

CC3200 SimpleLink™ Wi-Fi® and IoT Solution, a Single Chip Wireless MCU

Programmer's Guide



Literature Number: SWRU369C
June 2014–Revised February 2016

1	Introduction.....	6
1.1	Overview	6
1.2	Software Components	6
1.3	CC3200 LaunchPad Platform	8
2	Foundation SDK – Getting Started	9
2.1	Installation	9
2.2	Package Components Overview.....	11
2.3	Prerequisite: Tools to be Installed	13
3	Foundation SDK – Components	14
3.1	SimpleLink Component Library	14
3.2	Peripheral Driver Library	21
3.3	Reference Applications	22
3.4	CC3200 PinMux Utility	24
4	Getting Started With the CC3200 LaunchPad	25
5	Foundation SDK – Development Flow	26
5.1	Simple Networking Applications	26
5.2	SimpleLink APIs	40
5.3	Compilation, Build and Execution Procedure	40
5.4	Flashing and Running the .bin Using the Uniflash Tool	67
6	CC3200 ROM Services	67
6.1	CC3200 Bootloader	67
6.2	CC3200 Peripheral Driver Library Services in ROM	68
7	Additional Resources.....	71
	Revision History.....	72
	Revision History.....	72
	Revision History.....	72

List of Figures

1	CC3200 Overview of Peripherals	6
2	CC3200 Software Components	7
3	CC3200 LaunchPad Platform	8
4	CC3200 SDK Installation 1	9
5	CC3200 SDK Installation 2	9
6	CC3200 SDK Installation 3	10
7	SimpleLink Modular Composition	14
8	CC3200 SimpleLink IAR Config Switch	16
9	CC3200 CCS SimpleLink Config Switch	17
10	File Search Path	18
11	Edit Directory Path	19
12	Edit SimpleLink Library Path	20
13	Edit SimpleLink Library Path	20
14	Linking Code Example 1	21
15	Linking Code Example 2	21
16	CC3200 Programmer Guide Device Manager	25
17	TCP Socket Terminal	33
18	UDP Socket Terminal	37
19	CC3200 Transceiver Application on the Hyperterminal	39
20	CC3200 Programmer Guide IAR Project Options	41
21	CC3200 IAR Compiling Project	41
22	CC3200 IAR Linker Project	42
23	CC3200 IAR Linker Config	43
24	CC3200 IAR Generating Binary	44
25	CC3200 IAR Executing	45
26	CC3200 IAR Download and Run	45
27	CCS App Center	46
28	TI-PinMux Tool	47
29	Select CCS Projects to Import	48
30	CC3200 CCS Editing Existing Project	49
31	CC3200 CCS Creating Project	50
32	CC3200 CCS Compiling Project	51
33	CC3200 CCS Compiling Project 1	52
34	CC3200 CCS Compiling Project 2	53
35	CC3200 CCS Linking Project 1	54
36	CC3200 CCS Linking Project 2	55
37	TI-RTOS OS Dependency	56
38	CC3200 CCS Generating Binary	57
39	CC3200 CCS Executing 1	58
40	CC3200 CCS Executing 2	58
41	CC3200 CCS Launch Config	59
42	Target Configuration	60
43	CC3200 CCS Executing 4	60
44	CC3200 CCS Executing 5	61
45	Cygwin Setup	62
46	Zadig Options	63
47	Replace Driver	63

48	Editing common.h	63
49	Output Screen	64
50	Debugging wlan_station	65
51	Tera Term VT	65
52	Device Manager	66

List of Tables

1	Package Contents.....	11
2	CC3200 Prerequisite.....	13
3	End of RAM	68
4	ROM APIs	68
5	ROM Interrupts	68

CC3200 SimpleLink™ Wi-Fi® and IoT Solution, a Single Chip Wireless MCU

1 Introduction

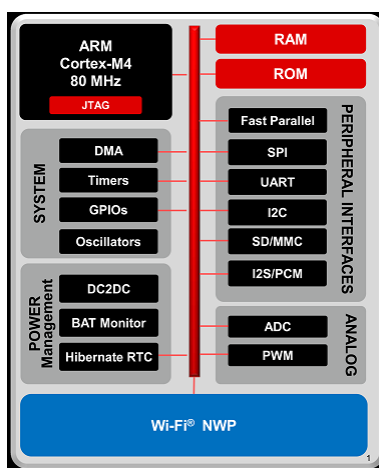
The CC3200 SimpleLink™ Wi-Fi® is the industry's first single-chip microcontroller (MCU) with built-in Wi-Fi connectivity, created for the Internet of Things (IoT). The CC3200 device is a wireless MCU that integrates a high-performance ARM Cortex-M4 MCU, allowing customers to develop an entire application with a single IC. This document introduces the user to the environment setup for the CC3200 SimpleLink Wi-Fi, along with programming 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 Texas Instruments royalty-free CC3200 Embedded Wi-Fi Foundation software development kit is a complete software platform for developing Wi-Fi applications. It is based on the CC3200, a complete Wi-Fi SoC (System-on-Chip) solution. The CC3200 solution combines a 2.4-GHz Wi-Fi PHY/MAC and TCP/IP networking engine with a microcontroller, 256 kB on-chip RAM, and a comprehensive range of peripherals.

Refer to the CC3200 Product Preview and Data Sheet ([SWAS032](#)) for more details on the CC3200 chip.

Figure 1. CC3200 Overview of Peripherals

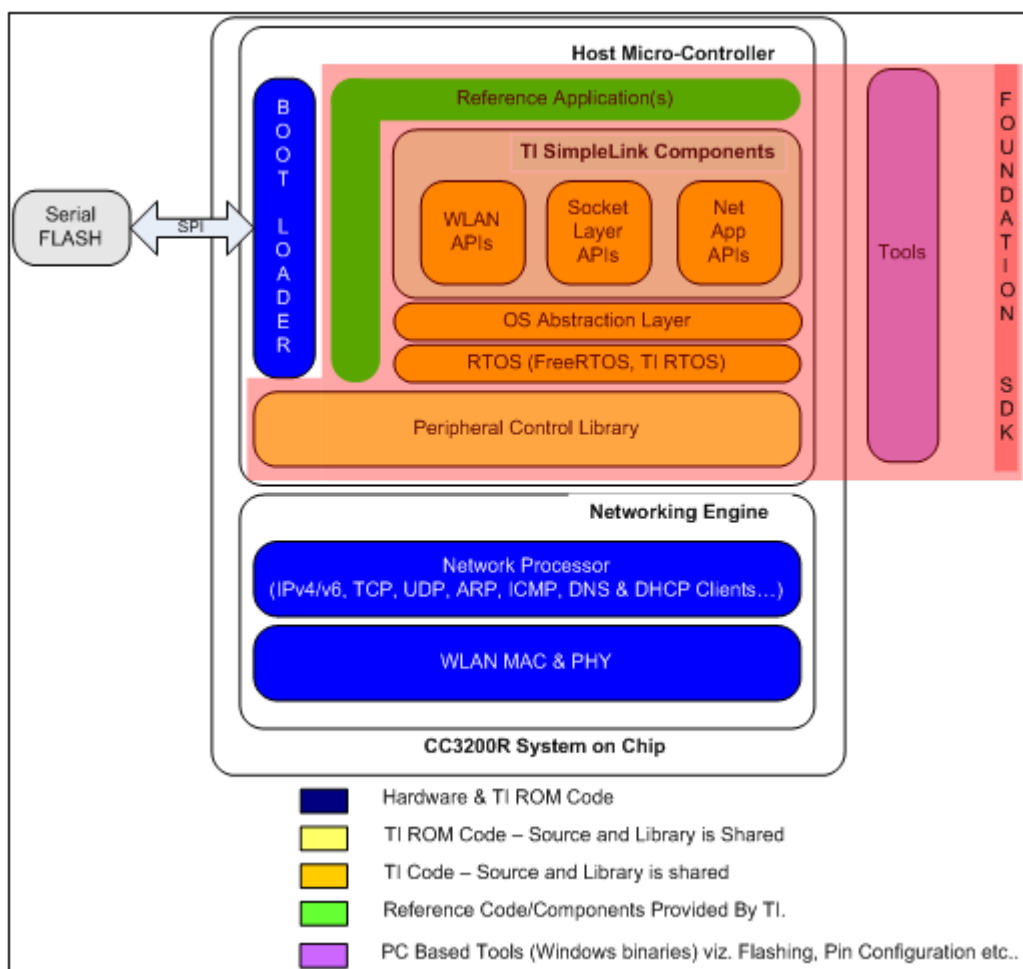


1.2 Software Components

The CC3200 platform includes a user-programmable host, along with a comprehensive networking solution combined with a Wi-Fi engine. The CC3200 Foundation software development kit provides an easy-to-use framework, hosted in the on-chip microcontroller, to use the WLAN networking services, along with a comprehensive listing of drivers for peripherals interfaced with the microcontroller. The kit also includes a reference code for peripheral usage and a few simple applications for networking services.

Figure 2 illustrates the various software components and their form in the CC3200 Foundation SDK.

Figure 2. CC3200 Software Components

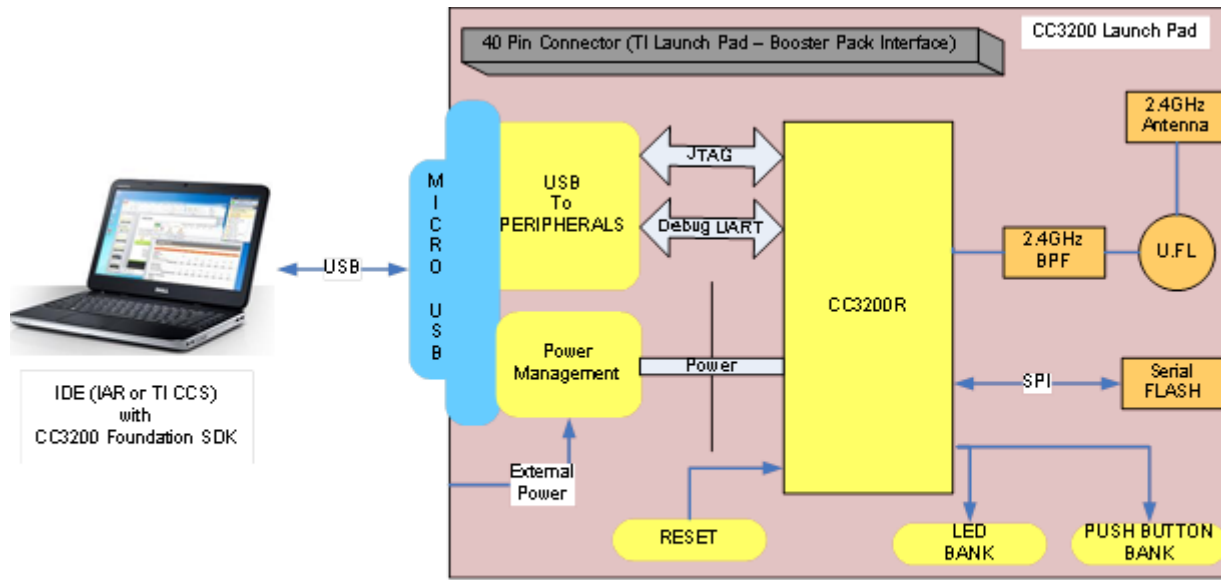


1.3 CC3200 LaunchPad Platform

The CC3200 LaunchPad board is the default hardware companion for the foundation SDK. This board hosts the CC3200 device, with interfaces designed for application software development and debugging. The CC3200 LaunchPad also supports the TI Booster Pack interface, allowing the user to interface with a rich repertoire of peripheral systems.

Refer to the CC3200 Launch Pad user manual ([SWRU372](#)) for more details.

Figure 3. CC3200 LaunchPad Platform



2 Foundation SDK – Getting Started

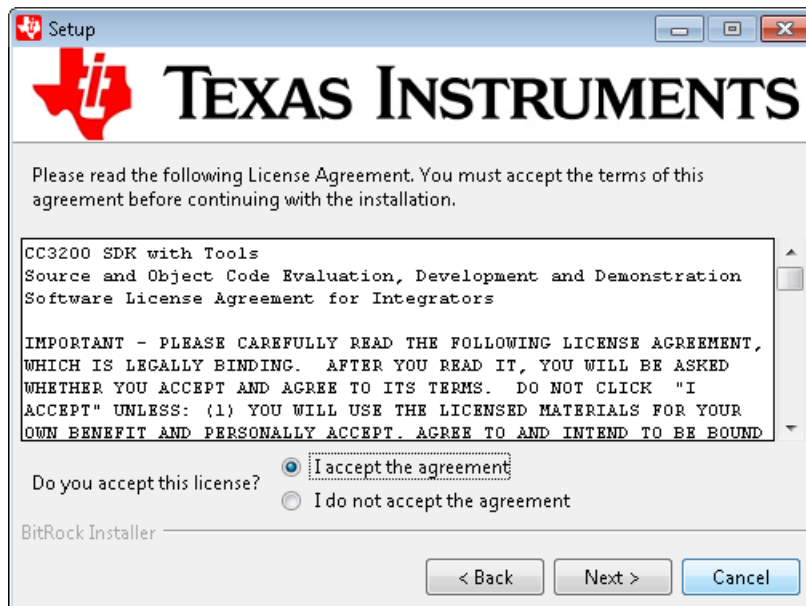
This section familiarizes the user with installation process and the directory structure of CC3200 Foundation SDK.

2.1 Installation

Run the installer by double-clicking the CC3200 SDK installer.

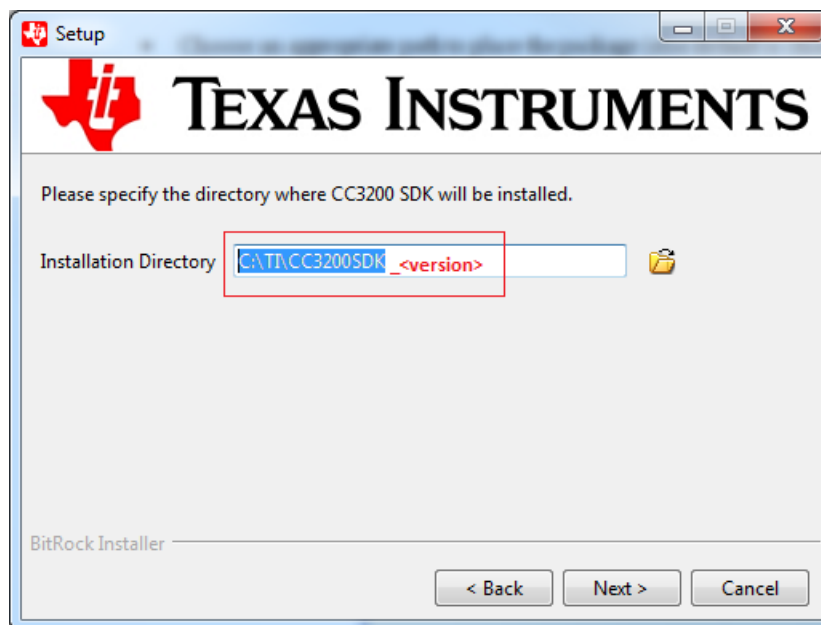
- Read and accept the license agreement to proceed.

Figure 4. CC3200 SDK Installation 1



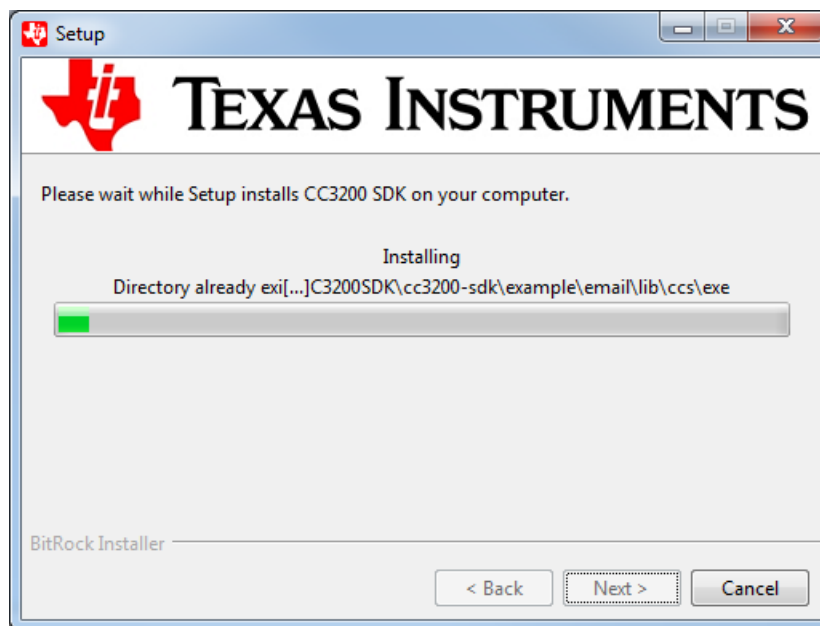
- Choose an appropriate path to place the package (else default is chosen).

Figure 5. CC3200 SDK Installation 2



- Proceed with the installation and click Finish once done.

Figure 6. CC3200 SDK Installation 3



2.2 Package Components Overview

Table 1. Package Contents

Directory Name	Information
docs	<ul style="list-style-type: none"> • CC3200 Programmer's Guide • Documentation for hardware details present in Hardware folder • Documentation for SimpleLink host driver in html format under <i>docs\simplelink_api</i> directory • Application notes for all the sample application present in <i>docs\examples</i> directory. • Peripheral driver library user's guide • Documentation for netapps libraries • SimpleLink OTA Extlib API user's guide
driverlib	<ul style="list-style-type: none"> • Contains the peripheral driver library source files. • The driverlib.a is also provided in the ccs and ewarm directories.
example	<ul style="list-style-type: none"> • Getting Started in STA Mode: Configures the CC3200 in STA mode. It verifies the connection by pinging the client connected to it. • Getting Started in AP Mode: Configures the CC3200 in AP mode. It verifies the connection by pinging the client connected to it. • TCP Socket: Demonstrates the connection scenario and basic TCP functionality. • UDP Socket: Demonstrates the connection scenario and basic UDP functionality. • Scan Policy: Demonstrates the scan-policy settings in the CC3200. • SSL: SSL certificates are designed to provide two principles, privacy and authentication. Privacy is achieved by encryption and decryption, and authentication is achieved by signature and verification. The application demonstrates using a certificate with SSL. • MAC Filters (NWP Filters): The Rx-Filters feature enables the user to define and manage the Rx-filtering process, to reduce the amount of traffic transferred to the host and achieve efficient power management. • File_operations: Demonstrates the use of file-system APIs. • Transceiver_mode: Demonstrates building a proprietary protocol on top of Wi-Fi PHY layer, with the user given full flexibility to build their own packet. The RX Statistics feature inspects the medium in terms of congestion, distance, validation of the RF hardware, and help using the RSSI information. • Provisioning with WPS: Demonstrates the usage of WPS Wi-Fi provisioning with the CC31xx/CC32xx. • Provisioning with SmartConfig: Demonstrates the usage of TI's SmartConfig™ Wi-Fi provisioning technique. • Hib: Showcases hibernate as a power saving tool in a networking context (in this case, as a UDP client). • Get Time: Connects to an SNTP server and requests for time information. • Get Weather: Connects to Open Weather Map and requests for weather data. • Email: Sends emails through SMTP. The email application sends a preconfigured email at the push of a button, or a user-configured email through the CLI. • XMPP: Demonstrates the connection scenario with an XMPP server. • Serial Wi-Fi: Serial Wi-Fi is a capability designed to provide easy, self-contained terminal access behavior, over a UART interface. • Connection Policy: Demonstrates the connection policies in the CC3200. The connection policies determine how the CC3200 connects to the AP. • ENT Wlan: Demonstrates the connection to an enterprise network using the flashed certificate. Certificate is flashed in SFLASH. • HTTP server: Demonstrates the HTTP server capability of the CC3200. • mDNS: Demonstrates the usage of mDNS functionality in the CC3200. The application showcases both mDNS advertise and mDNS listen functionality. • Mode config: Switches the CC3200 LP from STA to AP and vice-versa. • LED Blink Application: Showcases the usage of GPIO DriverLib APIs. The LEDs connected to the GPIOs on the LP are used to indicate the GPIO output. • Timer Demo Application: Showcases the usage of timer DriverLib APIs. This application uses 16-bit timers to generate interrupts, which in turn toggle the state of the GPIO (driving LEDs). • Watchdog Demo Application: Showcases the usage of watchdog timer (WDT) DriverLib APIs. The objective of this application is to showcase the watchdog timer functionality to reset the system when the system fails. • UART Demo Application: Showcases the usage of UART DriverLib APIs. The application demonstrates a simple echo of anything the user types on the terminal.

Table 1. Package Contents (continued)

Directory Name	Information
example	<ul style="list-style-type: none"> • Interrupt Application: Showcases the usage of interrupt DriverLib APIs. This is a sample application to showcase interrupt preemption and tail-chaining capabilities. • I2C Demo: Showcases the usage of I2C DriverLib APIs. It provides a user interface to read-from or write-to the I2C devices on the LaunchPad. • MCU Sleep: Exercises the sleep functionality of the MCU. • uDMA Application: Showcases the usage of UDMA DriverLib APIs. Various DMA mode functionalities are shown in this application. • FreeRTOS Demo Application: Showcases the FreeRTOS features, such as multiple task creation and intertask communication using queues. • AES Demo Application: Showcases the usage of AES Driverlib APIs. Provides a user interface to exercise various AES modes. • DES Demo Application: Showcases the usage of DES Driverlib APIs. Provides a user interface to exercise various DES modes. • CRC Demo Application: Showcases the usage of CRC Driverlib APIs. Provides a user interface to exercise various CRC modes. • SHA-MD5 Demo Application: Showcases the usage of SHA-MD5 Driverlib APIs. Provides a user interface to exercise various SHA-MD5 modes. • ADC Demo Application: Showcases the functionality of the CC3200 ADC module by using the Driverlib APIs. • PWM Demo Application: Showcases general 16-bit pulse-width modulation (PWM) mode feature supported by purpose timers (GPTs). • SD Host Application: Showcases the basic use case of initializing the controller to communicate with the attached card, reading and writing SD card block. • SD Host FatFS Application: Uses the FatFS to provide the block-level read/write access to the SD card, using the SD host controller on the CC3200. • SPI Demo Application: Displays the required initialization sequence to enable the CC3200 SPI module in full duplex 4-wire master and slave modes. • Wi-Fi Audio App: Demonstrates bi-directional audio application on a CC3200 LaunchPad setup. This application requires the audio boosterpack. • Camera Application: Demonstrates the camera feature on the CC3200. The user can invoke the image capture command on the web browser hosting on the CC3200 device. This application requires the camera boosterpack. • UART DMA Application: Showcases use of UART, along with uDMA and interrupts. • Antenna Selection: Gives the option to select an antenna with more signal for APs using a web-browser. • Out of Box Application: Demonstrates how the user can view different demo and SDK web links on their web-browser. • Peer to Peer Application: Demonstrates the Wi-Fi direct feature on the CC3200. • Timer Count Capture: Showcases the count capture feature of the timer to measure the frequency of an external signal. • Idle Profile: Exercises hibernation using power management framework (middleware). • Sensor Profile: Exercises low power modes (LPDS) using Power Management Framework (middleware). • Watchdog System Demo: Illustrates full system recovery using watchdog, including the network subsystem. • TFTP Client: Demonstrates file transfer using TFTP (Trivial File Transfer Protocol). Requires a TFTP server running on a connected device such as a PC or smart phone. • WebSocket Camera: Demonstrates websocket HTTP server functionality by transmitting continuous JPEG frames to a websocket client. This application requires a camera boosterpack and a connected PC or smartphone with a browser supporting HTML 5. • Application Bootloader: Showcases the secondary bootloader operations, to manage updates to application image. • HTTP Client Demo: Illustrates the usage of the HTTP client library, to enable the device as an HTTP client. • Idle Profile (Non OS): Exercises the low power modes (LPDS) using power management framework in a non-OS environment. • MQTT Client: Showcases the device acting as a MQTT client in a fully-functional MQTT network. • MQTT Server: Showcases the device acting as an MQTT server capable of managing multiple local clients, and allowing the local clients to communicate with remote MQTT clients. • Power Measurement: Allows the user to measure the current consumption for various low-power modes.

Table 1. Package Contents (continued)

Directory Name		Information
example		<ul style="list-style-type: none"> • OTA Update: Illustrates over-the-air (OTA) updates of the service pack, user application, and user files. • Dynamic Library Loader: Exercises an approach to enable dynamic loading of an application-binary from non-volatile memory while the program is being executed.
Inc		<ul style="list-style-type: none"> • Contains the register definition header files.
Oslib		<ul style="list-style-type: none"> • Contains the interface file to configure Free-RTOS or TI-RTOS.
Middleware		<ul style="list-style-type: none"> • Contains power management framework to provide a simple infrastructure for developers to create a power aware solution.
NetApps		<ul style="list-style-type: none"> • 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 MQTT (Message Queue Telemetry Transport) client and server library
Simplelink		<ul style="list-style-type: none"> • Contains SimpleLink host driver code.
Simplelink_extlib		<ul style="list-style-type: none"> • Contains the OTA (over the air) library
Documents		<ul style="list-style-type: none"> • Documentation for netapps libraries
Third party	FatFS	<ul style="list-style-type: none"> • Contains the FatFS source files.
	FreeRTOS	<ul style="list-style-type: none"> • Contains the FreeRTOS source files. Current SDK supports FreeRTOS v8.0.1.
Ti_rtos		<ul style="list-style-type: none"> • Contains the Ti RTOS config file and CCS, IAR , GCC projects to support TI-RTOS with all three IDEs. Current SDK supports TI-RTOS v2.15.00.17.
Tools		<ul style="list-style-type: none"> • ccs_patch – Contains the files required for CCS-FTDI-LP connection. • iar_patch – Contains the files required for IAR-FTDI-LP connection. • ftdi - Contains FTDI PC driver. • gcc_scripts - Contains the scripts to use GCC and openocd with CC3200.

2.3 Prerequisite: Tools to be Installed

Table 2. CC3200 Prerequisite

Tools	Remarks	Location
Development Environment		
IAR	IAR version 7.20 onwards must be installed. After the installation, follow the tools\iar_patch\readme.txt to debug over FTDI.	Installation: http://www.iar.com/Products/IAR-Embedded-Workbench/ARM/
Or/and		
CCS	CCS 6.1.1 version and TI v5.2.6 compiler version. After the installation, follow the tools\ccs_patch\readme.txt to debug over FTDI.	Installation: http://www.ti.com/tool/ccstudio
CC3200 Support package in CCSv6.1	Though CCS 6.1.1 provides an option to install this add-on during installation, the user must check and install if a newer version is available.	Refer to Section 5.3.2.1
Or/and		
GCC	To enable CC3200 SDK development on a Linux environment.	Refer to Section 5.3.3
CC32xx PinMux Utility		
CC32xx PinMux Utility.exe	Utility to assign a desired personality to the general purpose pins available at the CC3200 device boundary.	Installation: http://processors.wiki.ti.com/index.php/TI_PinMux_Tool or refer to Section 5.3.2.2

Table 2. CC3200 Prerequisite (continued)

Tools	Remarks	Location
CC32xx Programmer Utility		
Uniflash	Tool to download firmware, application images, and certificates to the CC3200 device.	http://www.ti.com/tool/uniflash
Support Tools		
HyperTerminal or Teraterm	Serial communication tool to communicate over the UART with the CC3200 device.	
Iperf	A useful tool for measuring TCP and UDP bandwidth performance.	
FTDI Driver	FTDI Windows drivers must be installed for a successful connection to the CC3200 LP over USB. This FTDI connection can be used for debugging over JTAG/SWD and communicating over UART.	tools\ftdi

3 Foundation SDK – Components

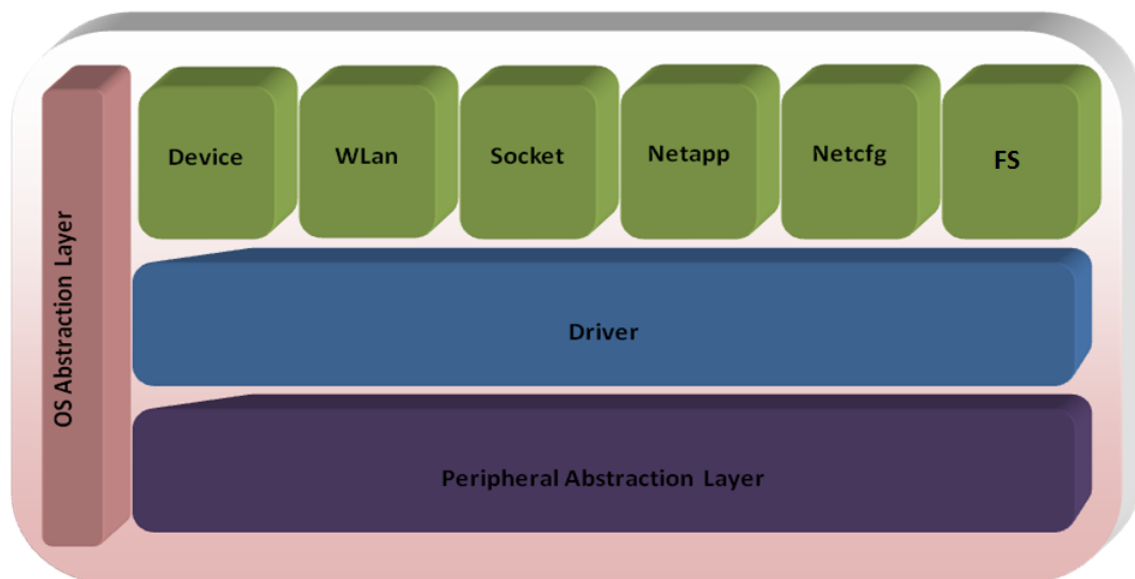
The CC3200 Foundation SDK package includes two main building blocks:

- SimpleLink Library – This library hosts APIs that serve the connectivity features.
- Peripheral Driver Library – This library hosts APIs to access MCU peripherals.

This section also lists the sample and reference applications packaged in the software development kit.

3.1 SimpleLink Component Library

3.1.1 SimpleLink Modular Decomposition

Figure 7. SimpleLink Modular Composition


TI SimpleLink Framework provides a wide set of capabilities, including basic device management through wireless network configuration, 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.

The following list enumerates the different components in the SimpleLink Framework:

Components	Functionality
device	<ul style="list-style-type: none"> Initializes the host Controls the communication with the network processor
wlan	<ul style="list-style-type: none"> Connection to the access point Scan access points Add or remove access point profiles WLAN security
socket	<ul style="list-style-type: none"> UDP/TCP client socket UDP/TCP server socket UDP/TCP Rx/Tx
netapp	<ul style="list-style-type: none"> DNS resolution Ping remote device Address resolution protocol
netcfg	<ul style="list-style-type: none"> IP and MAC address configuration
fs	<ul style="list-style-type: none"> File system read/write

3.1.2 Using the TI SimpleLink Framework

The TI 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_api\programmers_guide.html](#) available in the SDK.

The TI SimpleLink Framework has a ready-to-use port available in the CC3200 Foundation SDK. The source code is also shared if further customization is desired by the developer. The following note describes simple possible customizations and the associated procedure.

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

- **Modify user.h file** – Modify the user.h file that includes the default configurations and adjustments.
- **Select the capabilities set required for your application** – TI has focused on building a set of predefined capability sets that fit most target applications. TI recommends trying and choosing one of these predefined capabilities sets before building a customized set. Once a compatible set is found, skip the rest of this step. The available sets are:
 - # SL_TINY – Compatible with platforms with very limited resources. Provides the best-in-class footprint in terms of code and data consumption.
 - # SL_SMALL – Compatible with most common networking applications. Provides the most common APIs with a balance between code size, data size, functionality, and performance.
 - # SL_FULL – Provides access to all SimpleLink functionalities.
- **Memory management model** – The SimpleLink driver supports two memory models:
 - Static (default)
 - Dynamic

The CC3200 default configuration is static. In the dynamic model, the configuration uses the malloc and free, as defined by the operating system. To define your own memory management, define these interfaces.
- **Asynchronous event handlers routines** – The SimpleLink device generates asynchronous events in certain situations. These asynchronous events can be masked. Provide handler routines to catch these events. If a handler routine was not provided and the event is received, the driver drops this event without any indication of a drop.

- **Interface communication driver** – The CC3200 host driver implements an SPI communication interface. The interface for this communication channel includes four simple access functions:
 1. open
 2. close
 3. read
 4. write

The CC3200, SPI implementation uses DMA to increase the utilization of the communication channel.

- **OS adaptation** – The SimpleLink driver can run on two kinds of platforms:
 - Non-OS / single-threaded (default)
 - Multi-threaded

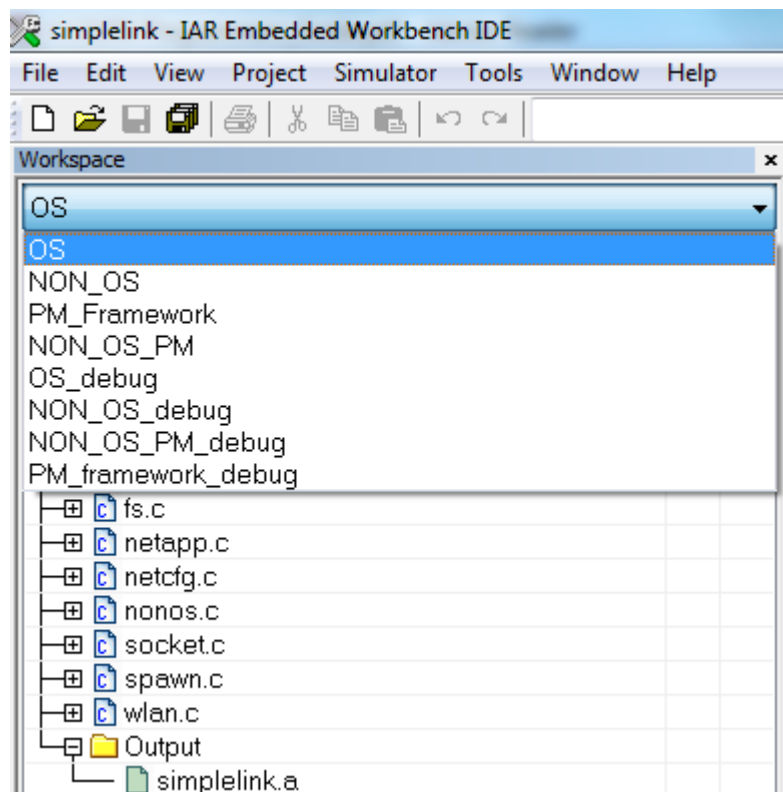
The CC3200 SimpleLink host driver is ported on both non-OS and multi-threaded OS environments. The host driver is made OS-independent by implementing an OS abstraction layer. Reference implementation for OS abstraction is available for FreeRTOS and TI-RTOS.

To work in a multi-threaded environment under a different operating system, provide some 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). In addition, the driver support runs without a dedicated thread allocated solely to the SimpleLink driver. To work in this mode, supply a spawn method that enables functions to run on a temporary context.

3.1.3 Switch Between OS, NON-OS, and Debug Configurations

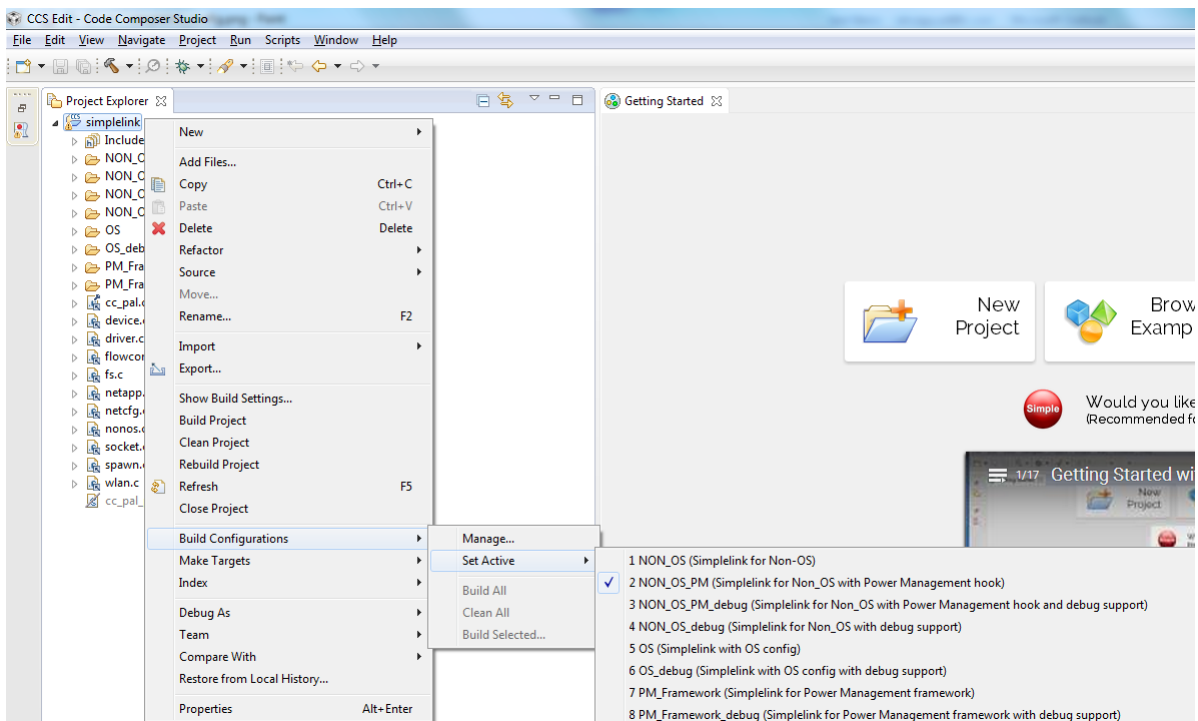
IAR: Choose configuration options from menu *Project->Edit configurations->OS/NON_OS/PM_Framework/NON_OS_PM/OS_debug/NON_OS_debug/PM_Framework_debug/NON_OS_PM_debug*, as indicated in [Figure 8](#).

Figure 8. CC3200 SimpleLink IAR Config Switch



CCS: Choose configuration options from menu *Project->Build Configurations->Set active*, or as indicated in Figure 9:

Figure 9. CC3200 CCS SimpleLink Config Switch



3.1.4 Development versus Deployment Scenario

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

This is required, as 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 gracefully stopped and started again. This results in additional delays and greater overall current consumption. As these additional steps and delay are required only for debugging purposes, they should not be a part of the deployment applications.

For ease of use, the CC3200 SDK latest package provides separate configurations of SimpleLink library for the development (debug) and deployment scenarios. The following four items should be used for the deployment scenario (this does not include the additional steps and delay) as per use-case.

- NON_OS – SimpleLink for NON-OS environment
- OS – SimpleLink for OS environment
- PM_Framework – SimpleLink with power management framework for OS environment
- NON_OS_PM – SimpleLink with power management framework hook

All of these configurations are pre-built 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.

- NON_OS_debug – SimpleLink for NON-OS environment with debug support
- OS_debug – SimpleLink for OS environment with debug support
- PM_Framework_debug – SimpleLink with power management framework for OS environment with debug support

- NON_OS_PM_debug – SimpleLink with power management framework hook with debug support

These configurations are not pre-built; the user must build them first before linking them to their application. Note that this will result 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 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 10](#) and [Figure 11](#).

Figure 10. File Search Path

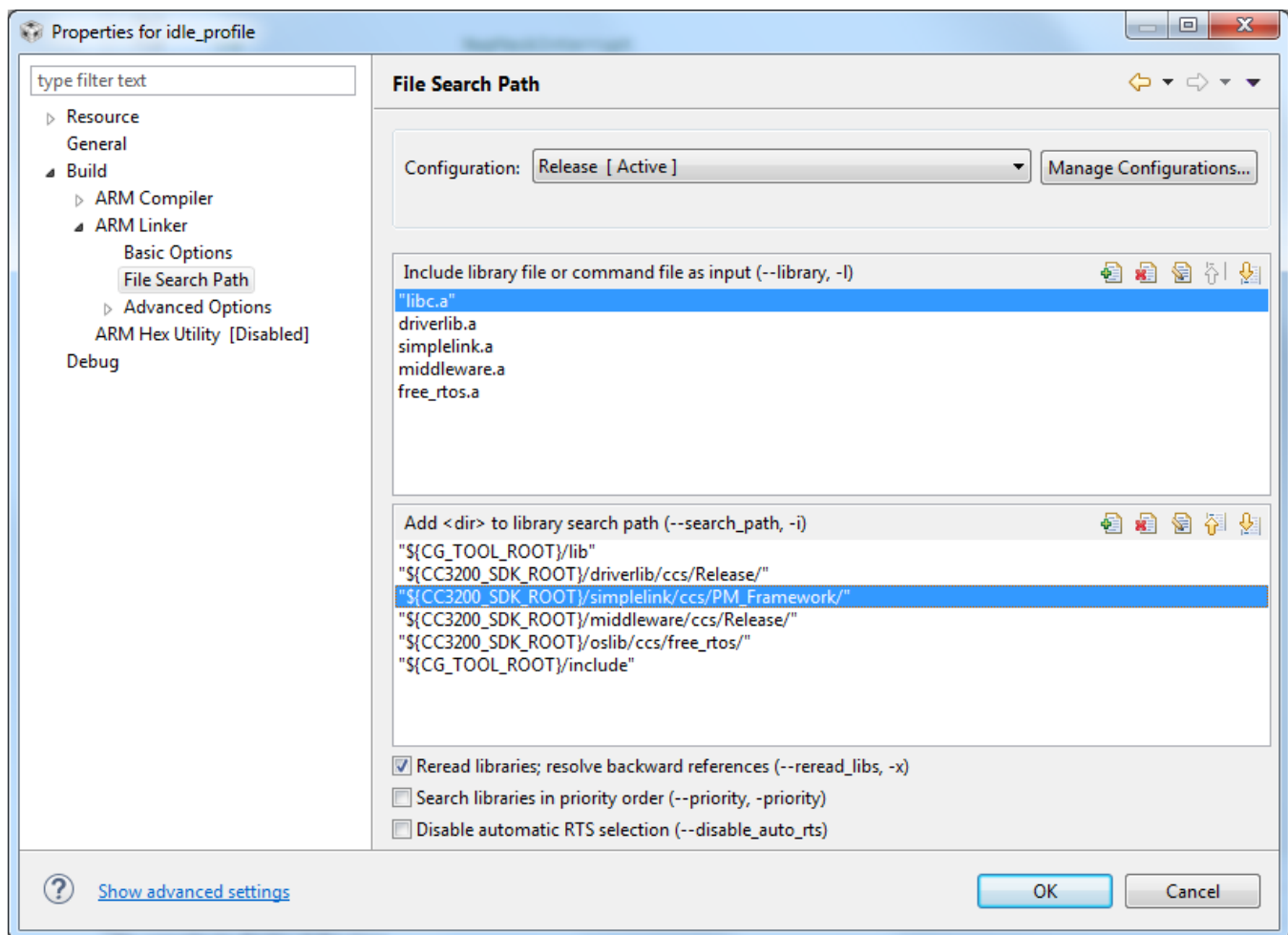
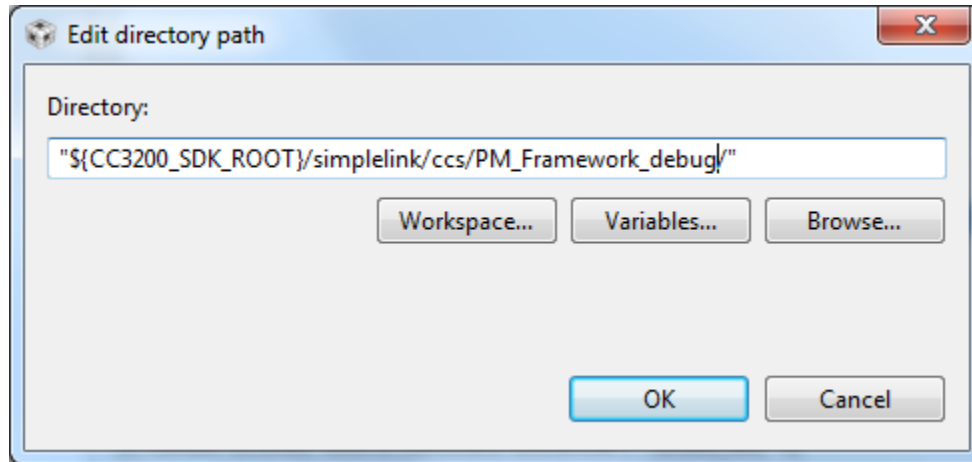
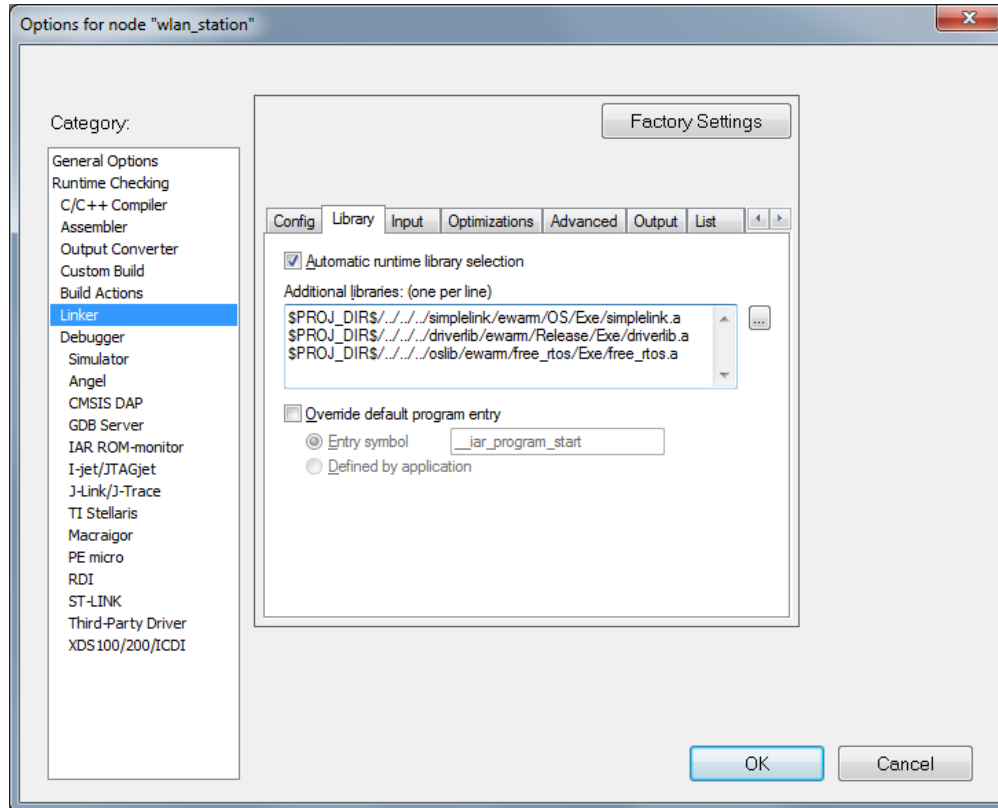
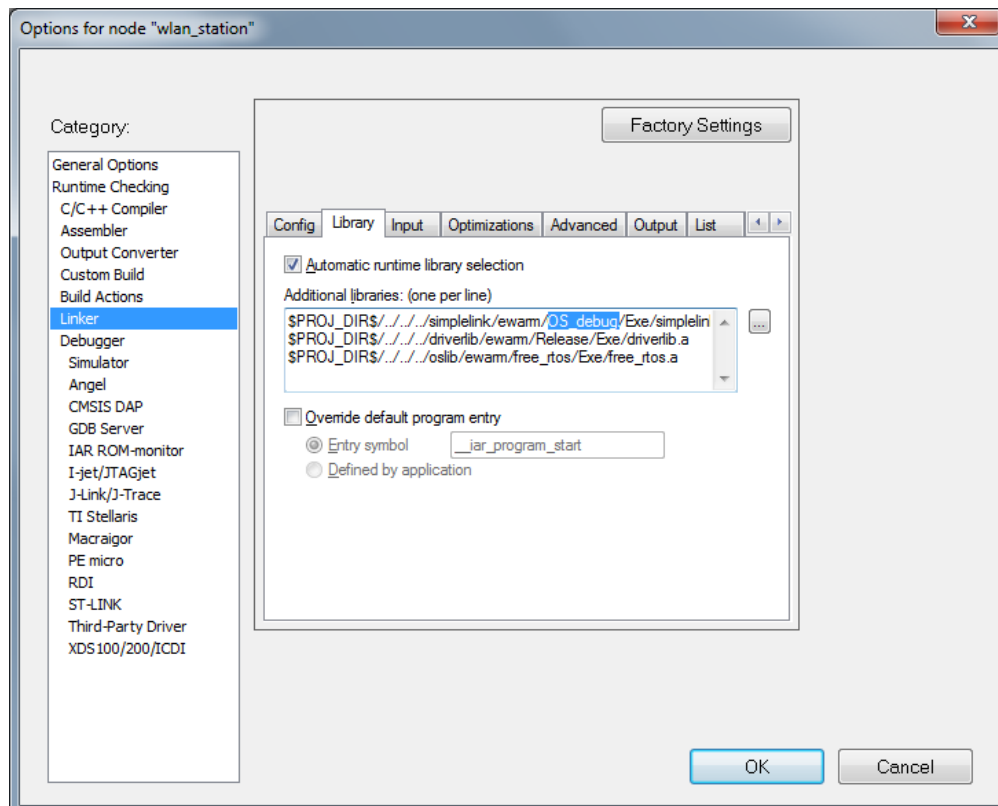


Figure 11. Edit Directory Path



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 12](#) and [Figure 13](#).

Figure 12. Edit SimpleLink Library Path

Figure 13. Edit SimpleLink Library Path


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's Makefile, and navigate to the linking portion.
3. Change the linking code as shown in [Figure 14](#) and [Figure 15](#).

Figure 14. Linking Code Example 1

```
#
# 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}/gpio_if.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/uart_if.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/startup_${COMPILER}.o
${BINDIR}/wlan_station.axf: ${ROOT}/simplelink/${COMPILER}/${BINDIR}/libsimplelink.a
${BINDIR}/wlan_station.axf: ${ROOT}/driverlib/${COMPILER}/${BINDIR}/libdriver.a
${BINDIR}/wlan_station.axf: ${ROOT}/oslib/${COMPILER}/${BINDIR}/FreeRTOS.a
SCATTERgcc_wlan_station=wlan_station.ld
ENTRY_wlan_station=ResetISR
```

Figure 15. Linking Code Example 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}/gpio_if.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/uart_if.o
${BINDIR}/wlan_station.axf: ${OBJDIR}/startup_${COMPILER}.o
${BINDIR}/wlan_station.axf: ${ROOT}/simplelink/${COMPILER}/${BINDIR}/libsimplelink debug.a
${BINDIR}/wlan_station.axf: ${ROOT}/driverlib/${COMPILER}/${BINDIR}/libdriver.a
${BINDIR}/wlan_station.axf: ${ROOT}/oslib/${COMPILER}/${BINDIR}/FreeRTOS.a
SCATTERgcc_wlan_station=wlan_station.ld
ENTRY_wlan_station=ResetISR
```

3.2 Peripheral Driver Library

The CC3200 ROM contains the peripheral driver library (DriverLib) and the boot loader. DriverLib can be utilized by applications to reduce their flash 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_api
- AES_Advanced_Encryption_Standard_api
- Camera_api
- CRC_Cyclic_Redundancy_Check_api
- DES_Data_Encryption_Standard_api
- Flash_api
- GPIO_General_Purpose_InputOutput_api
- HwSpinLock_api
- I2C_api
- I2S_api
- Interrupt_api

- Pin_api
- PRCM_Power_Reset_Clock_Module_api
- Secure_Digital_Host_api
- SHA_Secure_Hash_Algorithm_api
- SPI_Serial_Peripheral_Interface_api
- SysTick_api
- GPT_General_Purpose_Timer_api
- UART_api
- UDMA_Micro_Direct_Memory_Access_api
- Utils_api
- WDT_Watchdog_Timer_api

For detailed information on the APIs and their usage, refer to the document *docs\CC3200 Peripheral Driver Library User's Guide.chm*.

3.3 Reference Applications

The reference applications available as a part of the SDK package are example implementations which demonstrate key features and peripherals supported by the subsystem built around the CC3200 device on the LaunchPad. A brief description of the reference applications are tabulated below. Refer to the readme.txt present in the individual folders for further information. All examples are broadly divided into two types: the network reference and the MCU-only reference examples.

3.3.1 Network Reference Examples for the CC3200 LP

Application / Demo	Description	Peripheral/Feature exercised
Getting started with wLAN Station	Showcases the device's capability as a station in a typical networking system.	Networking (STA mode)
Getting started with wLAN AP	Showcases the device's capability as an AP in a typical networking system.	Networking (AP mode)
TCP Socket Application	Showcases the device's communication over network using TCP protocols.	Networking (Basics)
WLAN Scan Policies Application	Sets scan policy and enables the scan in the device.	Networking (Scan policies)
UDP Socket Application	Showcases the device's communication over network using TCP protocols.	Networking (Basics)
SSL Demo Application	Showcases SSL implementation on the CC3200 device.	Networking (SSL)
NWP Filter Application	Showcases the Rx-Filter feature on the CC3200 device.	Networking (MAC Filters)
File Operations Application	Showcases the file operation on the serial flash of the device.	SFlash (File operations)
Transceiver mode Application	Inspects the medium in terms of congestion and distance, validates the RF hardware, and help using the RSSI information.	Networking (Raw sockets), GPIO, UART, Timer
Provisioning - Smart Config	Demonstrates how to associate and connect the CC31xx/CC32xx to any access point.	Networking (Provisioning)
Hibernate Application	Showcases hibernate as a power saving tool in a networking context (in this case, a UDP client).	Networking, Low power modes (HIB), UART, GPIO, Timer
Info Center-Get Time Application	Connects to an SNTP server and requests for time information.	Networking (Internet)
Info Center-Get Weather Application	Connects to Open Weather Map and requests for weather data.	Networking (Internet)
Email Application	Sends emails through SMTP.	Networking, GPIO, UART, Timer
XMPP Reference Application	Demonstrates the connection scenario with an XMPP server.	Networking (Internet)
Provisioning-WPS Application	Demonstrates how to use WPS Wi-Fi provisioning with the CC3200.	Networking (Provisioning), GPIO

Application / Demo	Description	Peripheral/Feature exercised
Mode-Configuration Application	Configures the device either to a station or an AP mode.	Networking (STA/AP mode)
Serial Wi-Fi Application	Serial Wi-Fi is a capability designed to provide easy, self-contained terminal access behavior over a UART interface.	Networking
Connection Policy Application	Demonstrates the connection policies in the CC3200. The connection policies determine how the CC3200 is connected to AP.	Networking (STA Mode)
ENT WLAN Application	Demonstrates the connection to an enterprise network using the flashed certificate. Certificate is flashed in SFLASH.	Networking (STA mode)
HTTP Server Application	Demonstrates the HTTP server capability of the CC3200.	Networking (STA Mode)
mDNS Application	Demonstrates the usage of mDNS functionality in the CC3200. The application showcases both mDNS advertise and mDNS listen functionality.	Networking (STA Mode) , UART
Out of Box Application	Demonstrates how to view different demo and SDK web links on a web-browser.	Networking (AP/STA mode), I2C, GPIO
Wi-Fi Audio Application	Demonstrates bi-directional audio application on a CC3200 LaunchPad setup. The system is comprised of two LPs (in STA mode). Audio is streamed from one LP and rendered on another LP over Wi-Fi. This application requires the audio boosterpack.	Networking (STA/AP mode)
Antenna Selection Application	Allows the user to select the antenna with the highest signal for APs using a web-browser.	Networking (AP mode)
Camera Application	Demonstrates the camera feature on the CC3200 device. This application requires the camera boosterpack.	Networking (AP mode)
Peer to Peer Application	Demonstrates the Wi-Fi direct feature on the CC3200 device.	Networking (p2p mode)
Idle Profile	Exercises hibernation using power management framework (middleware).	Networking (STA Mode)
Sensor Profile	Exercises low-power modes (LPDS) using power management framework (middleware).	Networking (STA Mode)
File Download Application	Demonstrates file downloading from the web server and stores it to the device memory feature on the CC3200 device.	Networking (STA Mode)
Watchdog System Demo	Illustrates a full system recovery using watchdog, including the network subsystem.	Networking (STA Mode)
TFTP Client	Demonstrates file transfer using TFTP (Trivial File Transfer Protocol). Requires a TFTP server running on a connected device, such as a PC or smartphone.	Networking (STA Mode)
WebSocket Camera	Demonstrates websocket HTTP server functionality by transmitting continuous JPEG frames to a websocket client. This application requires camera boosterpack and a connected PC or smartphone with a browser supporting HTML 5.	Networking (AP Mode)
HTTP Client Demo	Illustrates the usage of the HTTP client library to enable the device as an HTTP client.	Networking (STA mode)
Idle Profile (Non OS)	Exercises the low-power modes (LPDS) using power management framework in a non-OS environment.	Networking (STA mode)
MQTT Client	Showcases the device acting as a MQTT client in a fully-functional MQTT network.	Networking (STA mode)
MQTT Server	Showcases the device acting as an MQTT server capable of managing multiple local clients, and allowing the local clients to communicate with remote MQTT clients.	Networking (STA mode)
OTA Update	Illustrates the over-the-air (OTA) update of the service pack, user application, and user files.	Networking (STA mode)
Power Measurement	Allows the user to measure the current consumption for various low-power modes.	Networking (STA mode)

3.3.2 MCU-Only Reference Examples for the CC3200 LP

Application / Demo	Description	Peripheral/Feature exercised
LED Blink Application	Showcases the blinking feature of available LEDs connected over GPIO on LP.	GPIO
Timer Demo Application	Showcases the usage of 16-bit timers to generate interrupts, which toggle the state of the GPIO.	Timer, GPIO, UART
Watchdog Demo Application	Showcases the watchdog timer functionality to reset the system whenever the system fails.	WDT, GPIO, UART
UART Demo Application	Showcases the use of UART.	UART
Interrupt Demo Application	Showcases interrupt preemption and tail-chaining capabilities.	NVIC, UART
I2C Demo	Showcases I2C read/write/read from features on the CC3200 device.	I2C, UART
MCU Sleep	Exercises the sleep functionality of the MCU.	Low-power mode (Sleep), UART
uDMA Application	Showcases different DMA modes of transfer.	uDMA, UART
Autorun non-OS Application	Showcases the basic packet send and receive functionality of the CC3200 in a non-OS environment.	0.2
AES Demo Application	Showcases the AES encryption feature on the CC3200 device.	Crypto, UART
DES Demo Application	Showcases the DES encryption feature on the CC3200 device.	Crypto, UART
CRC Demo Application	Showcases the CRC feature on the CC3200 device.	Crypto, UART
FreeRTOS Demo Application	Showcases the FreeRTOS features, such as multiple task creation and inter-task communication using queues.	UART
SHA-MD5 Demo Application	Showcases the SHA-MD5 hash algorithm on the CC3200 device	Crypto, UART
ADC Demo Application	Showcases the functionality of the CC3200 ADC module by using the Driverlib APIs.	ADC, UART
PWM Demo Application	Showcases the PWM mode of the CC3200 general purpose timers (GPTs). The GPTs support a 16-bit pulse-width modulation (PWM) mode with software-programmable output inversion of the PWM signal.	Timer, GPIO
SDHost Demo Application	Showcases the functionality of the SDHost module in the CC3200. The secure digital host (SD host) controller on the CC3200 provides an interface to standard SD memory cards in 1-bit transfer mode, and handles the SD protocol and data packing at transmission level with minimum CPU intervention.	UART, SDHOST
SDHost FatFS Demo Application	Uses the FatFS to provide the block level read/write access to the SD card, using the SD host controller on the CC3200.	UART, SDHOST
SPI Demo Application	Shows the required initialization sequence to enable the CC3200 SPI module in full duplex 4-wire master and slave modes.	UART, SPI
UART dma Application	Showcases the use of UART along with uDMA and interrupts.	UART, DMA
Timer Count Capture Application	Showcases the timer count capture feature to measure frequency of an external signal.	TIMER
Application Bootloader	Showcases the secondary bootloader operations to manage updates to application image.	SPI for serial flash access
Dynamic Library Loader	Exercises an approach to enable dynamic loading of an application-binary from non-volatile memory while the program is being executed.	

3.4 CC3200 PinMux Utility

The CC3200 pinmux utility provides a convenient interface to select the personality of the general purpose pins available at the CC3200 device boundary. The tool generates the source files based on the information selected, and can be directly included in the project. Refer to the [CC3200 PinMux tool wiki page](#) for further details.

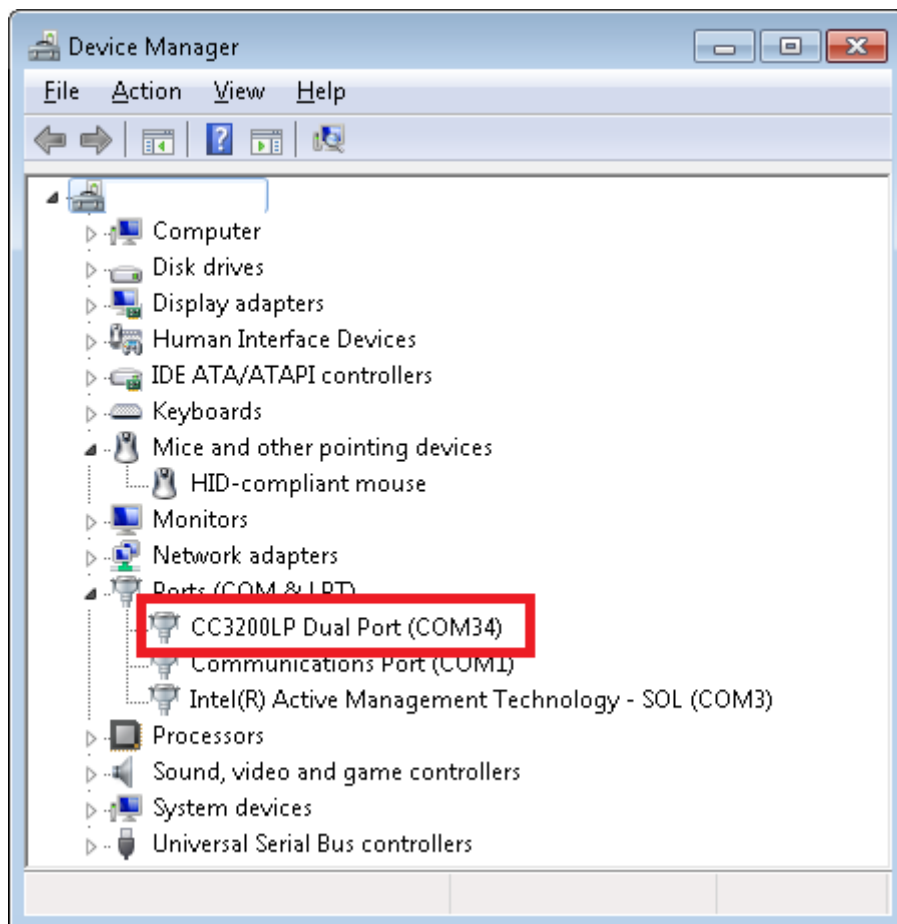
Refer to [Section 5.3.2.2](#) for how to get the new TI-Pinmux tool.

4 Getting Started With the CC3200 LaunchPad

To start with the CC3200 LaunchPad:

- Connect the CC3200 LaunchPad to the PC.
- As the FTDI driver is auto-installed from the SDK installer, the device will enumerate with one com port (CC3200LP Dual Port), as shown in [Figure 16](#).

Figure 16. CC3200 Programmer Guide Device Manager



- To configure the device into SWD/JTAG mode, refer to the *cc3200-sdk\docs\hardware\CC3200-LP_User's guide.pdf*.

5 Foundation SDK – Development Flow

This section familiarizes the developer with the typical development flow using the building blocks hosted in Foundation SDK, and emphasizes more of the network aspects of the CC3200. For this purpose, a suite of Getting Started applications are presented in the SDK. Start with a comprehensive description of these applications, build and execute procedure with the IDEs, and finally burn the application image in the non-volatile storage.

The SDK contains five simple network applications to demonstrate the connection and packet-handling functionality. These applications use the SimpleLink APIs to demonstrate the functionality. The source in these applications is modular, and can be referred or re-used by the developer.

	Application	Description
1.	Getting started with WLAN Station	Reference application to use the CC3200 in STA mode
2.	Getting started with WLAN AP	Reference application to use the CC3200 in AP (Access Point) mode
3.	TCP Socket Application	Reference application showcasing the TCP server and client functionality
4.	UDP Socket Application	Reference application showcasing the UDP server and client functionality
5.	Raw Socket (Transceiver Mode Application)	Reference application showcasing the Raw socket functionality

5.1 Simple Networking Applications

Any SimpleLink API (sl_*) must be invoked only after the sl_Start() API has completed. This function starts the SimpleLink (networking) device. This function also initializes the communication interface.

5.1.1 Getting Started With WLAN Station

5.1.1.1 Application Details

This application shows the CC3200 device as a station in a simple network. Developers and users can refer the function or re-use them while writing a new application. The device connects to an AP (access point), with AP configurations stored in the form of macros in the application. If the connection is successful, it tries to get the IP address of www.ti.com, then ping to the IP address. Zero is the expected return value. A different return code indicates that the internet connection is not available or that the ping was not successful. The application uses LEDs to indicate the test results; RED indicates an AP connection, GREEN indicates ping to AP, and ORANGE indicates a ping to www.ti.com.

Security Macros

```
#define SSID_NAME "cc3200demo"
#define SECURITY_TYPE SL_SEC_TYPE_OPEN
#define SECURITY_KEY ""
```

5.1.1.2 Source Files Briefly Explained

The application source can be found in *example\getting_started_with_wlan_station*.

- main.c – Main file creates the SimpleLink task which handles most of the network related operations, while a WlanStationMode task makes calls to the network-related APIs of the SimpleLink library.
- startup_ewarm.c – IAR workbench-specific vector table implementation for interrupts
- pinmux.c – Contains the configurations to mux the device pins
- gpio_if.c – Common interface file for LED use
- uart_if.c – Common interface file for UART prints

The ewarm folder contains the IAR workspace. The ccs folder contains the CCS Project, the driverlib folder contains all the driver files, the oslib folder contains the project to build the TI-RTOS/Free-RTOS library, and the simplelink folder contains SimpleLink host files.

5.1.1.3 Code Flow Connection

```

void WlanStationMode( void *pvParameters )
{
    ...
    //Start the SimpleLink

    lRetVal = sl_Start(0, 0, 0);
    ...

    // Connecting to WLAN AP

    lRetVal = WlanConnect(); /* ...

                                lRetVal = sl_WlanConnect(...);

                                ...

                                // Wait for WLAN Event

                                while((!IS_CONNECTED(g_ulStatus)) ||
                                    (!IS_IP_ACQUIRED(g_ulStatus))) { ... }

                                ...

                                */

    ...
    // Checking the Lan connection by pinging to AP gateway
    lRetVal = CheckLanConnection(); /* ...

                                // Check for LAN connection

                                lRetVal = sl_NetAppPingStart(...);

                                ...

                                // wait for Ping report event

                                while(!IS_PING_DONE(g_ulStatus)) { ... }

                                */

    ...
    // Checking the internet connection by pinging to external host
    lRetVal = CheckInternetConnection(); /* ...

                                // Get external host IP address

                                lRetVal = sl_NetAppDnsGetHostByName(...);

                                ...

                                // Try to ping HOST_NAME

                                lRetVal = sl_NetAppPingStart(...);

                                ...

                                // Wait for Ping done event

                                while(!IS_PING_DONE(g_ulStatus)) { ... }

                                */

    ...
}

```

Using the CC3200 as a STA is a three step process.

1. Start the SimpleLink by calling the `sl_Start()` API.
2. Connect to the AP by calling the `sl_WlanConnect()` API.
3. Ping to the AP and external host by calling the `sl_NetAppPingStart()` API.

Refer to the `main.c` file of the reference application for more details.

5.1.1.4 Usage

1. Run the application (*getting_started_with_wlan_sta*) from IAR/CCS, or flash the bin file to the device.
2. The device switches to STA mode if it is in the other mode.
3. The device tries to connect to open a pre-defined AP (`cc3200demo`). The red LED glows upon a successful connection.
4. The device pings to AP. If the ping is successful, the green LED glows.
5. The device checks for an internet connection by pinging to `www.ti.com`. If this ping is successful, the orange LED glows.

5.1.2 Getting Started with WLAN AP

5.1.2.1 Application Details

This application aims to exhibit the CC3200 device as an AP. Developers and users can refer or re-use the function while writing a new application. The device comes up as an AP (access point), then waits for a station to connect to it. If the connection is successful, it pings to that station. Zero is the expected return value. A different return code indicates that the ping to the station was unsuccessful.

5.1.2.2 Source Files Briefly Explained

- `main.c` – Main file creates the SimpleLink task which handles most of the network related operations, while a `WlanStationMode` task makes calls to the network-related APIs of the SimpleLink library.
- `startup_ewarm.c` – IAR workbench-specific vector table implementation for interrupts
- `pinmux.c` – Contains the configurations to mux the device pins
- `uart_if.c` – Common interface file for UART prints

The `ewarm` folder contains IAR workspace. The `ccs` folder contains CCS Project, the `driverlib` folder contains all the driver files, the `oslib` folder contains the project to build the TI-RTOS/Free-RTOS library, and the `simplelink` folder contains SimpleLink host files.

5.1.2.3 Code Flow Connection

```
void WlanAPMode( void *pvParameters )
{
    ...
    lRetVal = sl_Start(NULL,NULL,NULL);
    ...
    // Configure the networking mode ssid name (for AP mode)
    ConfigureMode(lRetVal) ;          /* ...
                                     lRetVal = sl_WlanSetMode(ROLE_AP);
                                     // set SSID name
                                     lRetVal = sl_WlanSet( ... );
                                     ...
                                     */

    while(!IS_IP_ACQUIRED(g_ulStatus))
    {
        //looping till ip is acquired
    }

    // get network configuration
    lRetVal = sl_NetCfgGet(SL_IPV4_STA_P2P_CL_GET_INFO,&ucDHCP,&len, (unsigned char *)&ipV4);
    while(!IS_IP_LEASED(g_ulStatus))
    {
        //waiting for the STA to connect
    }

    // Ping to connected client
    iTestResult = PingTest(ulIpAddr);
    ...
}
```

Using the CC3200 as an AP is a two step process:

- (a) Start the SimpleLink by calling the sl_Start() API.
- (b) Wait until the device gets an IP address.

After the device comes up in AP mode, follow these steps to ensure the device can act as an AP:

1. Wait for a station to connect to the device (the user must connect a machine to the device).
2. Ping the station (machine)

Refer to the main.c file of the reference application for more details.

NOTE: If the device is not able to ping to the connected machine, try disabling the antivirus on the machine.

5.1.2.4 Usage

1. Run the application (*getting_started_with_wlan_ap*) from IAR/CCS or flash to the device.
2. Application will switch to AP mode if it is not in AP mode.
3. After the client connects to the device, the device (AP) pings the client and prints the result over UART.
4. All results can be viewed on the terminal screen.
5. Observe the execution flow to understand the result.

5.1.3 TCP Socket Application

5.1.3.1 Application Details

This application illustrates how to use the device as a client or server for TCP communication. Developers and users can refer or re-use the function while writing new applications. The device connects to an AP (access point), with the SSID for AP stored as a macro in the application. Initially, the application implements a TCP client and sends 1000 TCP packets to a socket address, port number, and IP address specified as macros. Zero is the expected return code. A different return code indicates that a socket error has occurred. The default setting is defined as in the following macros, which can be changed either in the source code or at runtime.

```
#define SSID_NAME      "cc3200demo"
#define IP_ADDR        0xc0a8006E
#define PORT_NUM       5001
#define TCP_PACKET_COUNT 1000
```

5.1.3.2 Source Files Briefly Explained

- main.c – Main file calls SimpleLink APIs to connect to the network, creates a socket, and uses it to communicate over TCP by acting as a TCP client or server.
- pinmux.c – Pinmux file to mux the device to configure a UART peripheral.
- startup_ccs.c – CCS-specific vector table implementation for interrupts
- uart_if.c – Common interface file for UART prints
- udma_if.c – Common interface file for uDMA functionalities
- startup_ewarm.c – IAR workbench-specific vector table implementation for interrupts

The ewarm folder contains IAR workspace. The ccs folder contains CCS Project, the driverlib folder contains all the driver files, and the simplelink folder contains SimpleLink host files.

5.1.3.3 Code Flow Connection

```
void main()
{
    ...
    // Starting SimpleLink
    lRetVal = sl_Start(0, 0, 0);
    ...
    lRetVal = WlanConnect(); /* ...
                            lRetVal = sl_WlanConnect(...);
                            ...
                            // Wait for WLAN Event
                            while(!IS_CONNECTED(g_ulStatus)) ||

                                (!IS_IP_ACQUIRED(g_ulStatus))) { ... }
                            ...
                            */
    ...
    /* following calls depend on user's input at runtime */
    .

    // Before proceeding, please make sure to have a server waiting on PORT_NUM
    lRetVal = BsdTcpClient(PORT_NUM);

    // After calling this function, you can start sending data to CC3100 IP
    // address on PORT_NUM
    lRetVal = BsdTcpServer(PORT_NUM);
    ...
}
```

TCP Client

```
int BsdTcpClient(unsigned short usPort)
{
    ...
    //Open a socket with standard parameters
    iSockID = sl_Socket(SL_AF_INET,SL_SOCKET_STREAM, 0);
    if( iSockID < 0 )
    {
        // error
        ASSERT_ON_ERROR(TCP_CLIENT_FAILED);
    }

    //Connect to the server IP and port number
    iStatus = sl_Connect(iSockID, ( S1SockAddr_t *)&sAddr, iAddrSize);
    if( iStatus < 0 )
    {
        // error
        ASSERT_ON_ERROR(TCP_CLIENT_FAILED);
    }
    ...
    //Send packet using the sl_Send API call

    iStatus = sl_Send(iSockID, g_cBsdBuf, sTestBufLen, 0 );
    if( iStatus < 0 )
    {
        // error
        ASSERT_ON_ERROR(TCP_CLIENT_FAILED);
    }
    ...
    //Close the socket
    sl_Close(iSockID);

    SUCCESS
}
```

Sending the TCP packets is a four step process.

1. Open the socket
2. Connect to the server
3. Send the packets
4. Close the socket

TCP Server

```
int BsdTcpServer(unsigned short usPort)
{
    ...
    iSockID = sl_Socket(SL_AF_INET,SL_SOCKET_STREAM, 0);
    if( iSockID < 0 )
    {
        // error
        ASSERT_ON_ERROR(TCP_SERVER_FAILED);
    }

    .
    .
    .
    iStatus = sl_Bind(iSockID, (SlSockAddr_t *)&sLocalAddr, iAddrSize);
    if( iStatus < 0 )
    {
        // error
        ASSERT_ON_ERROR(TCP_SERVER_FAILED);
    }

    iStatus = sl_Listen(iSockID, 0);
    if( iStatus < 0 )
    {
        ASSERT_ON_ERROR(TCP_SERVER_FAILED);
    }

    iStatus = sl_SetSockOpt(iSockID, SL_SOL_SOCKET, SL_SO_NONBLOCKING,
                           &lNonBlocking, sizeof(lNonBlocking));
    ...
    iNewSockID = SL_EAGAIN;

    while( iNewSockID < 0 )
    {
        iNewSockID = sl_Accept(iSockID, ( struct SlSockAddr_t *)&sAddr,
                               (SlSocklen_t*)&iAddrSize);
        if( iNewSockID == SL_EAGAIN )
        {
            UtilsDelay(10000);
        }
        else if( iNewSockID < 0 )
        {
            // error
            ASSERT_ON_ERROR(TCP_SERVER_FAILED);
        }
    }

    iStatus = sl_Recv(iNewSockID, g_cBsdBuf, iTestBufLen, 0);
    if( iStatus <= 0 )
    {
        // error
        ASSERT_ON_ERROR(TCP_SERVER_FAILED);
    }

    sl_Close(iNewSockID);
    sl_Close(iSockID);

    SUCCESS
}
```

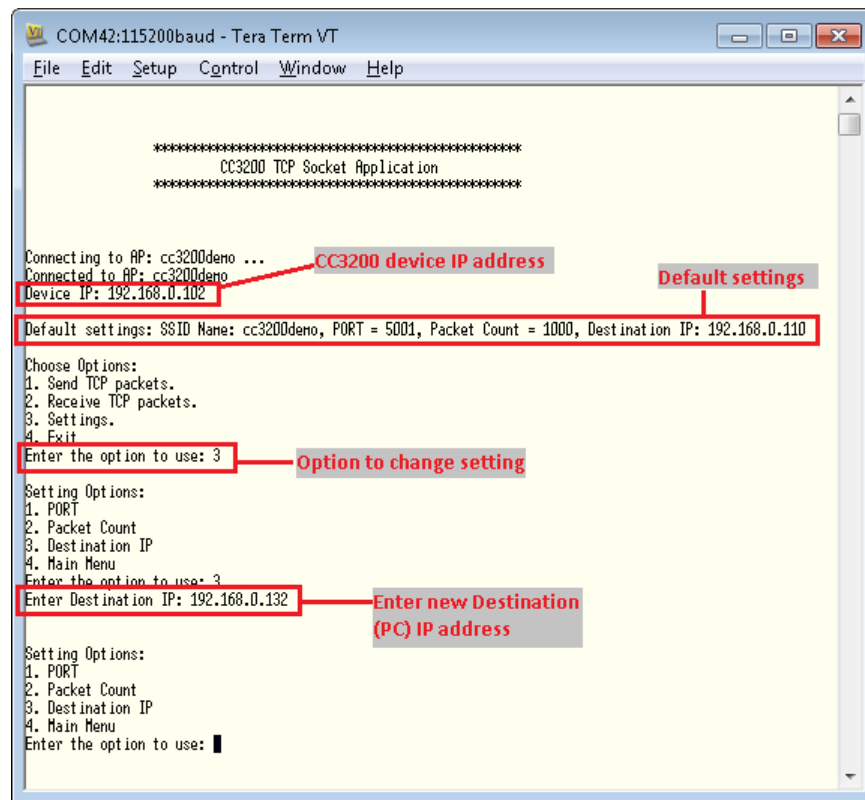

To receive TCP packets from a TCP client:

1. Open the socket
2. Create a TCP server
3. Listen for connection
4. Accept a connection
5. Receive packets
6. Close the socket

5.1.3.4 Usage

1. Setup a serial communication application (such as HyperTerminal or TeraTerm). For detailed information, visit the Setup Terminal on the host PC. The settings are:
 - Port: Enumerated COM port (CC3200LP Dual port)
 - Baud rate: 115200
 - Data: 8 bit
 - Parity: None
 - Stop: 1 bit
 - Flow control: None
2. Run the application (tcp_socket) from IAR/CCS, or flash the bin file to device.
3. Connect a PC to the AP connected to the device.
4. Get the IP address of the PC and fill this value for the IP_ADDR macro, or change the setting as specified in [Figure 17](#):

Figure 17. TCP Socket Terminal



5. Change the other setting (port, SSID name, packet count) as required.

6. Choose the options:
 - Send TCP packets
 - Receive TCP packets
7. After selecting one of the above options, run the iperf command on the PC command prompt as given in the TeraTerm/HyperTerminal screen.
8. Observe the execution flow to understand the results.

Note: Disable PC anti-virus while running iperf.

5.1.4 UDP Socket Application

5.1.4.1 Application Details

This application illustrates how to use the device as a client or server for UDP communication. Developers and users can refer or re-use the function while writing new applications. The device connects to an AP (access point), with the SSID for the AP stored as a macro in the application. Initially, the application implements a UDP client and sends 1000 UDP packets to a socket address, port number, and IP address specified as macros. Zero is the expected return code. A different return code indicates that a socket error has occurred. The default setting is defined in the following macros, changed either in the source code or at runtime.

```
#define SSID_NAME      "cc3200demo"
#define IP_ADDR        0xc0a8006E
#define PORT_NUM       5001
#define UDP_PACKET_COUNT 1000
```

5.1.4.2 Source Files Briefly Explained

The application source can be found in *example\udp_socket*.

- main.c – Main file calls SimpleLink APIs to connect to the network, creates a socket, and uses it to communicate over UDP by acting as a UDP client or server.
- pinmux.c – Pinmux file to mux the device to configure the UART peripheral.
- startup_ccs.c – CCS-specific vector table implementation for interrupts
- uart_if.c – Common interface file for UART prints
- udma_if.c – Common interface file for uDMA functionalities
- startup_ewarm.c – IAR workbench-specific vector table implementation for interrupts

The ewarm folder contains IAR workspace. The ccs folder contains CCS Project, the driverlib folder contains all the driver files, the oslib folder contains the project to build free_rtos library, the third_party folder contains FreeRTOS files, and the simplelink folder contains SimpleLink host files.

5.1.4.3 Code Flow Connection

```
void main()
{
    ...
    // Starting SimpleLink
    lRetVal = sl_Start(0, 0, 0);
    ...
    lRetVal = WlanConnect(); /* ...
                           lRetVal = sl_WlanConnect(...);
                           ...
                           // Wait for WLAN Event
                           while((!IS_CONNECTED(g_ulStatus)) || (!IS_IP_ACQUIRED(g_ulStatus)))
    { ... }

    ...
    */
    ...

    /* following calls depend on user's input at runtime */
    // Before proceeding, please make sure to have a server waiting on PORT_NUM
    lRetVal = BsdUdpClient(PORT_NUM);

    // After calling this function, you can start sending data to CC3200 IP
    // address on PORT_NUM
    lRetVal = BsdUdpServer(PORT_NUM);
    ...
}
```

UDP Client

```
int BsdUdpClient(unsigned short usPort)
{
    ...
    //Open a socket with standard parameters
    iSockID = sl_Socket(SL_AF_INET,SL_SOCKET_DGRAM, 0);
    if( iSockID < 0 )
    {
        // error
        ASSERT_ON_ERROR(UCP_CLIENT_FAILED);
    }

    //Send packet using the sl_Send API call

    iStatus = sl_SendTo(iSockID, g_cBsdBuf, sTestBufLen, 0,
                       ( S1SockAddr_t *)&sAddr, iAddrSize);
    if( iStatus <= 0 )
    {
        // error
        ASSERT_ON_ERROR(UCP_CLIENT_FAILED);
    }

    //Close the socket
    sl_Close(iSockID);

    SUCCESS
}
```

Sending the UDP packets is a three step process.

1. Open the socket
2. Send the packets
3. Close the socket

UDP Server

```
int BsdUdpServer(unsigned short usPort)
{
    ...
    iSockID = sl_Socket(SL_AF_INET,SL_SOCKET_STREAM, 0);
    if( iSockID < 0 )
    {
        // error
        ASSERT_ON_ERROR(UCP_SERVER_FAILED);
    }

    ...
    iStatus = sl_Bind(iSockID, (SlSockAddr_t *)&sLocalAddr, iAddrSize);
    if( iStatus < 0 )
    {
        // error
        ASSERT_ON_ERROR(UCP_SERVER_FAILED);
    }

    iStatus = sl_RecvFrom(iNewSockID, g_cBsdBuf, iTestBufLen, 0, &sAddr, &iAddrSize);

    sl_Close(iSockID);

    SUCCESS
}
```

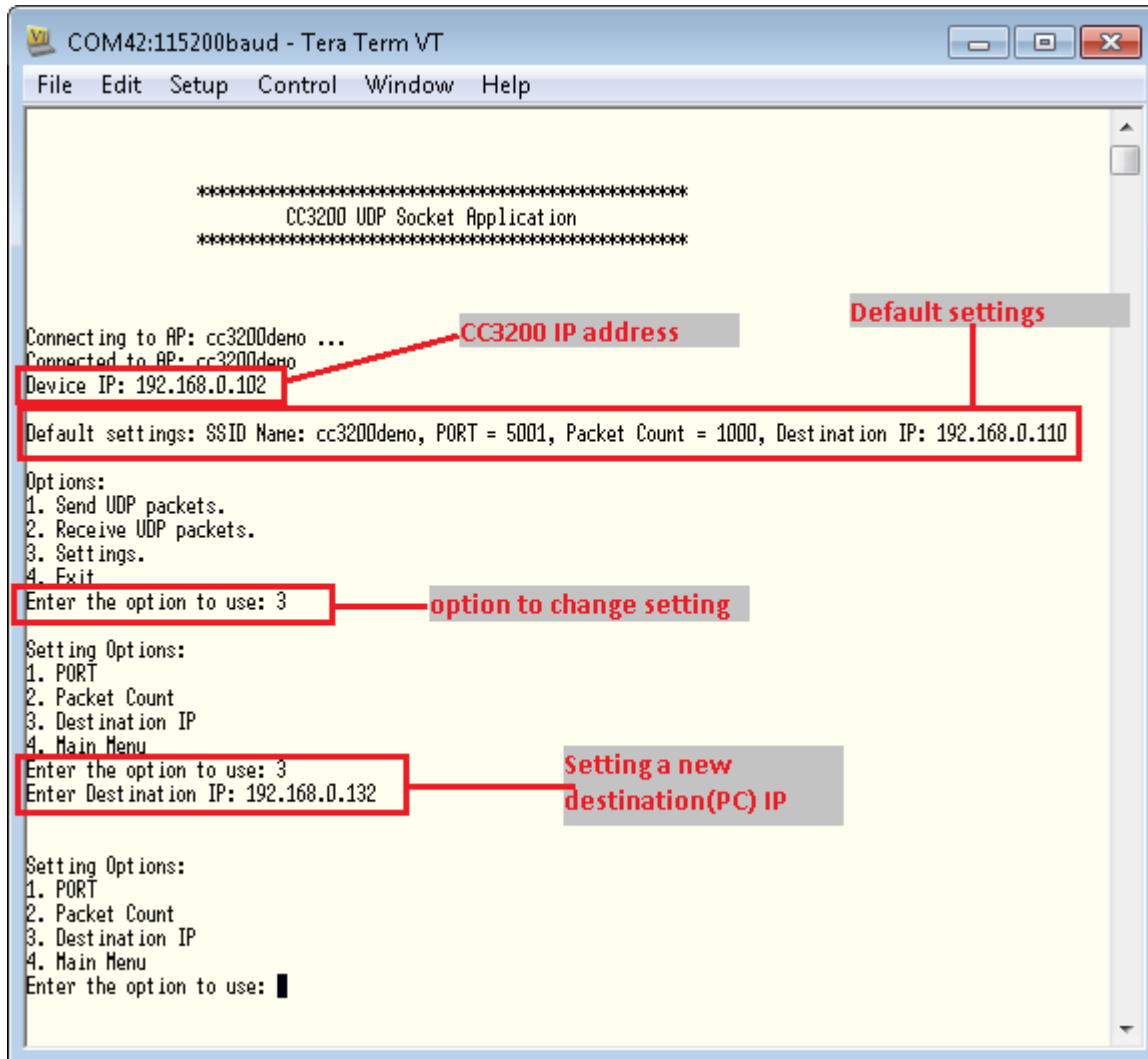
Steps for receiving UDP packets as a UDP server are as follows:

1. Open the socket
2. Create a UDP server
3. Receive packets
4. Close the socket

5.1.4.4 Usage

1. Set up a serial communication application (such as HyperTerminal or TeraTerm). For detailed information, visit the Setup Terminal on the host PC. The settings are:
 - Port: Enumerated COM port (CC3200LP Dual port)
 - Baud rate: 115200
 - Data: 8 bit
 - Parity: None
 - Stop: 1 bit
 - Flow control: None
2. Run the application (udp_socket) from IAR/CCS or flash the bin file to the device.
3. Connect a PC to the same AP that the device is connected to.
4. Get the IP address of the PC and fill this value for IP_ADDR macro, or change the setting as specified in [Figure 18](#).

Figure 18. UDP Socket Terminal



5. Change the other setting (port, SSID name, packet count) as required.
6. Choose the options:
 - Send UDP packets
 - Receive UDP packets
7. After selecting from the above options, run the iperf command on the PC command prompt as given in the TeraTerm/HyperTerminal screen.
8. Observe the execution flow to understand the results.

Note: Disable PC anti-virus while running iperf.

5.1.5 Raw Socket Application

5.1.5.1 Application Details

The transceiver mode application in the SDK showcases the use of raw socket usage in CC3200. This example demonstrates how to build a proprietary protocol on top of a Wi-Fi PHY layer, with the user given full flexibility to build their own packet.

The first two bytes of the raw data are Wi-Fi PHY layer-specific.

- 1st byte: Wi-Fi rate. Definition for rate options can be found in wlan.h, RateIndex_e structure.
- 2nd byte: 4 bits of power level and 4 bits of preamble type.

Defining a ping packet as a raw data structure

```
char RawData_Ping[] = {
/*----- wlan header start -----*/
    0x88,                                     /* version , type sub type */
    0x02,                                     /* Frame control flag */
    0x2C, 0x00,
    0x00, 0x23, 0x75, 0x55, 0x55, 0x55,      /* destination */
    0x00, 0x22, 0x75, 0x55, 0x55, 0x55,      /* bssid */
    0x08, 0x00, 0x28, 0x19, 0x02, 0x85,      /* source */
    0x80, 0x42, 0x00, 0x00,
    0xAA, 0xAA, 0x03, 0x00, 0x00, 0x00, 0x08, 0x00, /* LLC */
/*----- ip header start -----*/
    0x45, 0x00, 0x00, 0x54, 0x96, 0xA1, 0x00, 0x00, 0x40, 0x01,
    0x57, 0xFA,                                /* checksum */
    0xc0, 0xa8, 0x01, 0x64,                    /* src ip */
    0xc0, 0xa8, 0x01, 0x02,                    /* dest ip */
/* payload - ping/icmp */
    0x08, 0x00, 0xA5, 0x51,
    0x5E, 0x18, 0x00, 0x00, 0x41, 0x08, 0xBB, 0x8D, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00 .... };
```

Raw socket send

```
void TxContinues(int iChannel, RateIndex_e rate, int iNumberOfPackets, double dIntervalMiliSec)
{
    ...
    // Socket open
    iSoc = sl_Socket(SL_AF_RF, SL_SOCKET_RAW, iChannel);

    // Send the data
    for(ulIndex = 0; ulIndex < iNumberOfPackets; ulIndex++)
    {
        sl_Send(iSoc, RawData_Ping, sizeof(RawData_Ping), SL_RAW_RF_TX_PARAMS(iChannel, rate,
        iTxPowerLevel, PREAMBLE));
    }
    ...
    // Close the socket
    sl_Close(iSoc);
    ...
}
```

The Rx statistics feature inspects the medium in terms of congestion and distance, validates the RF hardware, and helps using the RSSI information.

5.1.5.2 Source Files Briefly Explained

- main – Demonstrates sending a raw ping packet in Tx continues, and usage of different API for getting the Rx statistics.
- uart_if – Displays status information over the UART.
- pinmux.c – Pinmux file to mux the device and configure the UART peripheral.
- startup_ccs.c – CCS-specific vector table implementation for interrupts
- startup_ewarm.c – IAR workbench-specific vector table implementation for interrupts

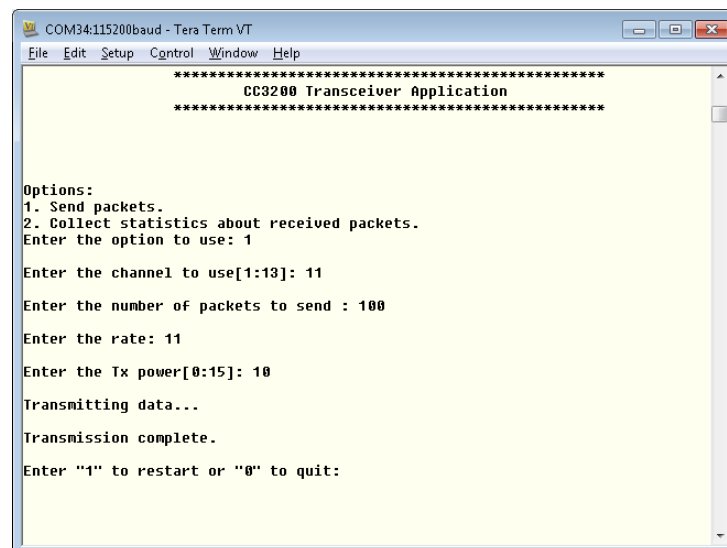
The ewarm folder contains IAR workspace. The ccs folder contains CCS Project, the driverlib folder contains all the driver files, and the simplelink folder contains SimpleLink host files.

5.1.5.3 Usage

- Setup a serial communication application (such as HyperTerminal or TeraTerm).
 - Port: Enumerated COM port (CC3200LP Dual Port)
 - Baud rate: 115200
 - Data: 8 bit
 - Parity: None
 - Stop: 1 bit
 - Flow control: None
- Run the reference application (Flashing the bin/IAR/CCS).
- Observe the status messages on the host over serial port in response to the user's selection of either "sending packets" or "Collect statistics of received packets" to understand the sequence of operations performed by the application.

Terminal snapshot when application runs on device is shown in [Figure 19](#).

Figure 19. CC3200 Transceiver Application on the Hyperterminal



```

COM34:115200baud - Tera Term VT
File Edit Setup Control Window Help
*****
CC3200 Transceiver Application
*****

Options:
1. Send packets.
2. Collect statistics about received packets.
Enter the option to use: 1

Enter the channel to use[1:13]: 11

Enter the number of packets to send : 100

Enter the rate: 11

Enter the Tx power[0:15]: 10

Transmitting data...
Transmission complete.
Enter "1" to restart or "0" to quit:
  
```

Common interface files are available under the example\common folder

5.2 SimpleLink APIs

	APIs	Description
WLAN APIs	sl_Start	This function starts the SimpleLink (Networking) device. This function initializes the communication interface. Once this is complete, LED1 starts blinking to indicate SL_start is complete and waiting to complete SL_Wlanconnect.
	sl_WlanConnect	This function connects the device to the AP specified as parameter. Once the connection status is set, LED1 will be ON until the device is disconnected from the AP. After a successful WLAN connect, the name of the AP to which the device is connected is displayed on the terminal.
	sl_IpConfigGet	While in DHCP mode, this function is used to get an IP address from the associated AP. The IP address is displayed on the terminal so that the client machine (iperf PC) can make use of this address to connect to the device.
	sl_WlanDisconnect	This function prompts the device to relinquish the connection (hence the association) with the AP.
Note – LED functionality depends on application implementation.		
Network APIs	sl_Socket	Creates UDP socket.
	sl_SendTo	Sends a Welcome message to the destination IP address, given as input. The destination IP address is taken from input. 200 packets are sent to the specific destination address if the address is mentioned in the input. If the destination address is not mentioned, the message is broadcasted. Once this is complete, an alert is given on UART. While the device is sending messages, LED2 will be on to indicate the device status.
	sl_Bind	Binds socket. The reception port is taken as 5002 in this application.
	sl_RecvFrom	Receives message from client. The device is waiting for 200 packets to receive. Once the device receives 200 packets, an alert indicating the same and the source address of the packets will be shown on UART. While receiving messages, LED4 will be on to indicate the device status.
	sl_Close	Closes the socket.
Note – Number of packets and port depend on application implementation.		

5.3 Compilation, Build and Execution Procedure

Refer to the IAR/CCS help documentation for detailed information on compiling, building, and executing user applications. The following sections highlight key projects exercised during the development process. The basic Wi-Fi application is taken up as a reference to demonstrate the development environment. Similar procedures apply for any reference application or new developments. Most of the steps mentioned here are already performed for all the reference applications (present in examples/folder) and captured in the project files. While using the debugger, clean and rebuild the libraries (driverlib, simplelink, FreeRTOS) to avoid any source file association problems.

NOTE: While creating new project under this SDK, call PRCMCC3200MCUInit() as the first call in main() for proper initialization of the device.

NOTE: Visit the [CC3200 TI-RTOS page](#) before creating any TI-RTOS-based application.

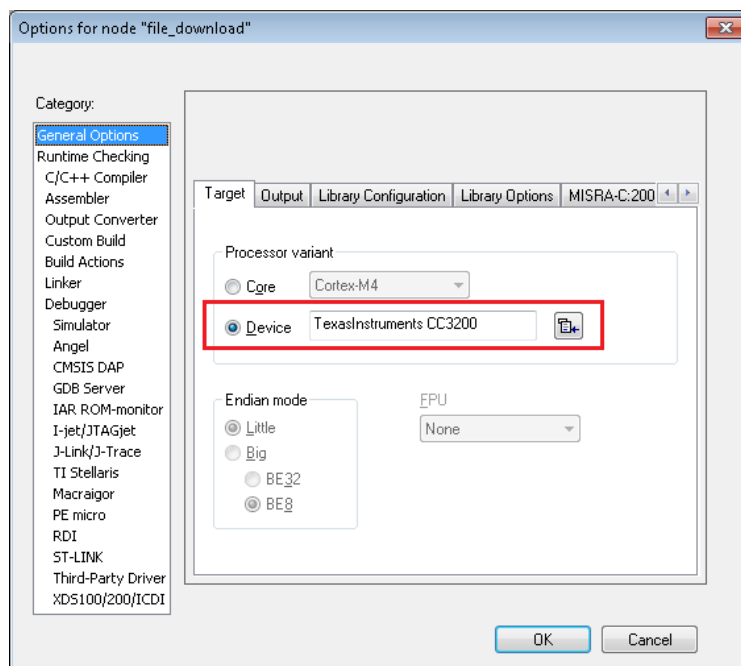
5.3.1 Development Environment – IAR

Follow the steps given in C:\TI\CC3200SDK\cc3200-sdk\tools\iar_patch\readme.txt to replace iarmLMIFTDI.dll.

5.3.1.1 Creating a Project

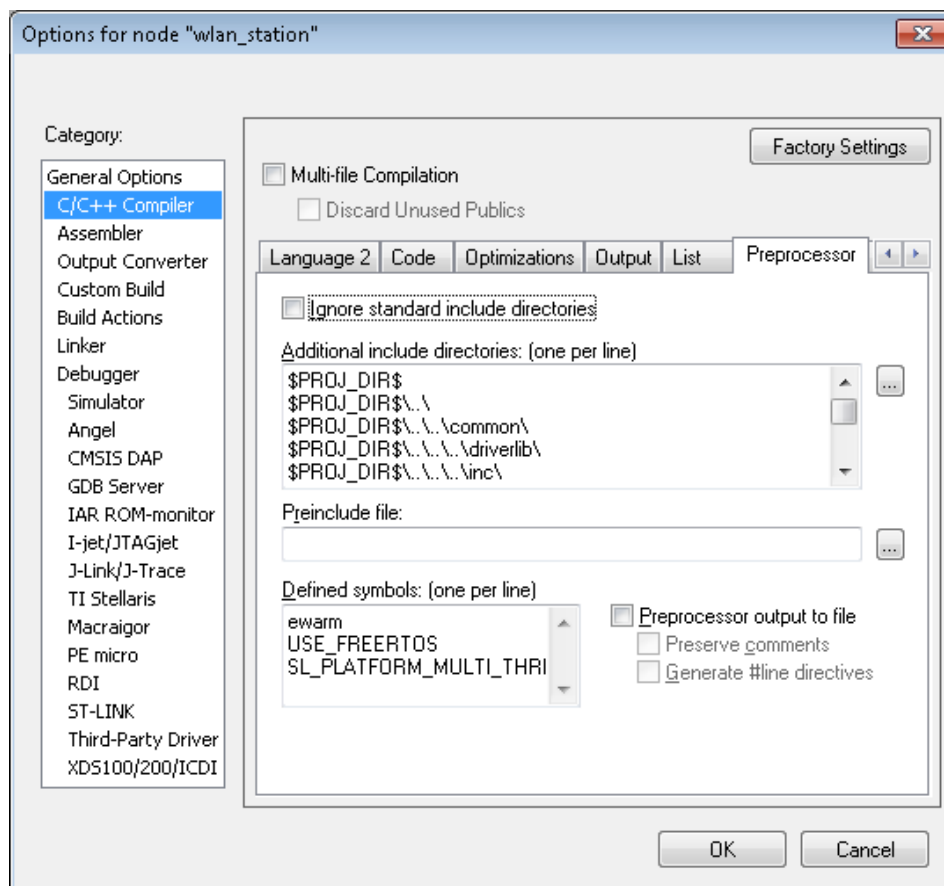
1. Go to File->New->Workspace
2. Click on Project->Create New Project (Tool Chain = ARM, Project Templates = C)
3. Open Project Option, and follow the settings as given in [Figure 20](#).

Figure 20. CC3200 Programmer Guide IAR Project Options



5.3.1.2 Compiling a Project

Figure 21. CC3200 IAR Compiling Project



Additional include directories:

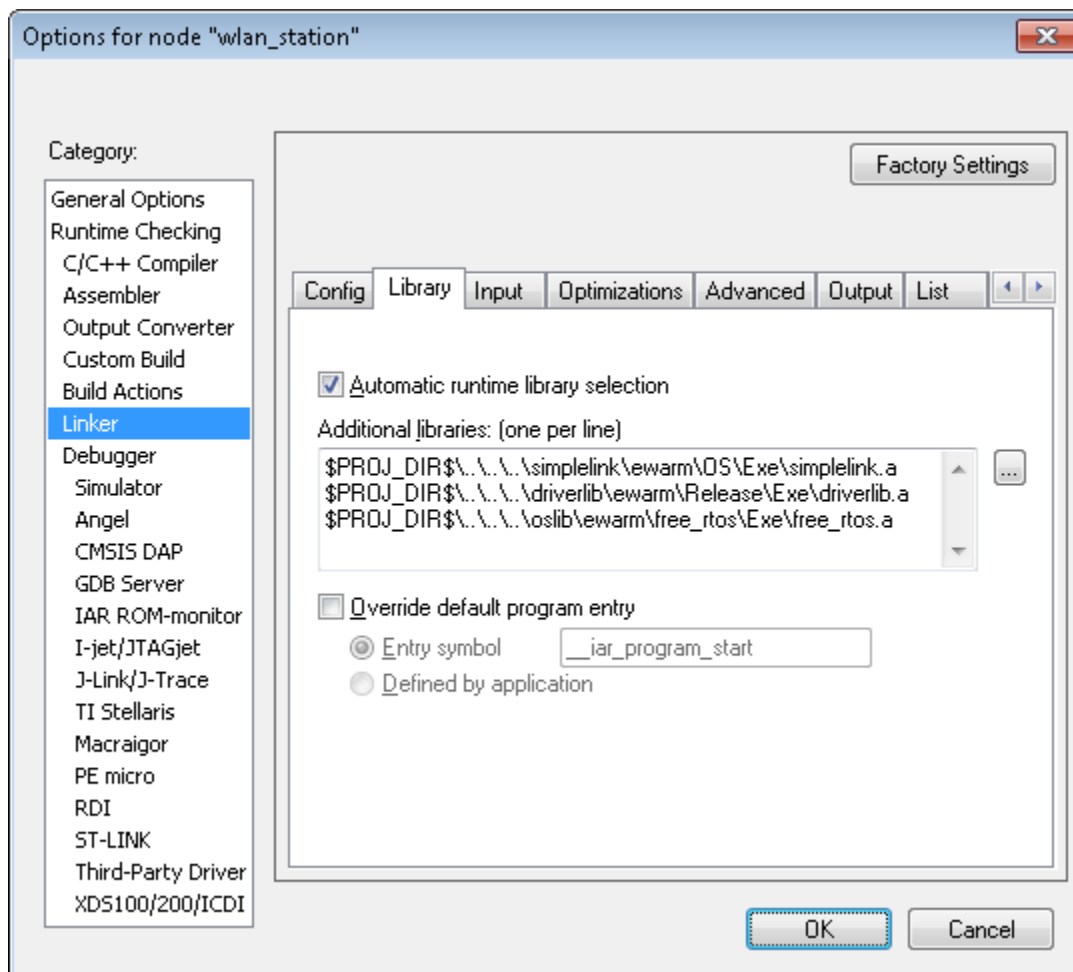
- To use Driverlib APIs – Include driverlib and inc folder path
- To use Simplelink APIs – Include simplelink, simplelink\Source, and simplelink\Include folder paths
- To use Free-RTOS TI-RTOS APIs – Include oslib path

Defined symbols and MACRO definition:

- USE_FREERTOS – If application uses Free-RTOS OS
- USE_TIRTOS – If application uses TI-RTOS OS
- ewarm – Define for IAR-based application
- SL_PLATFORM_MULTI_THREADED – If application uses any OS (Free-RTOS/TI-RTOS)
- NOTERM – If application does not need UART prints.

5.3.1.3 Linking a Project

Figure 22. CC3200 IAR Linker Project



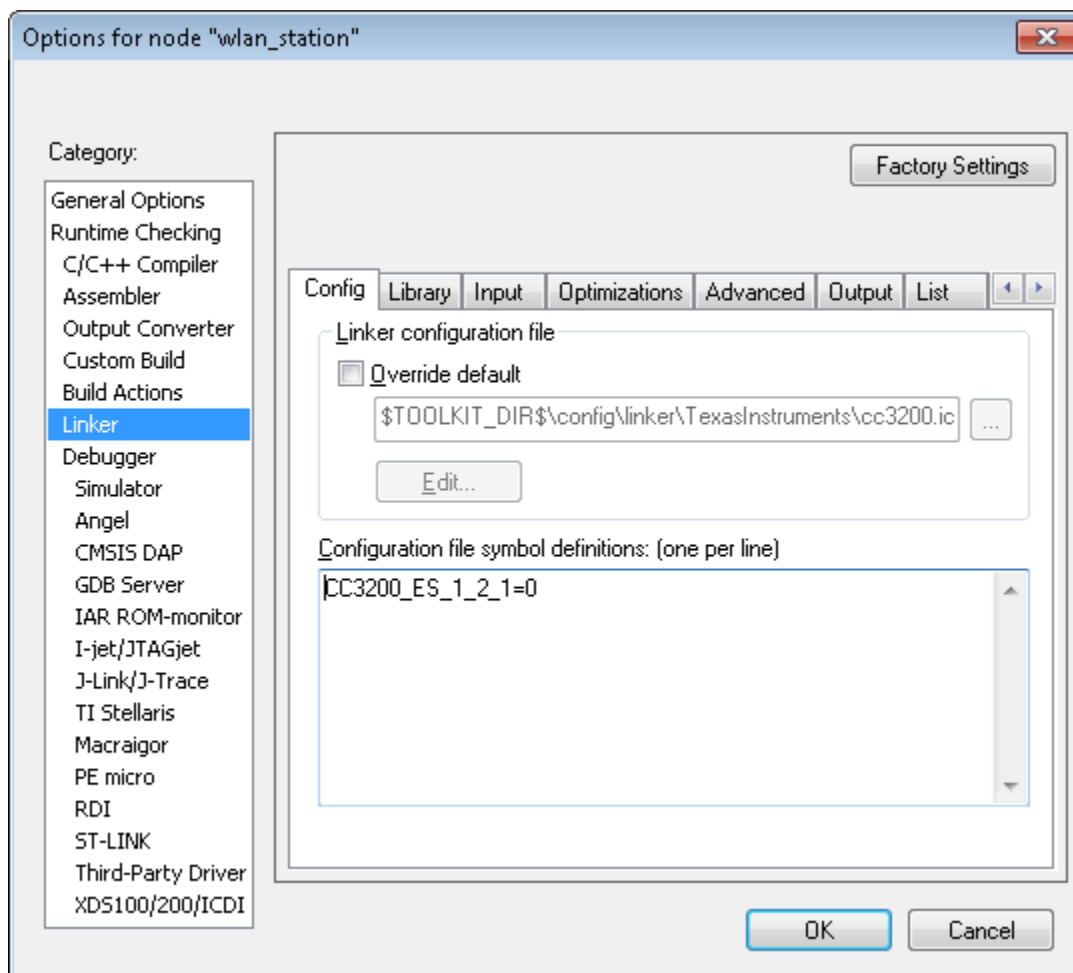
Additional libraries:

- Add library path as per application needs:
 - driverlib.a : Available under the *driverlib\ewarm\Release\Exe* folder.
 - simplelink.a : Available under *simplelink\ewarm\OS*, *simplelink\ewarm\WON_OS*, *simplelink\ewarm\WON_OS_PM*, and *simplelink\ewarm\PM_Framework* folder for the OS, Non-OS, Non-OS with power management hook, and power management configuration for OS environment, respectively.

- free_rtos.a : Available under `oslib\ewarm\free_rtos\Exe\` folder.
- ti_rtos.a : If application uses TI-RTOS, then the TI-RTOS library is available under the `oslib\ewarm\ti_rtos\Exe\` folder.

NOTE: Use the *_debug configuration of the SimpleLink library while debugging the application. Refer to [Section 3.1.4](#).

Figure 23. CC3200 IAR Linker Config



Linker configuration file:

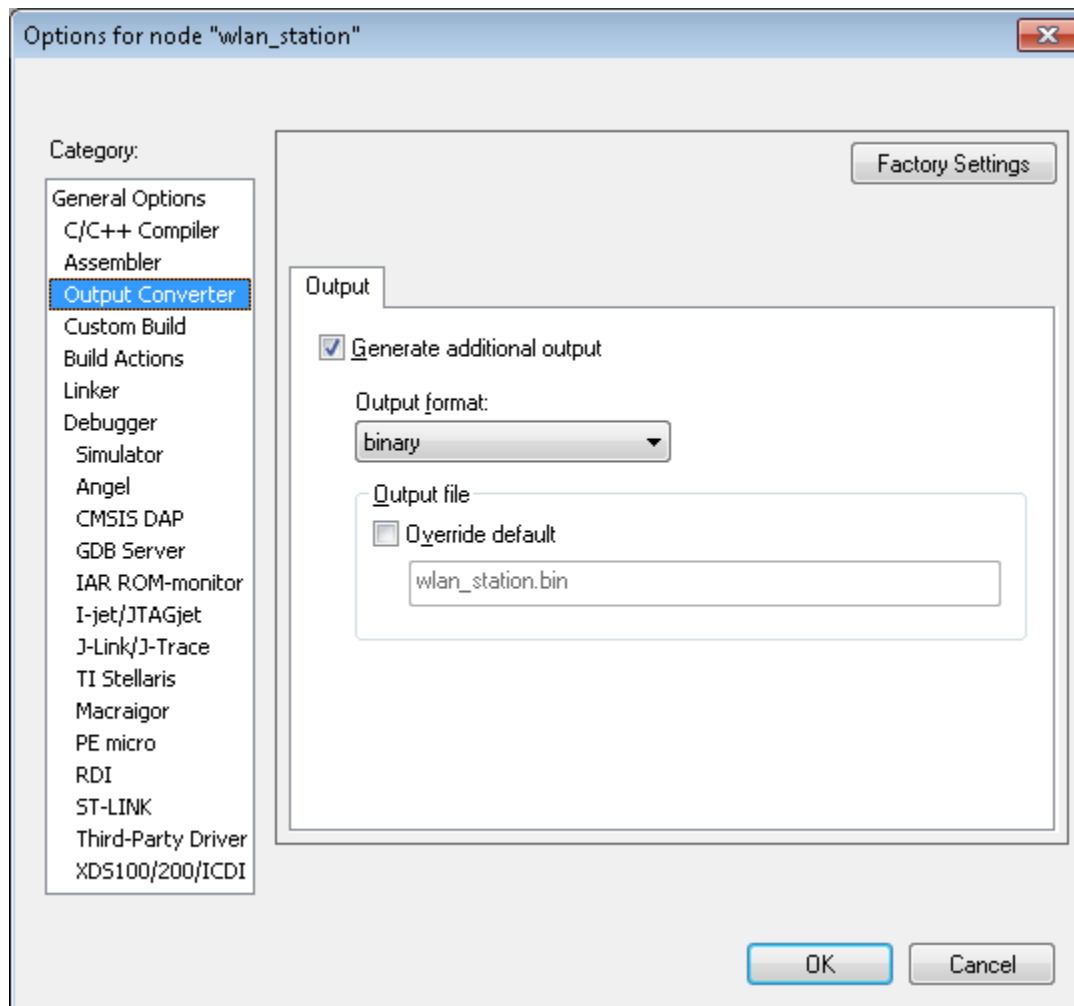
- Link to IAR linker file. By default, IAR links to CC3200.icf available in IAR installation. The developer can change this as per application requirement.

Symbol definitions:

- Define CC3200_ES_1_2_1=0 for production devices.

5.3.1.4 Generating the Binary (.bin)

Figure 24. CC3200 IAR Generating Binary



To generate additional output select the output format, a current SDK user must select the binary option and override the .bin path. In CC3200 SDK, IAR generates bin file in the *<example>lewarm\Release\Exe* folder.

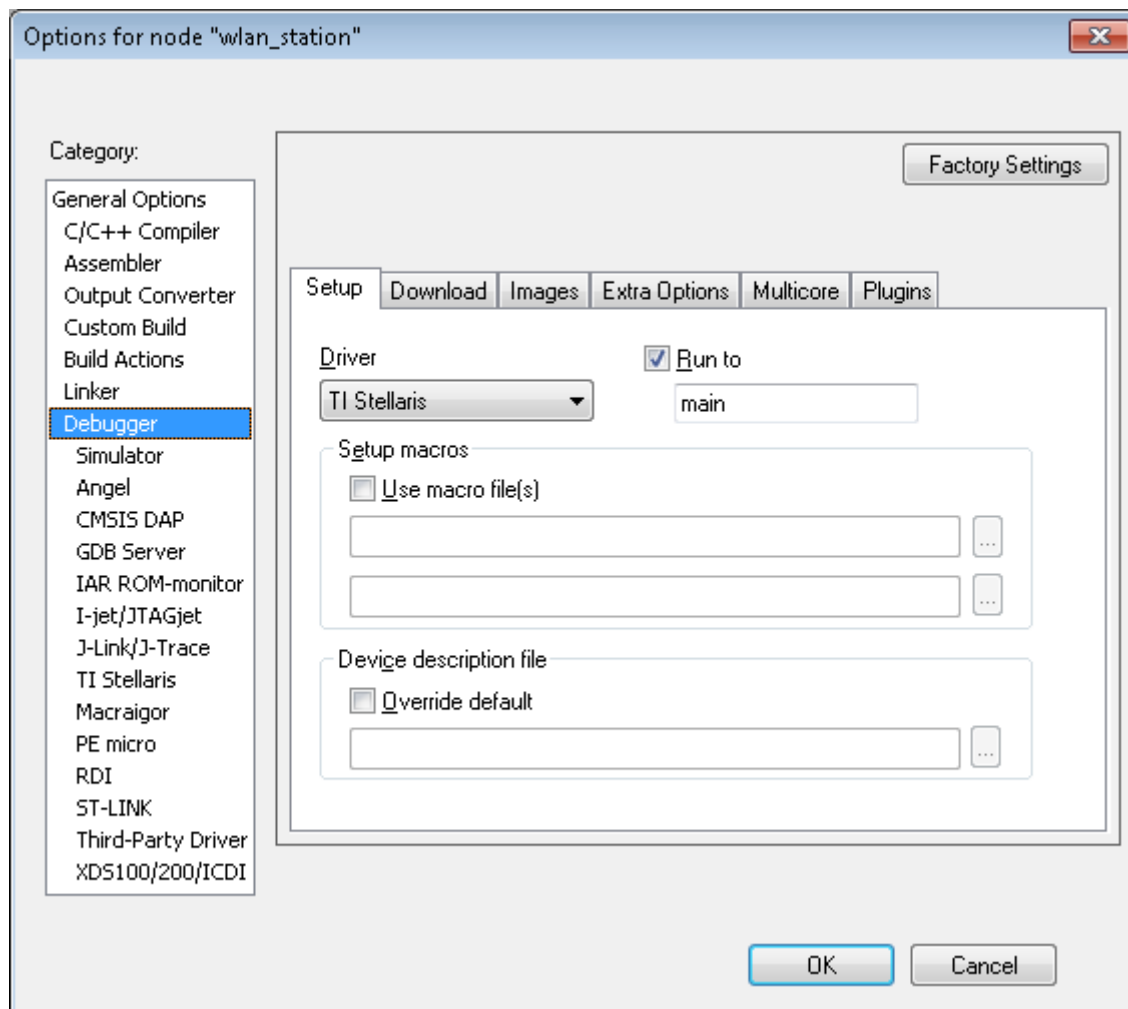
5.3.1.5 Executing a Project

To use the JTAG over FTDI, the TI Stellaris driver must be selected. The user can configure IAR to work with JTAG/SWD by choosing the option available in Options->Debugger->TI Stellaris->Interface. On the CC3200 LaunchPad:

- JTAG Mode – connect SOP-2 jumper only
- SWD mode – connect SOP-0 jumper only

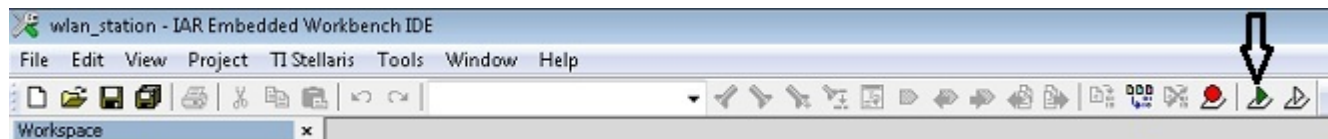
NOTE: The 4-wire JTAG works without connecting the SOP 2 jumper. However, TI recommends connecting the SOP 2 jumper to prevent the already-flashed code from executing while debugging.

Figure 25. CC3200 IAR Executing



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

Figure 26. CC3200 IAR Download and Run



If the application uses UART to print the output on the terminal, then the user must setup a terminal application (such as HyperTerminal or TeraTerm). These are the serial port settings:

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

NOTE: To enable UART prints in any application:

- Add UART_if.c/h to project and do pinmux for UART peripheral (refer to *example\mode_config\pinmux.c*).
- Disable the NOTERM macro and call InitTerm (*example\common\uart_if.c*) to initialize UART at the start of the application program.

5.3.2 Development Environment – TI Code Composer Studio

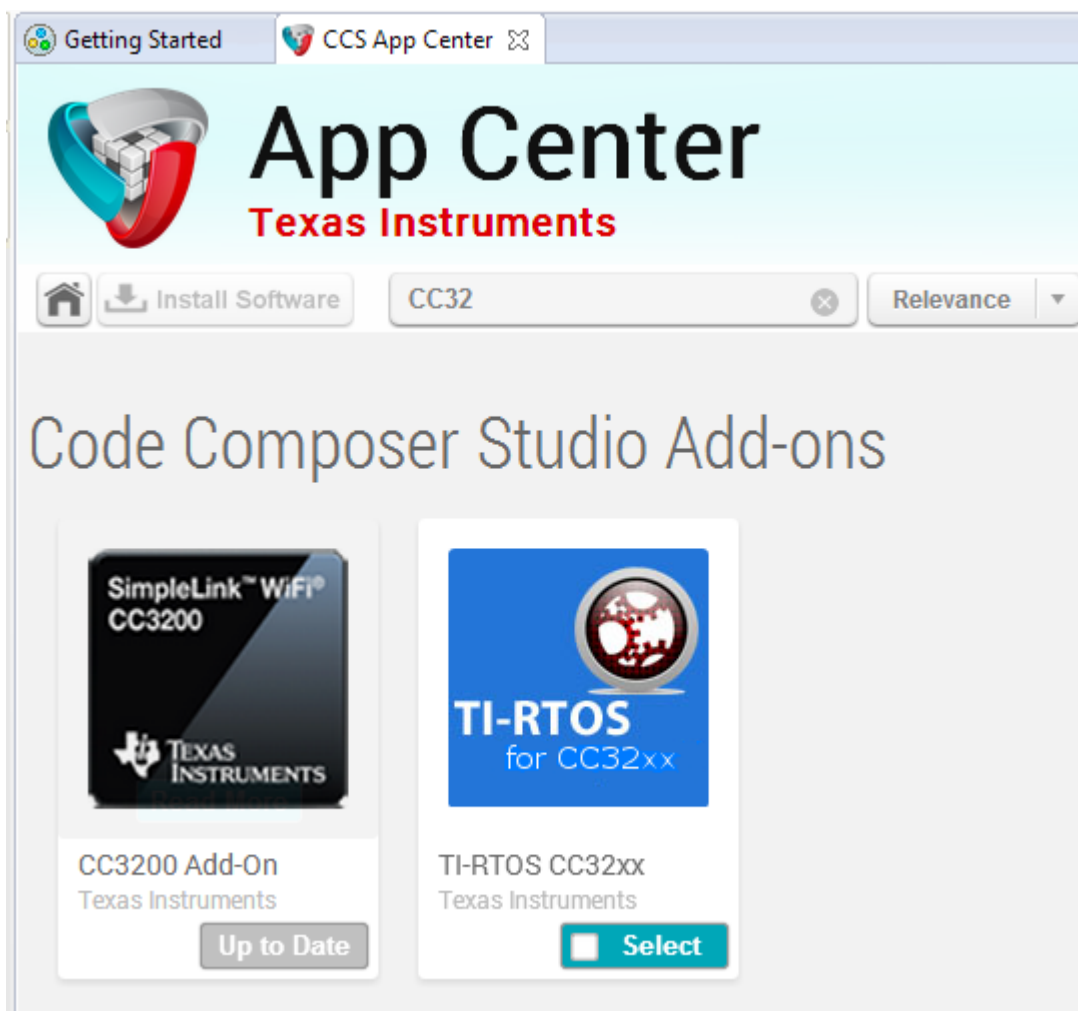
Current SDK supports CCS 6.1.1 version. The following are the steps to create a new project in CCS environment.

5.3.2.1 TI-RTOS in CCSv6

To install TI-RTOS under a CCS environment:

- Start the CCS and open the app center from the Help->Getting Started screen.
- Search 'CC3200' in the app center, which results in 'CC3200 Add-on' and 'TI-RTOS for SimpleLink.'
- Select and install in CCS 6.1.1

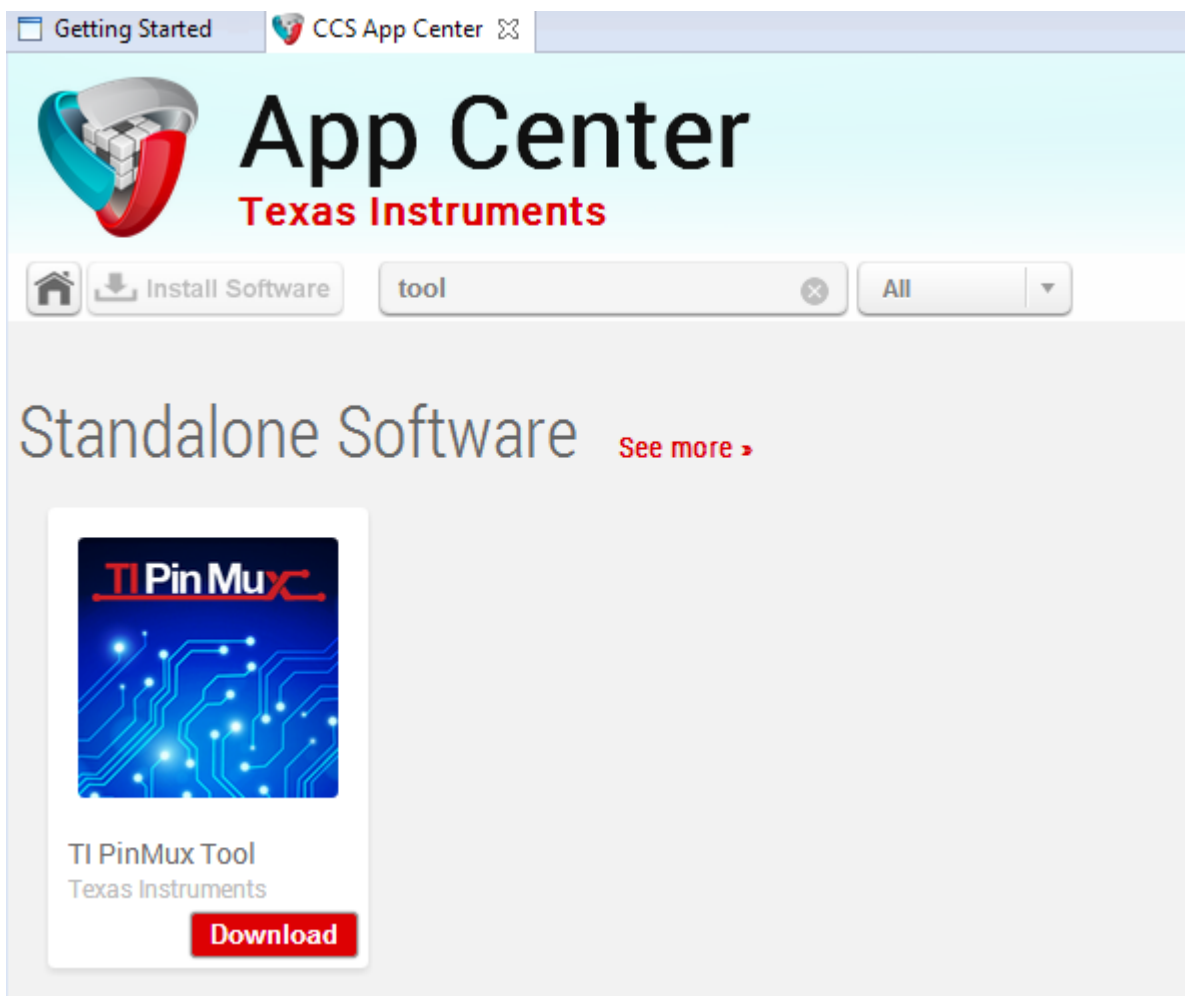
Figure 27. CCS App Center



5.3.2.2 Install TI-PinMux Tool

The user can install the TI-Pinmux Tool from <http://www.ti.com/tool/pinmuxtool> or as specified in Figure 28

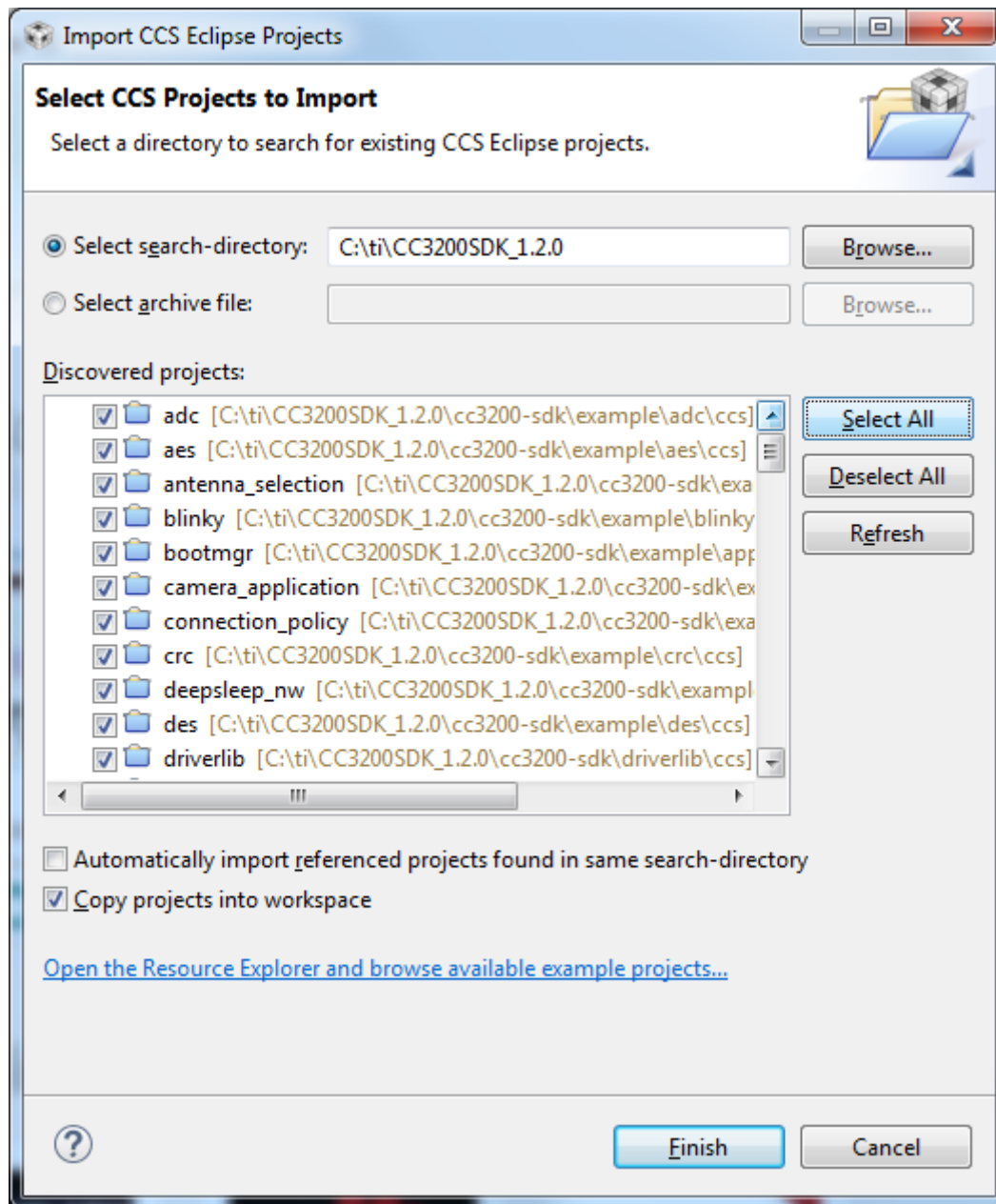
Figure 28. TI-PinMux Tool



5.3.2.3 Importing a Project

The current version of the SDK supports Copy Projects into Workspace, allowing the user to modify the example source as per requirement without touching the SDK repository.

Figure 29. Select CCS Projects to Import



If the user copies any library project (such as driverlib, simplelink, or netapps) to the workspace and modifies it, ensure that the concerning application connects to the latest and modified >library>.a, available in the workspace (see Figure 36).

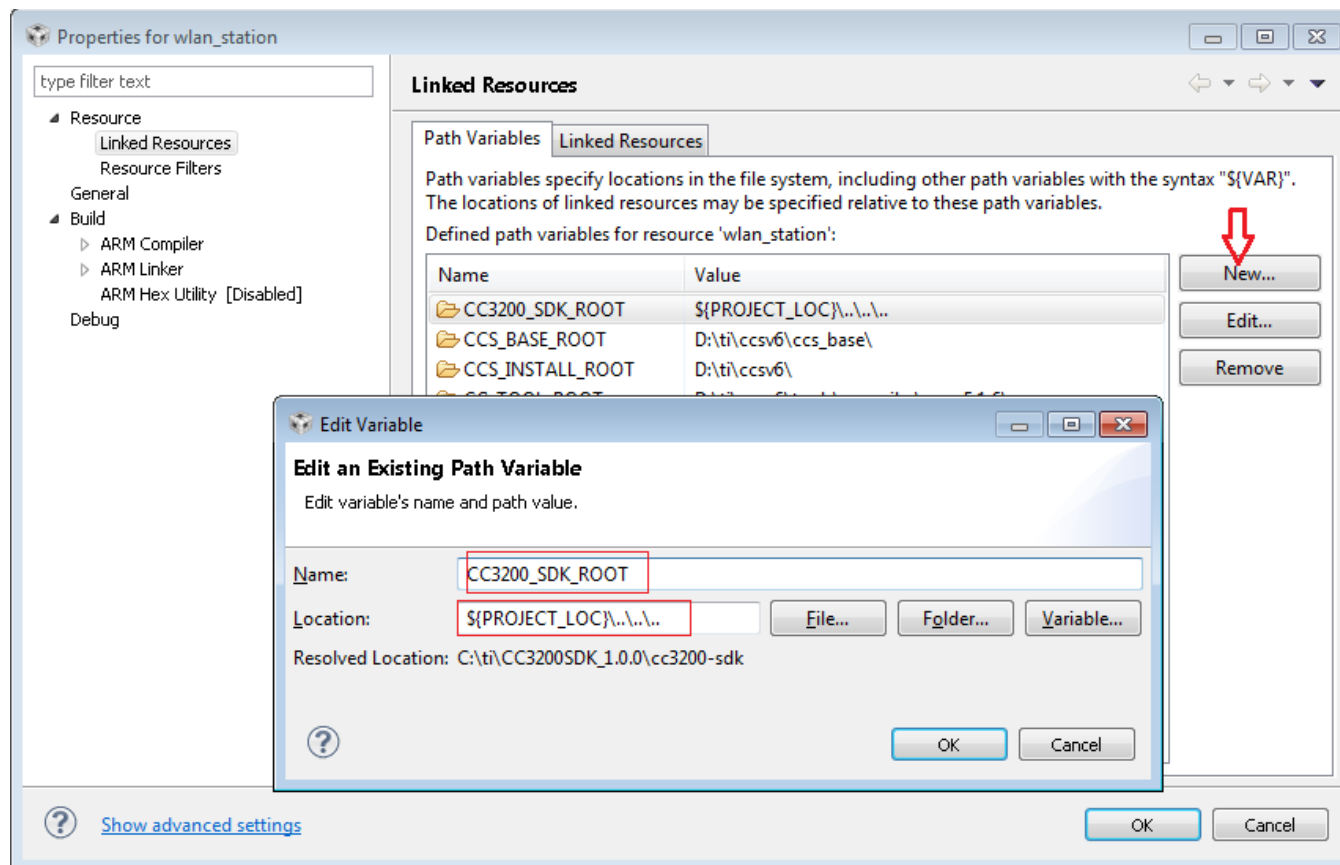
MACRO.ini : This file is used when the user copies any example folder to their working directory and tries to build a CCS project. Edit macro.ini and assign CC3200_SDK_ROOT to the SDK root path and import the project into the workspace, so that the example can connect to the SDK libraries and common files.
 CC3200_SDK_ROOT=C:\TI\CC3200SDK_<version>\cc3200-sdk

5.3.2.4 Creating a Project

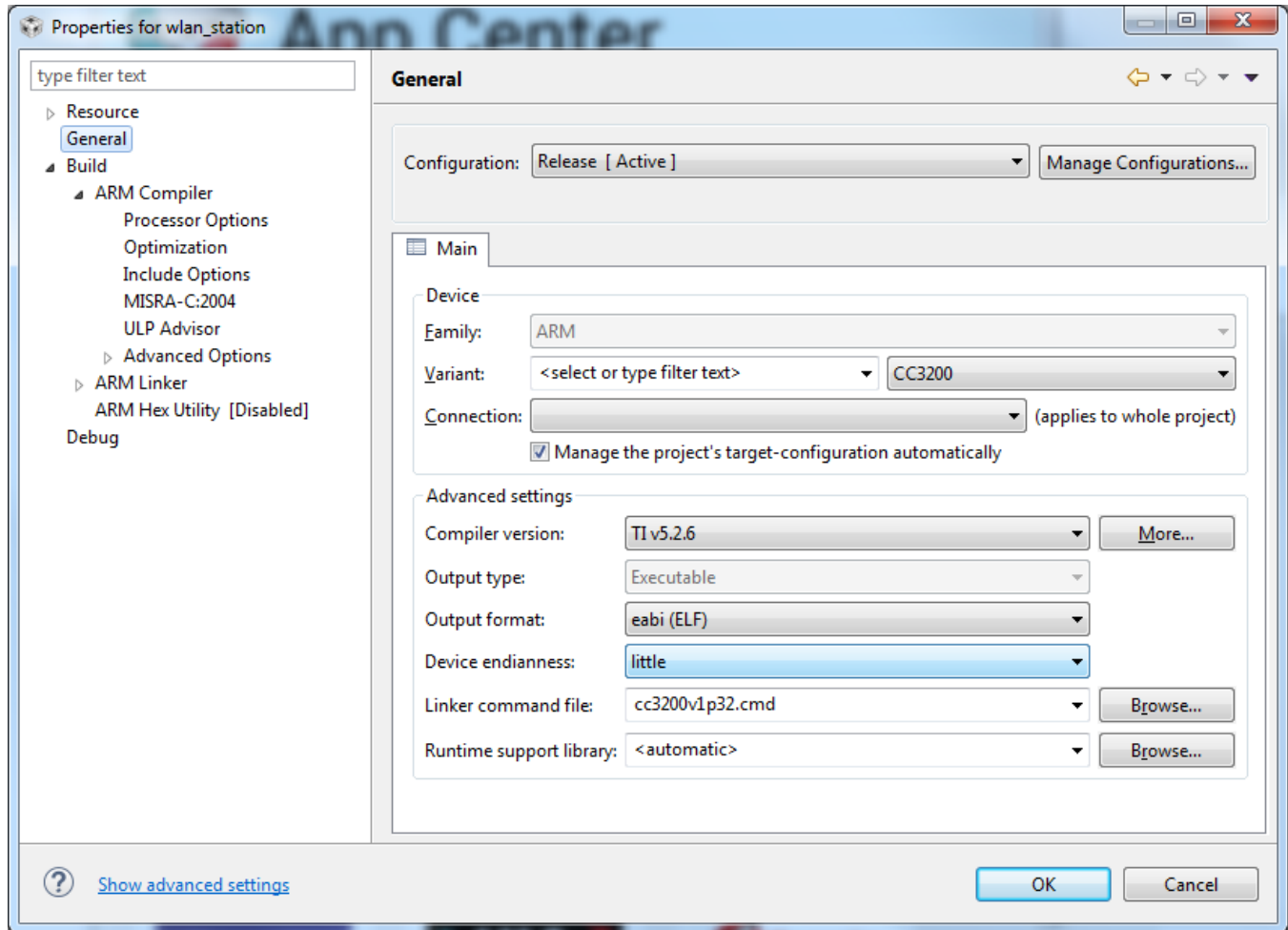
- File -> New -> CCS Project
- Target -> Wireless Connectivity MCU
- Device -> Variant -> CC3200
- Open Project Option, and follow the settings as given in the snapshots.

For the application to work with TI-RTOS, ti_rtos_config project must be imported into the CCS workspace. Refer to [docs\CC3200-TI-RTOS User Guide.pdf](#) or http://processors.wiki.ti.com/index.php/CC32xx_TI-RTOS.

Figure 30. CC3200 CSS Editing Existing Project



Add a path variable CC3200_SDK_ROOT in the CCS project, which locates to the root folder of the current SDK. The user can change this path to point to the desired version of the SDK path. This variable can be used to include paths, link libraries, and common source files.

Figure 31. CC3200 CCS Creating Project


Linker command file:

- cc3200v1p32.cmd: for XCC3200JR, CC3200R1M2 devices.

5.3.2.5 Compiling a Project

Figure 32. CC3200 CCS Compiling Project

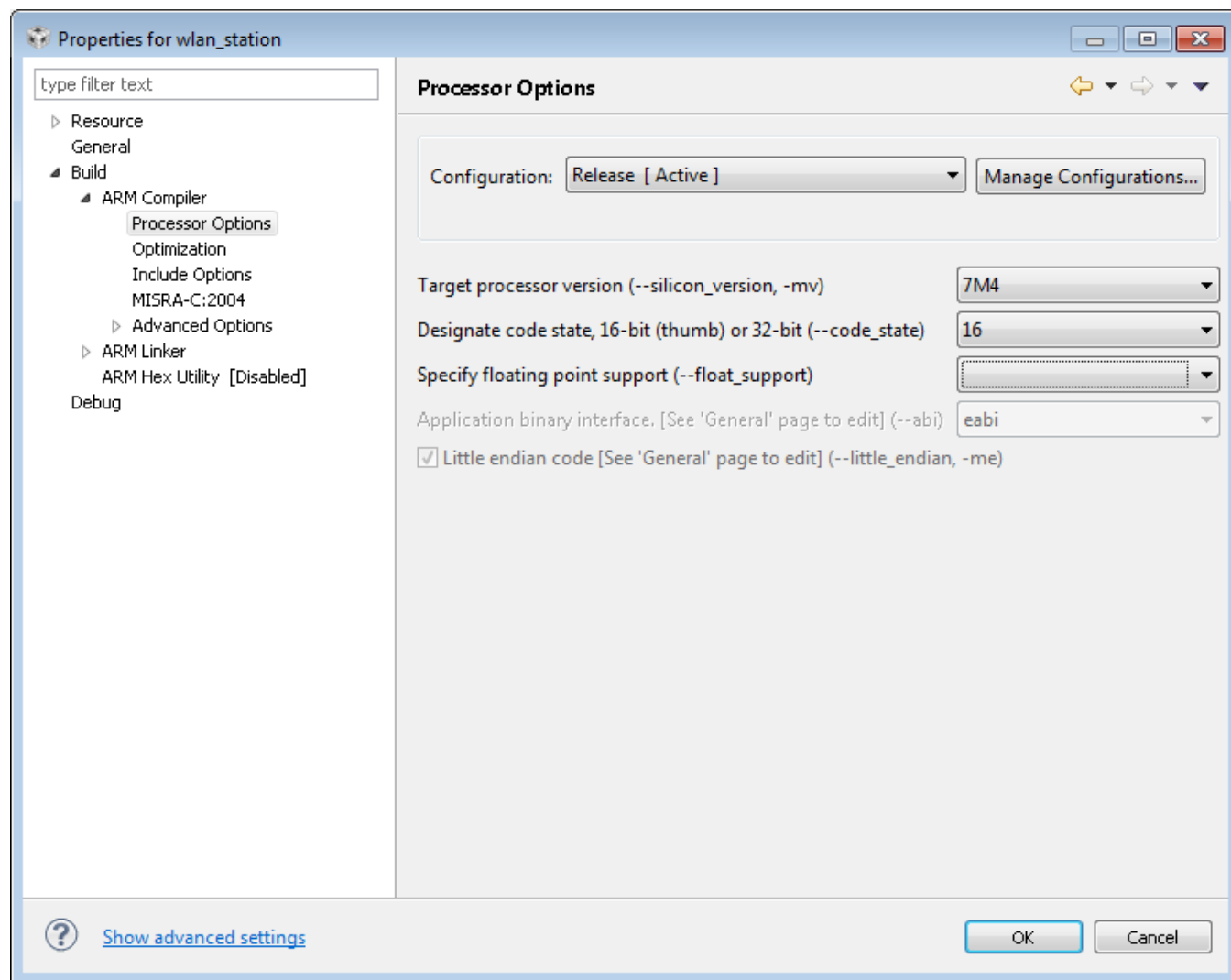
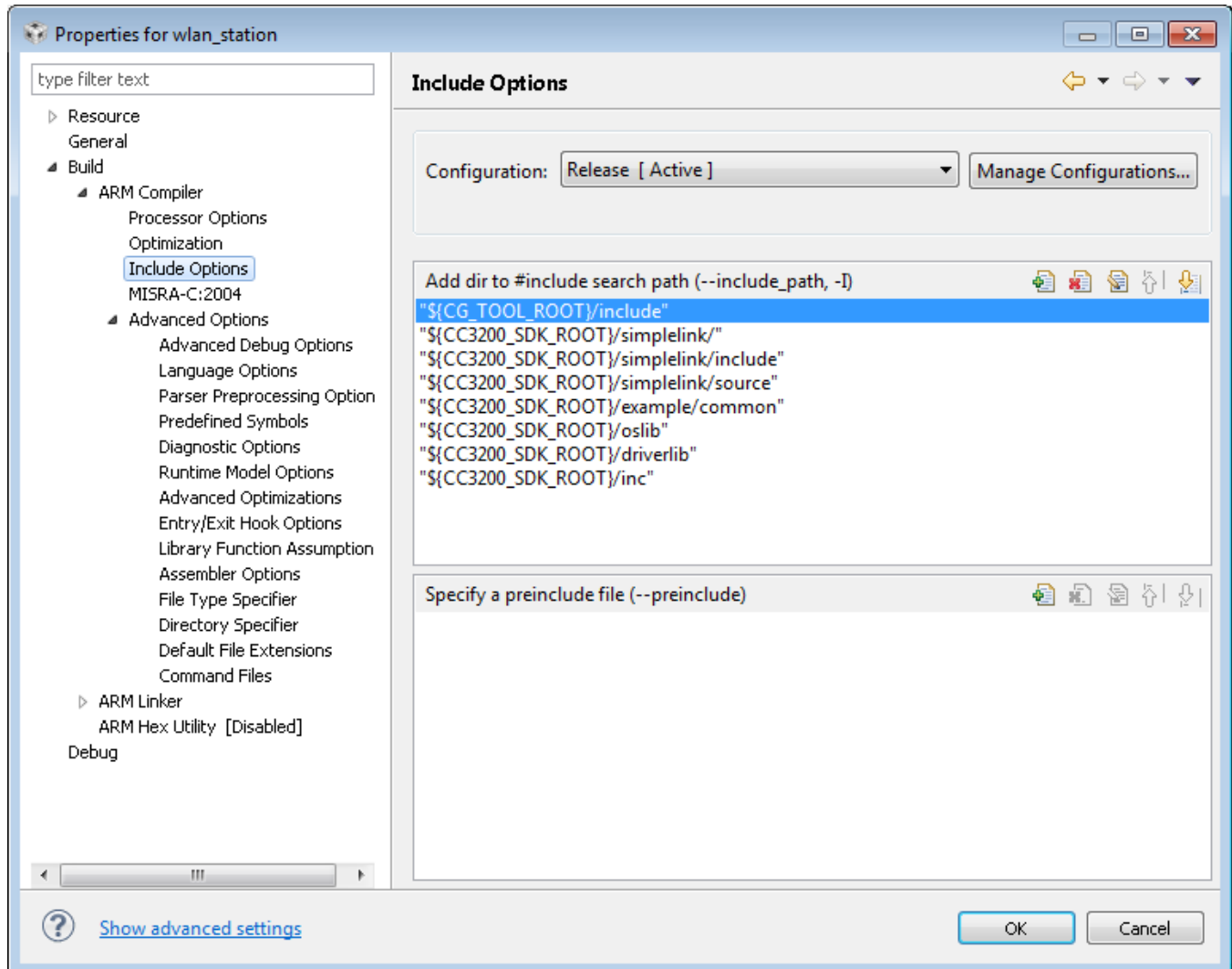
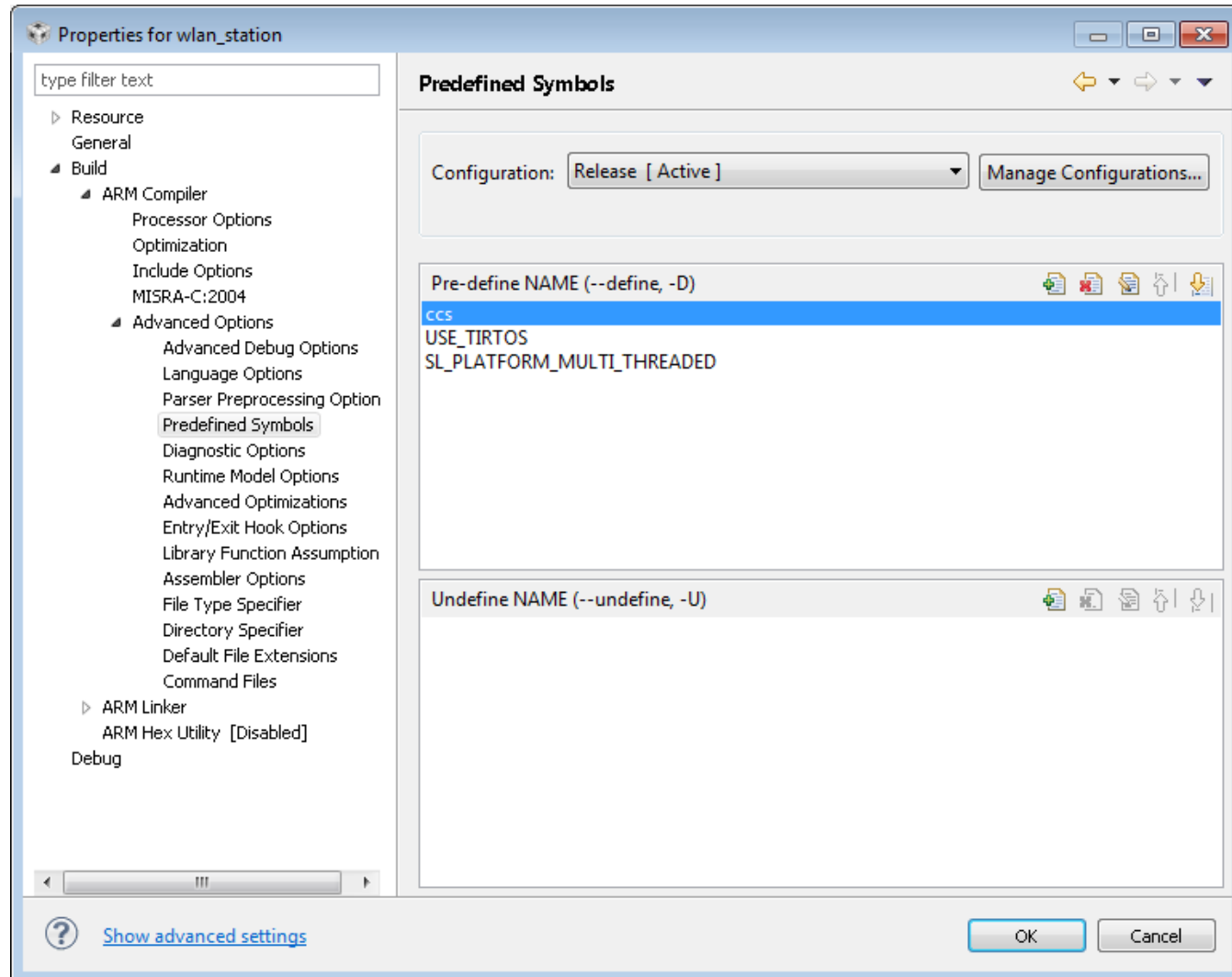


Figure 33. CC3200 CCS Compiling Project 1


Add dir to #include search path:

- To use Driverlib APIs – include driverlib and inc folder path.
- To use Simplelink APIs – include simplelink, simplelink\source, and simplelink\include folder path.
- To use TI-RTOS APIs – include oslib folder path.
- To use common interface APIs – include example\common folder path.

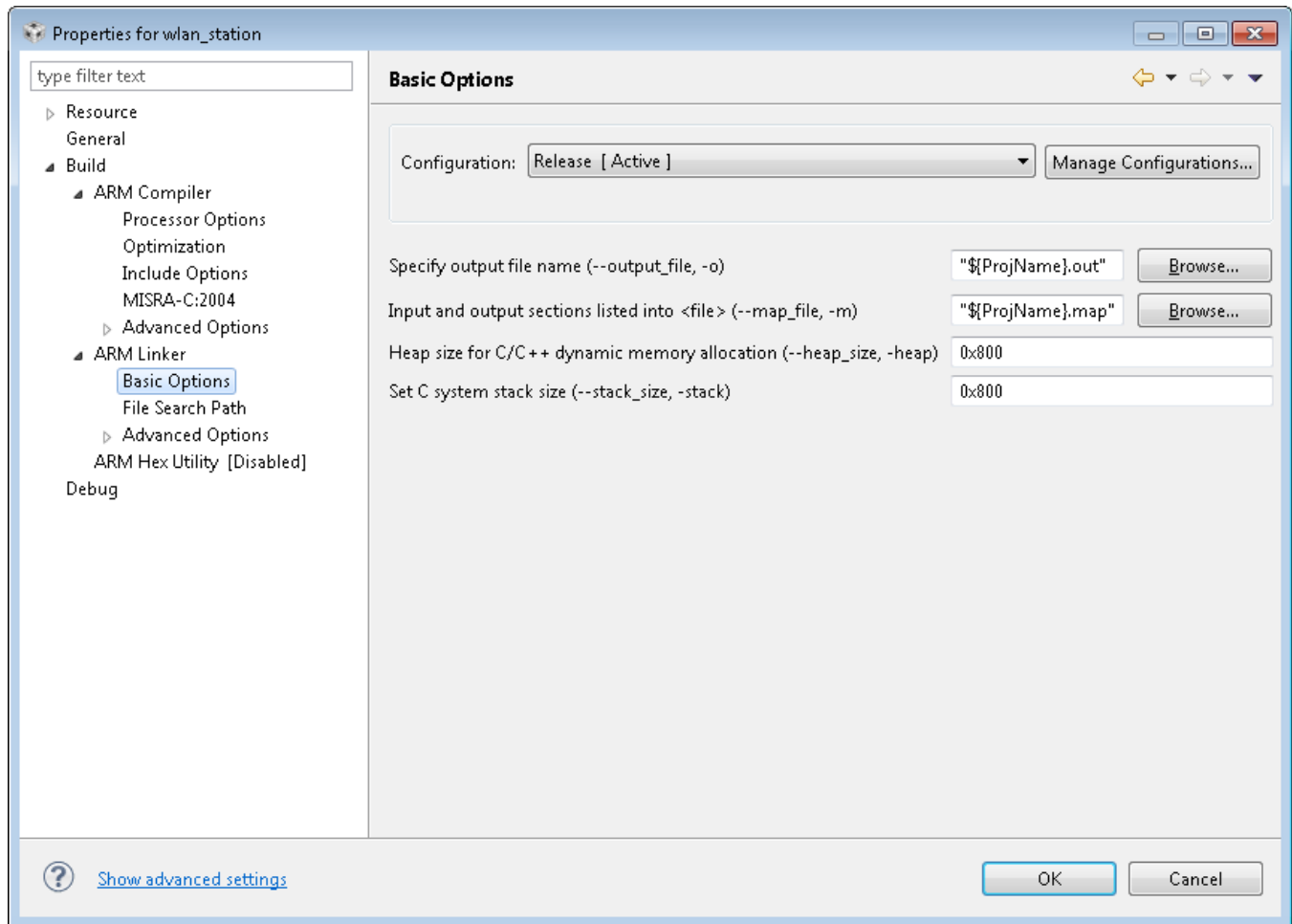
Figure 34. CC3200 CCS Compiling Project 2


Pre-define NAME:

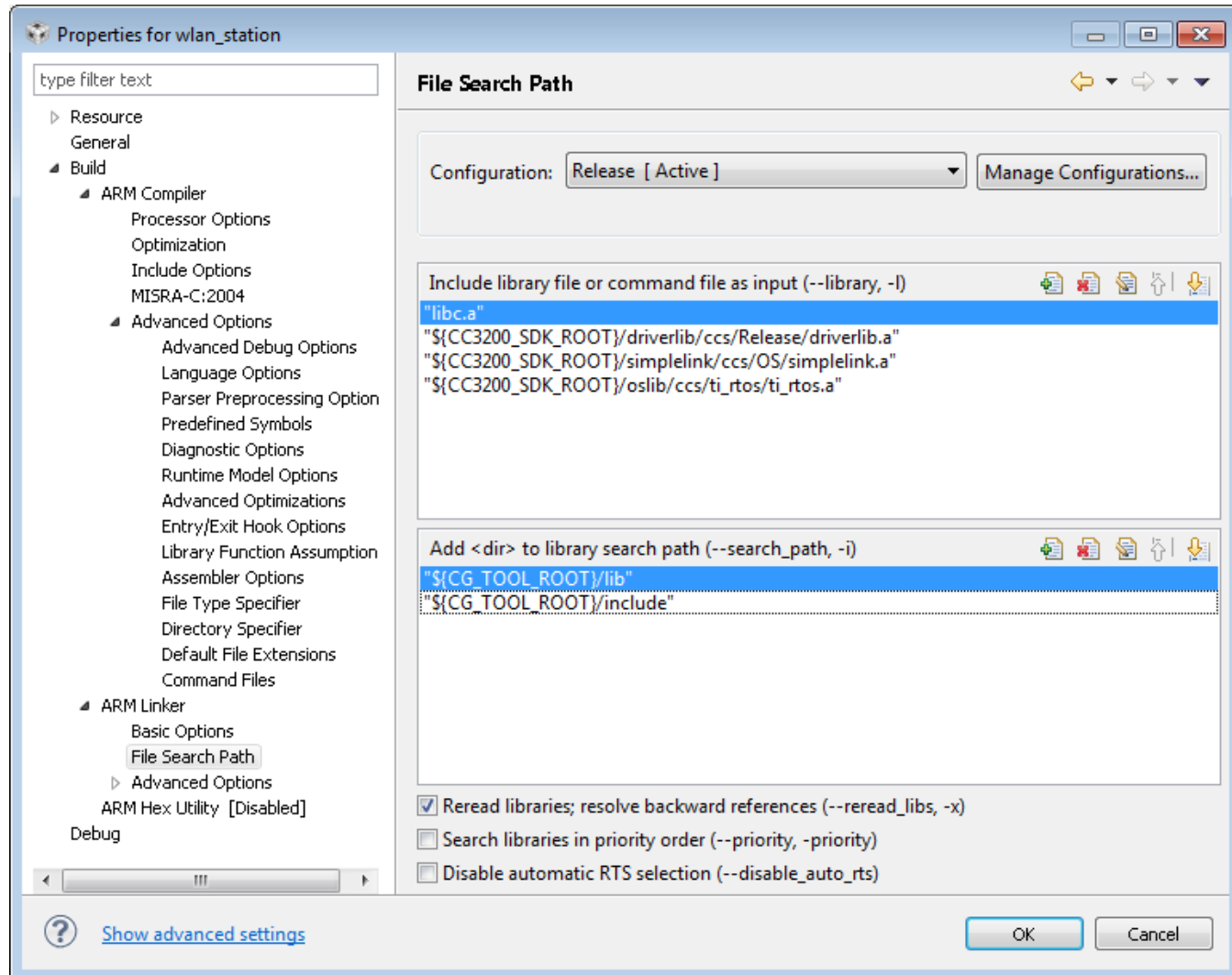
- USE_TIRTOS – To use TI-RTOS OS APIs.
- USE_FREERTOS – To use Free-RTOS OS APIs.
- SL_PLATFORM_MULTI_THREADED – If application uses any OS.
- ccs – For CCS-based application

5.3.2.6 Linking a Project

Figure 35. CC3200 CCS Linking Project 1



- Set heap and stack size as per the application requirements.

Figure 36. CC3200 CCS Linking Project 2


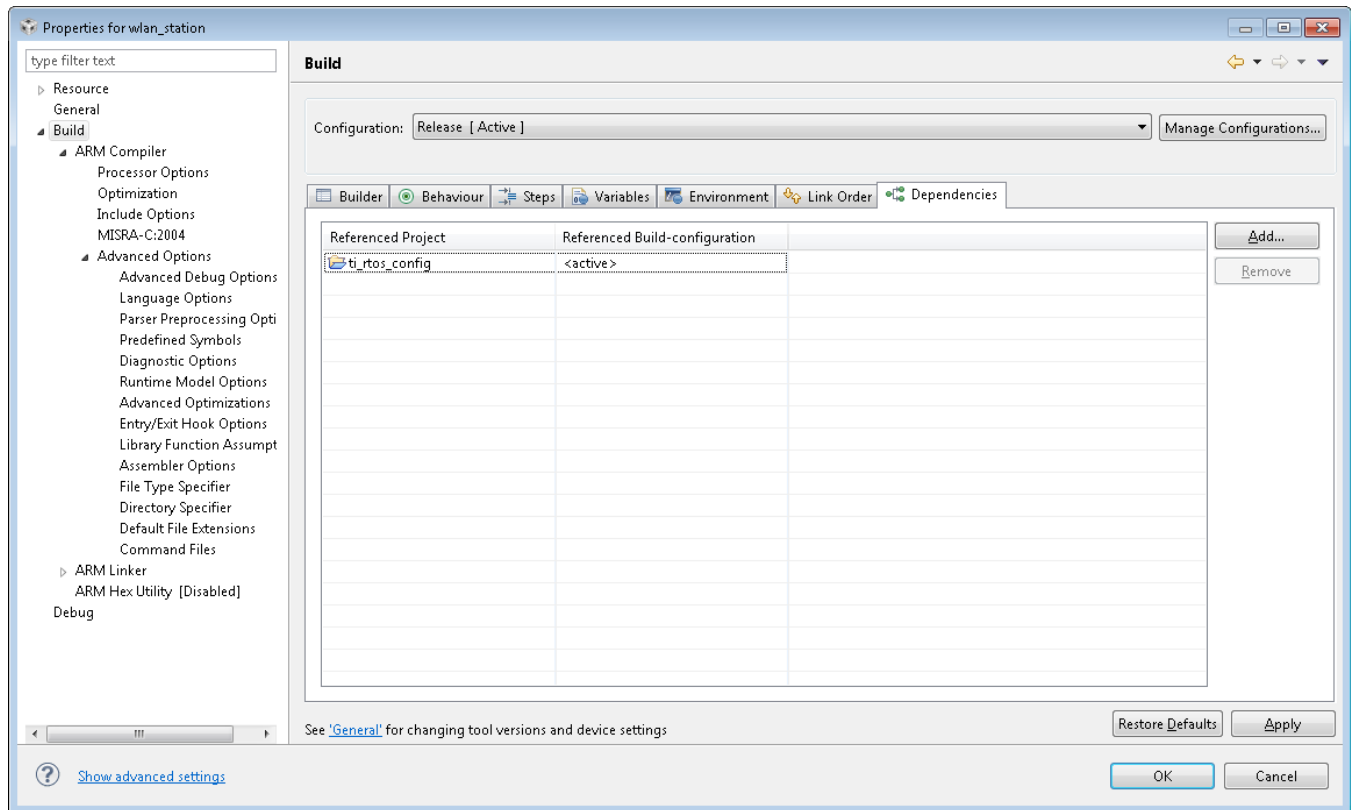
Include library file:

- As per the application requirements, include driverlib.a, simplelink.a, ti_rtos.a, or free_rtos.a
 - driverlib.a is available under the *driverlib\ccs\Release* folder.
 - simplelink.a is available under the *simplelink\ccs\OS*, *NON_OS*, *NON_OS_PM*, *PM_Framework* folder for OS, non-OS, power management for non-OS, or power management for OS-based applications respectively.
 - ti_rtos.a and free_rtos.a are present under the *oslib\ccs\ti_rtos* and *oslib\ccs\free_rtos* folders, respectively.

NOTE: Use the *_debug configuration of the SimpleLink library while debugging the application. Refer to [Section 3.1.4](#).

5.3.2.7 Dependency to Other Project

Figure 37. TI-RTOS OS Dependency



Dependencies:

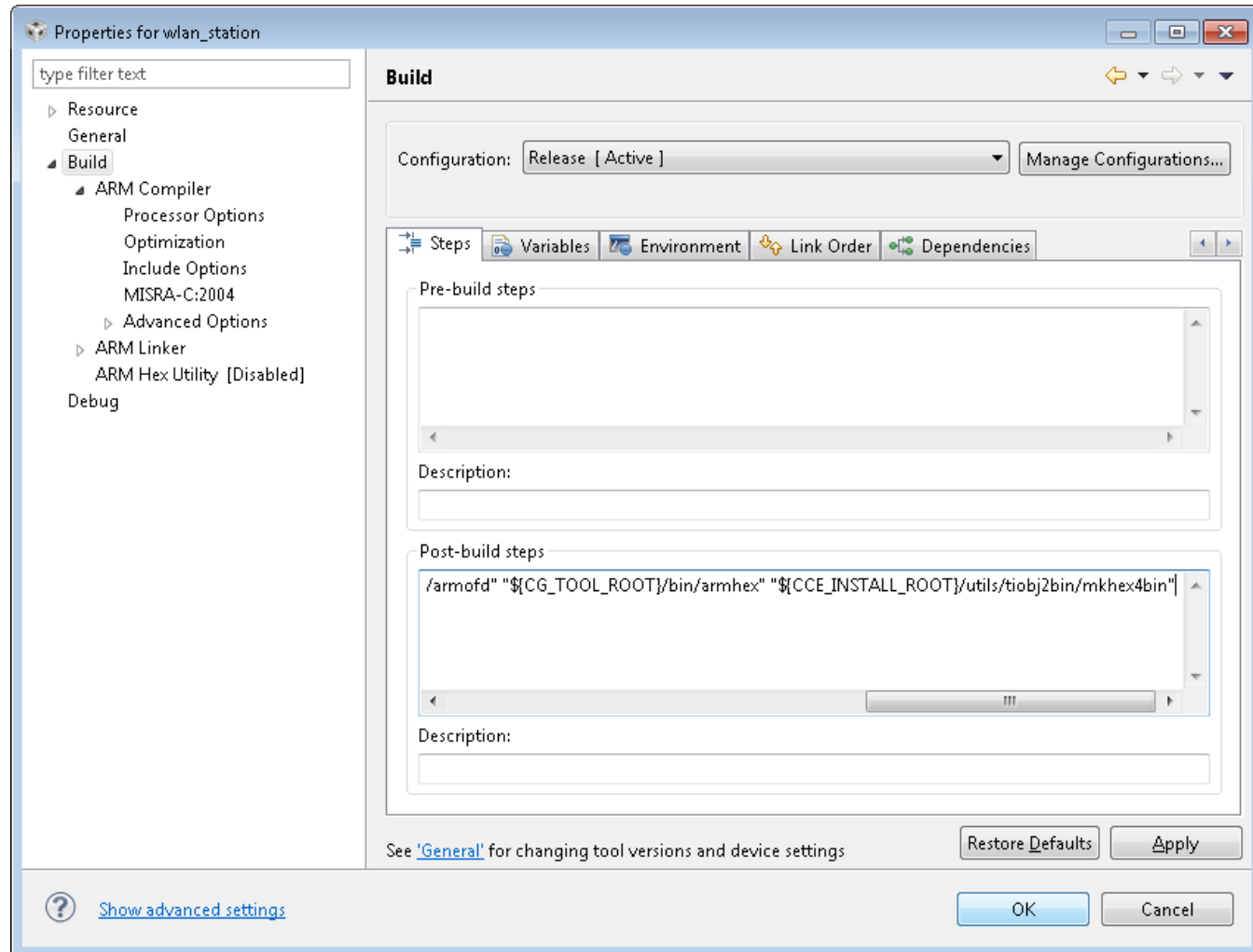
- If the application uses TI-RTOS OS, add ti_rtos_config project as dependency for the application.
 - ti_rtos_config project should be imported in CCS workspace for TI-RTOS-based application.
 - Current SDK supports TI-RTOS 2.15.0.x

5.3.2.8 Generating a Binary (.bin)

- Add the following script to generate a .bin file

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

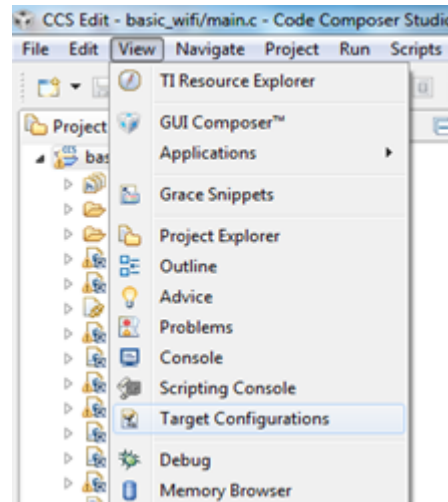

Figure 38. CC3200 CCS Generating Binary



5.3.2.9 Executing a Project

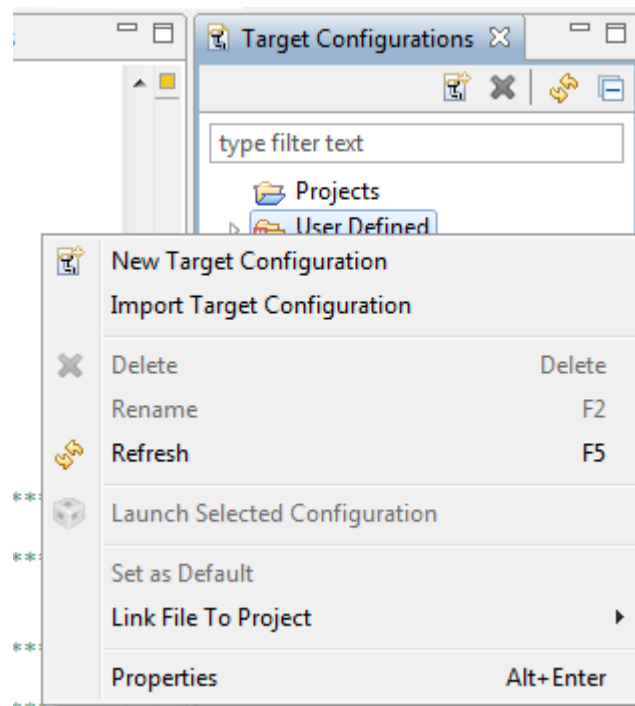
Click on the target configuration under View.

Figure 39. CC3200 CCS Executing 1



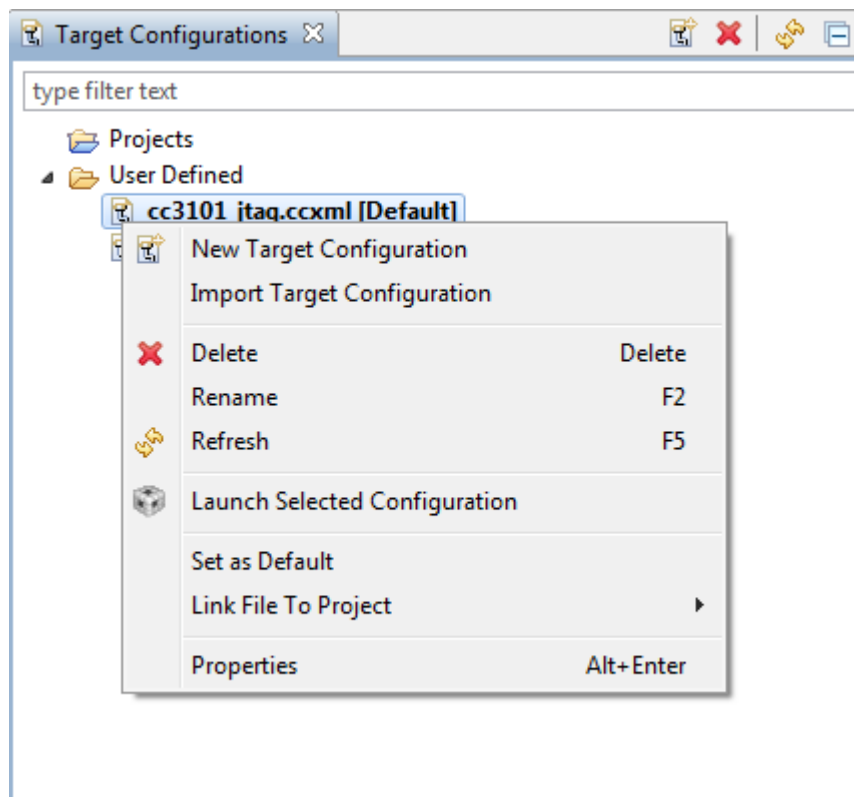
Right-click on the User Defined, click on Import Target Configuration, and select CC3200.ccxml from /tools/ccs/.

Figure 40. CC3200 CCS Executing 2



Set this new configuration as the default. Right-click on this configuration and select Launch Selected Configuration.

Figure 41. CC3200 CCS Launch Config

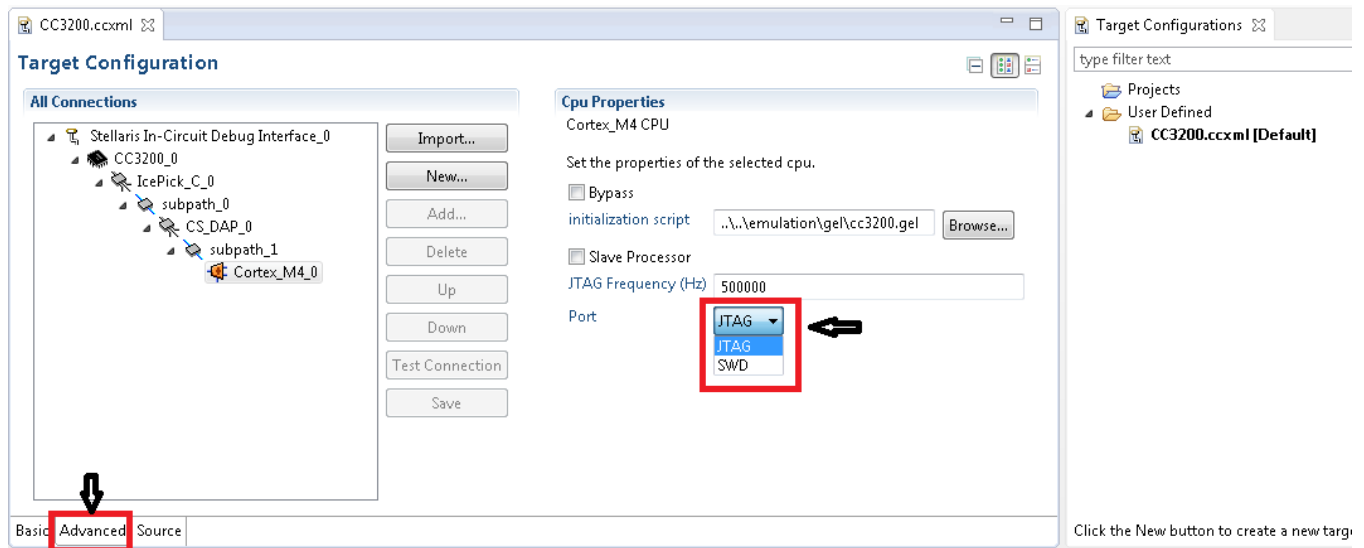


To switch between JTAG/SWD mode from CCS, follow the steps specified in [Figure 42](#). On the CC3200 LaunchPad, configure the board for either:

- JTAG Mode – connect SOP-2 jumper only
- SWD mode – connect SOP-0 jumper only

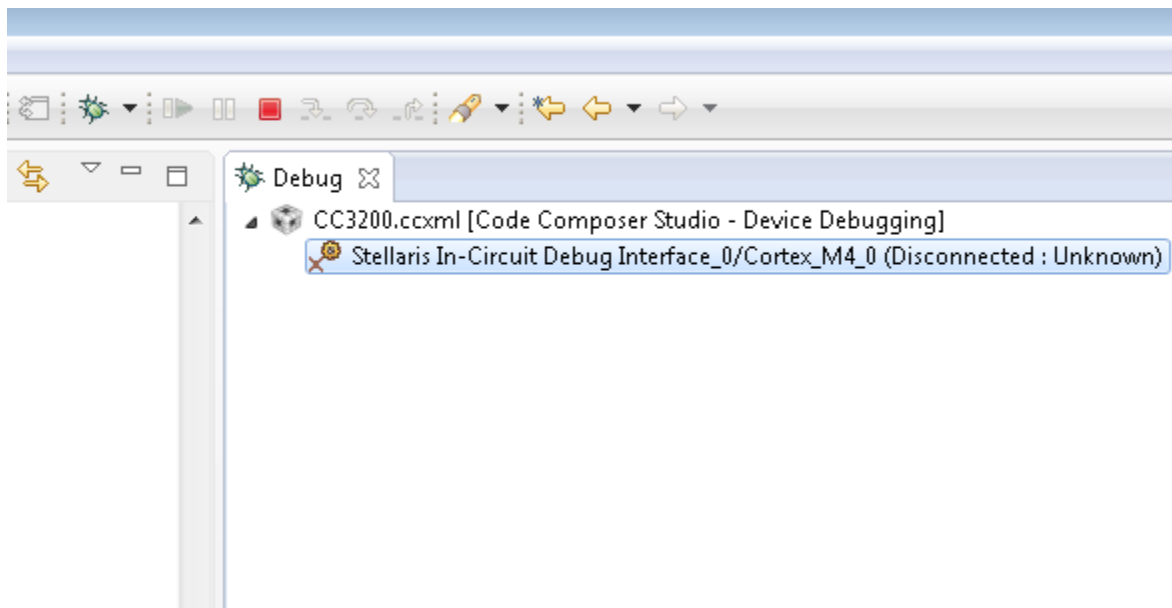
NOTE: 4-Wire JTAG works without connecting the SOP 2 jumper. However, TI recommends connecting the SOP 2 jumper to prevent the already-flashed code from executing while debugging.

Figure 42. Target Configuration

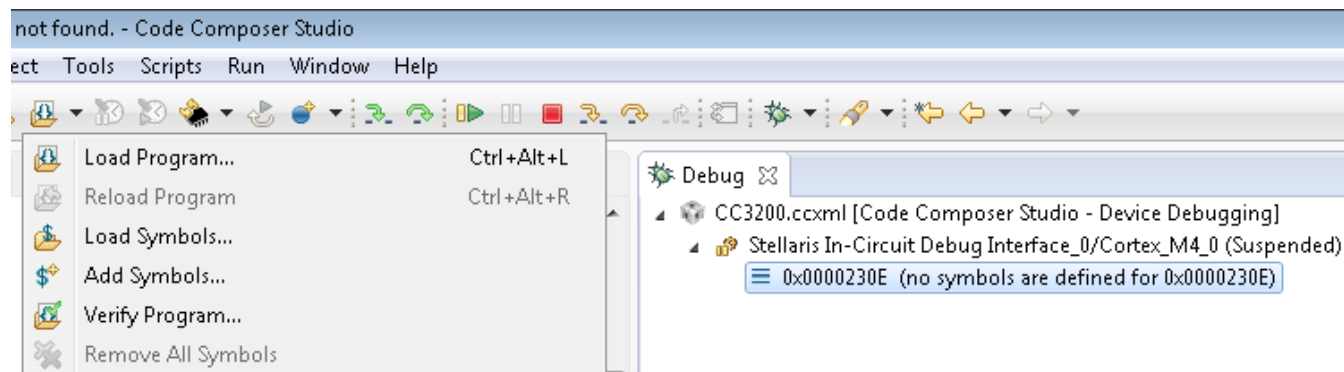


In the Debug window, right-click on Connect Target.

Figure 43. CC3200 CCS Executing 4



Once connected, load the .out file by selecting the appropriate application binary (Load Program).

Figure 44. CC3200 CCS Executing 5


The execution stops at the main function. Click the Go button (or F8) to run.

For UART-based applications, configure the terminal application:

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

5.3.3 Getting Started With GCC

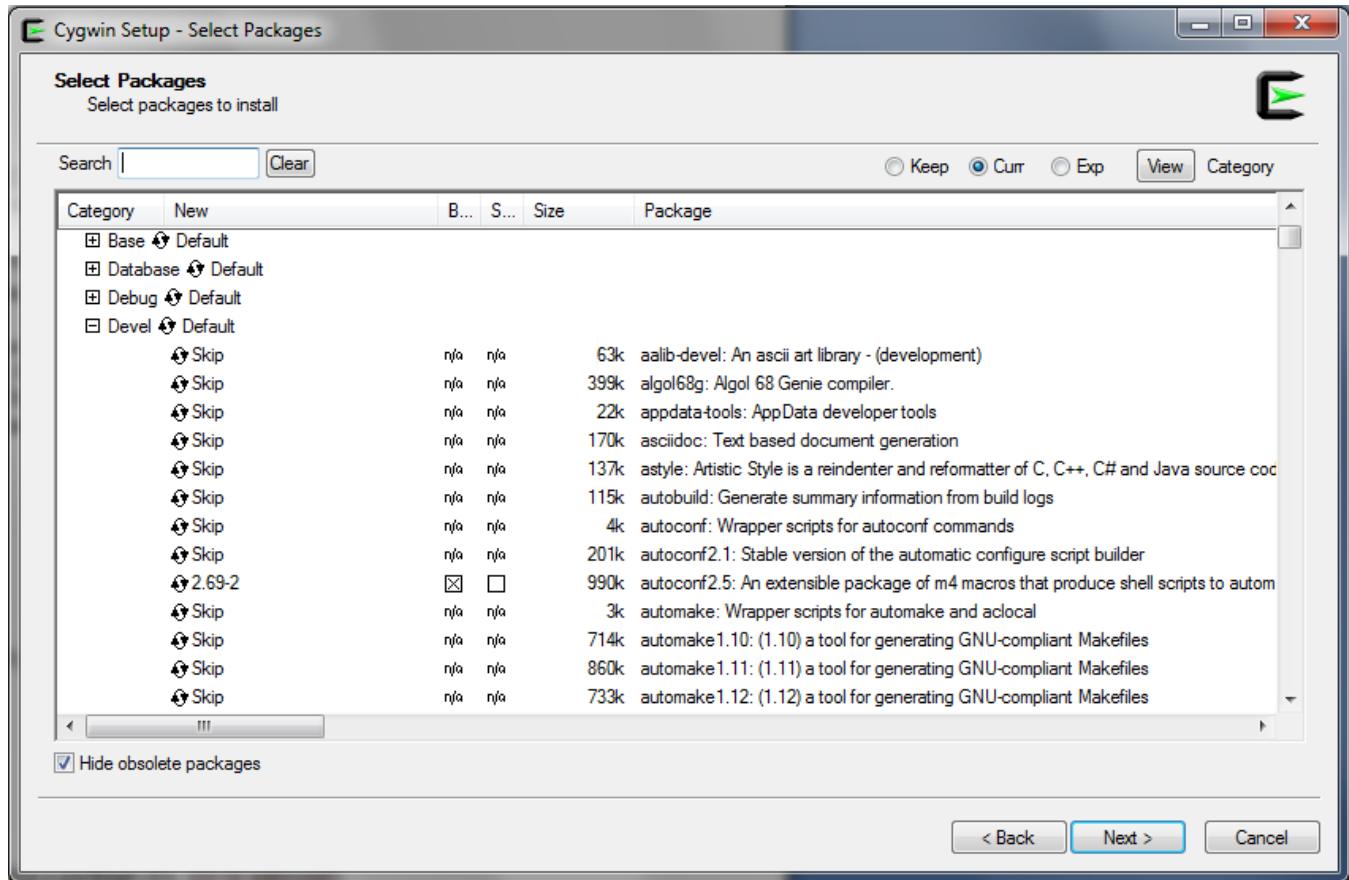
This platform enables open source tool chains. In this section, developers can learn how to get started with GCC/GDB using the CC3200 LaunchPad, including the dependencies associated with the environment setup for Windows OS under Cygwin. Included are a few validated sample applications with GCC, including building block libraries, such as the SimpleLink library and peripheral driver library.

5.3.3.1 Install Cygwin (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 (**Note:** if using TortoiseSVN/Windows7, skip this file)
 - Devel/gcc-core
 - Devel/gcc-g++
 - Devel/mingw-gcc-core
 - Devel/mingw-gcc-g++
 - Devel/mingw-runtime

See [Figure 45](#) for an example of selecting a package (as example: Devel/autoconf).

Figure 45. Cygwin Setup



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.

5.3.3.2 GNU Tools for ARM Embedded Processors

Download the latest version of `gcc-arm-none-eabi-<version>-win32.exe` from <https://launchpad.net/gcc-arm-embedded>, and install it 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.

5.3.3.3 Open On-Chip Debugger (OpenOCD)

1. Download prebuilt `openocd-0.9.0` for Windows from <http://sourceforge.net/projects/openocd/files/openocd/0.7.0/> 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.
3. Download the Zadig USB installation driver from <http://zadig.akeo.ie/>.
4. Run the `zadig_<version>.exe` and install the WinUSB driver for USB<->JTAG/SWD (Interface 0) debuggers (refer to Figure 46 and Figure 47). The IAR and CCS debugger do not work after this: the drivers must be reconfigured to work with these (refer to Section 5.3.3.6).

Figure 46. Zadig Options

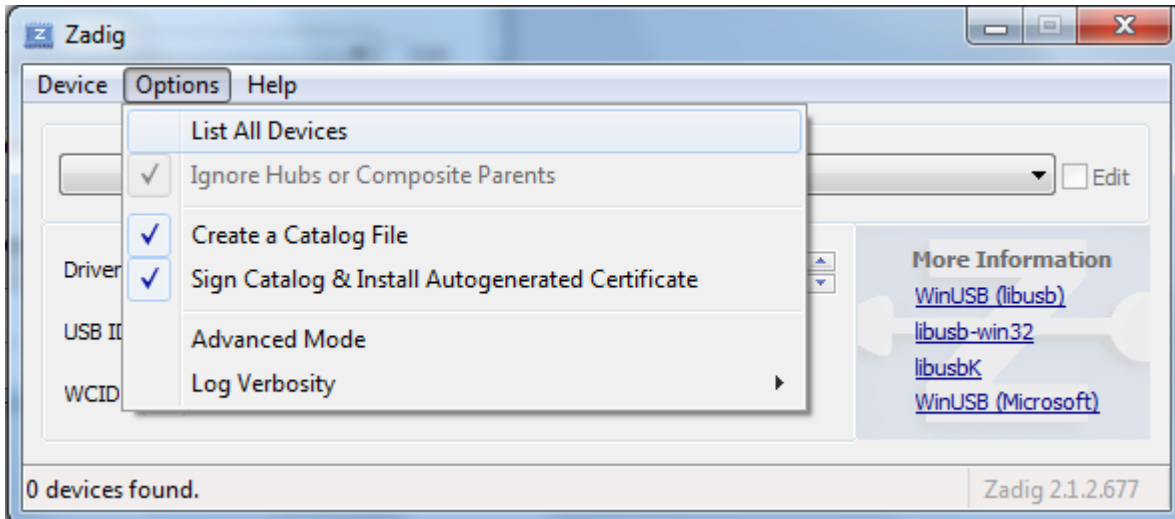
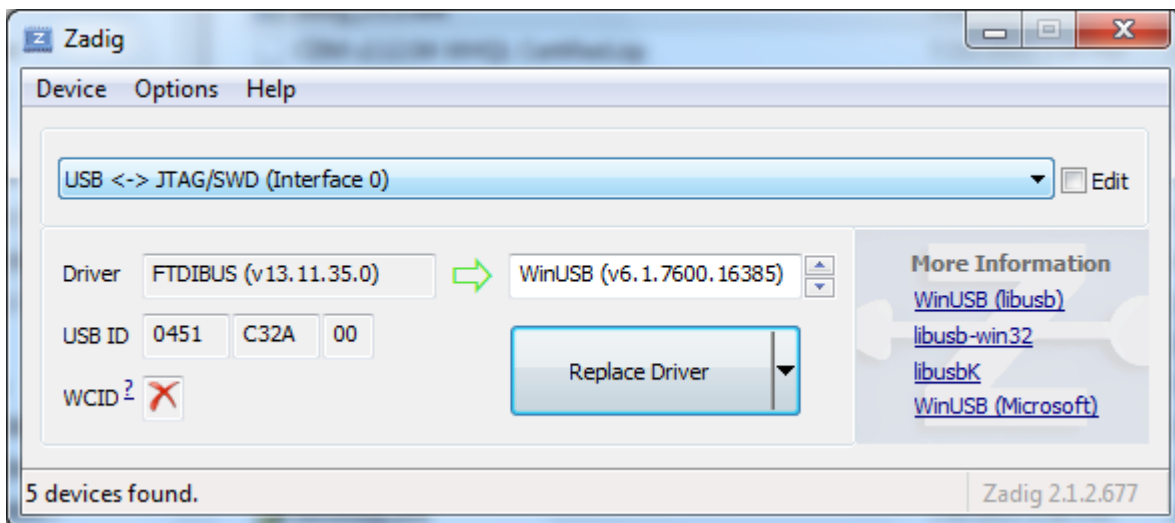


Figure 47. Replace Driver



5.3.3.4 Compile the GCC SDK Project

1. Open the common.h file located at the path C:\TI\CC3200SDK_1.2.0\cc3200-sdk\example\common\.
2. Edit common.h to use the SSID, security type, and security key of the AP. Edit the macros SSID_NAME, SECURITY_TYPE, and SECURITY_KEY to contain the AP information as shown in Figure 48. The security types supported for this demo are WPA/WPA2 and Open. For Open security, define SECURITY_TYPE as SL_SEC_TYPE_OPEN. For WPA and WPA2 security, define it as SL_SEC_TYPE_WPA.

Figure 48. Editing common.h

```
// Values for below macros shall be modified as per access-point(AP)
// SimpleLink device will connect to following AP when application
//
#define SSID_NAME "cc3200demo" /* AP SSID */
#define SECURITY_TYPE SL_SEC_TYPE_OPEN /* Security type (OPEN)
#define SECURITY_KEY "" /* Password of the security key
#define SSID_LEN_MAX 32
#define BSSID_LEN_MAX 6

// Values for below macros shall be modified as per access-point(AP)
// SimpleLink device will connect to following AP when application
//
#define SSID_NAME "Your_AP_Name_Here" /* AP SSID */
#define SECURITY_TYPE SL_SEC_TYPE_WPA /* Security type (WPA)
#define SECURITY_KEY "Your_AP_Security_Key_Here"
#define SSID_LEN_MAX (32)
#define BSSID_LEN_MAX (6)
```

3. Save common.h.
4. In the Cygwin terminal, change the directory to `C:\TI\CC3200SDK_1.2.0\cc3200-sdk\example\getting_started_with_wlan_station\gcc\` and run the following command:

```
make -f Makefile
```

Note that Cygwin uses forward slashes to separate the directories.
5. Go to `<cc3200-sdk>\example\getting_started_with_wlan_station\gcc\` in the command prompt and run following command:

```
make -f Makefile
```
6. This generates the wlan_station.axf file under the gcc\exe folder.

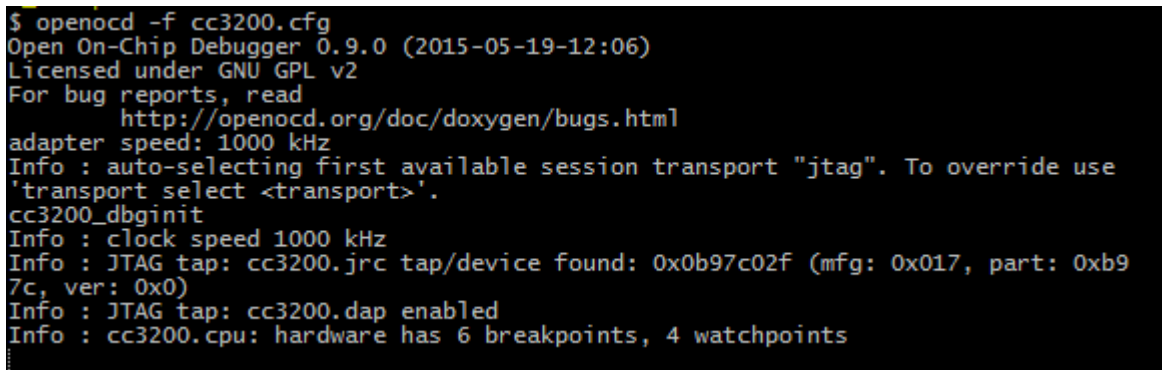
5.3.3.5 Target Connection and Debugging (GDB)

1. The OpenOCD configuration file for FTDI is present under the `C:\TI\CC3200SDK_1.2.0\cc3200-sdk\tools\gcc_scripts\` folder. To test the connection to the CC3200 FTDI Launchpad, navigate to the `<cc3200-sdk>\tools\gcc_scripts` folder, run the following command at the Cygwin prompt window, and check the output to see if the connection happened properly. Cygwin may need to be restarted before this step.

```
openocd -f cc3200.cfg
```

See [Figure 49](#) for the connection output screen while the CC3200 device is connected through GDB.

Figure 49. Output Screen



```
$ openocd -f cc3200.cfg
Open On-Chip Debugger 0.9.0 (2015-05-19-12:06)
Licensed under GNU GPL v2
For bug reports, read
  http://openocd.org/doc/doxygen/bugs.html
adapter speed: 1000 kHz
Info : auto-selecting first available session transport "jtag". To override use
'transport select <transport>'.
cc3200_dbginit
Info : clock speed 1000 kHz
Info : JTAG tap: cc3200.jrc tap/device found: 0x0b97c02f (mfg: 0x017, part: 0xb9
7c, ver: 0x0)
Info : JTAG tap: cc3200.dap enabled
Info : cc3200.cpu: hardware has 6 breakpoints, 4 watchpoints
```

2. Press `<ctrl>+c` to return to prompt.
3. Copy the wlan_station.axf file found in `C:\TI\CC3200SDK_1.2.0\cc3200-sdk\example\getting_started_with_wlan_station\gcc\exe\` to the directory `C:\TI\CC3200SDK_1.2.0\cc3200-sdk\tools\gcc_scripts\`.
4. Launch Tera Term, and create a new serial connection to the CC3200 Launchpad COM port.
5. To start debugging using GDB on CC3200, go to `C:\TI\CC3200SDK_1.2.0\cc3200-sdk\tools\gcc_scripts\` and run the following command at the Cygwin prompt:

```
arm-none-eabi-gdb -x gdbinit wlan_station.axf
```

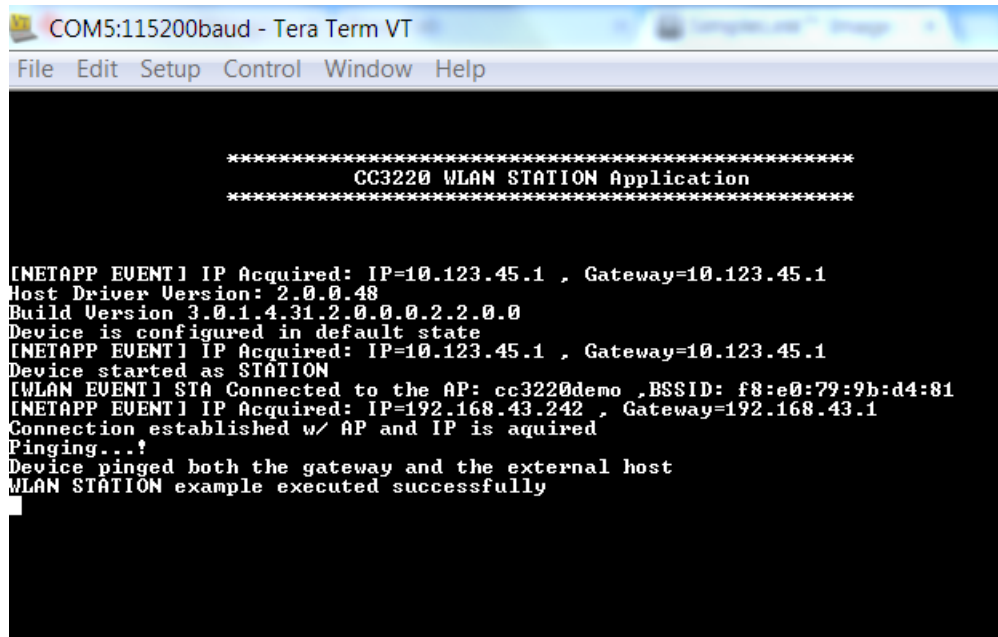

Figure 50. Debugging wlan_station

```
$ arm-none-eabi-gdb -x gdbinit wlan_station.axf
GNU gdb (GNU Tools for ARM Embedded Processors) 7.8.0.20150604-cvs
Copyright (C) 2014 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"...
Reading symbols from wlan_station.axf...done.
Open On-Chip Debugger 0.9.0 (2015-05-19-12:06)
Licensed under GNU GPL v2
For bug reports, read
http://openocd.org/doc/doxygen/bugs.html
0x20007940 in _sl_HandleAsync_PingResponse (pVoidBuf=0x8d1d)
at ../source/netapp.c:1007
1007         if (NULL != g_pCB->ObjPool[g_pCB->FunctionParams.AsyncExt.ActionIndex].pRespArgs)
Loading section .text, size 0xcd28 lma 0x20004000
Loading section .ARM, size 0x8 lma 0x20010d28
Loading section .data, size 0x898 lma 0x20010d30
Start address 0x200053c4, load size 54728
Transfer rate: 65 KB/sec, 3648 bytes/write.
Breakpoint 1 at 0x20004f16: file ../main.c, line 946.

Breakpoint 1, main () at ../main.c:946
946     long lRetVal = -1;
(gdb) :
```

This results in a GDB prompt. To continue, type 'continue' and press enter.

6. If the CC3200 successfully completes all steps, the serial output appears as shown in [Figure 51](#). See [Figure 50](#) for the result of debugging the wlan_station application from GCC.

Figure 51. Tera Term VT


```
COM5:115200baud - Tera Term VT
File Edit Setup Control Window Help

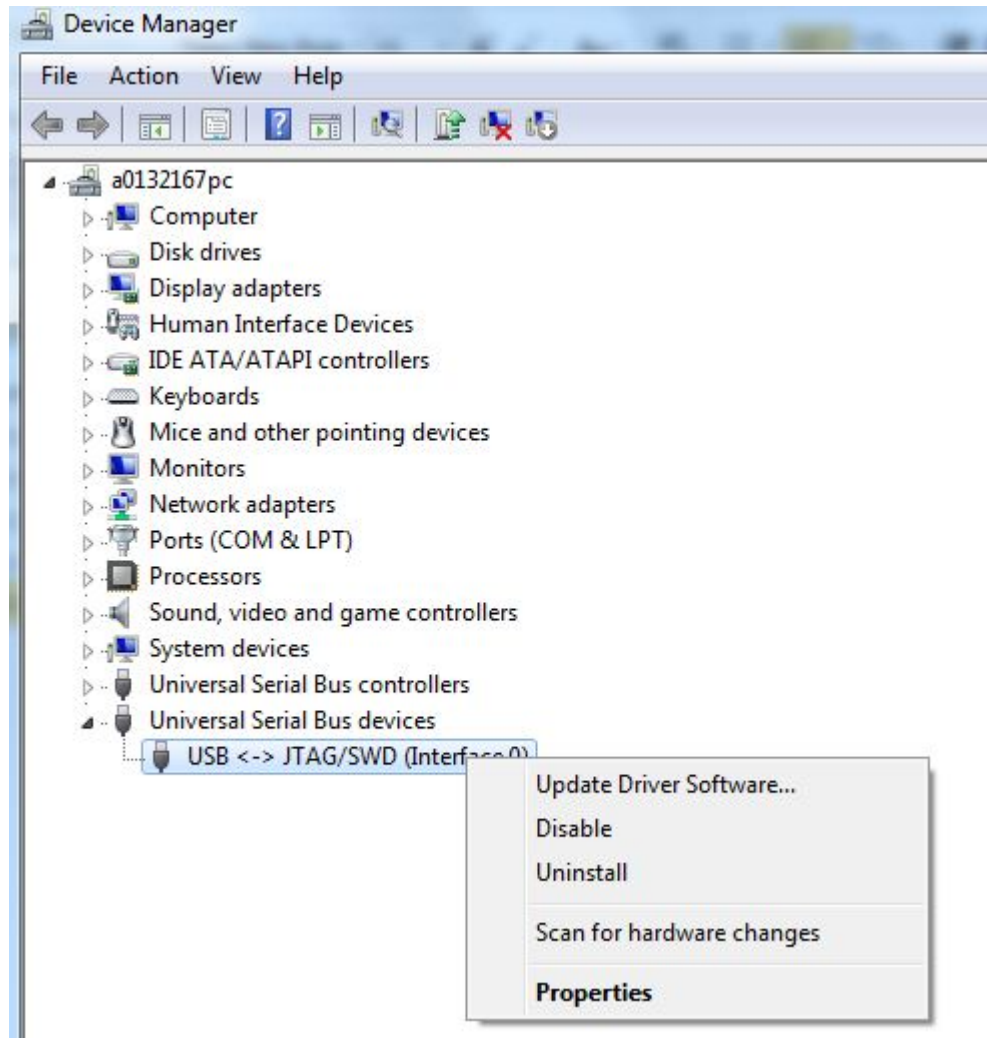
*****
CC3220 WLAN STATION Application
*****

[INETAPP EVENT] IP Acquired: IP=10.123.45.1 , Gateway=10.123.45.1
Host Driver Version: 2.0.0.48
Build Version 3.0.1.4.31.2.0.0.0.2.2.0.0
Device is configured in default state
[INETAPP EVENT] IP Acquired: IP=10.123.45.1 , Gateway=10.123.45.1
Device started as STATION
[WLAN EVENT] STA Connected to the AP: cc3220demo ,BSSID: f8:e0:79:9b:d4:81
[INETAPP EVENT] IP Acquired: IP=192.168.43.242 , Gateway=192.168.43.1
Connection established w/ AP and IP is aquired
Pinging...!
Device pinged both the gateway and the external host
WLAN STATION example executed successfully
```

5.3.3.6 Driver Reconfiguration

1. To work with CCS or IAR, go to the device manager.
2. Update the driver software for USB<->JTAG/SWD (Interface 0) under Universal Serial Bus devices.

3. Select the <cc3200-sdk>\tools\ftdi folder for the driver update, as shown in [Figure 52](#).

Figure 52. Device Manager


5.3.4 Setup the Terminal Application

To view the UART-based application output on the terminal screen, set up the terminal application (such as HyperTerminal or TeraTerm). Serial port settings:

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

5.4 Flashing and Running the .bin Using the Uniflash Tool

Once finalized, the binary images are flashed onto the non-volatile serial flash (SFlash) of the LaunchPad. The application starts the execution when the LaunchPad device is powered on. The Uniflash tool flashes the binaries onto the SFlash. The utility is available at http://processors.wiki.ti.com/index.php/Category:CCS_UniFlash.

Follow the [Uniflash Quick Start Guide](#) to download the .bin file to the CC3200 device.

NOTE: Connect the SOP 2 jumper on the LaunchPad before flashing any image to the device. After the flashing is done, remove the SOP 2 jumper and reset the board to boot-up the application.

The .bin files for the reference examples are available in the folder *example\<referenceapp>\ccs\Release* (as generated by CCS) and *example\<referenceapp>\ewarm\Release\exe* (as generated by IAR). The application image filename must be */sys/mcuimg.bin*.

6 CC3200 ROM Services

The CC3200 ROM hosts the bootloader and peripheral driver library. The peripheral driver library is a collection of routines that abstract the peripheral programming (refer to accompanying doxygen output in the CC3200 SDK packages). This library is provided in the ROM to let the developer reduce the RAM footprint of the application.

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

6.1 CC3200 Bootloader

The CC3200 bootloader resides in the ROM of the application processor.

- **Update/Download** – The bootloader downloads an application image from the PC to the CC3200 device. The Bootloader-DNLD functionality is only triggered when the board is in UARTLOAD Sense-On-Power (SOP) mode.
- **Bootstrap** – The bootloader is responsible for scanning a valid application image (binary) in the serial flash (for the CC3200R device). Subsequently, the image is loaded to internal memory and execution control is passed on to the user program.

6.1.1 Bootloader Modes – Impact of Device Sense-On-Power (SOP) Pin

The CC3200 device has three SOP pins. A detailed explanation of the functionality is described in the data sheet ([SWRS166](#)). In the context of the boot loader there are two modes:

- A setting corresponding to SOP[2:0] = 0b100, makes the bootloader enter the DOWNLOAD mode; in this mode an external intervention triggers an operation – for example a break signal on UART from the SimpleLink programming application, followed by a sequence to push the application image to the device serial flash.
- A setting corresponding to SOP[2:0] = 0b000 instructs the bootloader to load the application image from the SFLASH to the internal MCU RAM.

6.1.2 Bootloader and User Application – Sharing MCU RAM

In DOWNLOAD mode, the bootloader requires memory resources acquired from the MCU RAM. The amount of RAM used by the bootloader is 16 KB. This implies that for the production variant of a CC3200 device, the user application image must be restricted to 240 KB for the 256-KB MCU RAM variant of the CC3200. There are several key points for developers:

- **MCU RAM address range 0x20000000 - 0x20003FFF:** This area is shared between the application and the bootloader. The developer can only locate application data sections, as data sections are not part of the application image; this ensures that when the bootloader is loading the application image from 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.
- RAM range for different variants:
 - CC3200R1M1: 0x20004000 - 0x20020000
 - XCC3200JR, CC3200R1M2: 0x20004000 - 0x20040000

Refer to the Blinky Linker command file (CCS /IAR / GCC), which defines 240 KB of RAM for the XCC3200JR and CC3200R1M2 devices.

Table 3. End of RAM

Exclusively for application. Application should be part of this region and start at 0x20004000 [0x20004000]	END of RAM should be part of this region and start at 0x20004000 [0x20004000]
16 KB Shared between bootloader and application [0x20000000]	

6.2 CC3200 Peripheral Driver Library Services in ROM

Peripheral driver routines are used in the CC3200 MCU ROM for linking with user applications. Entire source codes of the peripheral driver routines are available in the CC3200 SDK. The developer could choose to build an application with the library while instructing the linker to use routines directly from RAM.

The focus of this section is to appraise the developer on how to use these routines, and the procedure to patch or extend any existing routines.

6.2.1 Peripheral Library Access in ROM

ROM APIs are invoked using the following re-direction flow to allow future extensions while retaining backward compatibility of location of access functions in the ROM memory map. While the API locations may change in future versions of the ROM, the API tables will not.

Two tables in the ROM resolve to the entry point of each supported API. Access is made through two levels; the main table contains one pointer per peripheral, which points to a secondary table that contains one pointer per API associated with that peripheral.

The main table is located at address 0x0000040C in the ROM. The following table shows a small portion of the API tables in a graphical form to illustrate the arrangement of the tables:

Table 4. ROM APIs

ROM_API TABLE (at 0x0000040C)
[0] = RESERVED
[1] = pointer to ROM_UARTTABLE
[2] = pointer to ROM_TIMERTABLE
[3] = pointer to ROM_WATCHDOGTABLE
[4] = pointer to ROM_INTERRUPTTABLE
[5] = pointer to ROM_UDMTABLE
[6] = pointer to ROM_PRCMTABLE
[7] = pointer to ROM_I2CTABLE
...

Table 5. ROM Interrupts

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

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

ROM_INTERRUPTTABLE is an array of pointers located at ROM_APITABLE[4].

ROM_IntMasterEnable is a function pointer located at ROM_INTERRUPTTABLE [1].

6.2.2 Linking User Application with ROM APIs

Below are the steps to use ROM driver lib APIs instead of the RAM APIs. These APIs could be used with the production version of a CC3200 device. These steps apply to all relevant source and project files that use driver lib APIs, such as the SimpleLink and oslib library.

1. All the .c files which use driver lib APIs should include these header files in order:
 - `#include rom.h`
 - `#include rom_map.h`
2. All the project files should add the global preprocessor define `TARGET_IS_CC3200`.
3. All driver lib APIs should be invoked 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 relevant projects.

6.2.3 Patching ROM APIs

Follow these steps to selectively patch the ROM driver lib APIs. Note that “patch” in this description means using the RAM driver lib API instead of the ROM driver lib API.

1. Add an entry in the file `\driverlib\rom_patch.h` for all APIs to be patched.
2. For example, to patch `MAP_UARTCharPut` and `MAP_UARTBreakCtl` entries in file `rom_patch.h`:
 - `#undef ROM_UARTCharPut`
 - `#undef ROM_UARTBreakCtl`
3. Rebuild all the relevant projects that use driver lib APIs.

6.2.4 Linking with RAM-based Peripheral Driver Library

To de-link all ROM driver lib APIs and use the RAM driver lib APIs, follow these steps:

1. Remove the global preprocessor define `TARGET_IS_CC3200` from all project files that use driver lib APIs.
2. Rebuild all the relevant projects that use driver lib APIs.

7 Additional Resources

Visit these links for additional resources on the SimpleLink Wi-Fi CC3200 and IoT Solution, a single-chip wireless MCU device.

- [CC32xx Wiki](#) – All additional resources.
- [TI Product Folder for CC32xx](#).
- [CC32xx SimpleLink Host Driver APIs](#) and [CC32xx Peripheral Drivers APIs](#).
- [CC32xx Technical Reference Manual](#).

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from March 20, 2015 to February 29, 2016 (from B Revision (March 2015) to C Revision)	Page
• Removed Provisioning with SmartConfig component.	11
• Replaced Deepsleep Usage with Provisioning with SmartConfig.	11
• Removed Provisioning AP component.	11
• Added Power Measurement example.	12
• Updated CCS v6.0.1 to 6.1.1.	13
• Updated CC3200 SimpleLink IAR Config Switch image.	16
• Updated CC3200 CCS SimpleLink Config Switch image.	17
• Added Development versus Deployment Scenario section.	17
• Updated CCS App Center image.	46
• Updated Select CCS Projects to Import image.	48
• Updated CC3200 CCS Creating Project image.	50
• Updated Getting Started With GCC section.	61
• Updated Compile the GCC SDK Project section.	63
• Updated Tera Term VT image.	65
• Added Driver Reconfiguration section.	65

Revision History

Changes from July 1, 2014 to March 11, 2015	Page
• Added Application Bootloader, HTTP Client Demo, Idle Profile (Non-OS), MQTT Client, MQTT Server, OTA Updated, Dynamic Library Loader to examples in Package Contents table.	13
• Added http, json, and mqtt to NetApps in Package Contents table.	13
• Added Simplelink_extlib and Documents values to Package Contents table.	13
• Added HTTP Client Demo, Idle Profile (Non OS), MQTT Client, MQTT Server, and OTA Update to table.	23
• Added Application Bootloader and Dynamic Library Loader to table.	24
• Updated Simple Networking Applications section.	26
• Added Note.	44
• Added Note.	59
• Updated RAM range.	68

Revision History

Changes from Original (June 2014) to A Revision	Page
• Added Watchdog System Demo, TFTP Client, and WebSocket Camera values	12
• Added NetApps values to Package Contents table.	13
• Added Watchdog System Demo, TFTP Client, and WebSocket Camera to table.	23
• Deleted security macro.	26
• Added pinmux, gpio, and uart values.	26
• Updated Code Flow Connection.	28
• Added pinmux and uart values.	28
• Updated Code Flow Connection.	29
• Updated Application Details section.	30
• Added uart and udma values.	30
• Updated Code Flow Connection.	31
• Updated Application Details.	34

• Added uart and udma values.	34
• Updated Code Flow Connection.	36
• Added pinmux, startup_ccs, and startup_ewarm values.....	39
• Added NOTERM bullet point	42
• Added Note	46
• Added Importing a Project section.....	48
• Added macro.ini paragraph.	48
• Added CC3200 CSS Editing Existing Project image.	49
• Updated Compiling a Project images.	51
• Updated CC3200 CCS Compiling Project 2 image.....	53
• Updated CC3200 CCS Linking Project 2 image.	55
• Replaced main.c with common.h.	63
• Replaced Editing common.h image.	63
• Added RAM range for different variants.	68
• Deleted Changed address of the ROM_INTERRUPTTABLE table from 0x0000042C to 0x0000041C.....	69
• Changed ROM_INTERRUPTTABLE from ROM_APITABLE[5] to ROM_APITABLE[4].....	69

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information 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.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have **not** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products

Audio	www.ti.com/audio
Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
OMAP Applications Processors	www.ti.com/omap
Wireless Connectivity	www.ti.com/wirelessconnectivity

Applications

Automotive and Transportation	www.ti.com/automotive
Communications and Telecom	www.ti.com/communications
Computers and Peripherals	www.ti.com/computers
Consumer Electronics	www.ti.com/consumer-apps
Energy and Lighting	www.ti.com/energy
Industrial	www.ti.com/industrial
Medical	www.ti.com/medical
Security	www.ti.com/security
Space, Avionics and Defense	www.ti.com/space-avionics-defense
Video and Imaging	www.ti.com/video

TI E2E Community

e2e.ti.com