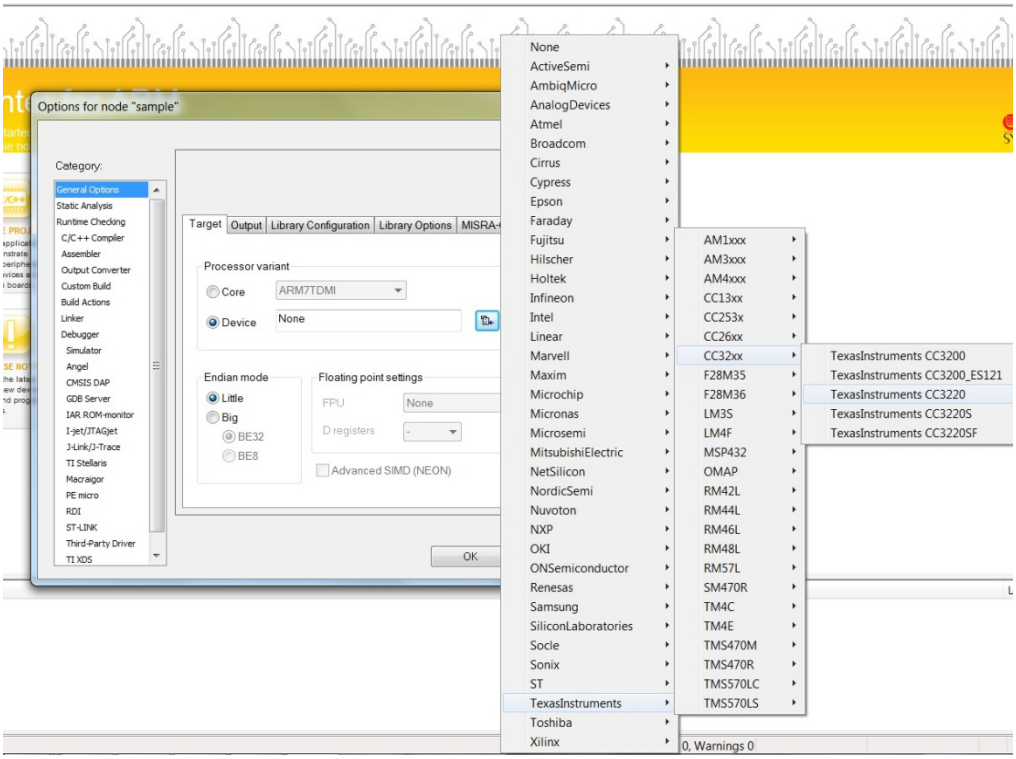

Figure 5-1. IAR Project Settings

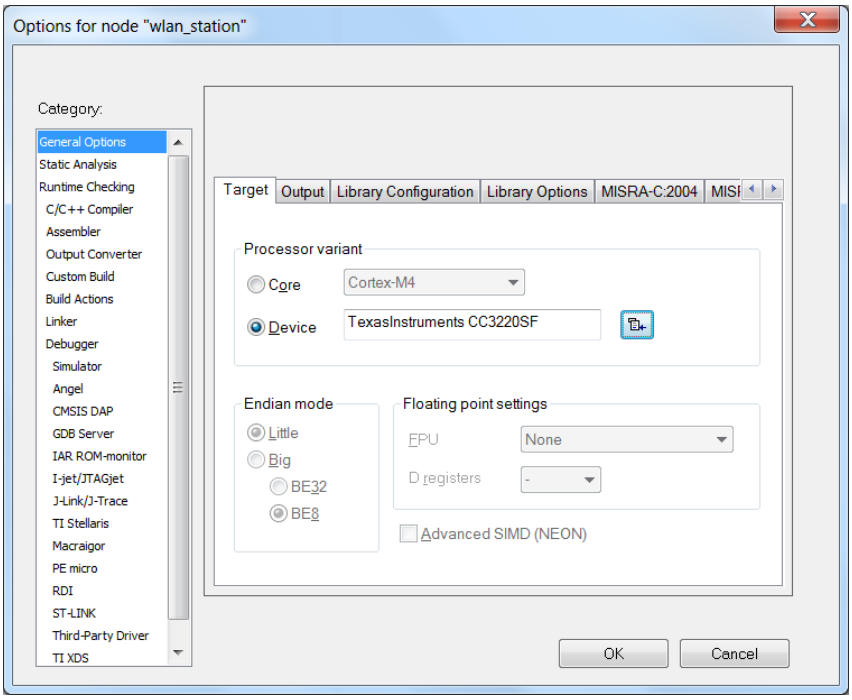

Figure 5-2. Specify the Device Variant (IAR)

5.1.1.2 Adding the Files

Although user applications vary according to the use case, developers are encouraged to refer or reuse the existing application code in their application. The following list is the file structure for most reference applications.

- cc_launchpad.c/.h: Contains the initialization structure and function for peripheral
- board.h: Interface header file
- pinmux.c/.h: Pin-muxing for the peripherals
- main.c: Contains main function and other functions
- startup_*: Compiler-specific start-up file (not required for TI-RTOS-based example)
- Other application-specific files

Consider the following points while developing the application:

- If working with FreeRTOS or non-OS environment, the project requires a compiler-specific start-up file (startup_ccs/startup_ewarm/startup_gcc), which contains the vector table for the processor.
- If working with FreeRTOS, the application must provide the definition for the following hooks (FreeRTOS-specific).
  - void vAssertCalled(const char *pcFile, _u32 ulLine)
  - void vApplicationIdleHook(void)
  - void vApplicationMallocFailedHook()
  - void vApplicationStackOverflowHook(OsiTaskHandle *pxTask, signed char *pcTaskName)

5.1.1.3 Compiling

Figure 5-3 shows the IAR compiler options.

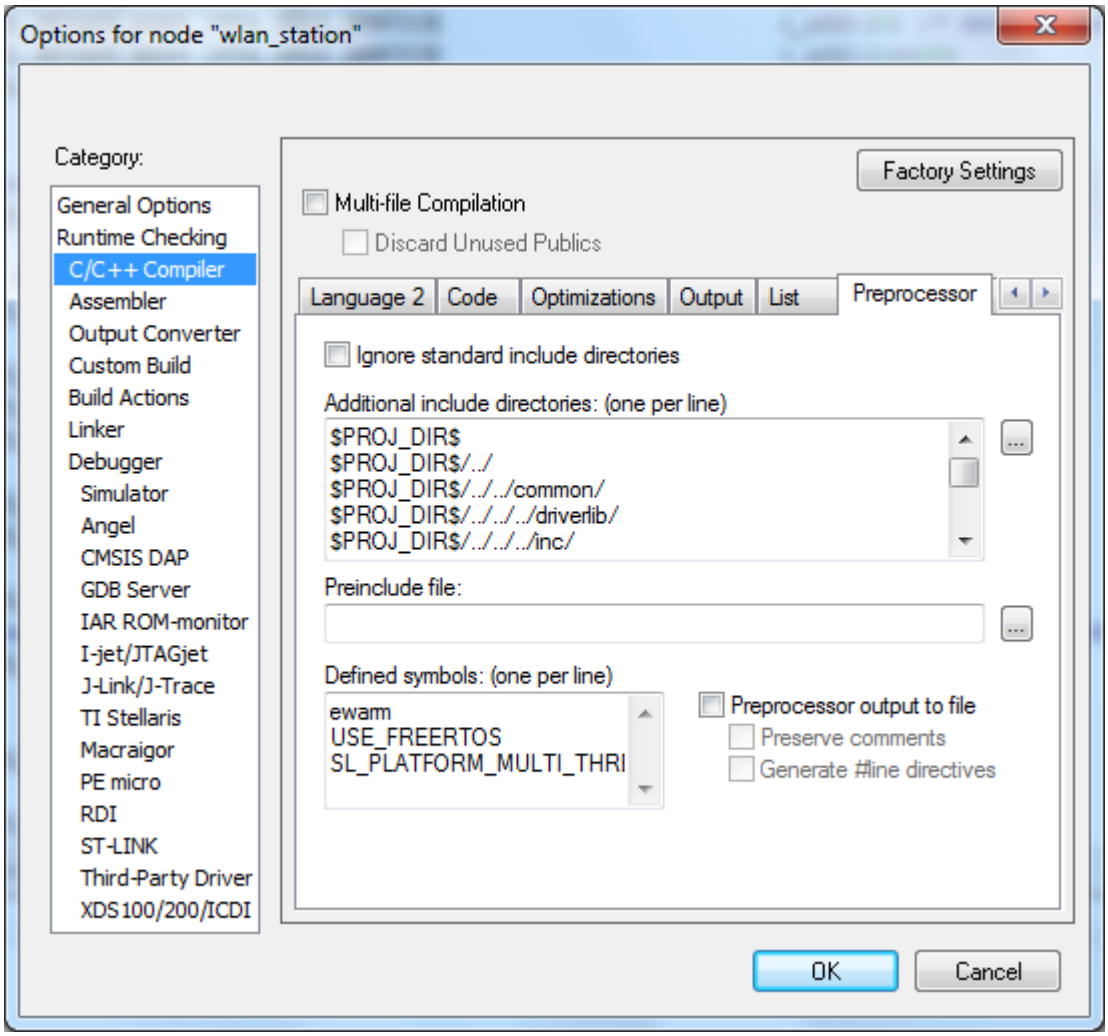

Figure 5-3. IAR Compiler Options

Additional include directories:

- To use driverlib APIs, include *driverlib* and *inc* folder paths.
- To use SimpleLink APIs, include *simplelink*, *simplelink/source*, and *simplelink\include* folder paths.
- To use TI Drivers, include *tidrivers_cc32xx/packages/*.
- To use Free-RTOS APIs, include *oslib*, *third_party/FreeRTOS/source/*, *third_party/FreeRTOS/source/include/*, and *third_party/FreeRTOS/source/portable/IAR/ARM_CM4/* paths.
- Paths for any other library required by the application.

Defined symbols:

- USE_TIRTOS, to use TI-RTOS OS APIs
- USE_FREERTOS if the application uses Free-RTOS OS
- ewarm, to define for IAR-based applications
- SL_PLATFORM_MULTI_THREADED, if the application uses any OS (Free-RTOS/ TI-RTOS)
- Any other application-specific defined symbol

5.1.1.4 Linking

Linker command file: specify the path for the linker command file for IAR(.icf) (see Figure 5-4).

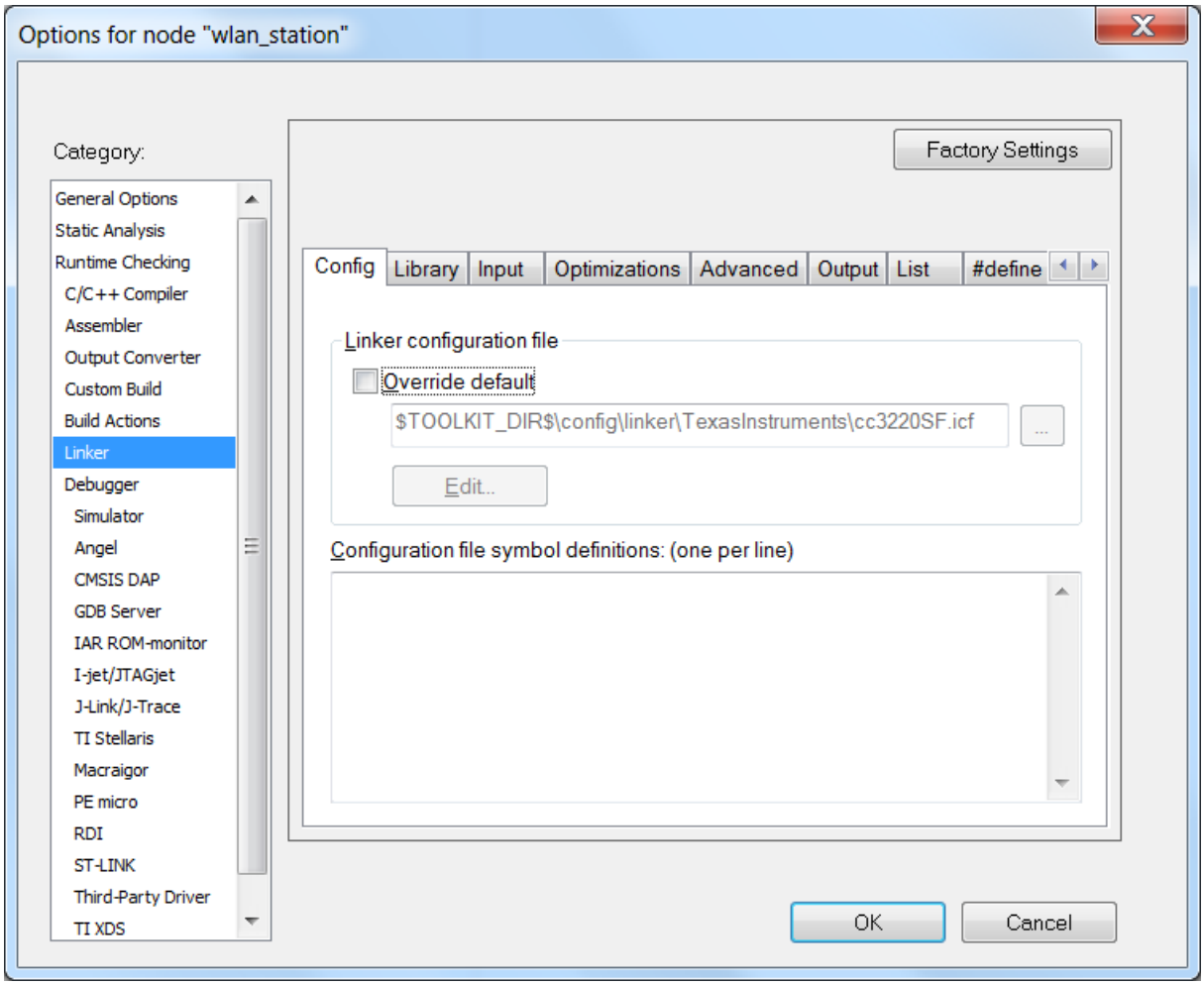

Figure 5-4. IAR Linking Options

Additional libraries: add a library path, in accordance with application needs (driverlib.a, simplelink.a, freertos.a, osal_tidrivers, tidrivers_cc32xx/packages/ti/drivers/lib/drivers_cc32xxware.arm4). See Figure 5-5.

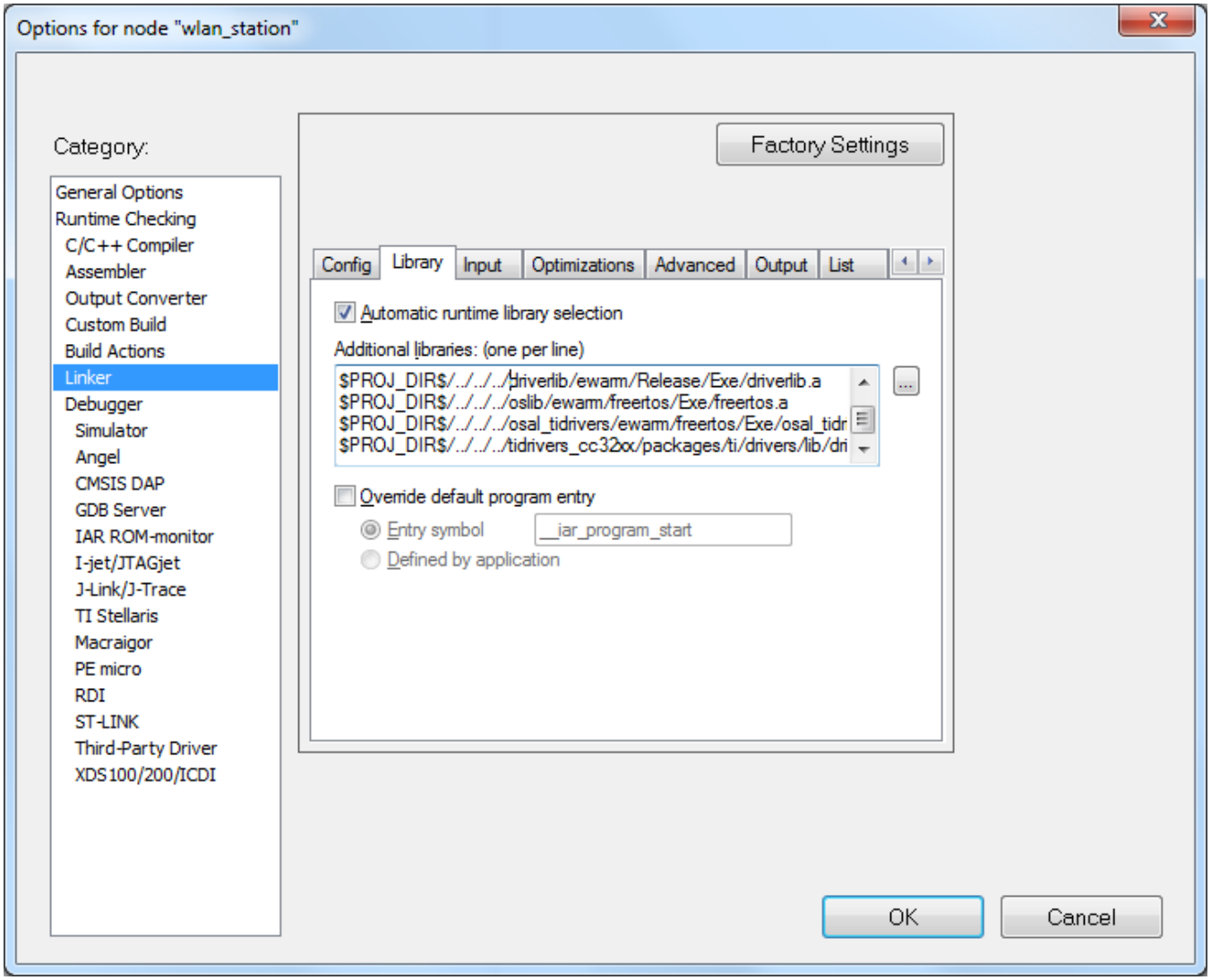

Figure 5-5. IAR Linking Options

5.1.1.5 Generating the Binary (.bin)

To generate additional output, select the output format; a current SDK user can select the binary option to override the .bin path, as shown in Figure 5-6.

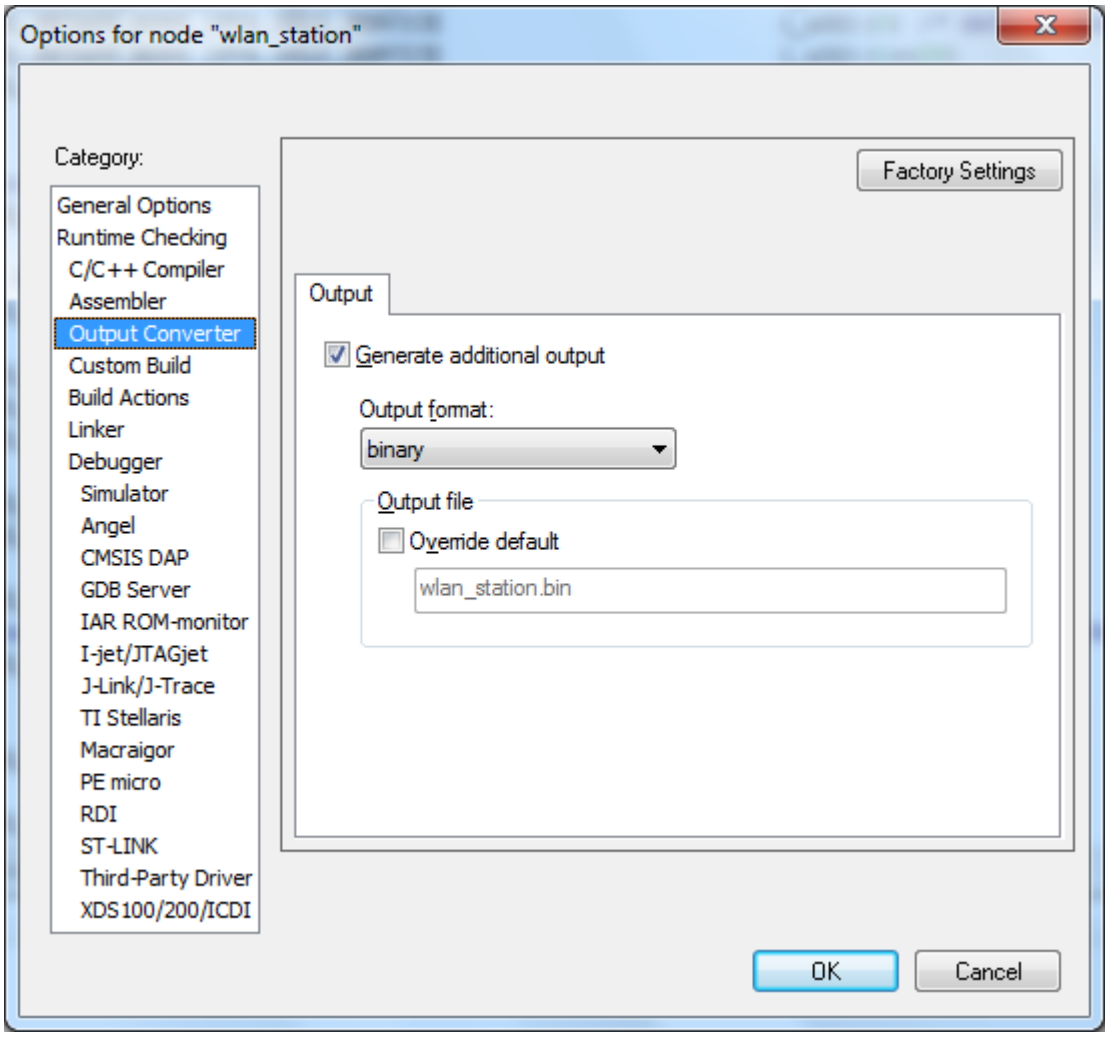

Figure 5-6. IAR Output Conversion

5.1.1.6 Executing

1. If using the JTAG over XDS110, select the TI XDS driver option.

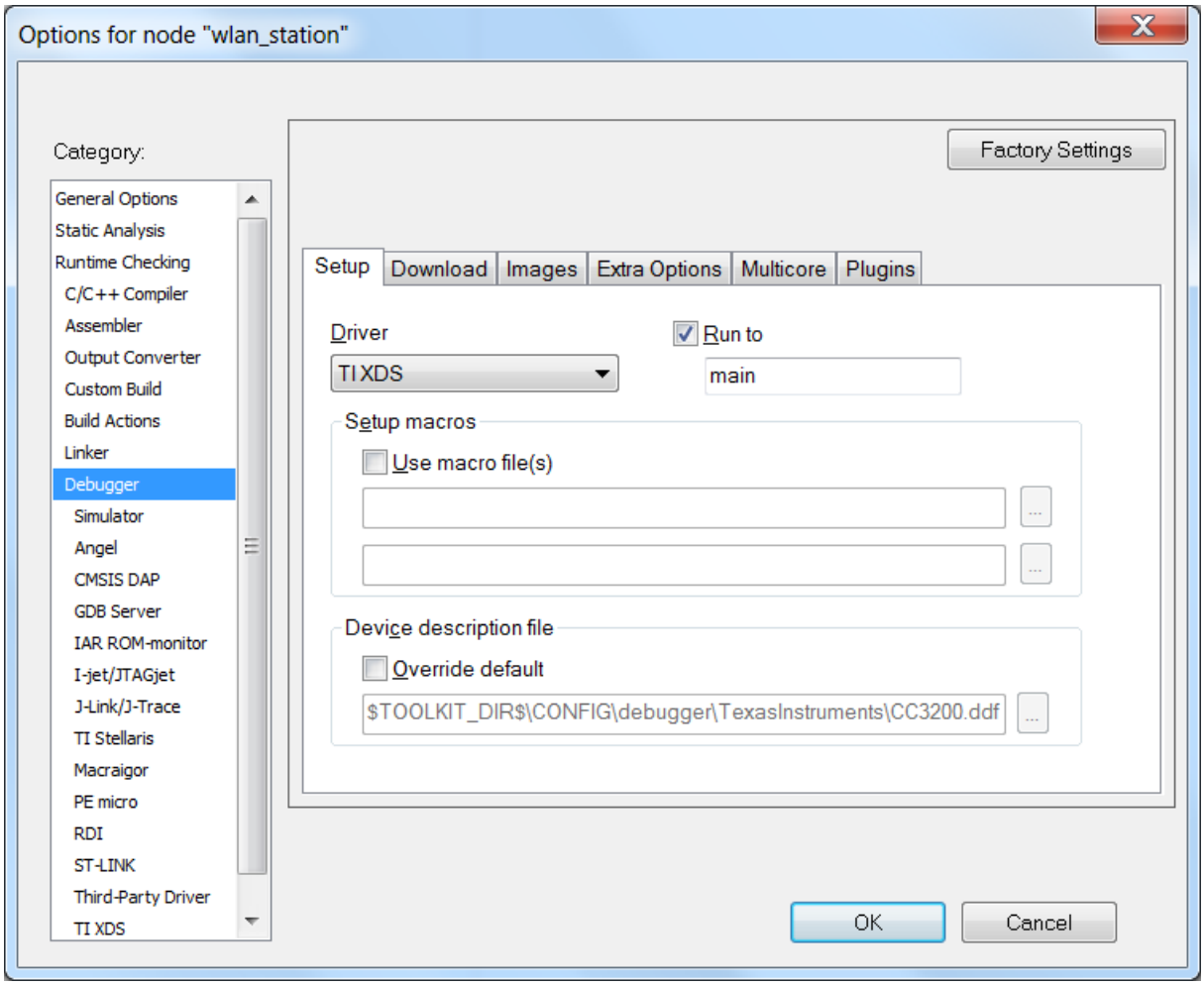

Figure 5-7. IAR Debug Options

2. If working with the CC3220SF device, check the box for “Use flash loader(s)” in the *Debugger -> Download* tab, as shown in Figure 5-8.

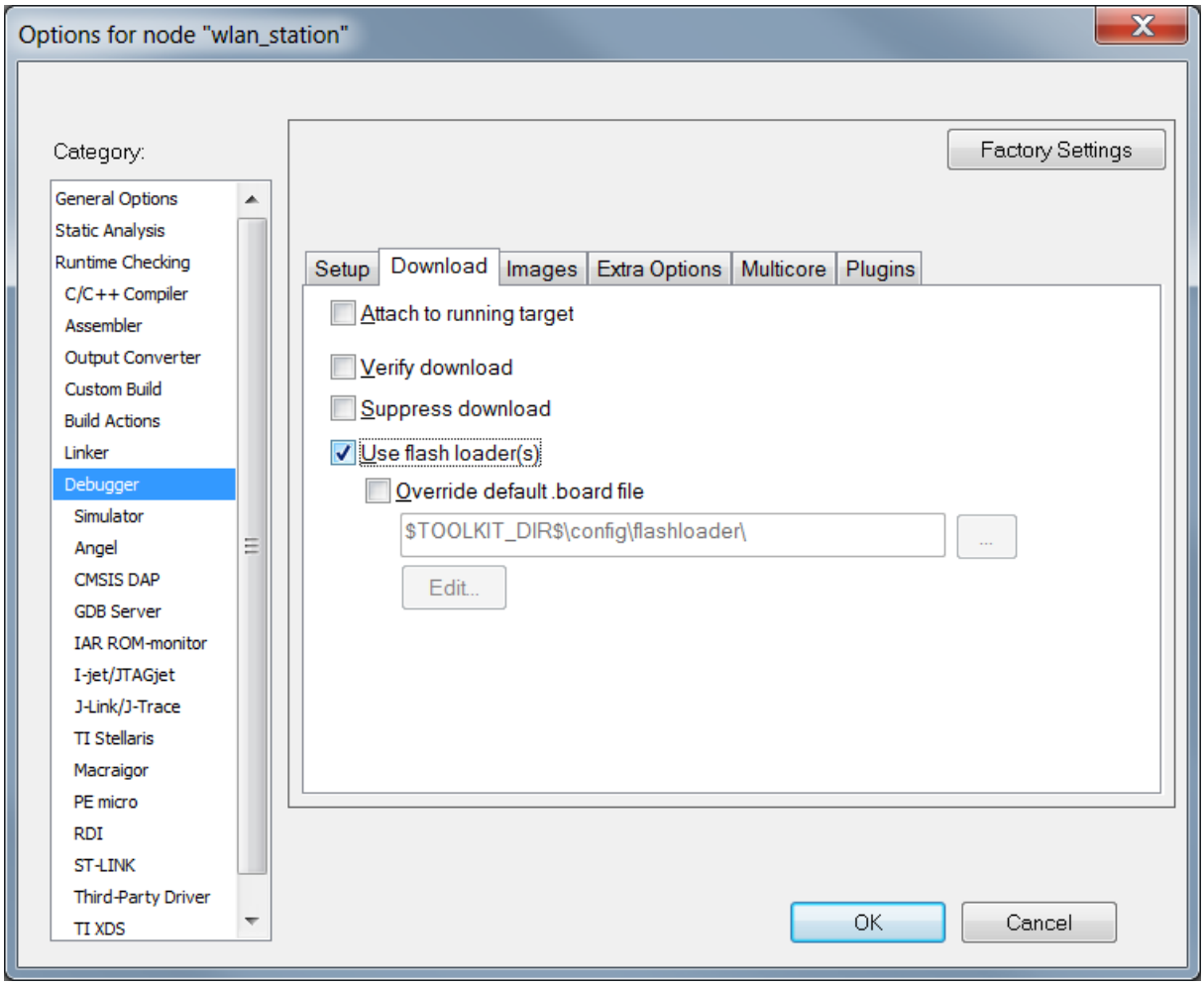

Figure 5-8. Using Flash Loader

3. Go to the TI XDS option and, if required, override the path for the TI Emulation package installation (default is *C:\ti\ccs_base*).

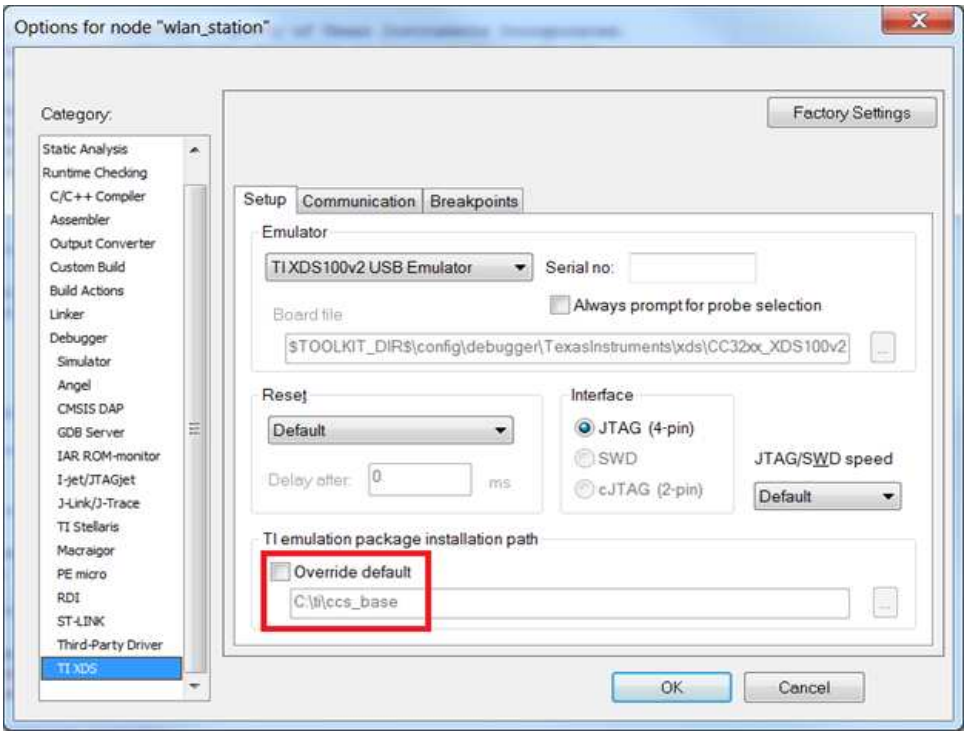

Figure 5-9. IAR XDS Settings

4. Click on the Download and Debug button to start the execution. The execution stops at the main function.
5. Click the Go button (or F5) to run.

NOTE: The user must recompile and link the debug version of SimpleLink in debug mode. Refer to Section 3.1.4.

Figure 5-10. IAR Download and Debug

If the application uses UART to print output on the terminal, the user must set up the terminal application (such as HyperTerminal or TeraTerm). The serial port settings are:

- Baud rate – 115200
- Data – 8 bits
- Parity – none
- Stop – 1 bit
- Flow control – none

5.1.2 Development Environment – TI's Code Composer Studio™

Code Composer Studio™ (CCS) is an integrated development environment (IDE) to develop applications for TI embedded processors. This section details the steps to create a new project in a CCS environment for the CC3220 wireless MCU.

5.1.2.1 Configure CCS for the CC3220 Device

The user must install the CC3220 support package in CCSv6 so the IDE can better support the CC32xx devices. Perform the following steps to install support package in CCS:

1. Open the App Center from the *Help→CCS App Center*.
2. Type cc32xx in the search box and select all the packages, including TI-RTOS and other add-ons, as shown in Figure 5-11.

Figure 5-11. CCS Support Package Installation

3. A restart (for CCS IDE) may be required after the installation is complete.

5.1.2.2 Creating a New Project

1. Go to *File->New-> CCS Project*.
2. Keep Target = Wireless Connectivity MCU and Device Variant = CC3220SF/CC3220S/CC3220.
3. Specify the project name (wlan_station in used for demonstration), and choose a location to create the project, as shown in Figure 5-12.

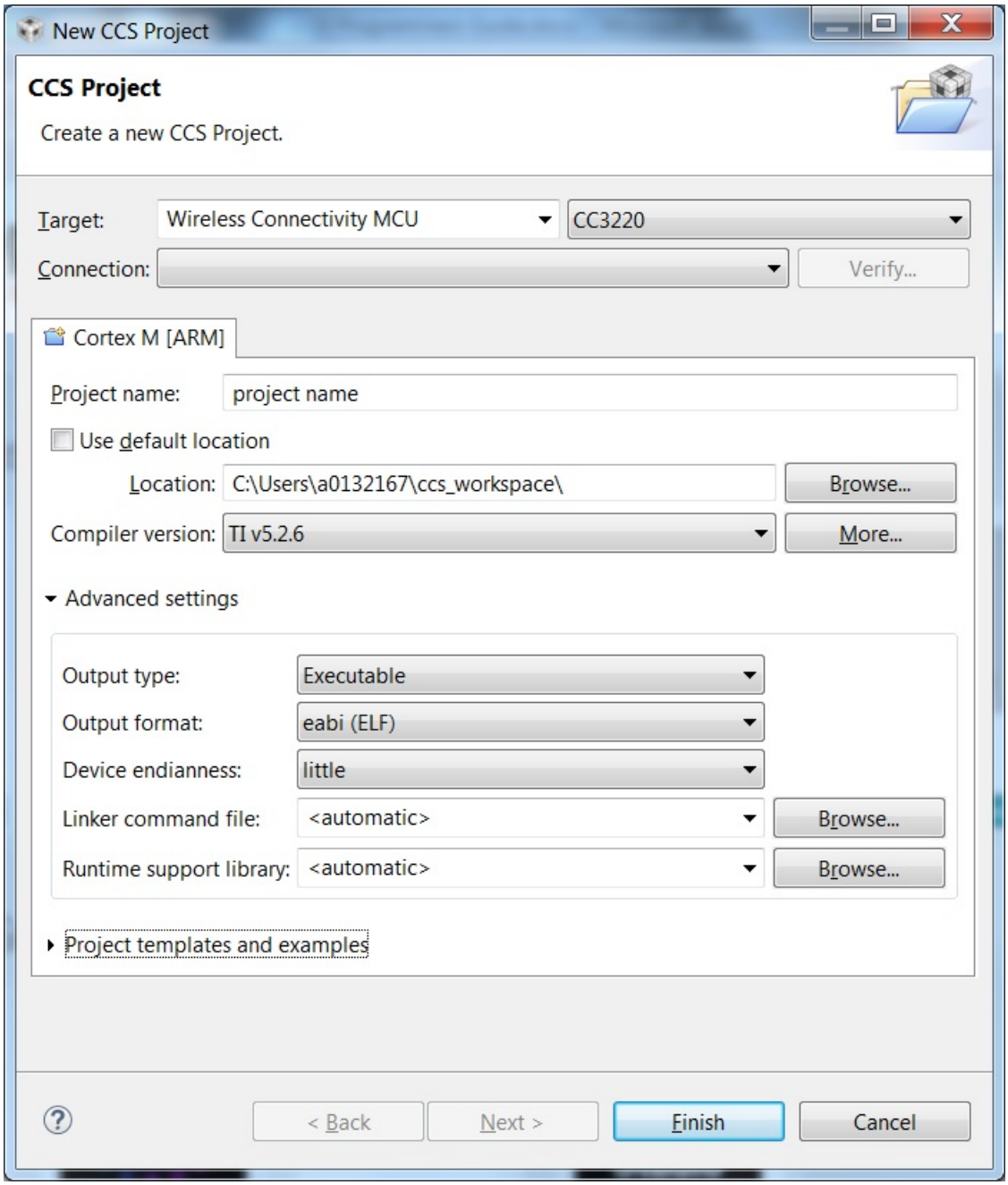

Figure 5-12. CCS Project Settings

4. Select the project template, and click *Finish*, as shown in Figure 5-13.

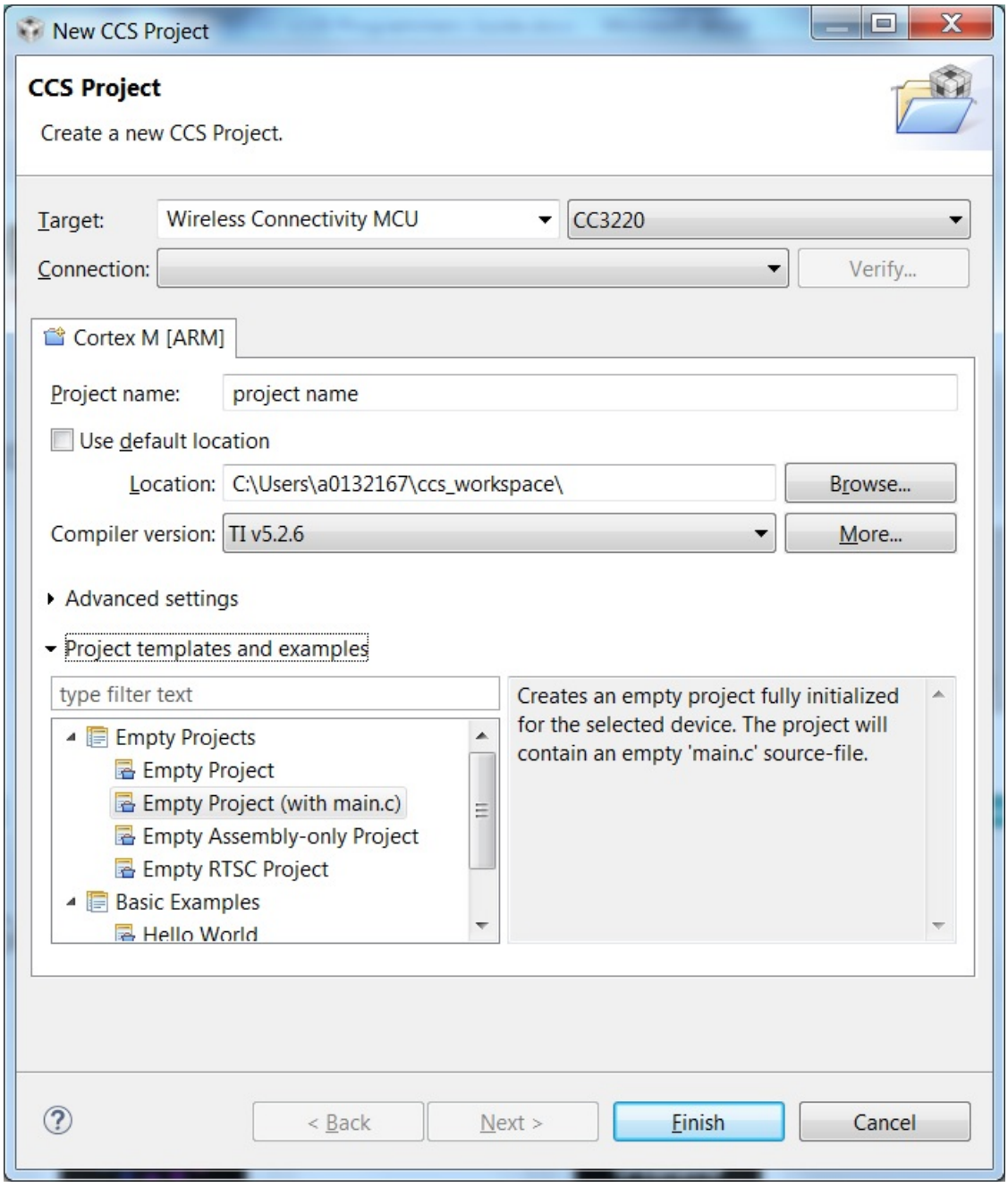

Figure 5-13. CCS Project Settings

5.1.2.3 Adding the Files

This section is common for all the supported tool chains. Refer to Section 5.1.1.2.

5.1.2.4 Compiling

Add directories to the #include search path:

- To use Driverlib APIs – include *driverlib* and *inc* folder paths
- To use Simplelink APIs - include *simplelink*, *simplelink\Source*, and *simplelink\include* folder paths
- To use TI drivers – include *tidrivers_cc32xx/packages/*
- To use TI-RTOS APIs – include *oslib* folder path
- To use common interface APIs – include *example\common* folder path
- Path for any other library that is required by the application

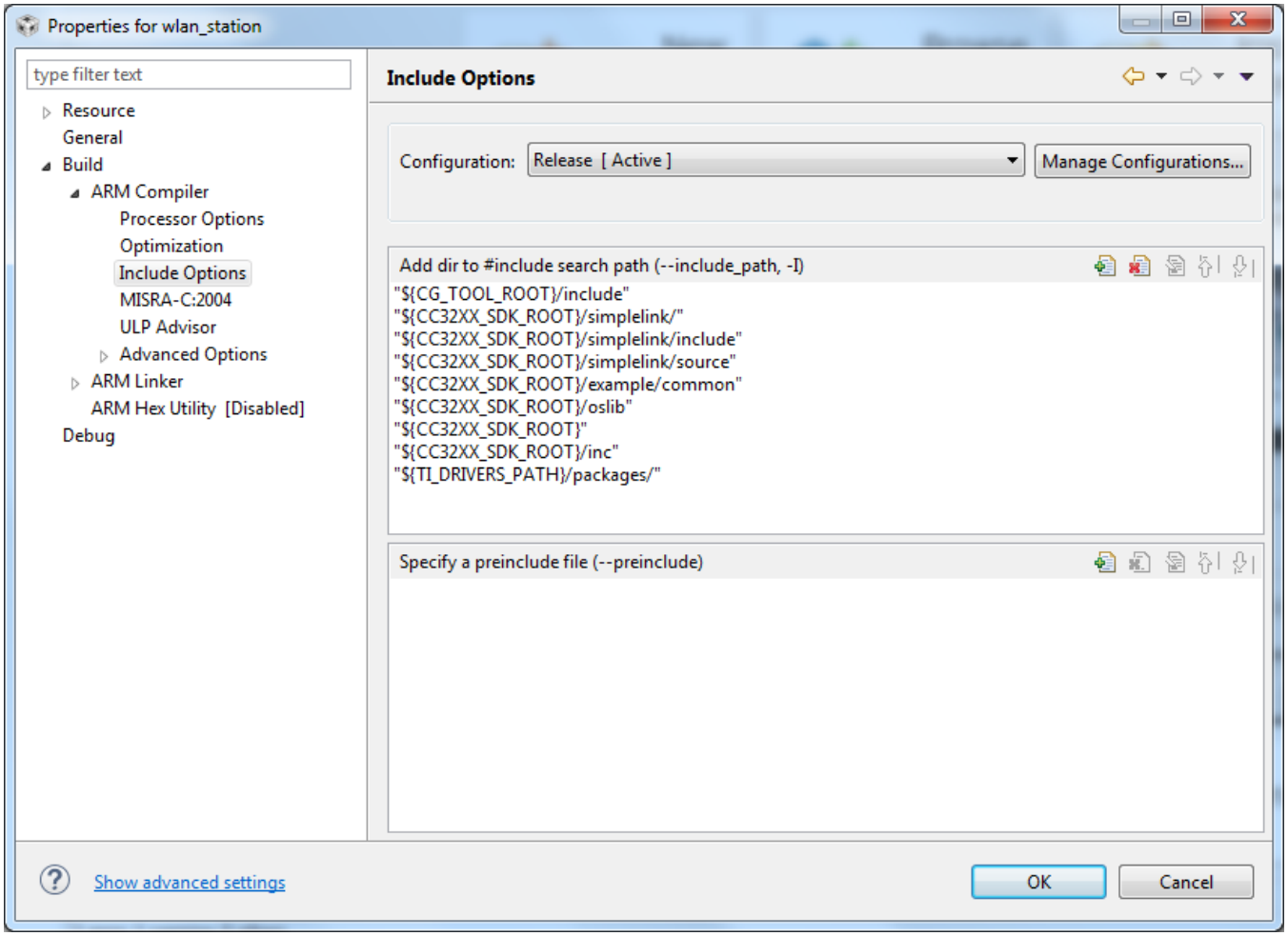

Figure 5-14. CCS Include Settings

Predefine NAME:

- USE_TIRTOS – To use TI-RTOS OS APIs
- USE_FREERTOS – To use FREE-RTOS OS APIs
- SL_PLATFORM_MULTI_THREADED – If the application uses any OS
- ccs – For a CCS-based application

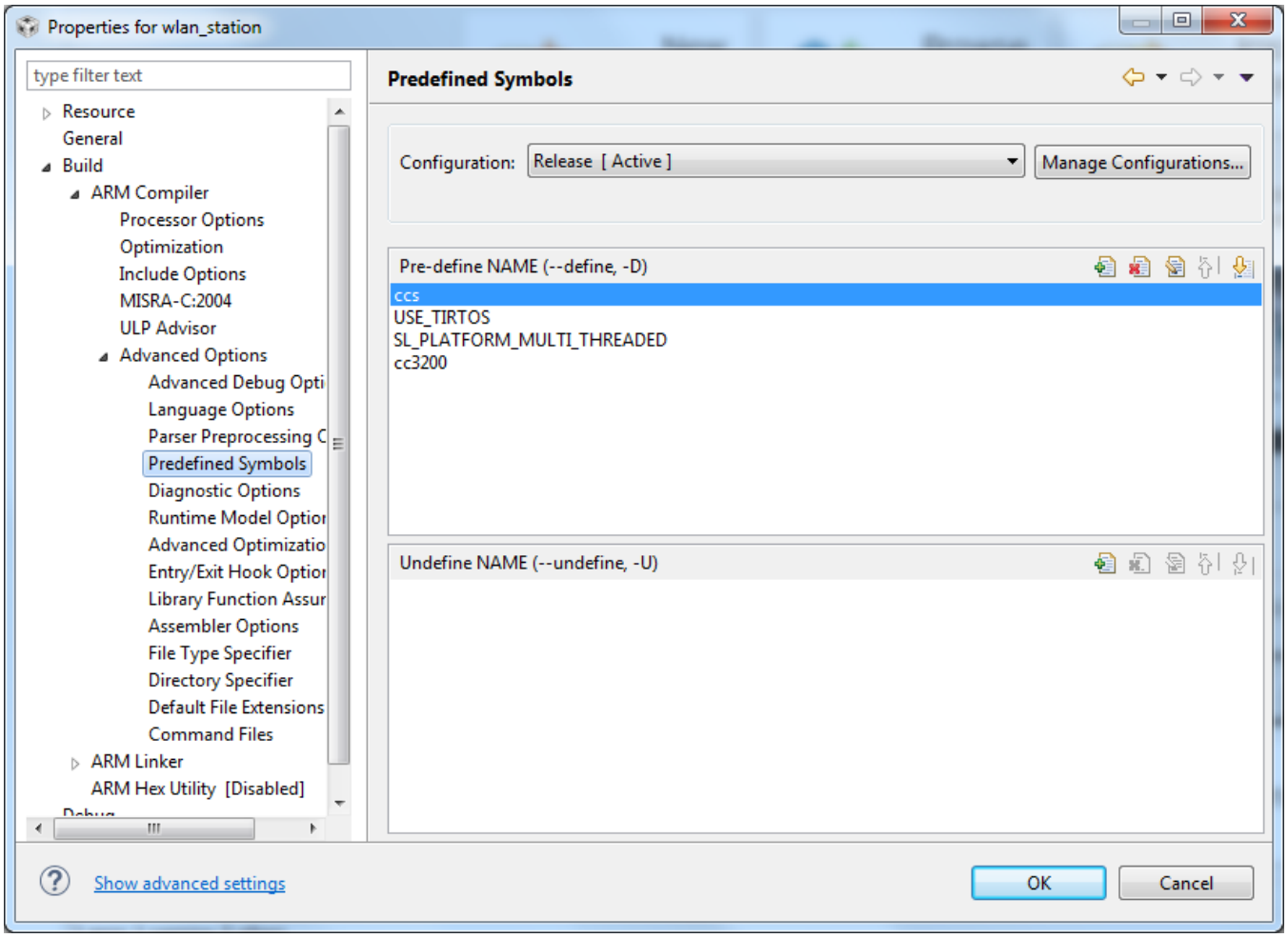

Figure 5-15. CCS Predefined Symbols

5.1.2.5 Linking

Link library files:

1. Add the library path as per application needs (driverlib.a, simplelink.a, freertos.a/ tirtos.a, osal_tidrivers.a/tirtosport.aem4, tidrivers_cc32xx/packages/ti/drivers/lib/drivers_cc32xxware.aem4), as shown in Figure 5-16.

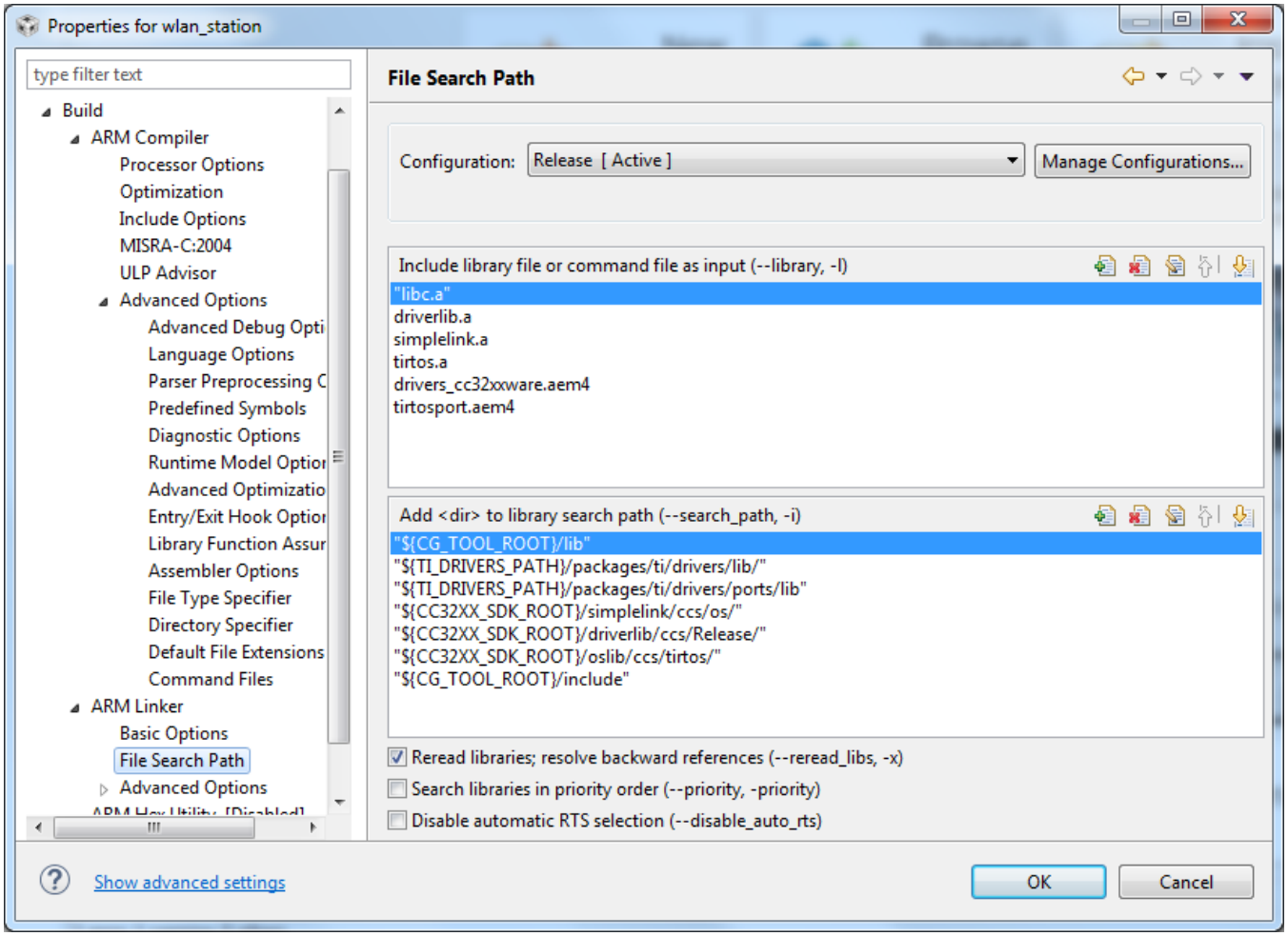

Figure 5-16. CCS Linker Settings

2. Specify the stack and heap values, along with the output file settings, as shown in Figure 5-17.

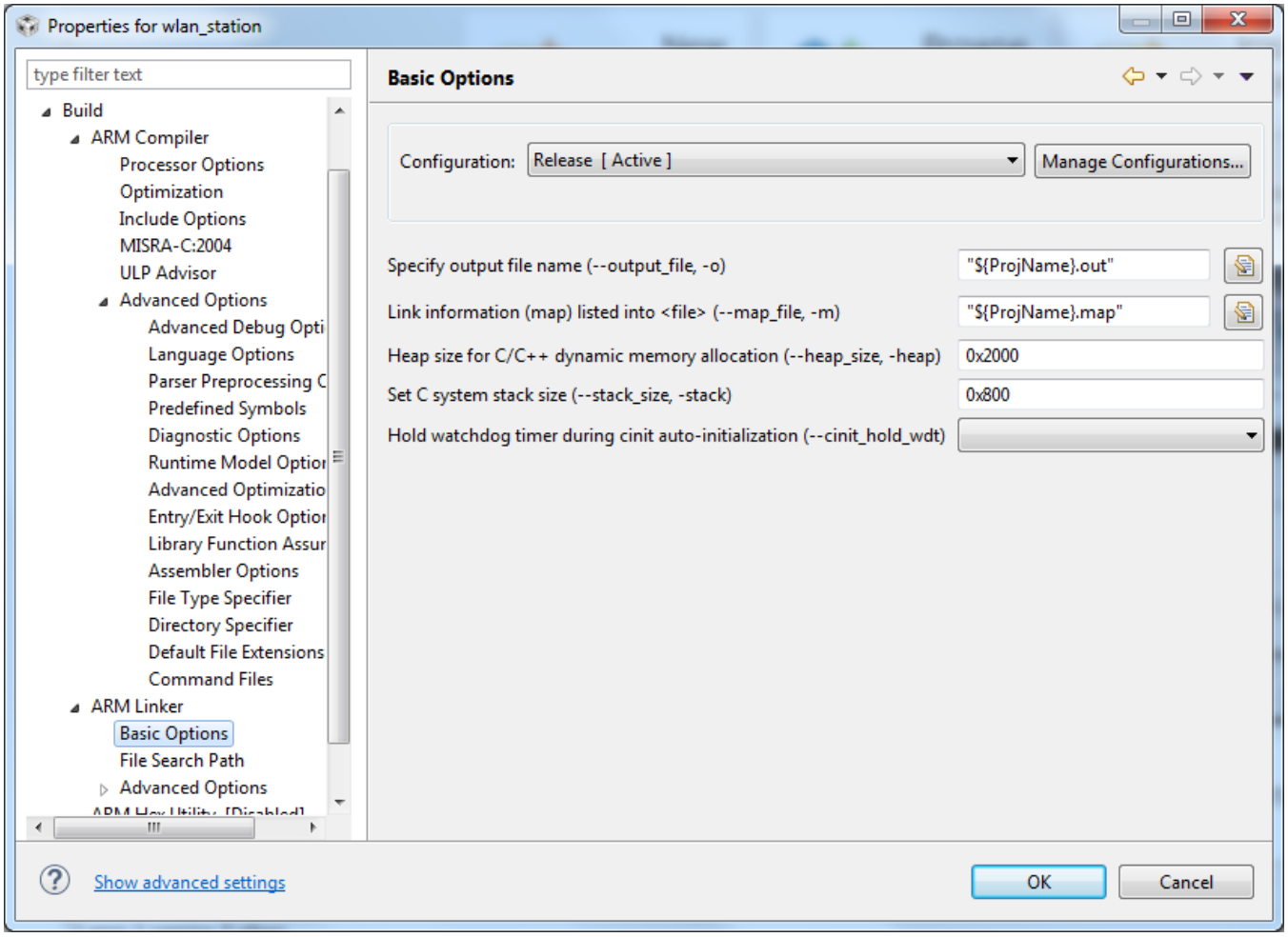

Figure 5-17. CCS Stack and Heap Values

NOTE: If tirtos is being used, the heap setting in Figure 5-17 will have no effect, as the heap values are specified in the app.cfg file in the tirtos_config project.

NOTE: The user must recompile and link the debug version of SimpleLink in debug mode. Refer to Section 3.1.4.

5.1.2.6 Dependency to Another Project

Dependencies:

- If the application uses TI-RTOS OS, then add the tirtos_config project as a dependency for the application.
- The tirtos_config project should be imported in the CCS workspace for a TI-RTOS-based application, as shown in Figure 5-18.

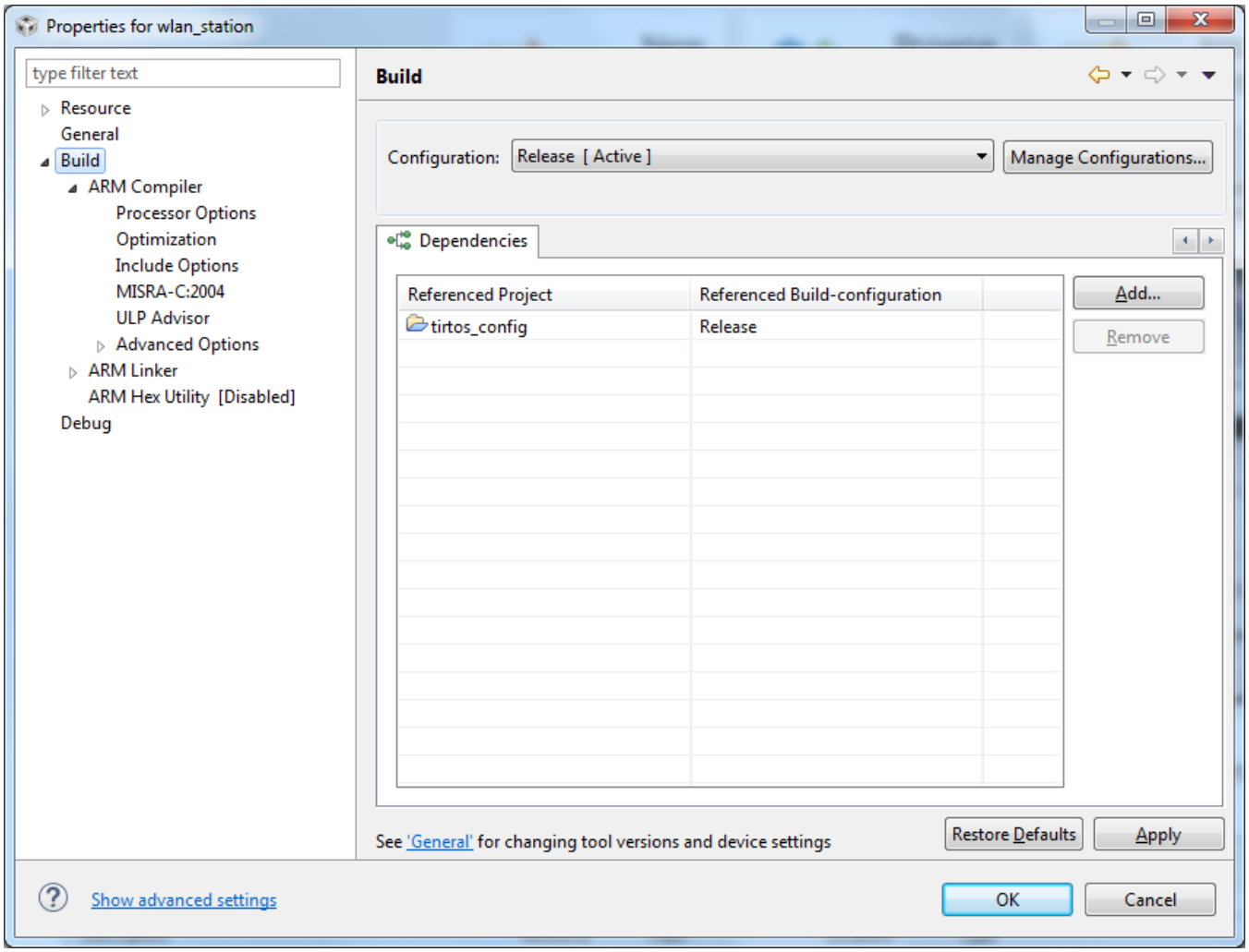

Figure 5-18. CCS Dependencies

NOTE: In cases of dependency on another project, the other project should be imported into the workspace before compiling the target project.

5.1.2.7 Recompiling TI-RTOS

The TI-RTOS library in the SDK is based on a specific version of TI-RTOS. The user can choose to update the TI-RTOS library to a later version (if available).

1. Refer to Section 2.8 of the *CC3220 SimpleLink Wi-Fi and Internet of Things Software Getting Started Guide* to install the latest version of TI-RTOS.
2. Restart the CCS.
3. After CCS discovers the recently added components, select the TI-RTOS package and finish the installation process.
4. Right-click on *tirtos_config project*, and go to *Properties->General->RTSC*.
5. Select the latest version for TI-RTOS under TI-RTOS for CC32XX.
6. Rebuild the tirtos_config project.
7. Recompile the tirtos configuration for the oslib library.

5.1.2.8 Generating the Binary (.bin)

Add the following script to generate the .bin file:

```
"${CCE_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin" "${BuildArtifactFileName}"
"${BuildArtifactFileBaseName}.bin" "${CG_TOOL_ROOT}/bin/armofd"
"${CG_TOOL_ROOT}/bin/armhex" "${CCE_INSTALL_ROOT}/utils/tiobj2bin/mkhex4bin"
```

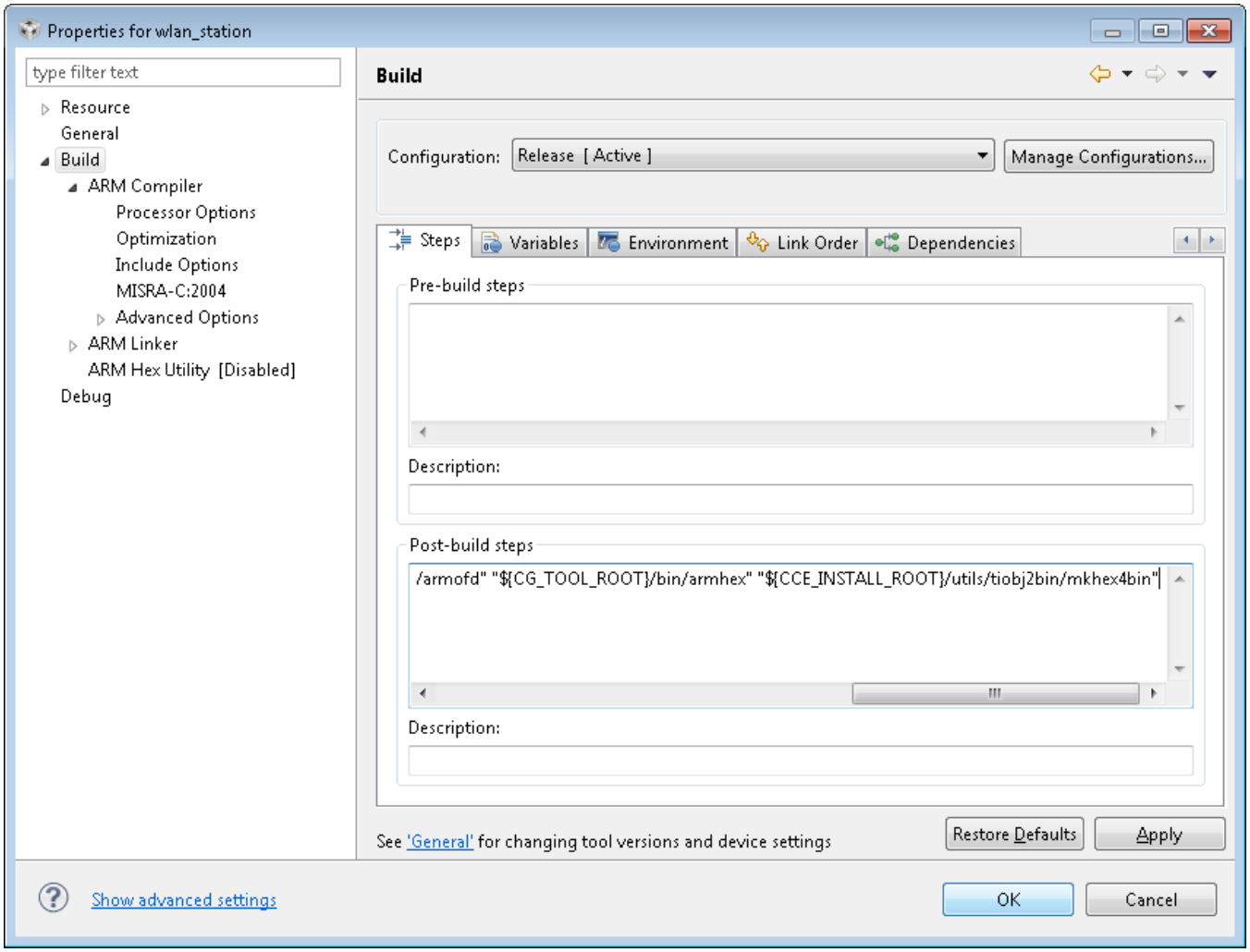

Figure 5-19. CCS Generating the .bin File

5.1.2.9 Executing

1. The target configuration must be set before debugging from CCS. Navigate to *View→Target Configurations*, as shown in Figure 5-20.

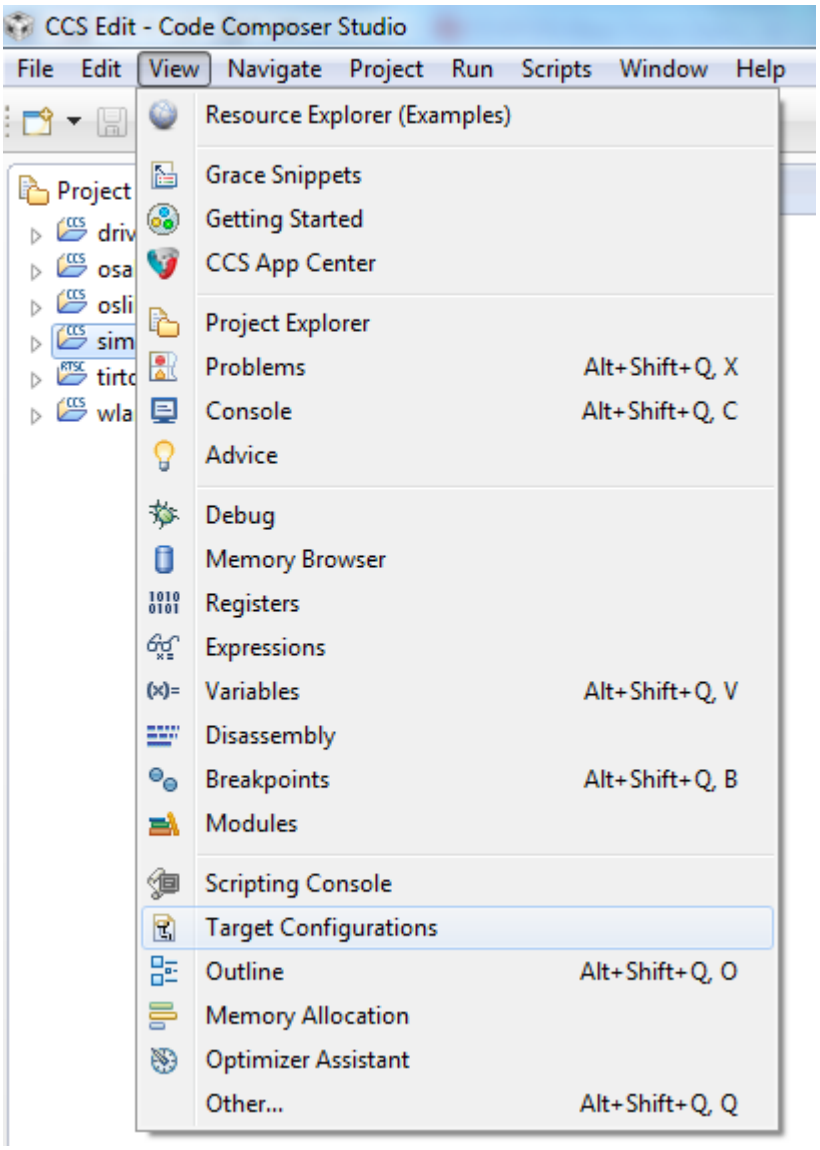

Figure 5-20. Set Target Configuration

2. Right-click on *User Defined*, select *Import Target Configuration* (see Figure 5-21), and select the file *CC3220_XDS110.ccxml* from *<sdk-installation-directory>\cc3220-sdk\tools\ccs_target_configurations*. Select the *Copy files* option when prompted.

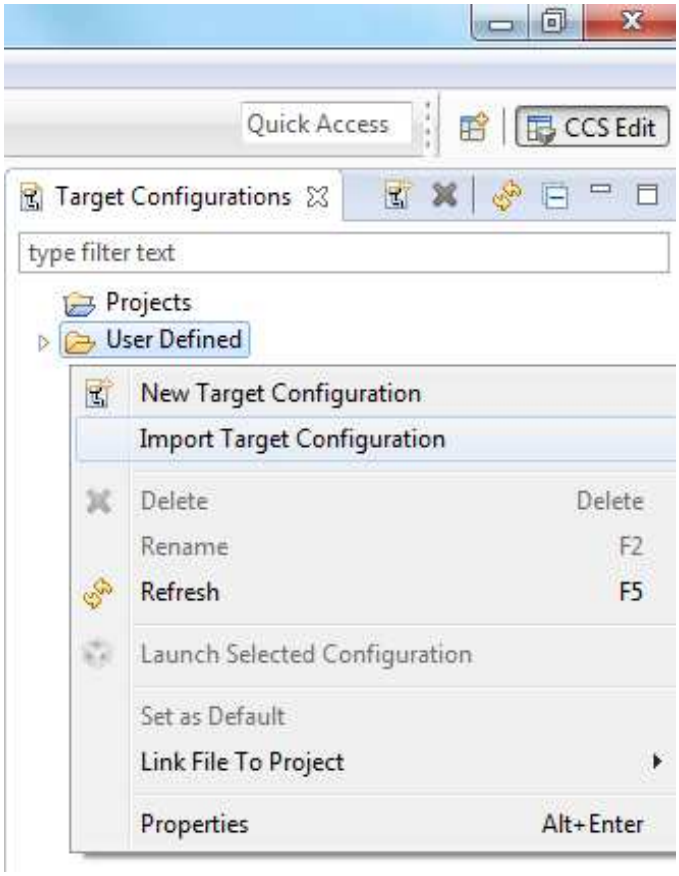

Figure 5-21. User Defined

3. Set this new configuration as the default by right-clicking on the filename and selecting *Set as Default*.
4. Launch the application. Select the project in *Project Explorer* and click on the debug icon, as shown in Figure 5-22, to download code to the device and begin debugging. Press F8 to begin execution.

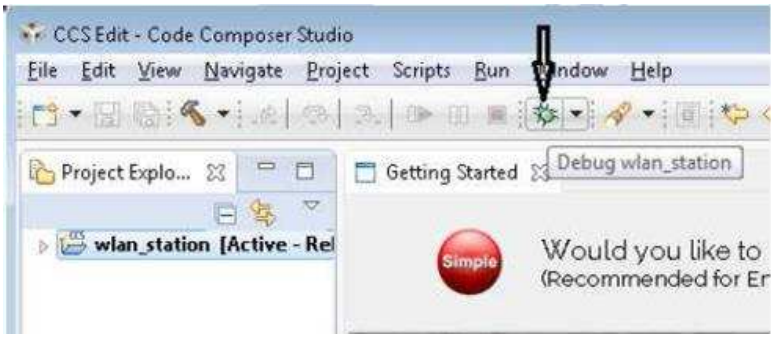

Figure 5-22. Debug Icon

5.1.2.10 Setting up the Serial Terminal

If the application uses a UART terminal to print any information, a serial terminal application such as TeraTerm or Hyperterminal must be installed and configured on the host machine. For installing and configuring the serial terminal, refer to the *CC3220 SimpleLink Wi-Fi and Internet of Things Software Getting Started Guide*. The configuration is as follows:

- Baud rate – 115200
- Data – 8 bits
- Parity – none
- Stop – 1 bit

5.1.3 Development Environment – Open Source [GCC/GDB]

This software is designed to be compatible with open source tool chains. This section details how to get started with GCC/GDB with the CC3220 LaunchPad development kit, and lists all the dependencies associated with the environment setup for Windows® OS under Cygwin.

TI has validated some sample applications with GCC (including building block libraries such as SimpleLink Library, Peripheral Driver Library, and so forth).

5.1.3.1 Environment Setup

5.1.3.1.1 Cygwin Installation (Windows)

1. Download setup-x86.exe from http://cygwin.com/install.html and run it. Select the Install from Internet option.
2. Specify a proxy if necessary, depending on the network.
3. Choose a download site (for example, http://mirrors.kernel.org).
4. Include the latest versions of the following packages in the Cygwin installation (in addition to those included in the base installation):
   - Archive/unzip
   - Archive/zip
   - Devel/autoconf
   - Devel/automake
   - Devel/libtool
   - Devel/make
   - Devel/subversion (if using TortoiseSVN/Windows 7, skip this file)
   - Devel/gcc-core
   - Devel/gcc-g++
   - Devel/mingw-gcc-core
   - Devel/mingw-gcc-g++
   - Devel/mingw-runtime

See Figure 5-23 for an example of selecting a package (for example: Devel/autoconf).

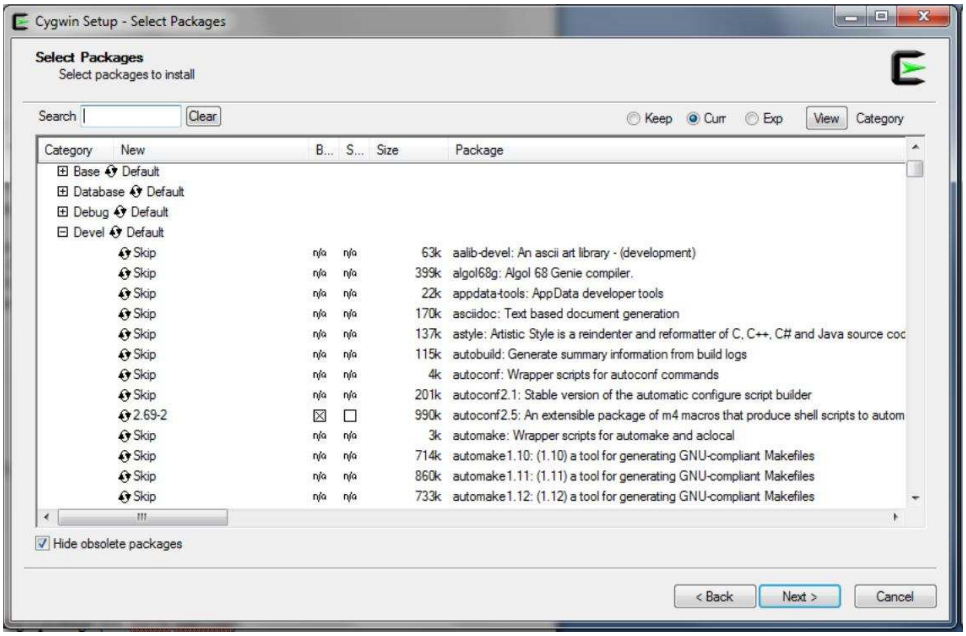

Figure 5-23. Selecting a Package

5. The system will find dependencies. Press Next.
6. After a successful Cygwin installation, add its path (*c:\cygwin\bin*) to the Windows environment variable PATH by going into Control Panel>System>Advanced System Settings>Environment Variables. Under System Variables, select PATH and press Edit. Append ";C:\cygwin\bin\" to the end of the line and press OK.

5.1.3.1.2 GNU Tools for ARM-Embedded Processors

Download latest version of gcc-arm-none-eabi-<version>-win32.exe from following https://launchpad.net/gcc-arm-embedded, and install under the Cygwin root directory (Default: c:\cygwin). Add the path *<installation_dir>\bin* (for example *c:\cygwin\4.9 2015q2\bin*) to the windows PATH environment variable.

5.1.3.1.3 Open On-Chip Debugger (OpenOCD)

1. Download prebuilt openocd-0.9.0 for windows from http://www.freddiechopin.info/download/category/4-openocd, and unzip the package to the Cygwin root directory (default: c:\cygwin).
2. Add the path for the openocd.exe (.*openocd-0.9.0\bin*\) to the Windows PATH environment variable.

5.1.3.2 Creating a New Project

TI recommends using an existing Makefile in the SDK as the template for a new project. Make the necessary changes for the new application.

5.1.3.3 Adding the Files

This section is common for all supported tool chains. Refer to Section 5.1.1.2.

5.1.3.4 Compilation

In the Cygwin terminal, go to the project (Makefile) path and run the following command:

- For a CC3220SF device:

```
make -f  Makefile
```

- For a CC3220 device:

```
make -f  Makefile DEVICE=CC3220
```

- For a CC3220S device:

```
make -f  Makefile DEVICE=CC3220S
```

NOTE: Cygwin uses forward slashes to separate the directories.

The command will generate *<project_name>.axf* and *<project_name>.bin* files under the *gcc\exe* folder.

5.2 Flashing the Binary

For the flashing procedure, refer to the *CC3220 SimpleLink Wi-Fi and Internet of Things Software Getting Started Guide*.

CC3220 ROM Services

The CC3220 ROM hosts the bootloader and peripheral driver library. The peripheral driver library is a collection of routines that abstracts the peripheral programming. This library is provided in the ROM to give the developer an option to reduce the RAM footprint of the application.

The peripheral driver library (driverlib) inside ROM corresponds to driverlib version 1.50.1.00.

Bootloader services let the user update the application binary image, along with other user files in serial flash, and are responsible for loading the user application from the serial flash to MCU RAM.

6.1 CC3220 Bootloader

The CC3220 bootloader resides in the ROM of the application processor. It is responsible for the following operations:

- **Update and Download** – The bootloader enables the developer to download an application image from the PC to the CC3220 device (in serial flash). The bootloader-download functionality can be triggered only when the board is in UARTLOAD sense-on-power (SOP) mode.
- **Bootstrap** – The bootloader is also responsible for scanning a valid application image (binary) in the serial flash (for the CC3220 device). Subsequently, the image is loaded to internal RAM, and execution control is passed to the user program.

6.1.1 Bootloader Modes – Impact of Device SOP Pin

The CC3220 device has three SOP pins. A detailed explanation of the functionality is described in the *CC3220x SimpleLink™ Wi-Fi® Wireless and Internet-of-Things Solution, a Single-Chip Wireless MCU Data Sheet*. In the context of the bootloader, there are three modes:

- Download only – This mode corresponds to the SOP [2:0] being 0b100, and makes the bootloader enter the download mode. This mode expects a break signal on UART from the programming application, which is followed by a sequence to the application image to the serial flash of the device.
- Download/Execute – The SOP [2:0] setting for this case would be 0b010. The user can download the application and execute it in this mode.
- Execute – A setting corresponding to SOP [2:0] = 0b000. For the CC3220S and the CC3220 devices, this instructs the bootloader to load the application image from the SFLASH to internal MCU RAM. For the 3220SF, the bootloader jumps to the application image in the on-chip flash, and executes.

NOTE: An SOP setting of 010 or 100 is needed for image programming through UART. During development and debugging, these SOP modes can be used.

However, once the programming is done and before putting the device to final production use case, TI recommends using SOP mode 000. If the device is not SOP 000, then the Restore to Factory by the Host feature may not work in a failsafe manner. The limitations and workarounds if in SOP mode 010 or 100 are listed below.

For the CC3220 device, if the SOP is configured as 010 or 100, the following scenarios may occur:

Limitation:

- Host requests Restore to factory feature.
- If a device reset occurs while Restore to factory is underway, the device could get into a stuck state.

Solution:

1. Set the SOP to 000.
2. POR the device, and the Restore to factory is completed successfully.

6.1.2 Bootloader and User Application – Sharing MCU RAM

The bootloader uses a portion of the MCU RAM for its own execution. The amount of RAM used by the bootloader is 16KB, which implies that for the CC3220S and CC3220 variants, the user application image must be restricted to 240KB for the 256-KB MCU RAM variant of the CC3220 and CC3220S devices. For the CC3220SF variant, the application image resides in the on-chip flash, and the whole SRAM is available for the application. There are several key points for the developer when working with the CC3220 and CC3220S variants:

- MCU RAM address range 0x20000000 to 0x20003FFF: This area is shared between the application and the bootloader. The developer can locate only the application data sections, because the data sections are not part of the application image; this ensures that when the bootloader is loading the application image from the serial flash to RAM, this memory region is made exclusive to the bootloader. Once the bootloader launches the application, this memory region can be used by the application for its data sections.
- 0x20004000 to END of RAM: This RAM area is exclusively for the application. The application image should always be within this region, and start at 0x20004000.

Figure 6-1. CC3220 and CC3220S Device SRAM

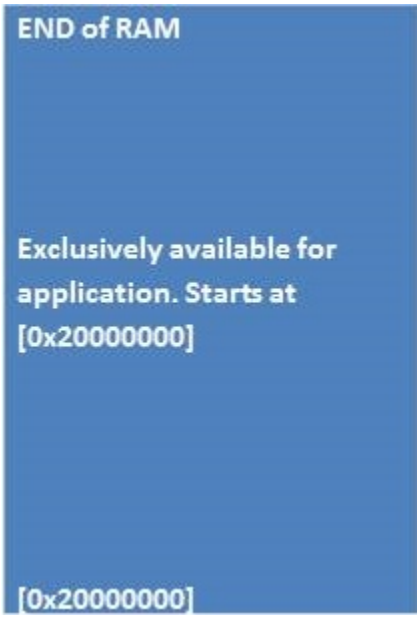

Figure 6-2. CC3220SF Device SRAM

6.2 CC3220 Peripheral Driver Library Services in ROM

Peripheral driver routines are provided in the CC3220 MCU ROM. The source for these routines, along with their project files, are also available in the CC3220 SDK as a library (driverlib). The developer can choose to link their application to the either of these routines.

This section focuses on how to use these routines, and the procedure to patch or extend any existing routines. TI recommends using these routines from ROM, because this saves the RAM space. If any modification to these routines is required, the developer can make the required changes in the library and link the application to the modified library.

6.2.1 Peripheral Access in ROM

The ROM APIs are indexed at a fixed location in the ROM map, which allows future extensions while retaining backward compatibility. The API locations might change in the future versions of the ROM, but API tables will remain the same.

Two levels of indexing are done, which means two tables in the ROM resolve to the entry point of each supported API. The main table contains one pointer per peripheral, which points to a secondary table that contains a pointer corresponding to each API associated with that peripheral.

The main table is at address 0x0000040C in the ROM. Table 6-1 and Table 6-2 list a small portion of the API tables showing the arrangement of the tables.

Table 6-1. ROM_API Table (at 0x0000040C)

[0] = RESERVED
[1] = pointer to ROM_UART TABLE
[2] = pointer to ROM_TIMER TABLE
[3] = pointer to ROM_WATCHDOG TABLE
[4] = pointer to ROM_INTERRUPT TABLE
[5] = pointer to ROM_UDMA TABLE
[6] = pointer to ROM_PRCM TABLE
[7] = pointer to ROM_I2C TABLE
... ...

Table 6-2. ROM_INTERRUPT Table

[0] = pointer to ROM_IntEnable
[1] = pointer to ROM_IntMasterEnable
[2] = pointer to ROM_IntMasterDisable

The address of ROM_INTERRUPT TABLE is in the memory location at 0x0000041C. The address of the ROM_IntMasterEnable () function is at offset 0x4 from that table. In the function documentation, ROM_API TABLE is an array of pointers at 0x0000040C.

ROM_INTERRUPT TABLE is an array of pointers at ROM_API TABLE [4].

ROM_IntMasterEnable is a function pointer at ROM_INTERRUPT TABLE [1].

6.2.2 Linking the User Application With ROM APIs

The following steps describe how to use ROM driverlib APIs instead of the RAM APIs. These steps apply to all relevant source and project files that use driverlib APIs.

NOTE: Follow these steps for the application as well as the libraries and projects which are driverlib-dependent and used by the application. The libraries which might be used by the application are simplelink, oslib, netapps, osal_tidrivers, ti_drivers, and netapps libraries (http, mqtt, json, smtp, tftp, xmpp).

1. Include the following header files, in order, in all .c files using the driverlib APIs:

```
    #include "rom.h"
#include "rom_map.h"
```

2. Add the global preprocessor define USE_CC3220_ROM_DRV_API to all project files.
3. Invoke all driverlib APIs by MAP_APIname instead of APIname. For example, use MAP_UARTCharPut instead of UARTCharPut. Any changes or additions should follow this approach.
4. Rebuild all the relevant libraries and projects.

6.2.3 Patching ROM APIs

Follow these steps to selectively patch the ROM driverlib APIs. "Patch" in this description means using the RAM driverlib API instead of the ROM driverlib API.

1. Add an entry in the file \driverlib\rom_patch.h for all APIs to be patched. For example, to patch MAP_UARTCharPut and MAP_UARTBreakCtl entries in file rom_patch.h:

```
#undef ROM_UARTCharPut
#undef ROM_UARTBreakCtl
```

2. Rebuild all the relevant projects that use driverlib APIs.

6.2.4 Linking With RAM-Based Peripheral Driver Library

To delink all ROM driverlib APIs and use the RAM driverlib APIs, follow these steps:

1. Remove the global preprocessor define USE_CC3220_ROM_DRV_API from all project files that use the driverlib APIs.
2. Rebuild all the relevant projects that use driverlib APIs.

Revision History

Date	Revision	Notes
February 2017	SWRU464*	Initial release

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265