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

Programmer's Guide

Literature Number: SWRU369A
June 2014–Revised June 2014

Contents

List of Figures

List of Tables

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

1 Introduction

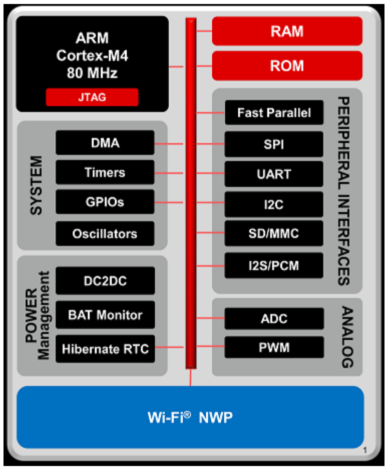
The CC3200 SimpleLink™ Wi-Fi®™ is the industry's first single-chip microcontroller (MCU) with built-in Wi-Fi connectivity, created for the Internet of Things (IoT). The CC3200 device is a wireless MCU that integrates a high-performance ARM Cortex-M4 MCU, allowing customers to develop an entire application 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 2.4GHz Wi-Fi PHY/MAC and TCP/IP networking engine with a microcontroller, up to 256kB on-chip RAM (In ES 1.32 and ES 1.21 devices, only 176kb of RAM is available for applications) 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

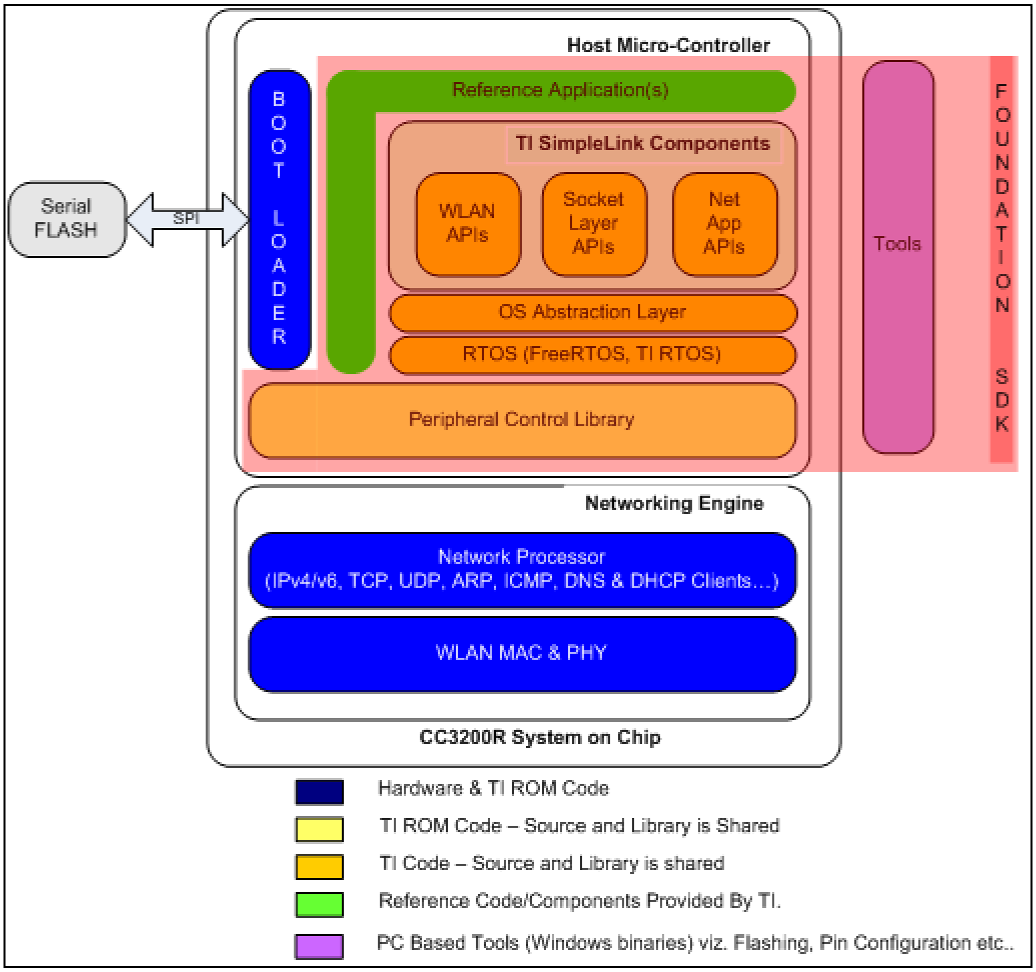


1.2 Software Components

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 and 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.

The following figure illustrates the various software components and their form in the CC3200 Foundation SDK.

Figure 2. CC3200 Software Components

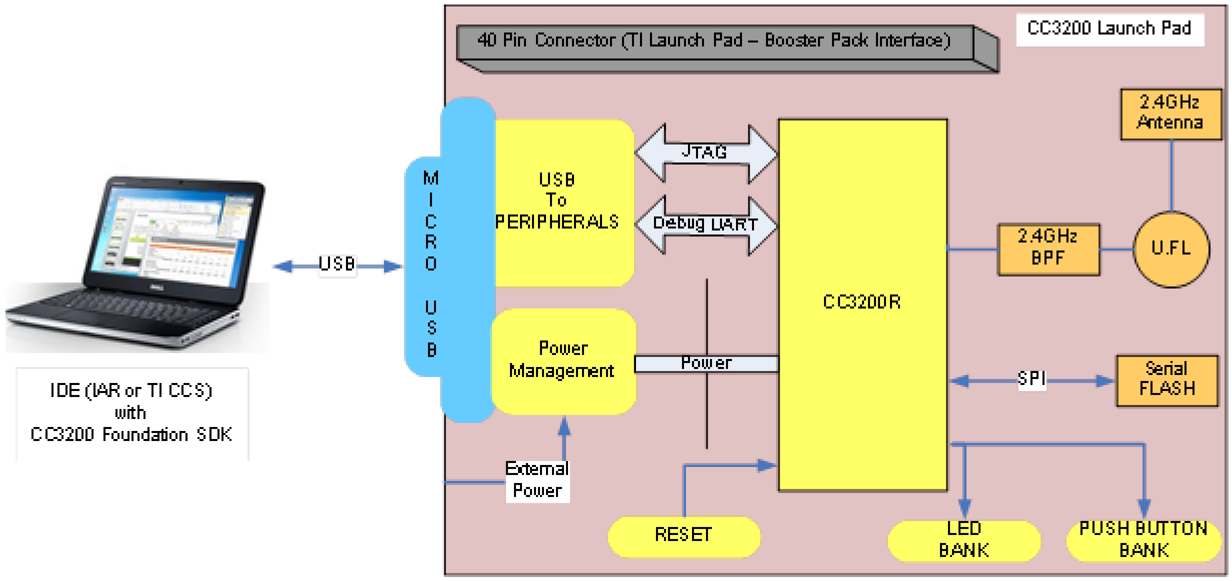


1.3 CC3200 LaunchPad Platform

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

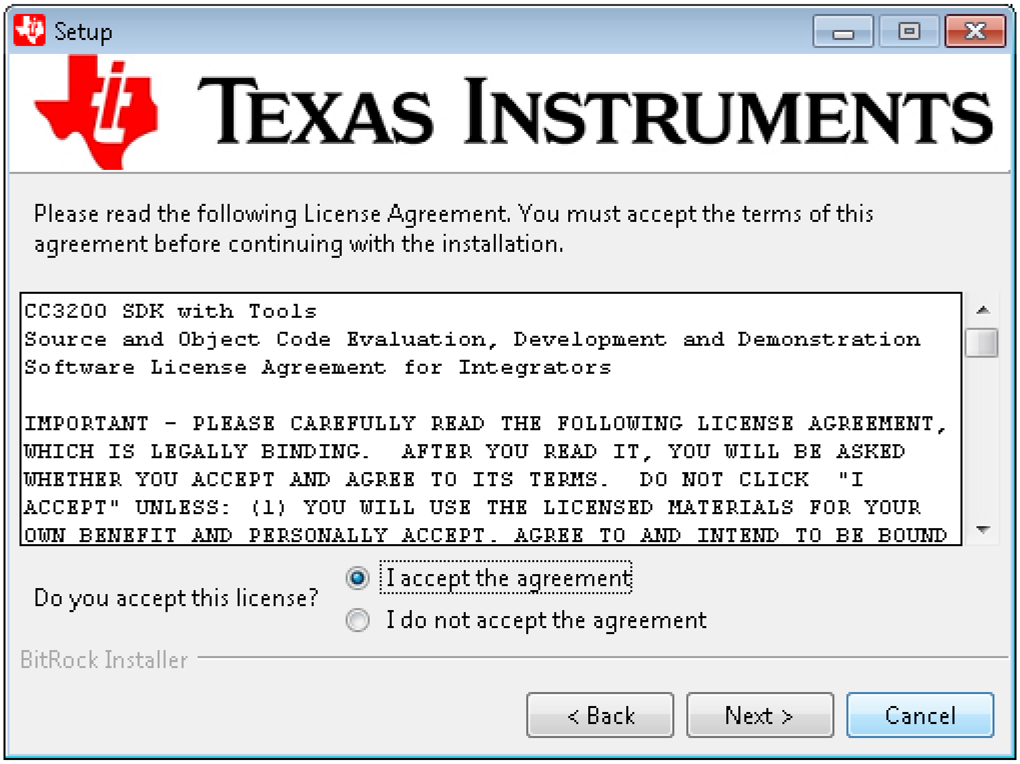
This section familiarizes the user with installation process and the directory structure of CC3200 Foundation SDK.

2.1 Installation

Run the installer by double clicking the CC3200 SDK installer.

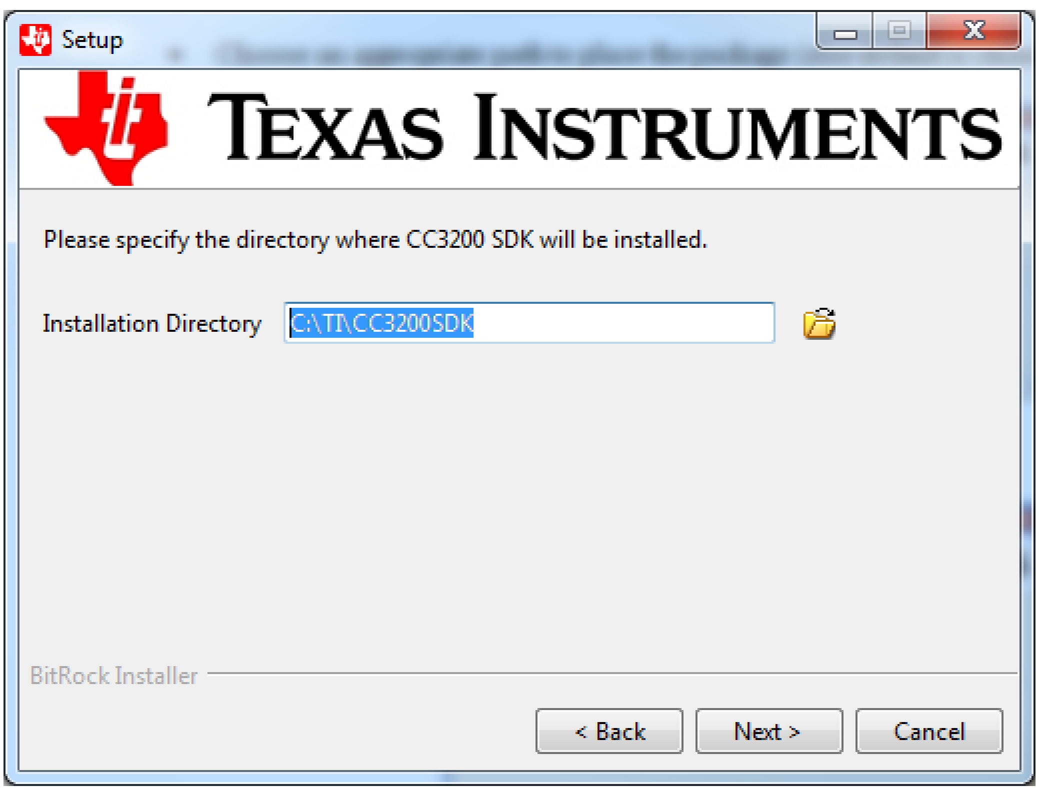
- Read and accept the license agreement to proceed.

Figure 4. CC3200 SDK Installation 1



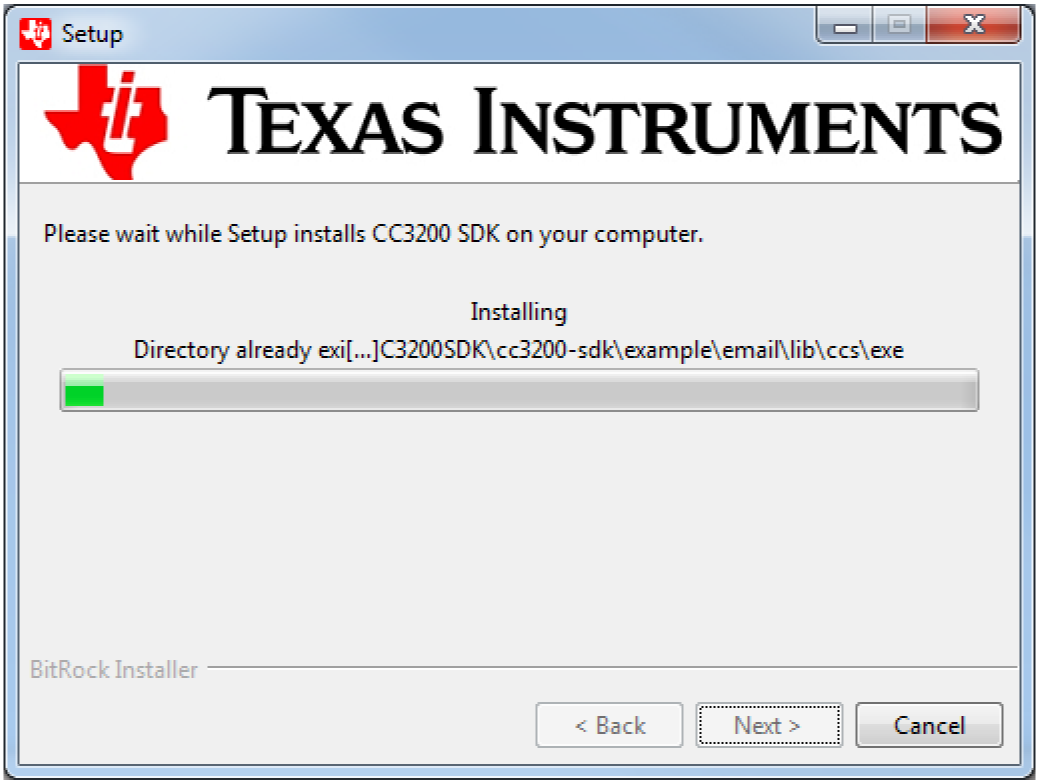
- Choose an appropriate path to place the package (else default is chosen).

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	• CC3200 Programmer's Guide • Documentation for hardware details present in 'hardware' folder • Documentation for 'SimpleLink Host Driver' in html format under *'docs\simplelink_api' directory* • Application Notes for all the sample application present in '*docs\examples*' directory. • Peripheral Driver Library User's Guide
driverlib	• Contains the peripheral driver library source files. • The driverlib.a is also provided in the ccs and ewarm directories.
example	• **Getting Started in STA Mode:** Configures CC3200 in STA mode. It verifies the connection by pinging the client connected to it. • **Getting Started in AP Mode:** Configures 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 CC3200. • **SSL:** SSL certificates are designed to provide two principles, privacy and authentication. Privacy is achieved by encryption/decryption and authentication is achieved by signature/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 SmartConfig:** Demonstrates the usage of TI's SmartConfig™ Wi-Fi provisioning technique. • **Provisioning with WPS:** Demonstrates the usage of WPS Wi-Fi provisioning with CC31xx/CC32xx. • **Deepsleep Usage:** Showcases deepsleep as a power saving tool in a networking context. • **Hib:** Showcases the hibernate as a power saving tool in a networking context (in this case an 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 via 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 UART interface. • **Connection Policy:** Demonstrates the connection policies in CC3200. The connection policies determine how the CC3200 connects to 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 CC3200. • **mDNS:** Demonstrates the usage of mDNS functionality in CC3200. The application showcases both "mDNS advertise" and "mDNS listen" functionality. • **Provisioning AP:** Demonstrates the AP provisioning feature. • **Mode config:** Used to switch 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 whenever the system fails.

Table 1. Package Contents (continued)

Directory Name		Information
example		• **UART Demo Application:** Showcases the usage of UART DriverLib APIs. The application demonstrates a simple echo of anything the user types on the terminal. • **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 Launch-Pad. • **MCU Sleep-DS:** Exercises the Sleep and Deepsleep 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 like multiple task creation and inter task 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 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 SD card, using the SD Host controller on CC3200. • **SPI Demo Application:** Displays the required initialization sequence to enable the CC3200 SPI module in full duplex 4-wire master and slave mode(s). • **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 CC3200 device. 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 CC3200 device. • **Timer Count Capture**: Showcases Timer's count capture feature 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).
Inc		• Contains the register definition header files.
Oslib		• Contains the interface file to configure Free-RTOS or TI-RTOS.
Middleware		• Contains power management framework to provide a simple infrastructure for developers to create a power aware solution.
Simplelink		• Contains 'SimpleLink Host Driver' code.
Third_ party	**FatFS**	• Contains the FatFS source files.
	FreeRT OS	• Contains the FreeRTOS source files.
Ti_rtos		• Contains the Ti RTOS config file and CCS project.

Table 1. Package Contents (continued)

Directory Name	Information
Tools	• **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 needs to be installed. After the installation, follow the tools\iar_patch\readme.txt to be able to debug over FTDI.	Installation: http://www.iar.com/Products/IAR-Embedded-Workbench/ARM/
Or/and		
CCS	CCS 6.0 version and 'TI v5.1.5' compiler version. After the installation, follow the tools\ccs_patch\readme.txt to be able to debug over FTDI.	Installation: http://www.ti.com/tool/ccstudio
CC3200 Support package in CCSv6.0	This package needs to be installed within CCS so that CCSv6.0 can support CC3200 device.	Refer to Section 5.3.2.1
Or/and		
GCC	To enable CC3200 SDK development on 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
CC32xx Programmer Utility		
Uniflash	Tool to download firmware, application image, and certificate to 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 need to 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:

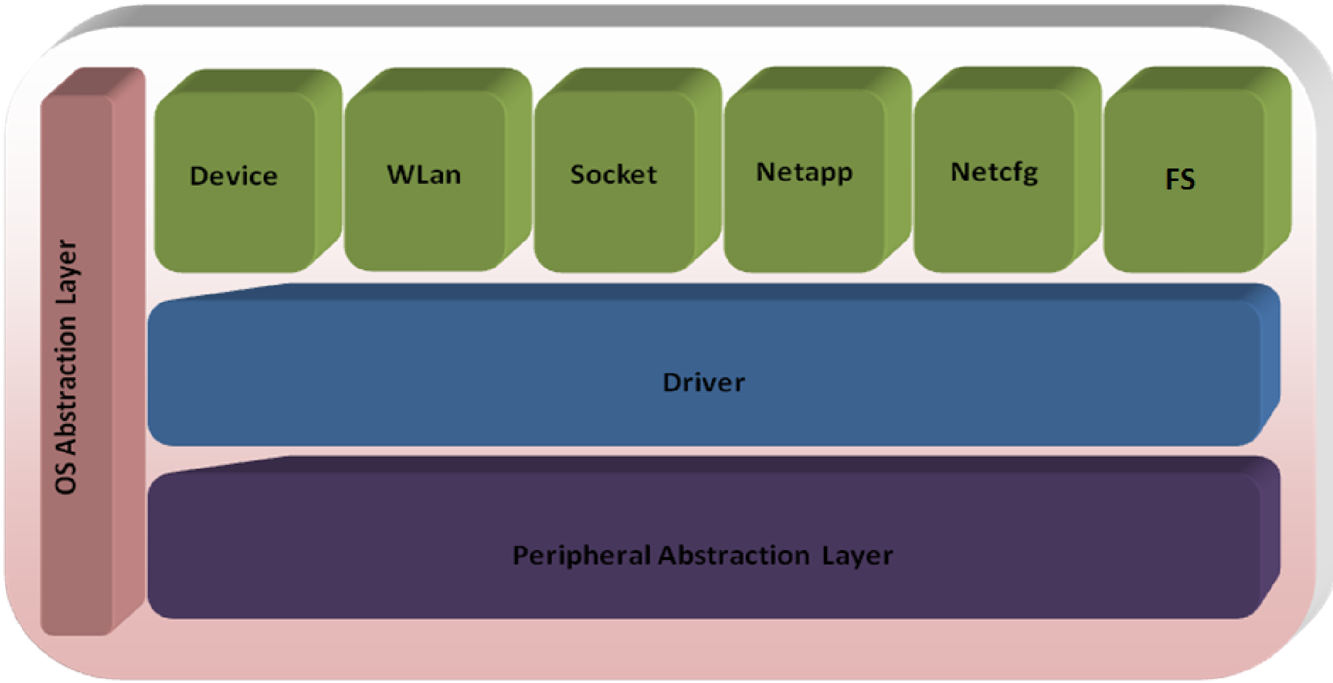
- SimpleLink Library – This library hosts APIs that serve the connectivity features.
- Peripheral Driver Library – This library hosts APIs to access MCU peripherals.

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 ranging from 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/capability of the SimpleLink Framework.

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

Components	Functionality
device	• Initializes the host • Controls the communication with the Network Processor
wlan	• Connection to the access point • Scan access points • Add/Remove access point profiles • WLAN Security
socket	• UDP/TCP Client Socket • UDP/TCP Server Socket • UDP/TCP Rx/Tx
netapp	• DNS Resolution • Ping remote device • Address Resolution Protocol
netcfg	• IP/MAC address configuration
fs	• File system Read/Write

3.1.2 Using the TI SimpleLink Framework

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.

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.

Memory management model – The SimpleLink driver supports two memory models:

- Static (default)
- Dynamic

The CC3200 default configuration is Static. In the dynamic model it will use the malloc and free as defined by the OS/operating system. If the user wishes to define their own memory management, they can define these interfaces.

Asynchronous event handlers routines – The SimpleLink device generates asynchronous events in several situations. These asynchronous events could be masked. Provide handler routines to catch these events. Note that if a handler routine was not provided and the event is received, the driver will drop this event without any indication of a drop.

Interface communication driver – The SimpleLink device supports several standard communication protocols among SPI and UART. The CC3200 Host Driver implements 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. If the user prefers to use UART, the above interfaces will need to be redefined.

OS adaptation – The SimpleLink driver can run on two kinds of platforms:

- Non-Os / Single Threaded (default)
- Multi-Threaded

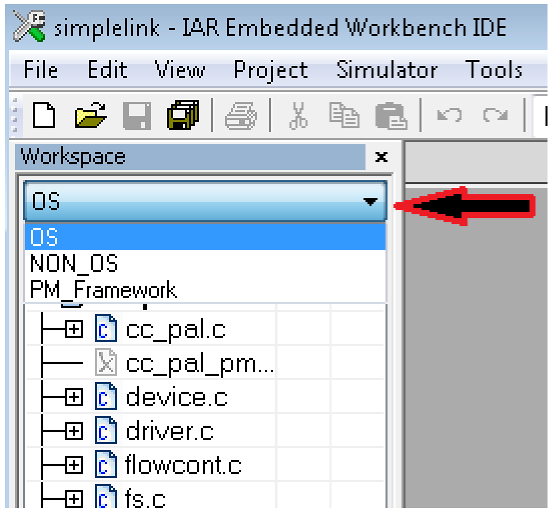
The CC3200 SimpleLink Host Driver is ported on both Non-Os and Multi-Threaded OS environment. The Host driver is made OS independent by implementing an OS Abstraction layer. Reference implementation for OS Abstraction is available for FreeRTOS and TI-RTOS.

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 will enable functions to run on a temporary context.

3.1.3 Switch Between OS and NON-OS Configuration

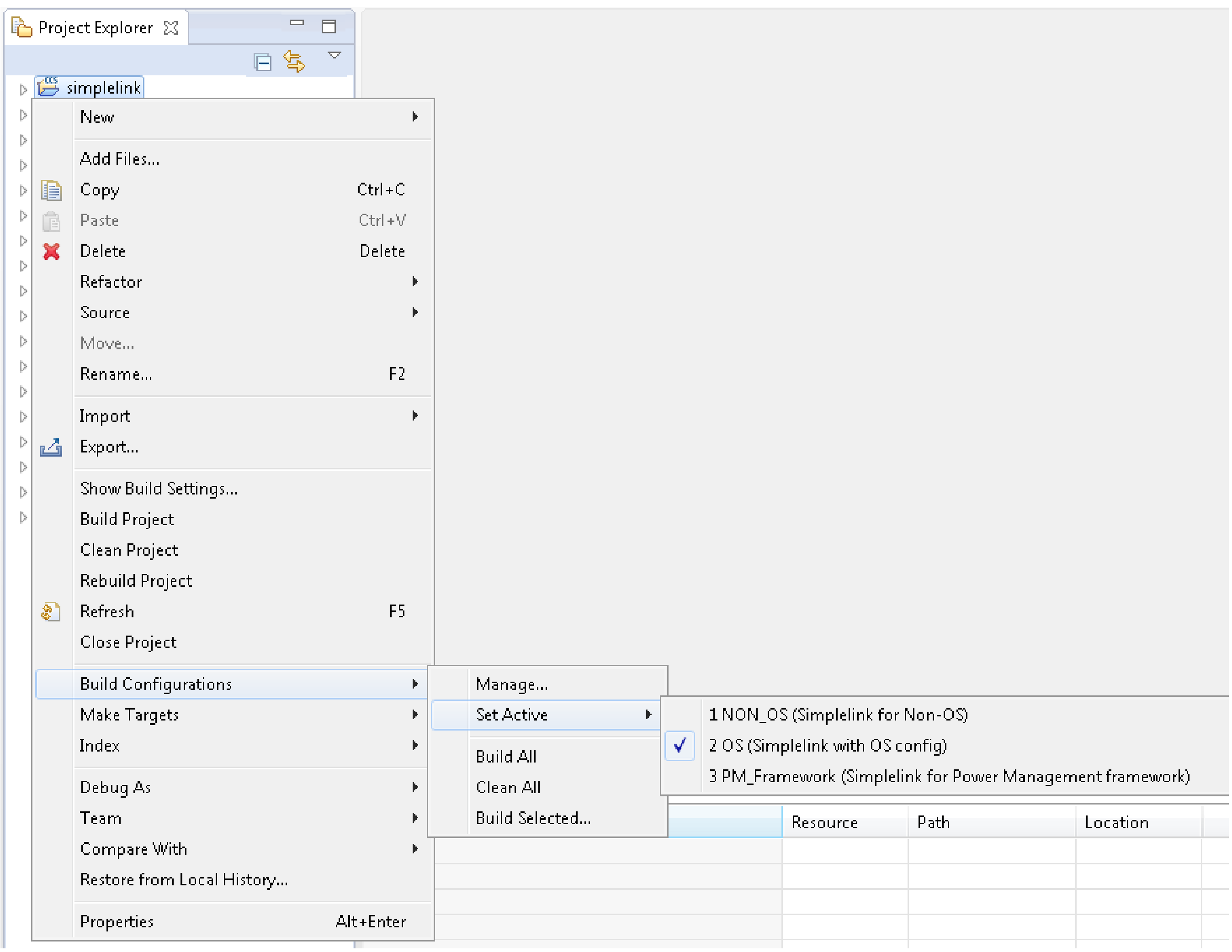
IAR: Choose configuration options from menu *Project->Edit configurations->OS/NON_OS/PM_Framework* as indicated in the snapshot below:

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



3.2 *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	This application showcases the device's capability as a station in a typical networking system.	Networking (STA mode)
Getting started with wLAN AP	This application showcases the device's capability as an AP in a typical networking system.	Networking (AP mode)
TCP Socket Application	This application showcases the device's communication over network using TCP protocols.	Networking (Basics)
WLAN Scan Policies Application	This application sets scan policy and enables the scan in the device.	Networking (Scan policies)

Application / Demo	Description	Peripheral/Feature exercised
UDP Socket Application	This application showcases the device's communication over network using TCP protocols.	Networking (Basics)
SSL Demo Application	This application showcases SSL implementation on CC3200 device.	Networking (SSL)
NWP Filter Application	This application showcases the Rx-Filter feature on CC3200 device.	Networking (MAC Filters)
File Operations Application	This application showcases the file operation on the serial flash of the device.	SFlash (File operations)
Transceiver mode Application	The RX Statistics feature inspects the medium in terms of congestion and distance, validates he RF hardware, and help using the RSSI information.	Networking (Raw sockets), GPIO, UART, Timer
Provisioning - Smart Config Application	The application demonstrates how to associate/connect CC31xx/CC32xx to any access point.	Networking (Provisioning)
Deep-Sleep Application	This application showcases the deepsleep as a power saving tool in a networking context of CC3200 device.	Networking, Low power modes (DeepSleep), UART, GPIO, Timer
Hibernate Application	This 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	The application connects to an SNTP server and requests for time information.	Networking (Internet)
Info Center-Get Weather Application	The application connects to ‘Open Weather Map’ and requests for weather data.	Networking (Internet)
Email Application	The email application on the CC3200 sends emails via SMTP.	Networking, GPIO, UART, Timer
XMPP Reference Application	The application demonstrates the connection scenario with an XMPP server.	Networking (Internet)
Provisioning-WPS Application	This application demonstrates how to use WPS Wi-Fi provisioning with CC3200.	Networking (Provisioning), GPIO
Mode-Configuration Application	This 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	The application demonstrates the connection policies in CC3200. The connection policies determine how the CC3200 is connected to AP.	Networking(STA Mode)
ENT WLAN Application	The example demonstrates the connection to an enterprise network using the flashed certificate. Certificate is flashed in SFLASH.	Networking(STA mode)
HTTP Server Application	The application demonstrates the Http server capability of CC3200.	Networking(STA Mode)
mDNS Application	The application demonstrates the usage of mDNS functionality in CC3200. The application showcases both “mDNS advertise” and “mDNS listen” functionality.	Networking(STA Mode) , UART
Provisioning-AP Application	This application demonstrates the AP provisioning feature. CC3200 AP Provisioning is method by which user can configure the AP information on the CC3200 device from a Browser.	Networking(AP Mode)
Out of Box Application	This application demonstrates the how the user can view different demo and SDK web links on a web-browser.	Networking(AP/STA mode), I2C, GPIO
Wi-Fi Audio Application	This example 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	This example allows the user to select the antenna with the highest signal for APs using a web-browser.	Networking(AP mode)
Camera Application	This example demonstrates the camera feature on CC3200 device. This application requires the camera boosterpack.	Networking(AP mode)
Peer to Peer Application	This example demonstrates the Wi-Fi direct feature on CC3200 device.	Networking(p2p mode)

Application / Demo	Description	Peripheral/Feature exercised
Idle Profile	This application exercises hibernation using Power Management Framework (middleware).	Networking(STA Mode)
Sensor Profile	This application exercises low power modes (lpds) using Power Management Framework (middleware).	Networking(STA Mode)
File Download Application	This application demonstrates file downloading from the web server and stores it to the device memory feature on the CC3200 device.	Networking (STA Mode)

3.3.2 MCU Only Reference Examples for the CC3200 LP

Application / Demo	Description	Peripheral/Feature exercised
LED Blink Application	This application showcases the blinking feature of available LEDs connected over GPIO on LP.	GPIO
Timer Demo Application	This application showcases the usage of 16 bit timers to generate interrupts which toggle the state of the GPIO.	Timer, GPIO, UART
Watchdog Demo Application	This application showcases the watchdog timer functionality to reset the system whenever the system fails.	WDT, GPIO, UART
UART Demo Application	This application showcases the use of UART.	UART
Interrupt Demo Application	A sample application showcasing interrupt preemption and tail-chaining capabilities.	NVIC, UART
I2C Demo	A sample application showing i2c read/write/read from features on CC3200 device.	I2C, UART
MCU Sleep-DeepSleep	This application exercises the Sleep and DeepSleep functionality of the MCU.	Low power modes (Sleep, DeepSleep), UART
uDMA Application	This application showcases different DMA modes of transfer.	uDMA, UART
Autorun non-OS Application	This application showcases the basic packet send and receive functionality of CC3200 in a non OS environment.	0.2
AES Demo Application	This application showcases the AES Encryption feature on CC3200 device.	Crypto, UART
DES Demo Application	This application showcases the DES Encryption feature on CC3200 device.	Crypto, UART
CRC Demo Application	This application showcases the CRC feature on CC3200 device.	Crypto, UART
FeeRTOS Demo Application	This application showcases the FreeRTOS features like Multiple task creation, Inter task communication using queues.	UART
SHA-MD5 Demo Application	This application showcases the SHA-MD5 Hash Algorithm on CC3200 device	Crypto, UART
ADC Demo Application	This application showcases the functionality of CC3200 ADC module by using the Driverlib APIs.	ADC, UART
PWM Demo Application	This application showcases the PWM mode of CC3200 GPTs. The general purpose timers (GPTs) support a 16-bit pulse-width modulation (PWM) mode with software-programmable output inversion of the PWM signal.	Timer, GPIO
SDHost Demo Application	This application showcases the functionality of SDHost module in CC3200. The Secure Digital Host (SD Host)controller on 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	This application uses the FatFS to provide the block level read/write access to SD card, using the SD Host controller on CC3200.	UART, SDHOST
SPI Demo Application	The demo application shows the required initialization sequence to enable the CC3200 SPI module in full duplex 4-wire master and slave mode(s).	UART, SPI
UART dma Application	The demo application showcases use of UART along with uDMA and interrupts.	UART, DMA
Timer Count Capture Application	This application showcases Timer's count capture feature to measure frequency of an external signal.	TIMER

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 using the tool and can be directly included in the project. Refer to the CC3200 PinMux tool wiki page for further details.

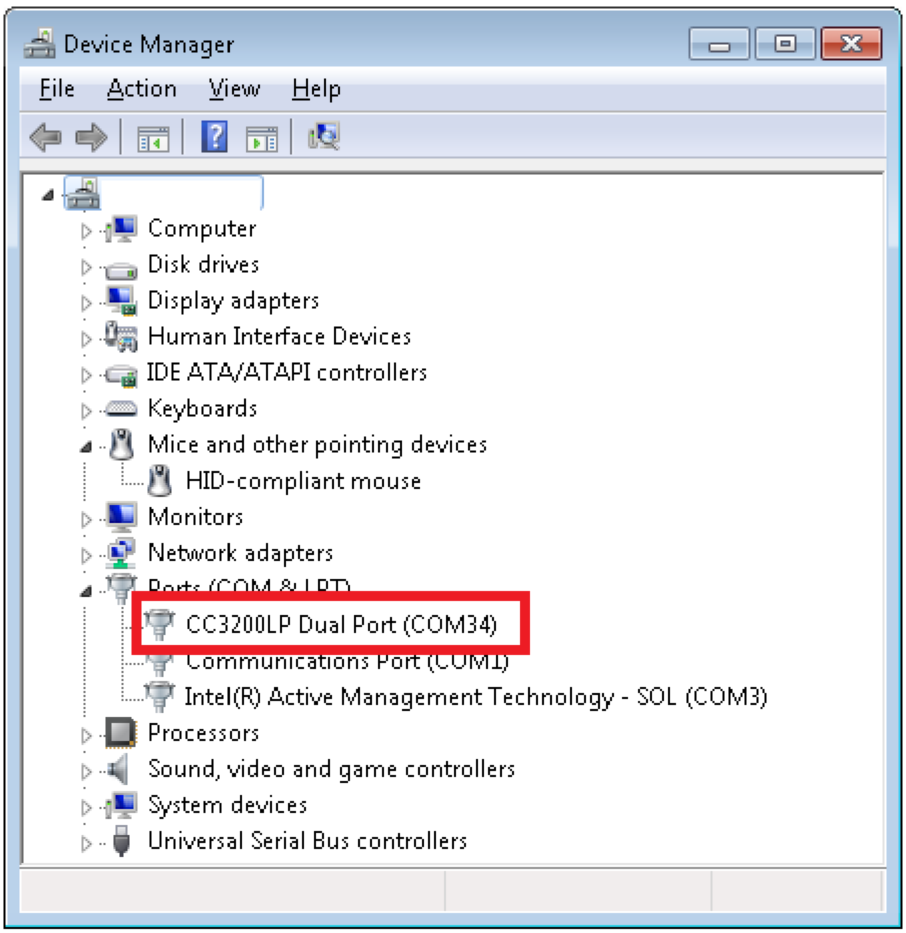
Refer to Section 5.3.2.2 for getting the new TI-Pinmux Tool.

4 Getting Started with CC3200 LP 3.x revB Board

Follow these steps to start with CC3200 LP 3.x revB board:

- Install the FTDI driver from CC3200-sdk\tools\FTDI available in SDK package.
- After installation the device will enumerate with one com port (CC3200LP Dual Port) as in this snapshot:

Figure 10. CC3200 Programmer Guide Device Manager



- To configure device into SWD/JTAG mode refer to 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 simple "Getting Started" applications are presented in the SDK. Start with a comprehensive description of these applications, build/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

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 application will be used without the UART terminal. 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 will try to get the IP address of “www.ti.com” and 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.

Security Macros

```
#define SSID_NAME "cc3200demo"
#define SECURITY_TYPE SL_SEC_TYPE_OPEN
#define SECURITY_KEY ""
#define WEP_KEY_ID 1
```

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.

Supporting folders/files

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 Simple Link Host Files

5.1.1.3 Code Flow Connection

```
void WlanStationMode( void *pvParameters )
```

```
{
.
.
.
.
//Start the SimpleLink

sl_Start(NULL,NULL,InitCallback);
.
.
.

// Connecting to WLAN AP
// After this call we will be connected and have IP address */

sl_WlanConnect(SSID_NAME, strlen(SSID_NAME), 0, &amp;secParams, 0);

// Read the IP address and use it for further operations

SL_STA_IPV4_ADDR_GET(&ulIP,&ulSubMask,&ulDefaultGateway,&ulDNSServer,
                       &ucDHCP);
.
.
.
.
iTestResult = PingTest(ulDefaultGateway);

}
```

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 Access point by calling the **sl_WlanConnect( )** API.
3. Use the **sl_NetCfgGet** API to get the IP address of the device.

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 will switch to STA mode if it is in AP mode.
3. The device will try to connect to open a pre-defined AP (cc3200demo). The red LED will glow upon a successful connection.
4. The device will ping to AP. If the ping is successful, the green LED will glow.
5. Observe the LED's state to confirm the proper execution.

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 will ping 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.

Supporting folders/files

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 Simple Link Host Files

5.1.2.3 Code Flow Connection

```
void WlanAPMode( void *pvParameters )
{
.
.
.
.
sl_Start(NULL,NULL,InitCallback);
while (g_uiIpObtained == 0)
{
//looping till ip is acquired
}

SL_STA_IPV4_ADDR_GET(&ulIP,&ulSubnetMask,&ulDefaultGateway,&ulDNSServer,
                       &ucDHCP);
while((g_uiIpLeased == 0) || (g_uiStaConnected == 0))
{
//wating for the STA to connect
}

ulIpAddr = g_ulStaIp;
iTestResult = PingTest(ulIpAddr);
.
.
.
}
```

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

1a. Start the SimpleLink by calling the **sl_Start()** API.

1b. Wait until the device gets an IP address.

After the device has come up in AP mode, follow these two steps to ensure the device can act as an AP.

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

2b. Ping the station (machine).

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

NOTE: 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) will ping the client and print 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 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 will be 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         0xc0a80167
#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.
- startup_ewarm.c – IAR workbench specific vector table implementation for interrupts.

Supporting folders/files

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 Simple Link Host Files

5.1.3.3 Code Flow Connection

```
void main()
{
.
.
.
.
// staring simplelink
sl_Start(NULL,NULL,InitCallback);
.
.
.
// Connecting to WLAN AP - Set with static parameters defined at the top
// After this call we will be connected and have IP address
sl_WlanConnect(SSID_NAME, strlen(SSID_NAME), 0, &amp;secParams, 0);

/* following calls depend on user's input at runtime */
.

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

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

TCP Client

```
int BsdTcpClient(unsigned short usPort)
{
.
.
.
.
//Open a socket with standard parameters
iSockID = sl_Socket(SL_AF_INET,SL_SOCK_STREAM, 0);
if( iSockID &lt; 0 )
{
// error
return -1;
}

//Connect to the server IP and port number
iStatus = sl_Connect(iSockID, ( SlSockAddr_t *)&amp;sAddr, iAddrSize);
if( iStatus &lt; 0 )
{
// error
return -1;
}

//Send packet using the sl_Send API call

iStatus = sl_Send(iSockID, g_cBsdBuf, sTestBufLen, 0 );
if( iStatus &lt;= 0 )
{
// error
return -1;
}

//Close the socket
sl_Close(iSockID);

return 0;
}
```

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_SOCK_STREAM, 0);
if( iSockID &lt; 0 )
{
// error
return -1;
}
```

```
.
.
.
iStatus = sl_Bind(iSockID, (SlSockAddr_t *)&amp;sLocalAddr, iAddrSize);
if( iStatus &lt; 0 )
{
// error
return -1;
}

iStatus = sl_Listen(iSockID, 0);
if( iStatus &lt; 0 )
{
return -1;
}

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

while( iNewSockID &lt; 0 )
{
iNewSockID = sl_Accept(iSockID, ( struct SlSockAddr_t *)&amp;sAddr, (SlSocklen_t*)&amp;iAddrSize);
if( iNewSockID == SL_EAGAIN )
{
UtilsDelay(10000);
}
else if( iNewSockID &lt; 0 )
{
// error
return -1;
}
}

iStatus = sl_Recv(iNewSockID, g_cBsdBuf, iTestBufLen, 0);
if( iStatus &lt;= 0 )
{
// error
return -1;
}

sl_Close(iNewSockID);
sl_Close(iSockID);

return 0;
}
```

Steps for receiving TCP Packets from TCP client are as follows:

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 (HyperTerminal/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 same AP that the device is connected to.
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 11:

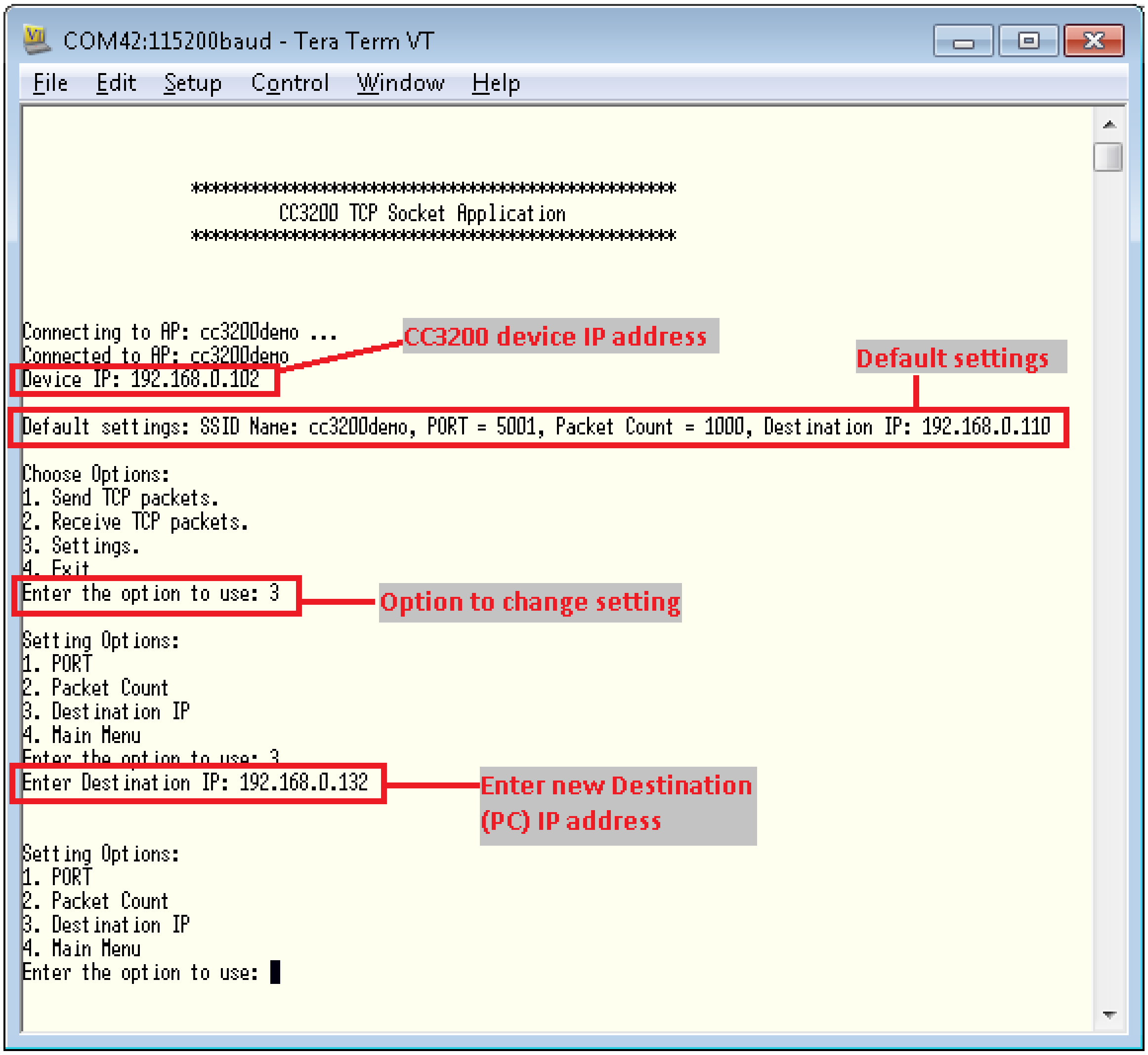


Figure 11. TCP Socket Terminal

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 will connect to an AP (access point), with 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 will be the expected return code. A different return code indicates that a socket error has occurred. The default setting is defined in the following macros, which can be changed either in the source code or at runtime.

```
#define SSID_NAME       "cc3200demo"
#define IP_ADDR         0xc0a80167
#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.
- startup_ewarm.c – IAR workbench specific vector table implementation for interrupts.

Supporting folders/files

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 Simple Link Host Files

5.1.4.3 Code Flow Connection

```
void main()
{
.
.
.
.
// staring simplelink
sl_Start(NULL,NULL,InitCallback);
.
.
.
// Connecting to WLAN AP - Set with static parameters defined at the top
// After this call we will be connected and have IP address
sl_WlanConnect(SSID_NAME, strlen(SSID_NAME), 0, &amp;secParams, 0);

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

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

UDP Client

```
int BsdUdpClient(unsigned short usPort)
{
.
.
.
.
//Open a socket with standard parameters
iSockID = sl_Socket(SL_AF_INET,SL_SOCK_DGRAM, 0);
if( iSockID &lt; 0 )
{
// error
return -1;
}

//Send packet using the sl_Send API call

iStatus = sl_SendTo(iSockID, g_cBsdBuf, sTestBufLen, 0, &amp;sAddr, iAddrSize);
if( iStatus &lt;= 0 )
{
// error
return -1;
}

//Close the socket
sl_Close(iSockID);

return 0;
}
```

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_SOCK_STREAM, 0);
if( iSockID &lt; 0 )
{
// error
return -1;
}

.
.
.
iStatus = sl_Bind(iSockID, (SlSockAddr_t *)&amp;sLocalAddr, iAddrSize);
if( iStatus &lt; 0 )
{
// error
return -1;
}

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

sl_Close(iSockID);
```

```
return 0;
}
```

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 (HyperTerminal/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 12.

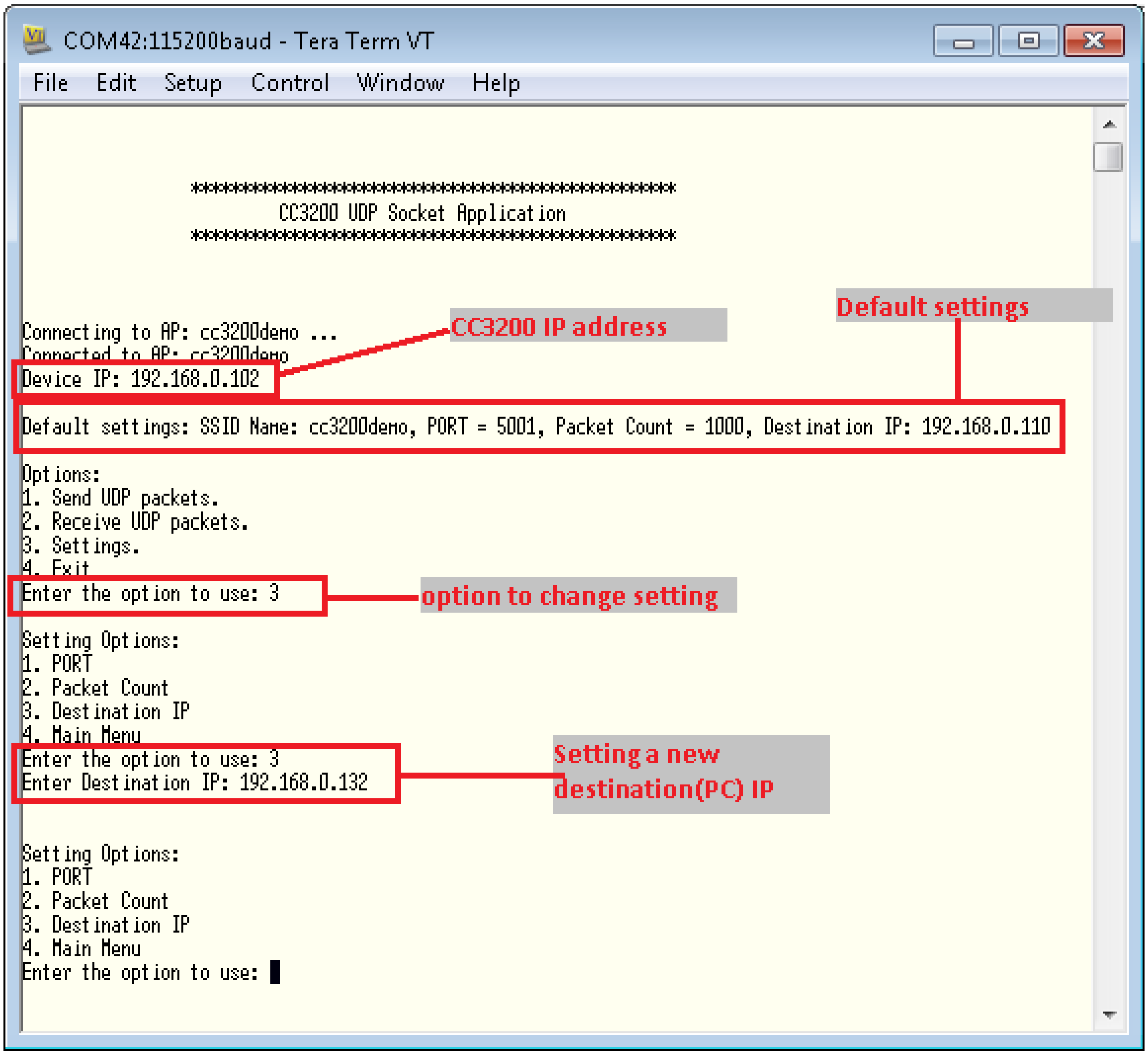


Figure 12. UDP Socket Terminal

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. The example demonstrates how to build a proprietary protocol on top of Wi-Fi PHY layer with the user given full flexibility to build their own packet.

Note that 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_SOCK_RAW,iChannel);

// Send the data
for(ulIndex = 0 ; ulIndex &lt; 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.

Supporting folders/files

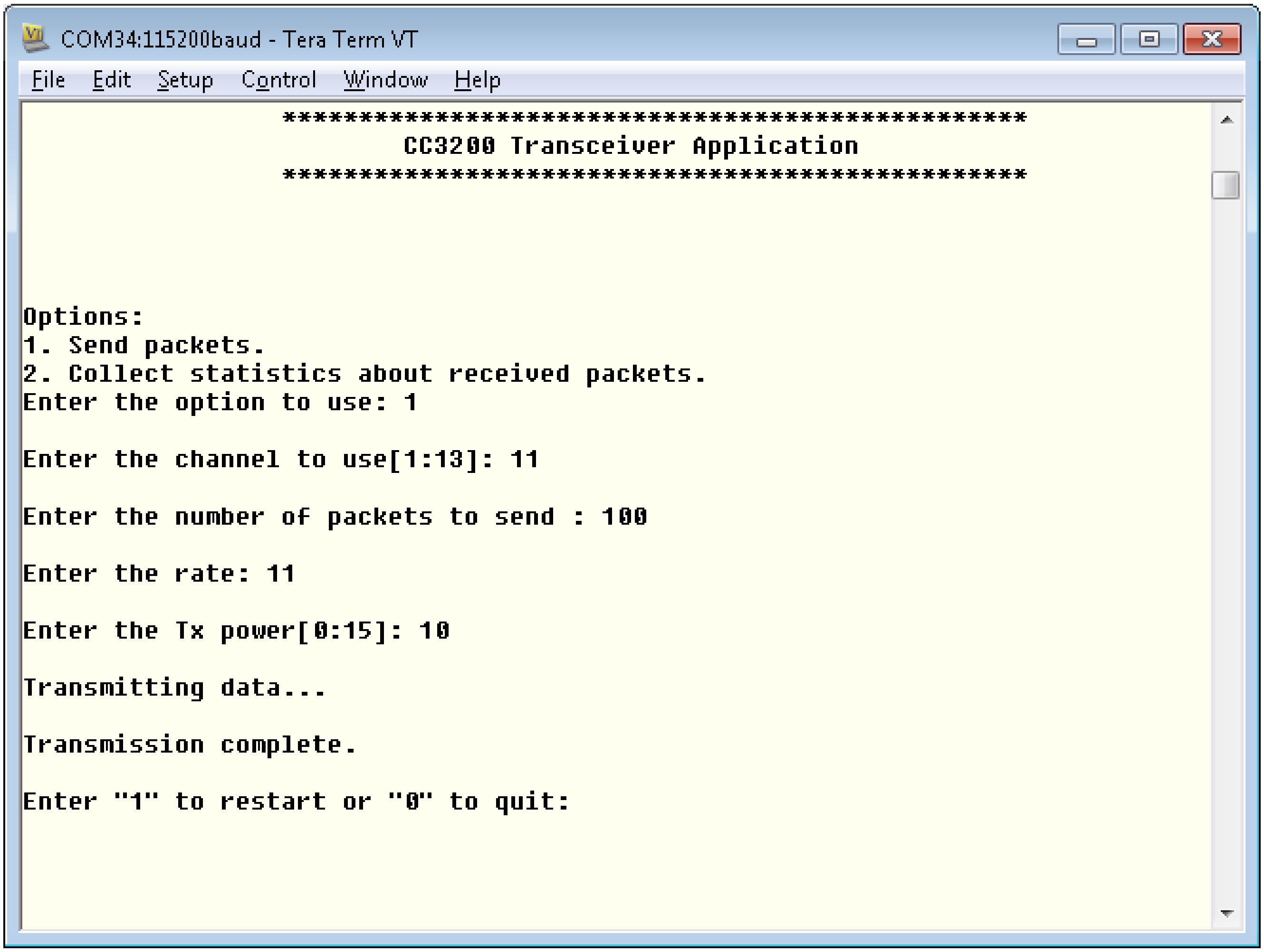
The *ewarm* folder contains IAR workspace. The *ccs* folder contains CCS Project, the *driverlib* folder contains all the driver files, the *oslib* folder contains the project to build free_rtos library, the *third_party* folder contains FreeRTOS files, and the *simplelink* folder contains Simple Link Host Files

5.1.5.3 Usage

- Setup a serial communication application (HyperTerminal/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:

Figure 13. CC3200 Transceiver Application on the Hyperterminal



Common interface files: 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 AP that is 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 will be 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.

	APIs	Description
	sl_SendTo	Sends a Welcome message to the destination IP address, given as input. The destination IP address is taken from Input. 200 packets will be 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 will be 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 will be 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 that contains 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 using any reference application or for 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.

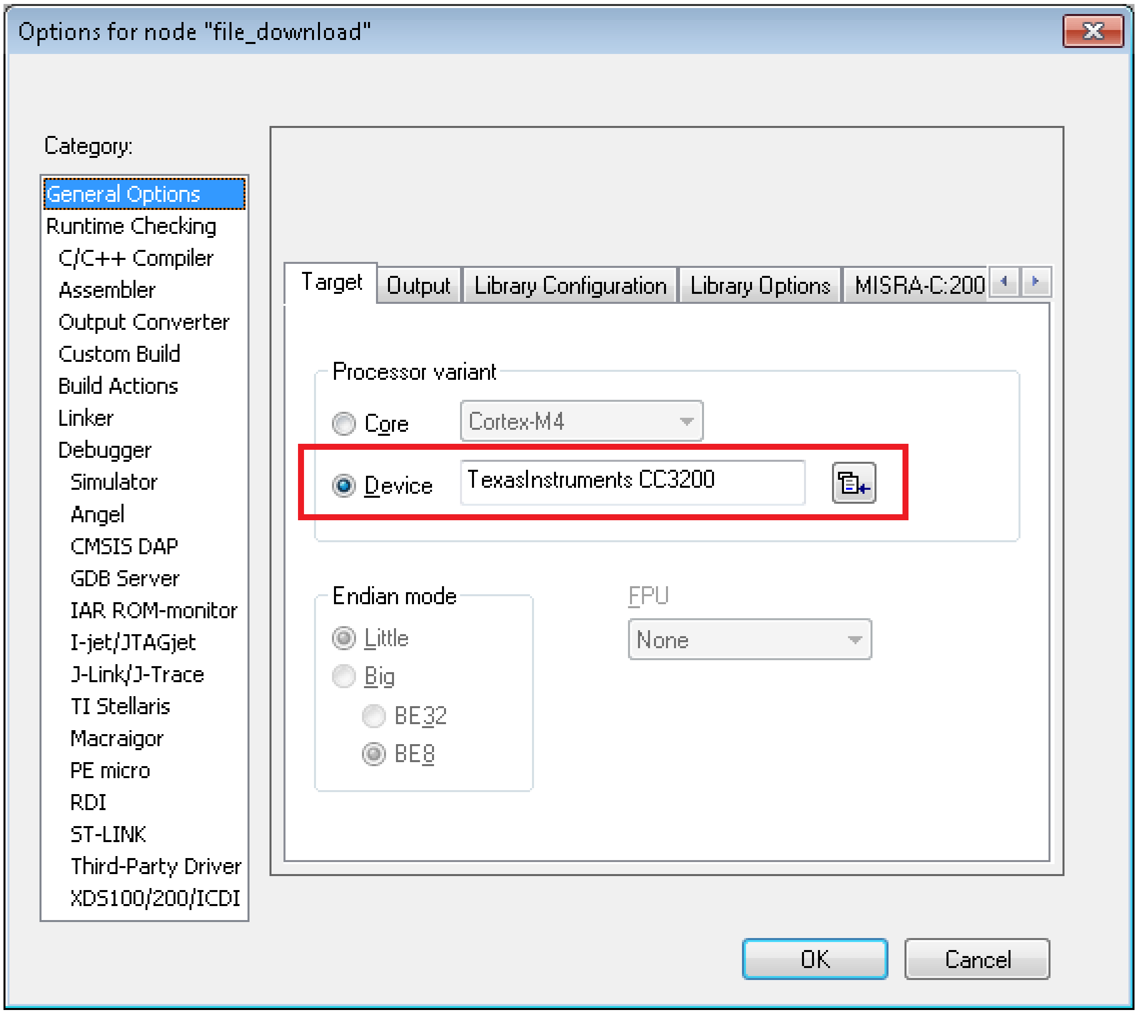
5.3.1 Development Environment – IAR

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

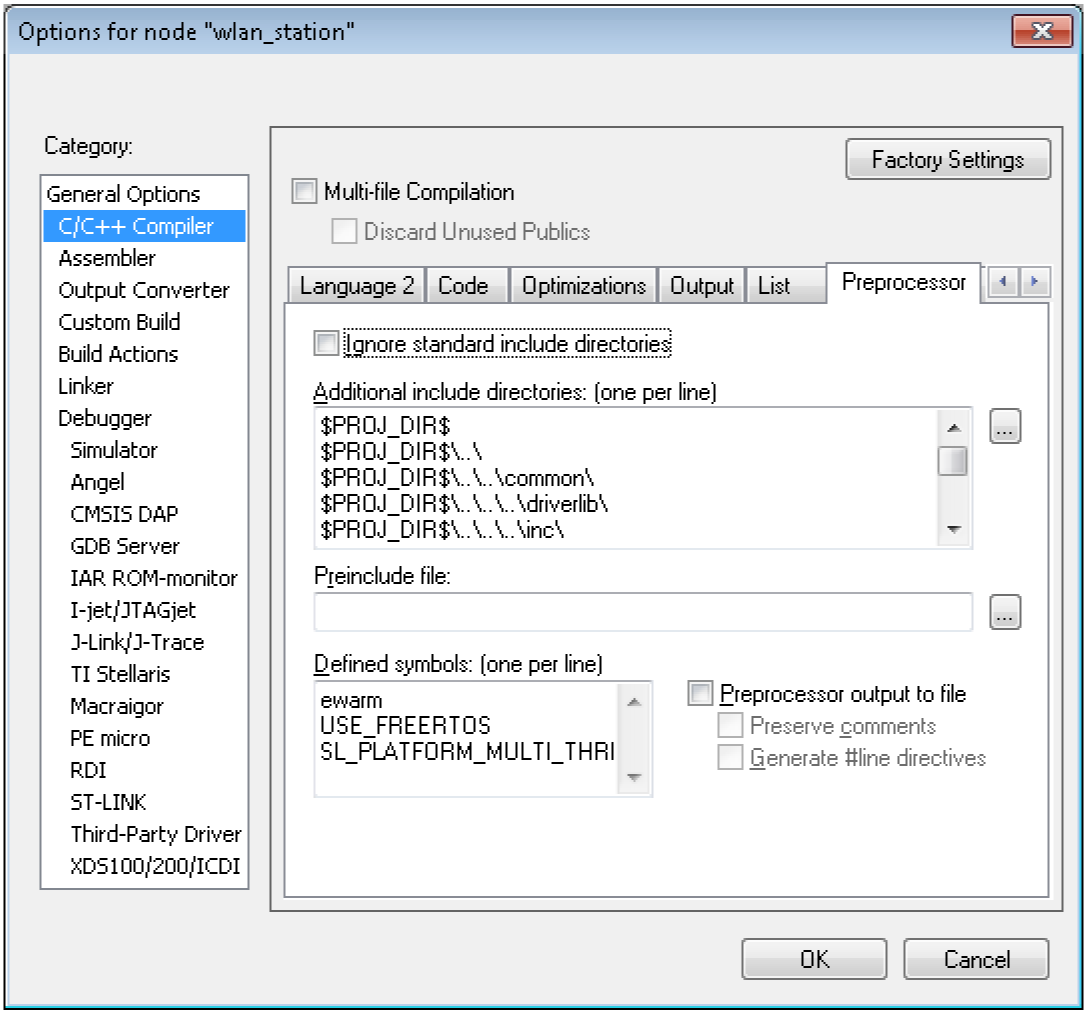
- File->New->Workspace
- Project->Create New Project (Tool Chain = ARM, Project Templates = 'C')
- Open Project Option and follow the settings as given in these snapshots:

Figure 14. CC3200 Programmer Guide IAR Project Options



5.3.1.2 Compiling a Project

Figure 15. CC3200 IAR Compiling Project



Additional include directories

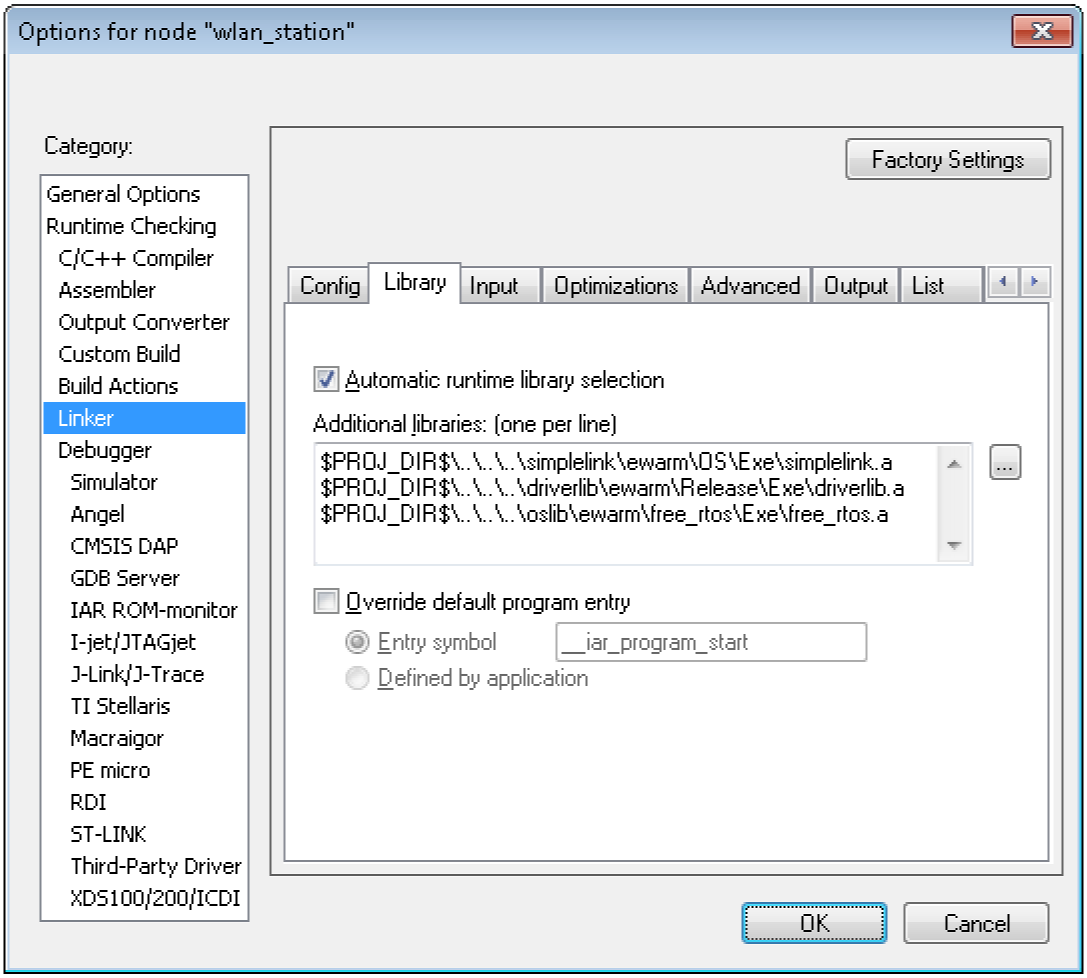
- To use Driverlib APIs – Include ‘driverlib’ and ‘inc’ folder path.
- To use Simplelink APIs – Include ‘simplelink’, ‘simplelink\Soure’ and ‘simplelink\Include’ folder paths.
- To use Free-RTOS APIs – Include ‘oslib’, ‘third_party\FreeRTOS\source\’, ‘third_party\FreeRTOS\source\include\’ and ‘third_party\FreeRTOS\source\portable\IAR\ARM_CM4\’ paths.

Defined symbols (MACRO definition)

- USE_FREERTOS – If application uses Free-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

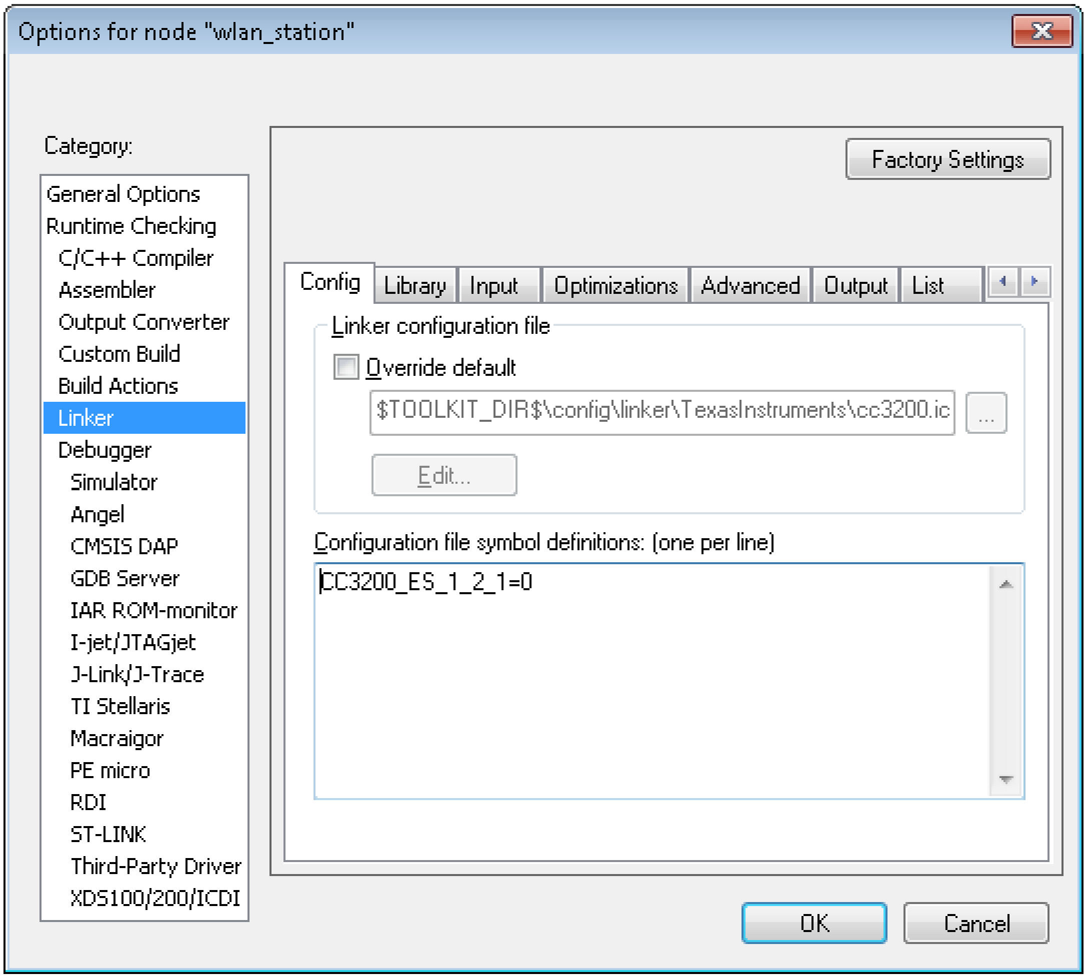
Figure 16. CC3200 IAR Linker Project



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\NON_OS* and *simplelink\ewarm\PM_Framework* folder for the OS, Non-OS and Power management configurations, respectively.
  - free_rtos.a : Available under *oslib\ewarm\free_rtos\Exe* folder.

Figure 17. CC3200 IAR Linker Config



Linker configuration file

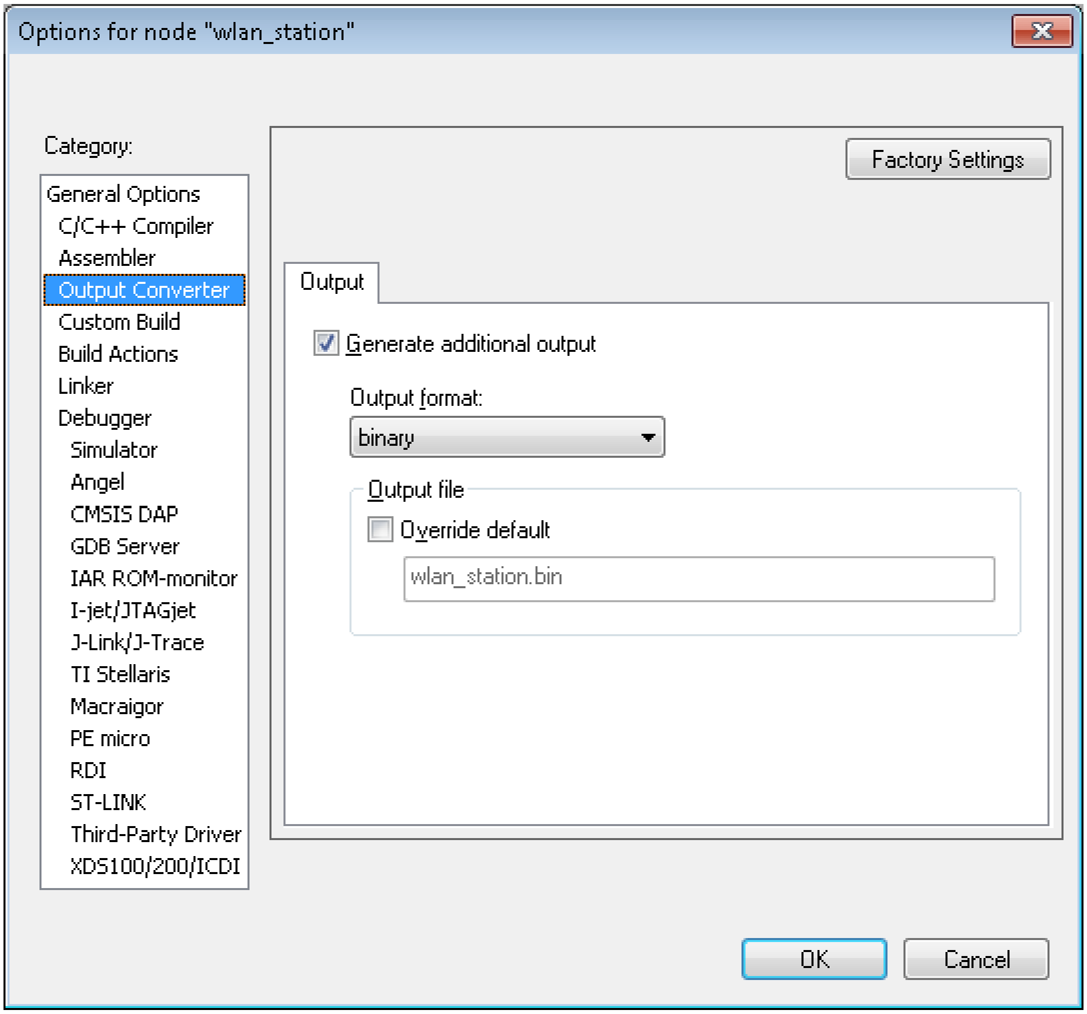
- Link to IAR linker file, by default IAR links to CC3200.icf available in IAR installation. The developer can change this as per application requirement.

Symbol definitions

- Define ‘CC3200_ES_1_2_1=0’ for ES 1.32 device and ‘CC3200_ES_1_2_1=1’ for ES 1.21 device.

5.3.1.4 Generating the Binary (.bin)

Figure 18. CC3200 IAR Generating Binary



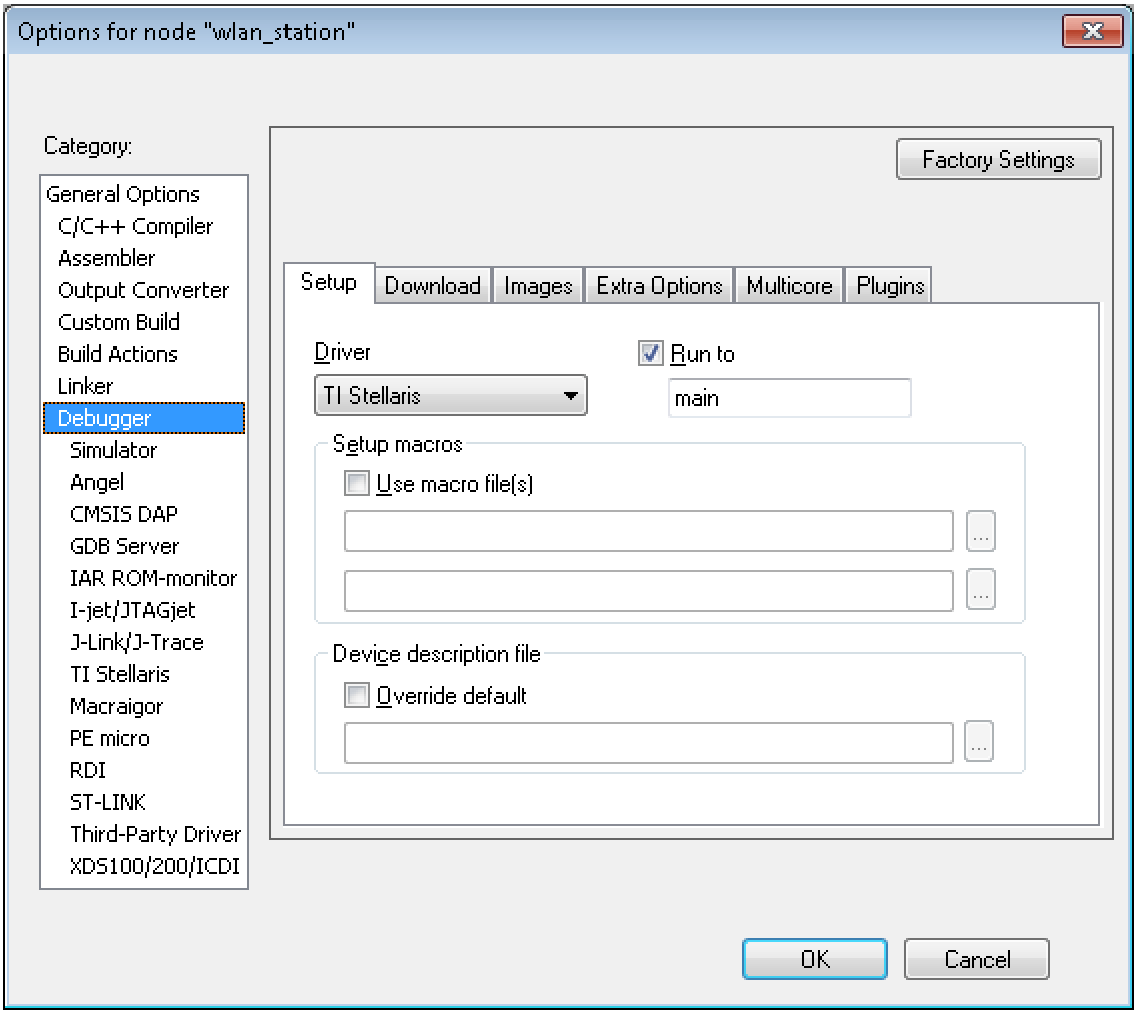
To generate additional output select the output format. A current SDK user needs to select the 'binary' option and can override the .bin path. In CC3200 SDK IAR generates bin file in '*<example>\ewarm\Release\Exe*' folder.

5.3.1.5 Executing a Project

To use the JTAG over FTDI, the TI Stellaris driver needs to 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:

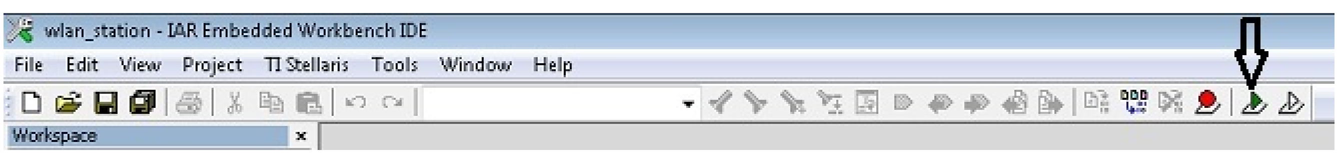
- JTAG Mode - connect SOP-2 jumper only
- SWD mode - connect SOP-0 jumper only

Figure 19. 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 20. CC3200 IAR Download and Run



If the application uses UART to print the output on the terminal, then the user needs to 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 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 6.0 version. These are the steps to create a new project in CCS environment.

5.3.2.1 *TI-RTOS 2.0 in CCSv6*

Follow these steps 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 it in CCS 6.0

Figure 21

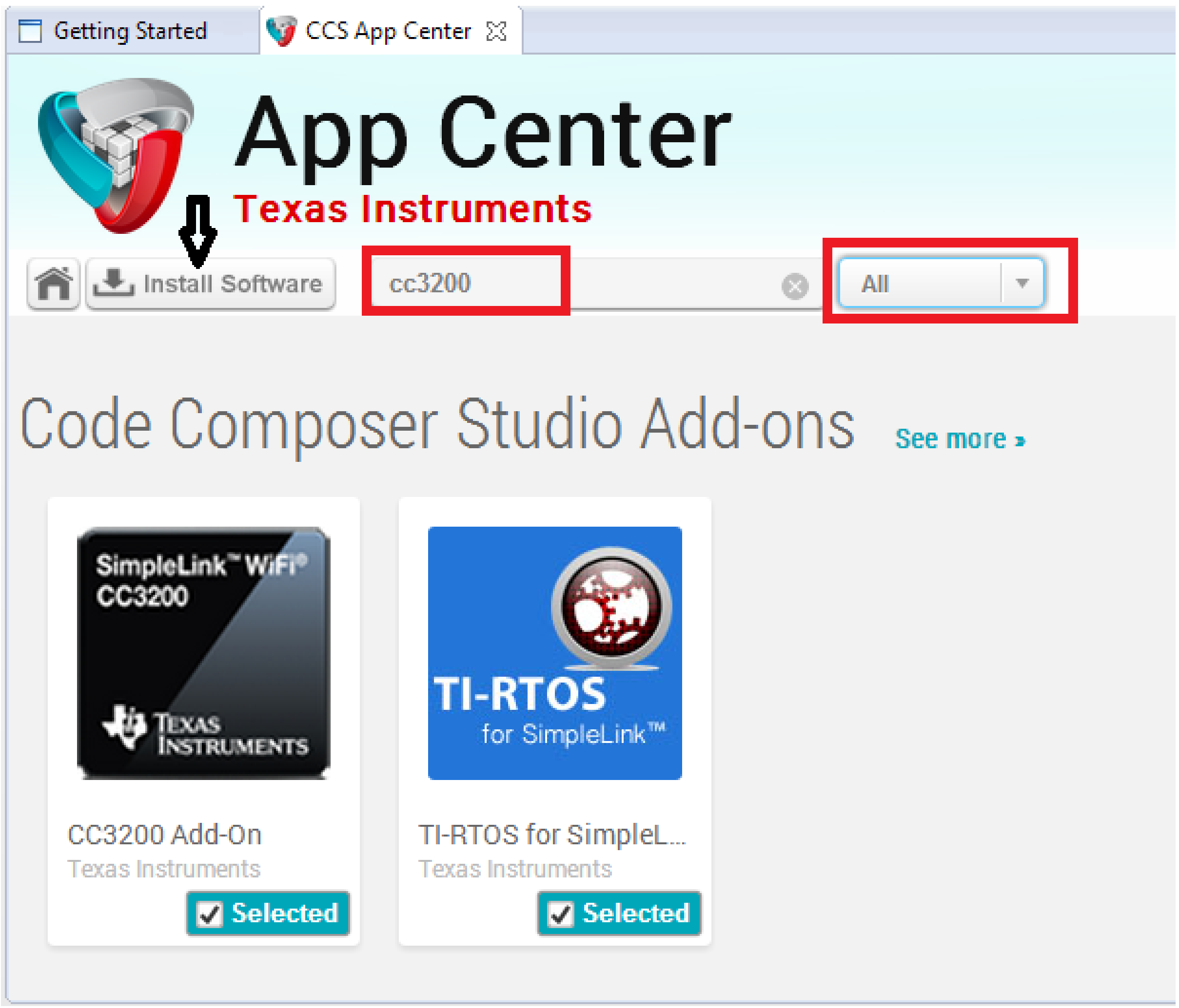


Figure 21. CCS App Center

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 22

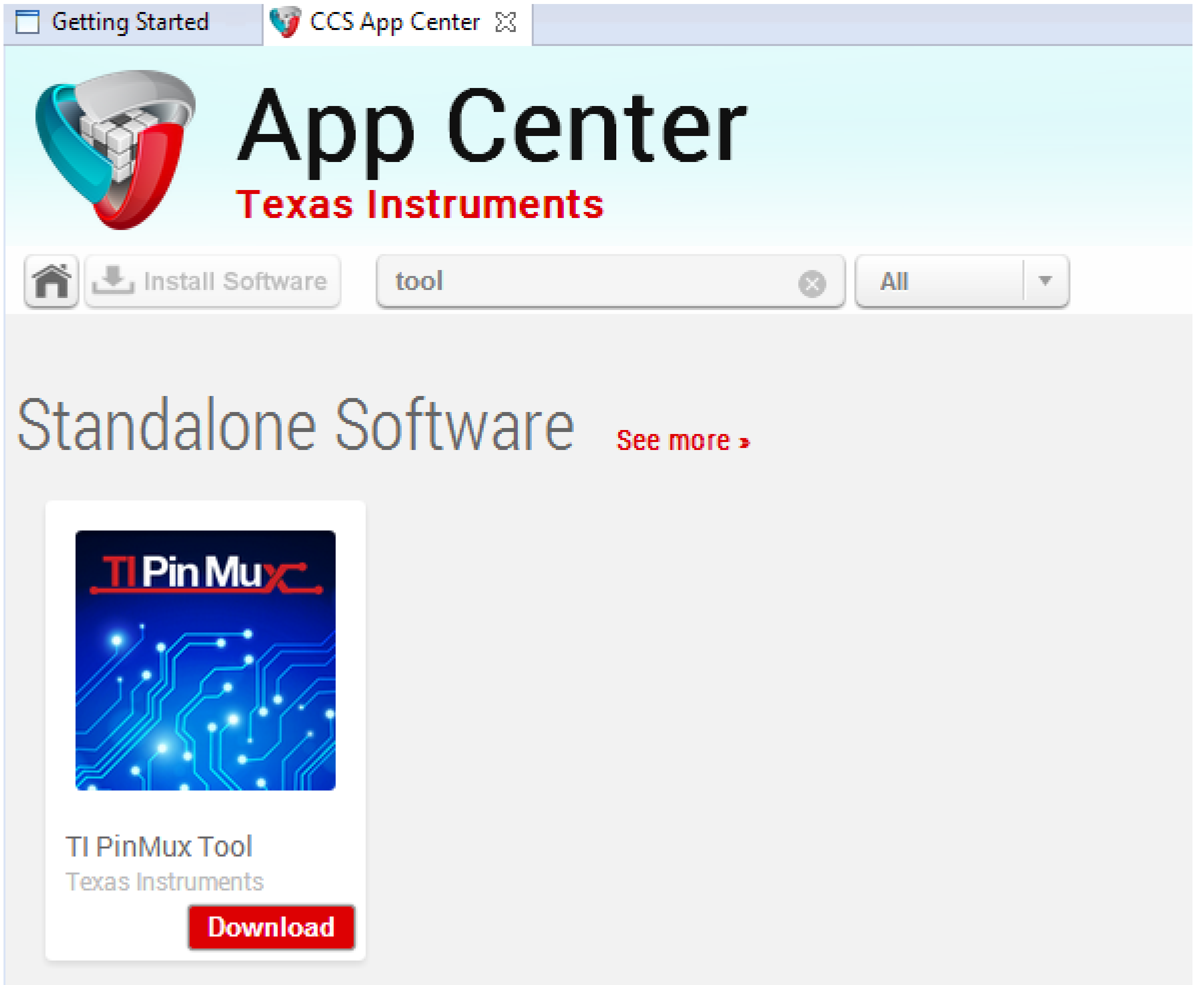


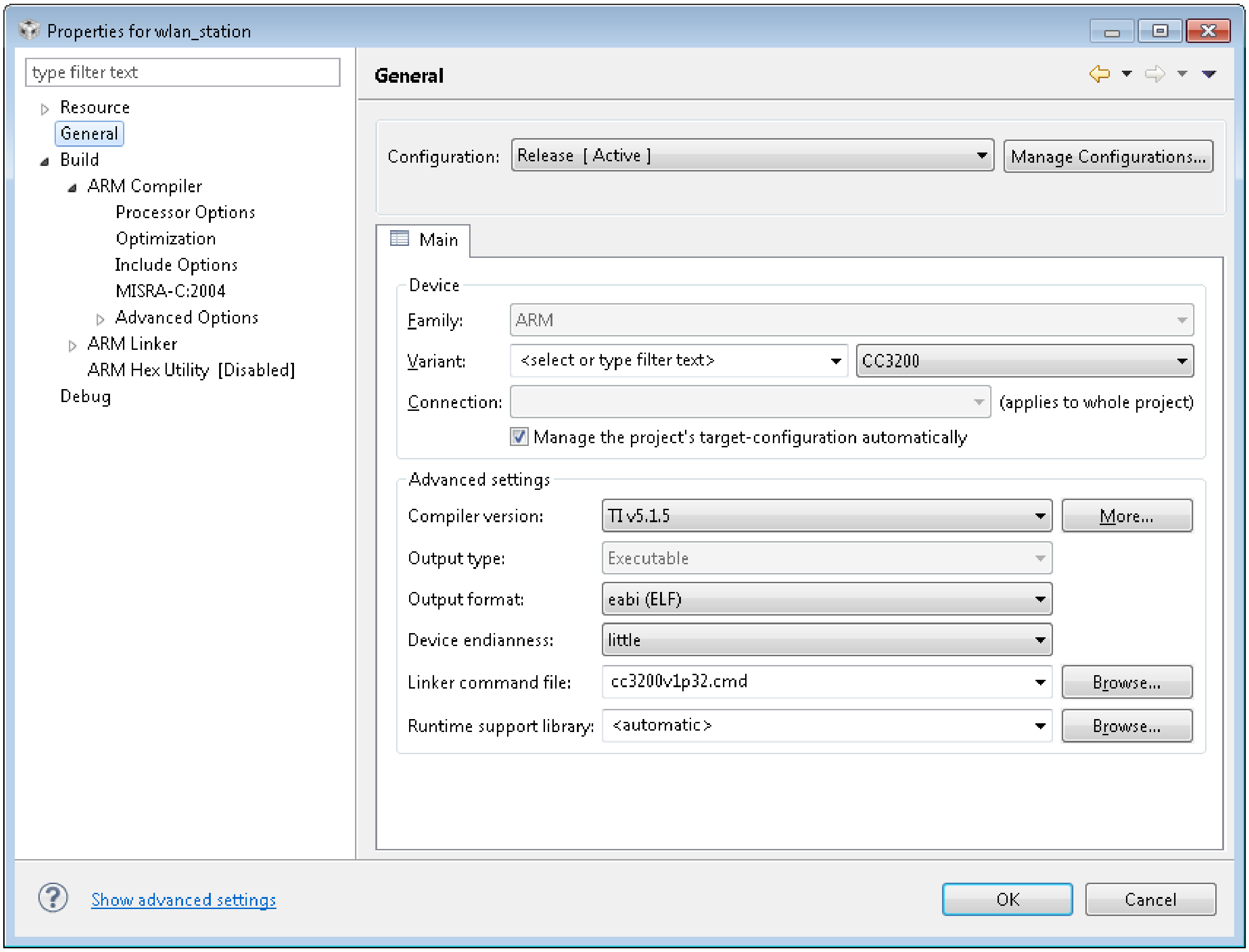
Figure 22. TI-PinMux Tool

5.3.2.3 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 need to 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 23. CC3200 CCS Creating Project



Linker command file:

- cc3200v1p32.cmd: for ES 1.32 onward device.
- cc3200v1p21.cmd for ES 1.21 device.

5.3.2.4 Compiling a Project

Figure 24. CC3200 CCS Compiling Project

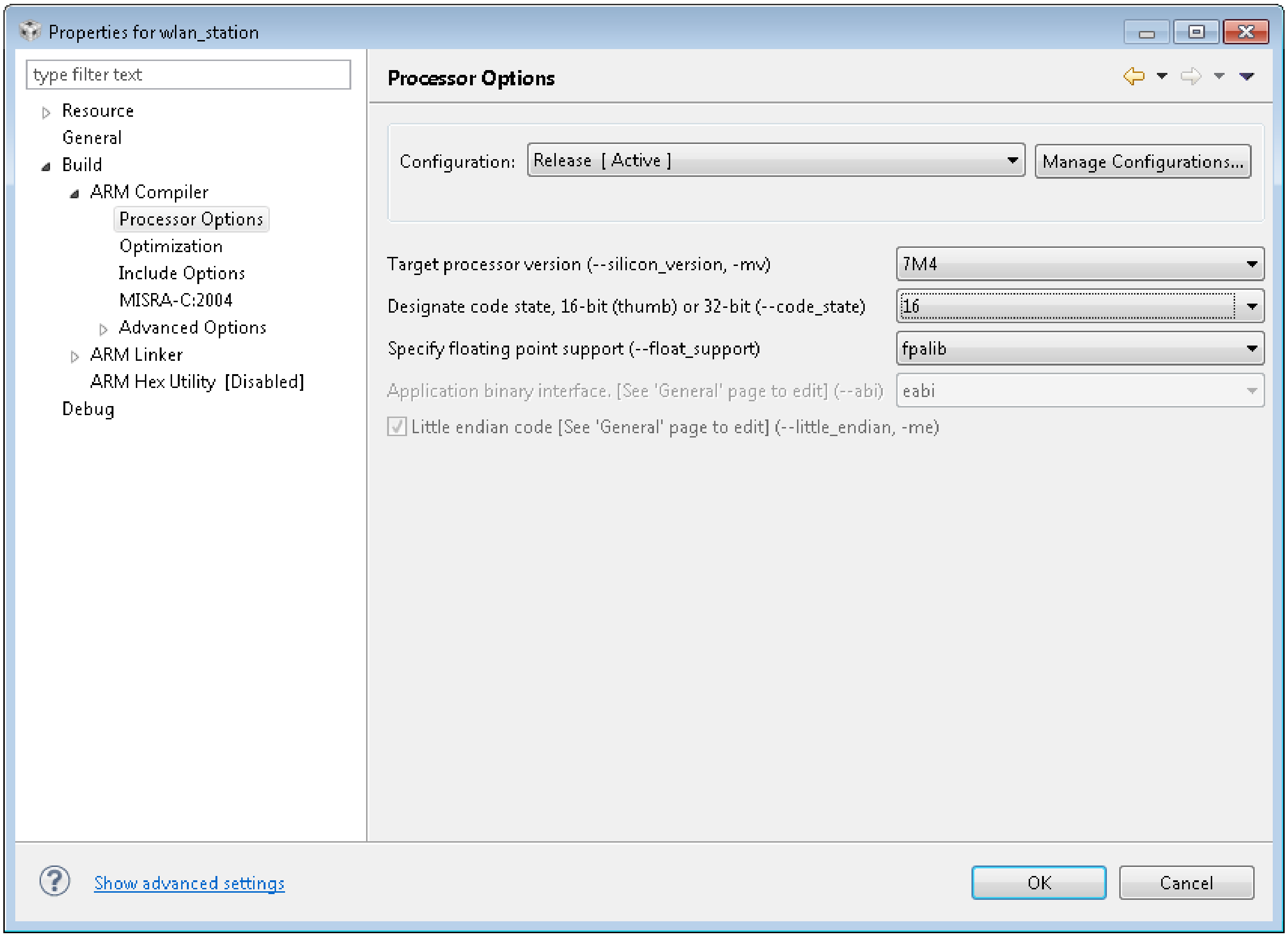
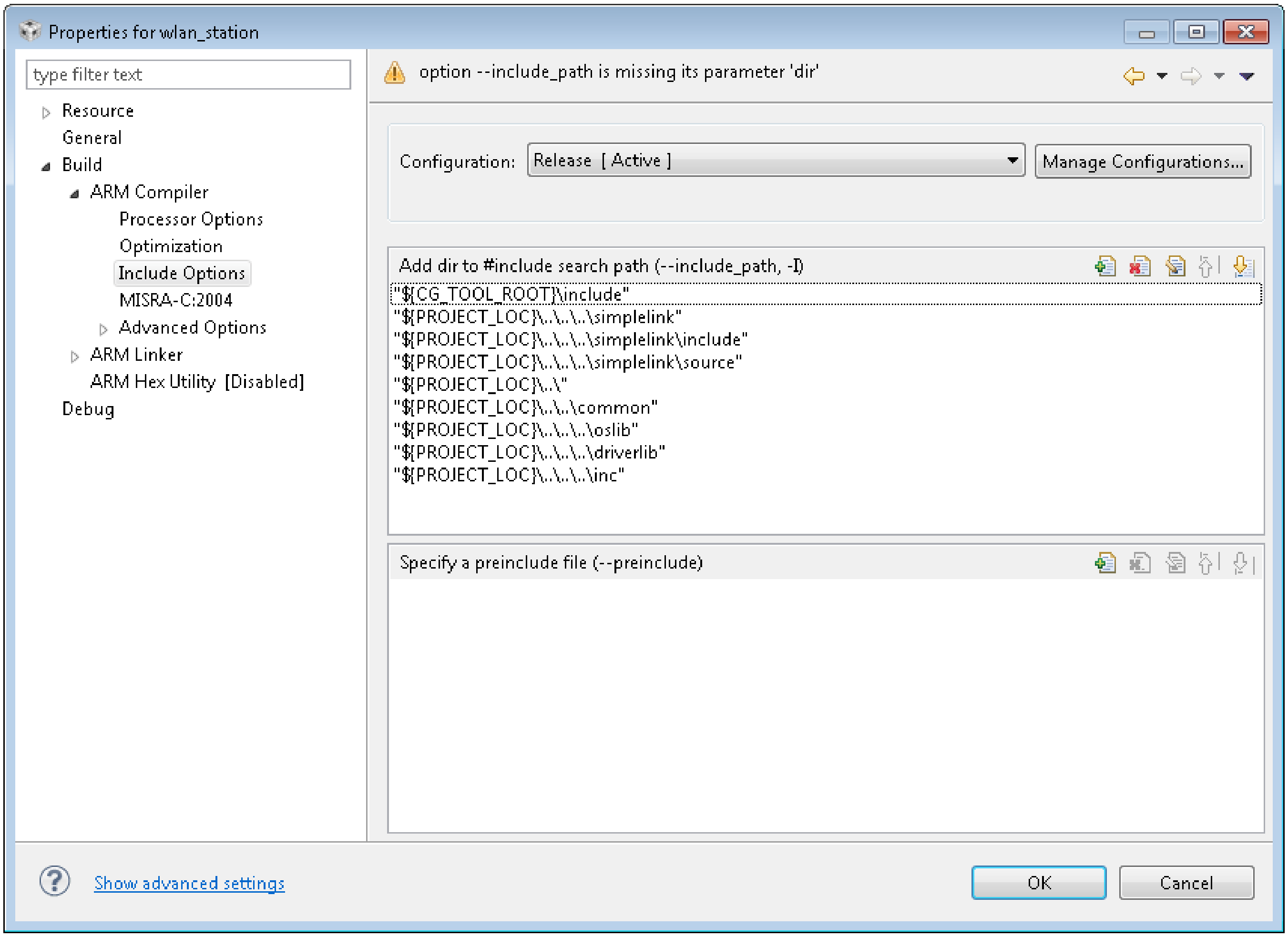


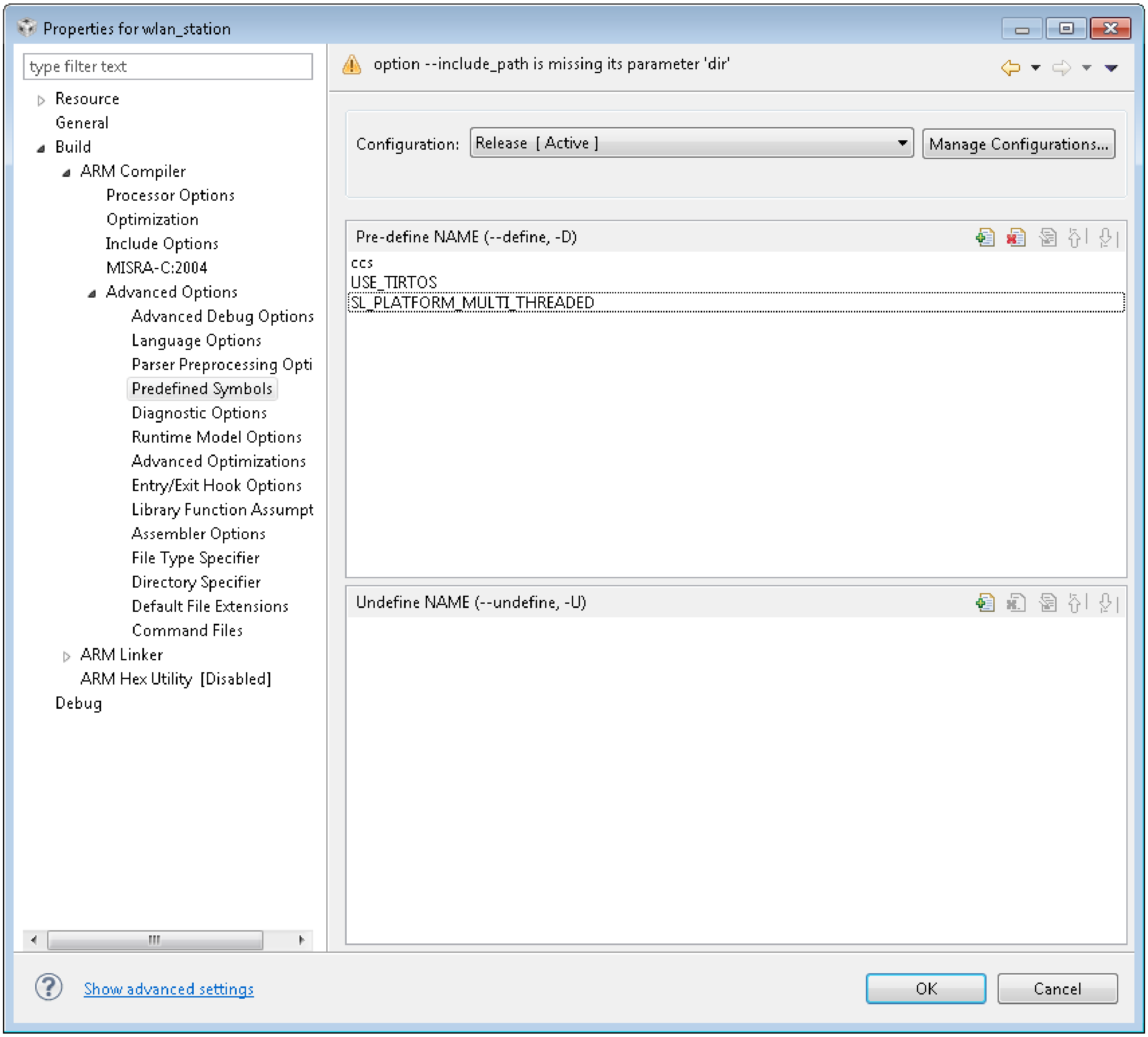
Figure 25. CC3200 CCS Compiling Project 1



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 26. CC3200 CCS Compiling Project 2

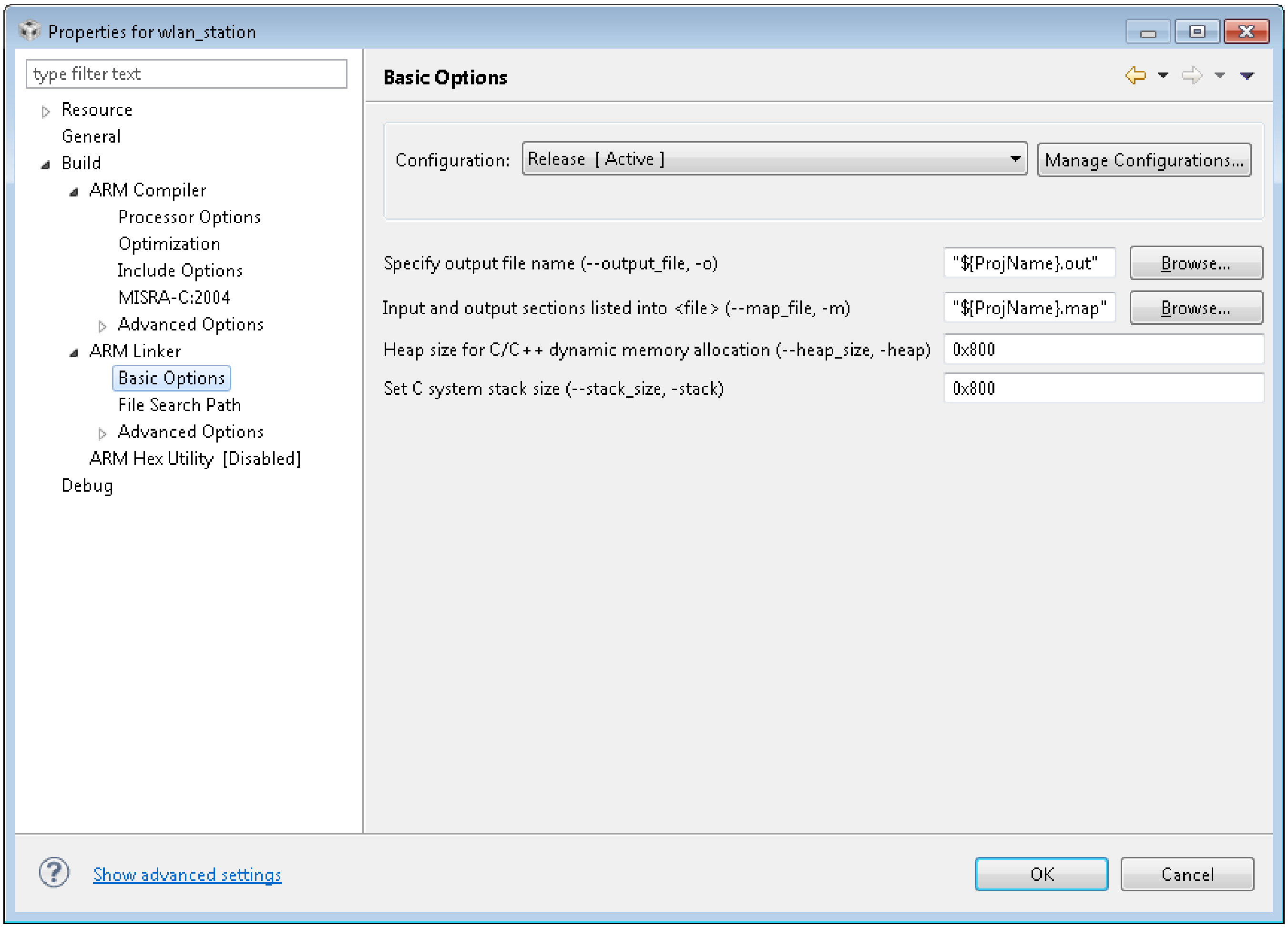


Pre-define NAME

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

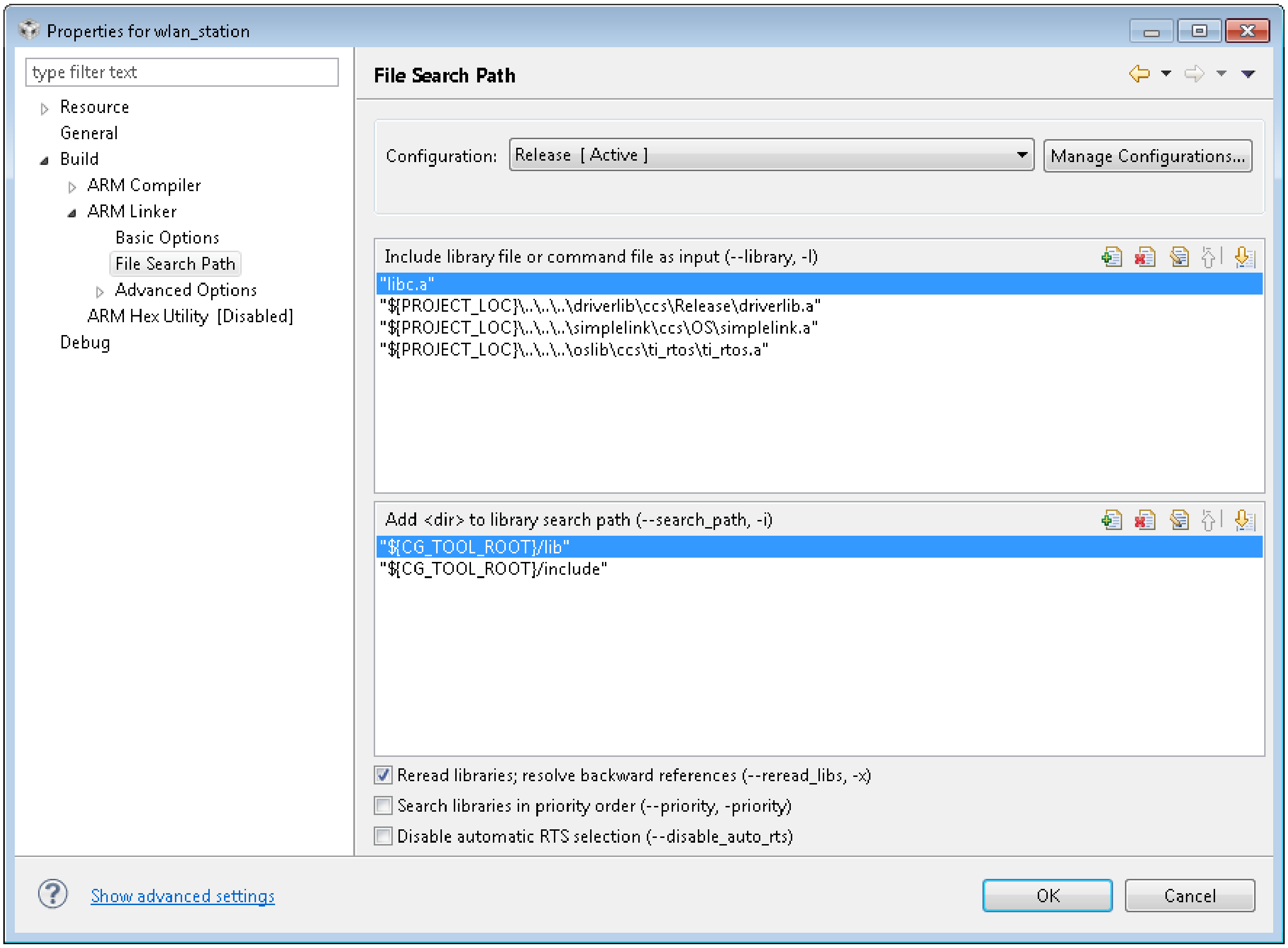
5.3.2.5 Linking a Project

Figure 27. CC3200 CCS Linking Project 1



- Set heap and stack size as per the application's requirement.

Figure 28. 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', 'NOON_OS*, *PM_Framework* folder for OS, non-OS or power management 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..

5.3.2.6 Dependency to Other Project

TI-RTOS OS Dependencydocato-extra-info-title Dependencies

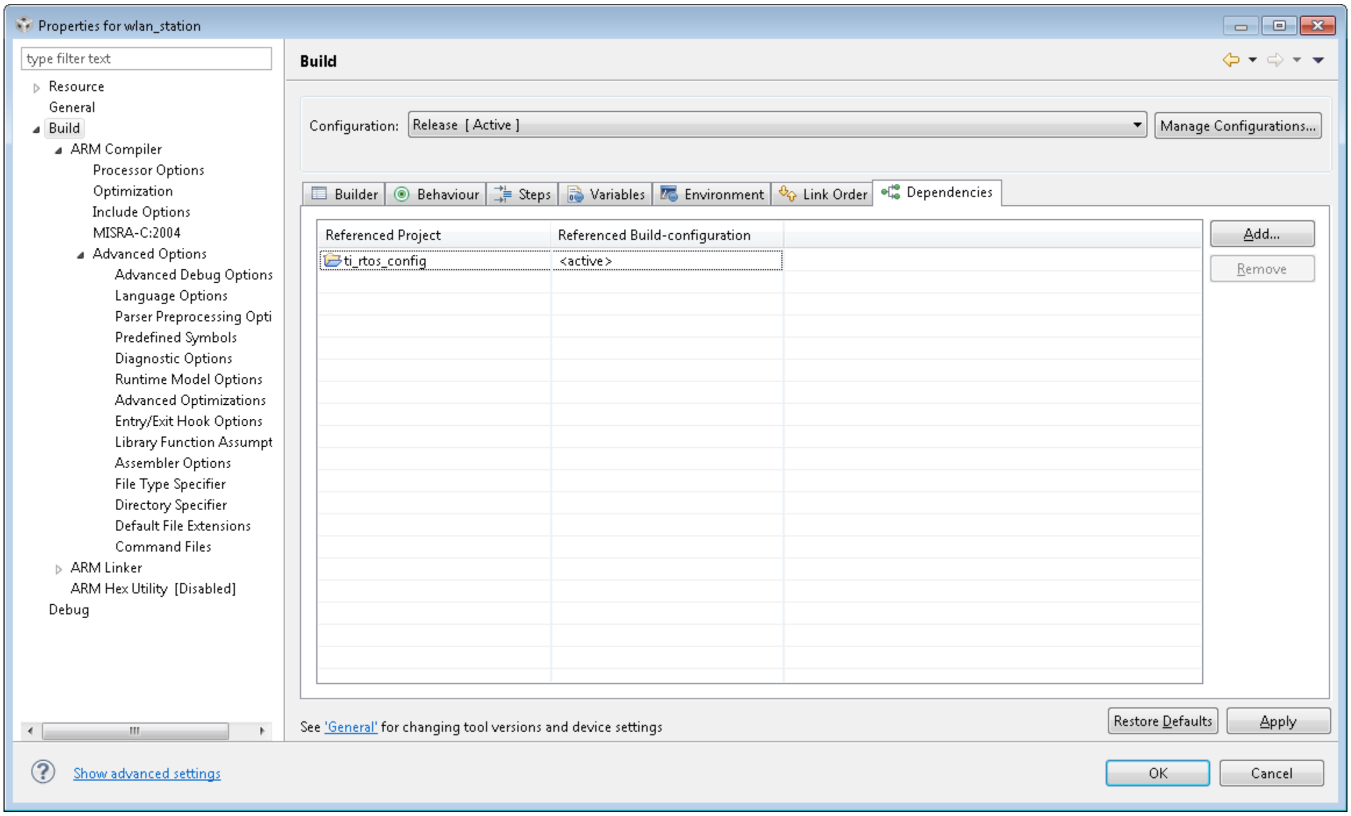


Figure 29. Dependencies

Dependencies

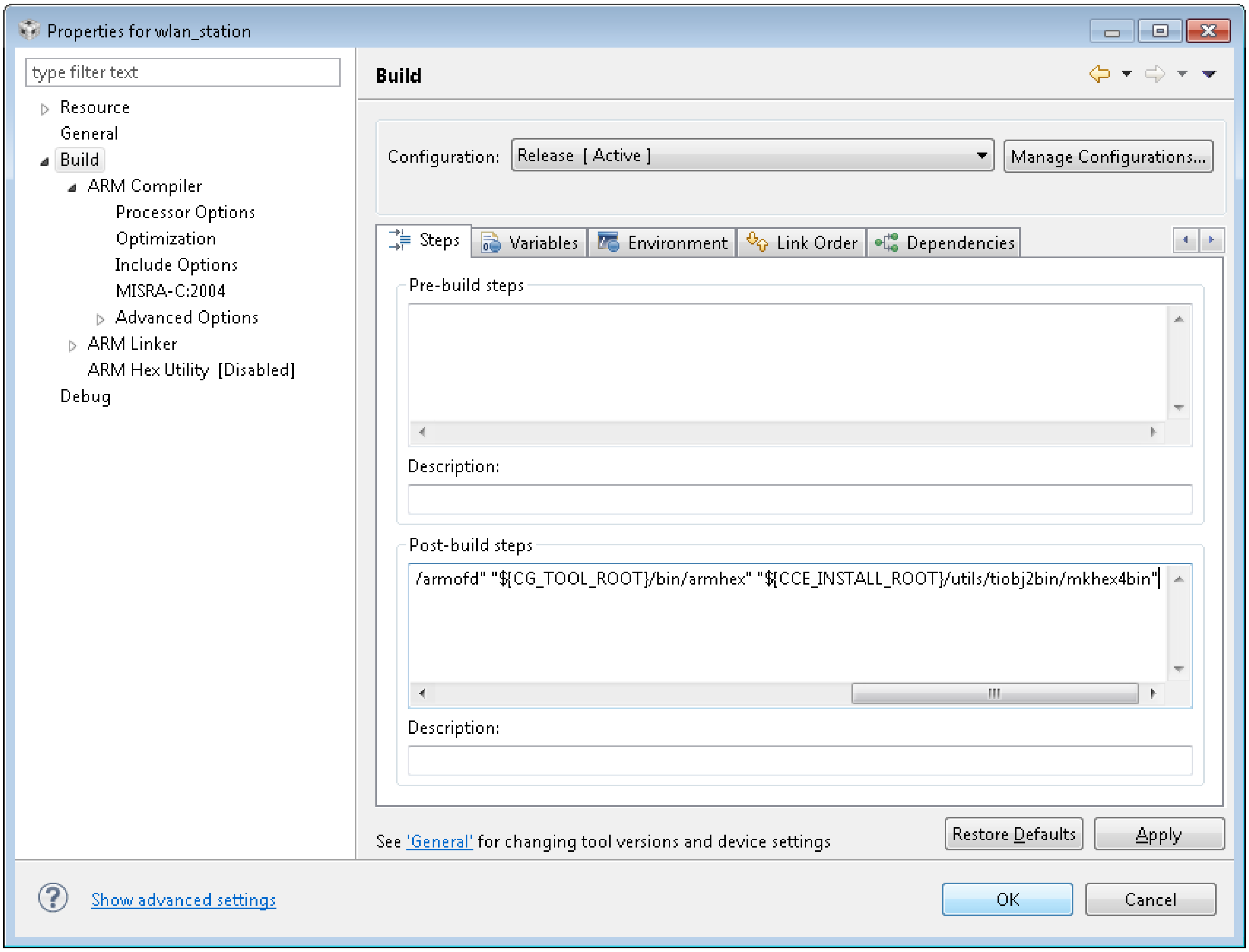
- If the application uses TI-RTOS OS, add 'ti_rtos_config' project as dependency for the application.
  - '**ti_rtos_config**' project should be imported in CCS workspace for TI-RTOS based application.
  - Current SDK supports TI-RTOS 2.0.x

5.3.2.7 Generating a Binary (.bin)

- Add the following script to generate a .bin file

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

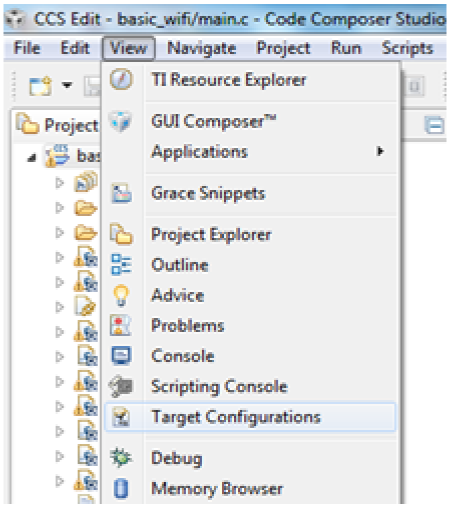
Figure 30. CC3200 CCS Generating Binary



5.3.2.8 *Executing a Project*

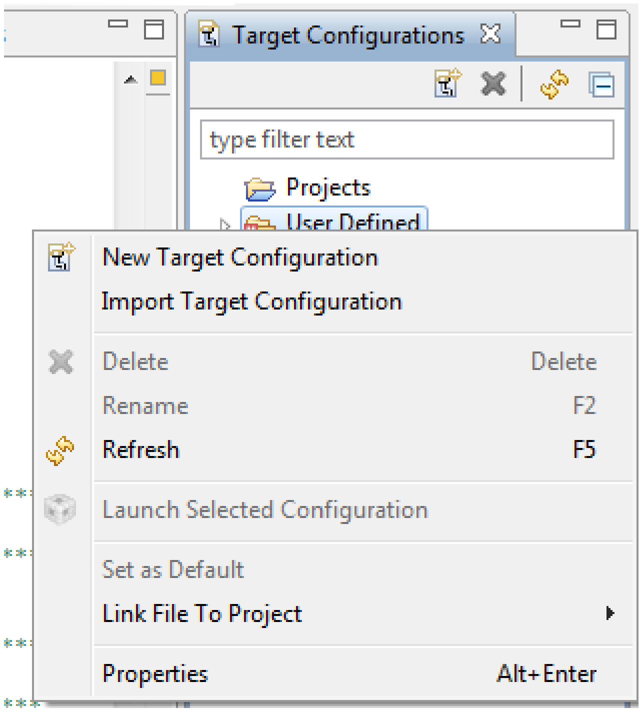
Click on the target configuration under View.

Figure 31. CC3200 CCS Executing 1



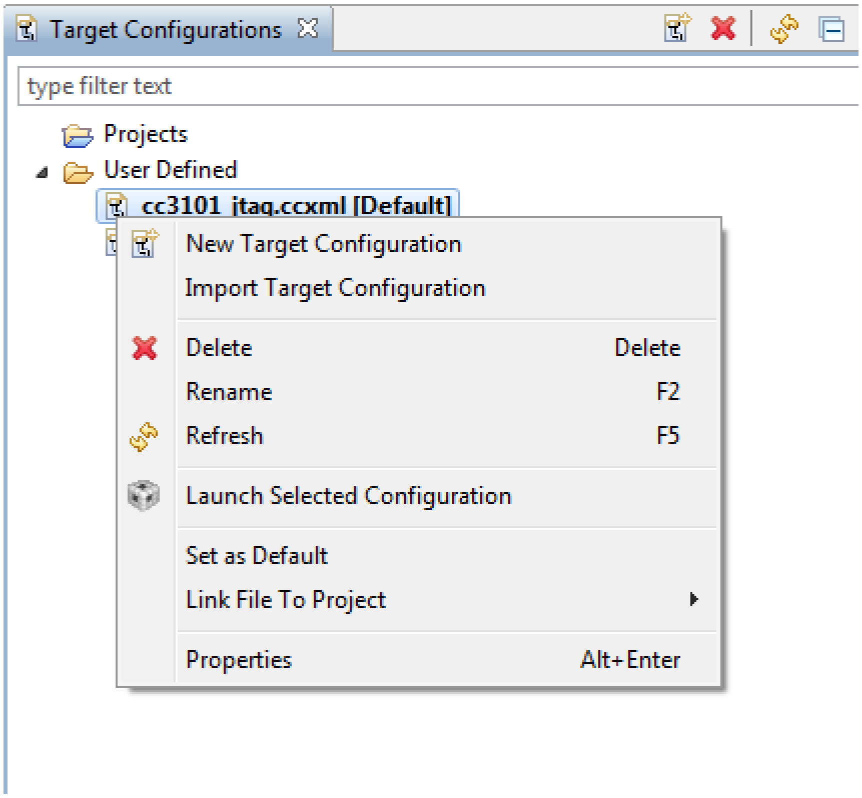
Right Click on the "User Defined," click on "Import Target Configuration" and select **CC3200.ccxml** from ***/tools/ccs/***.

Figure 32. CC3200 CCS Executing 2



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

Figure 33. CC3200 CCS Launch Config



To switch between JTAG/SWD mode from CCS, follow the steps specified in Figure 34. On the CC3200 LaunchPad, configure the board for either:

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

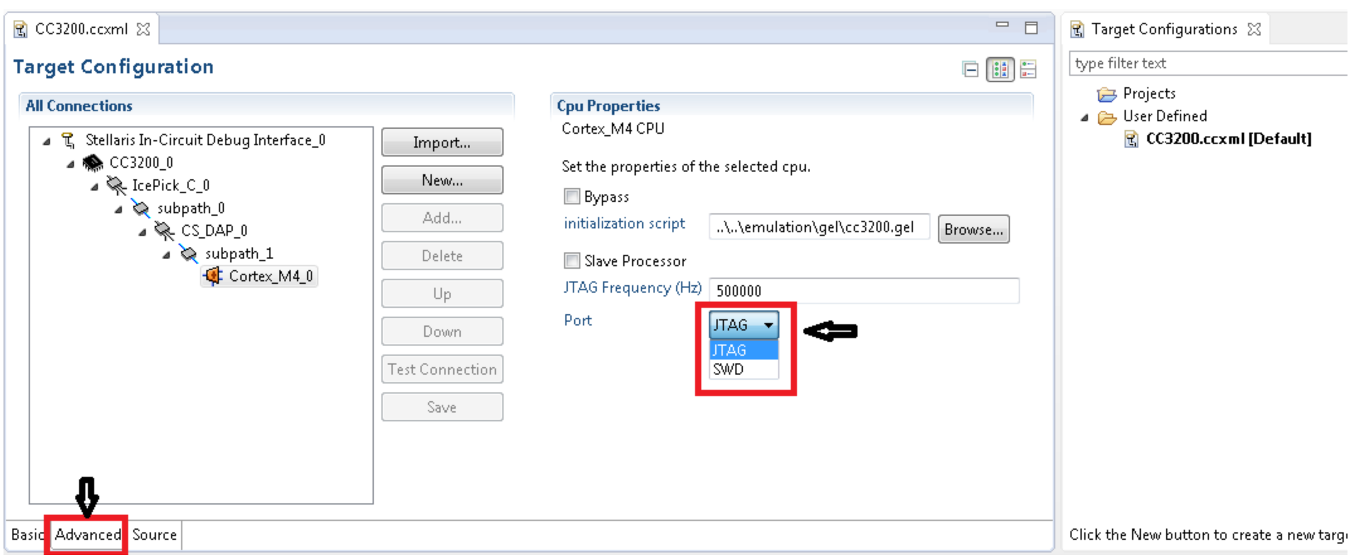
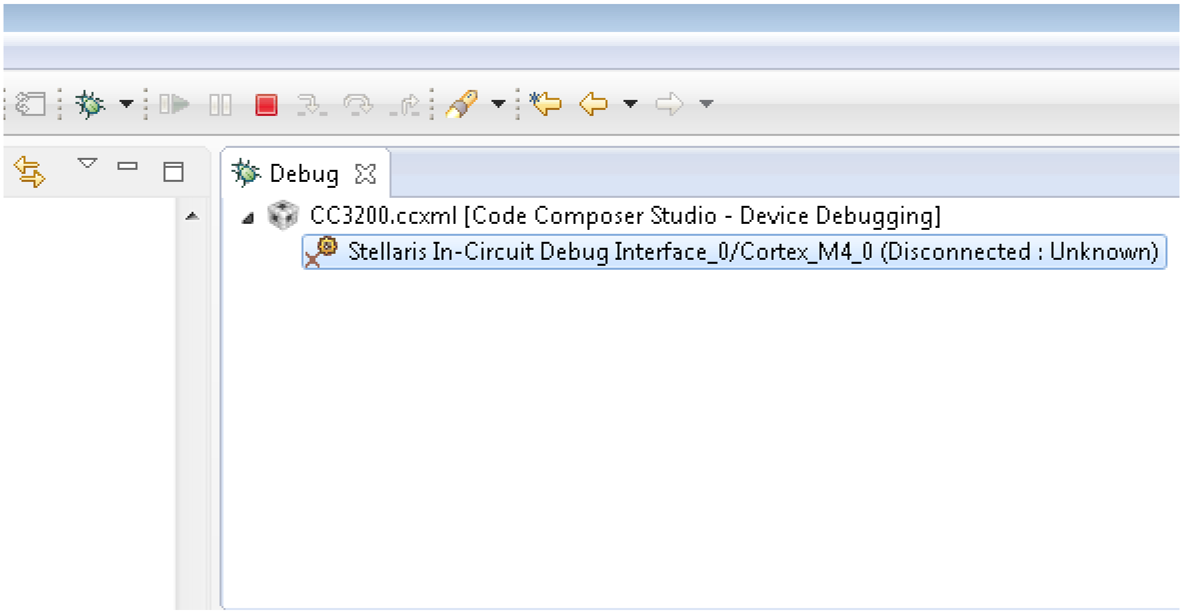


Figure 34. Target Configuration

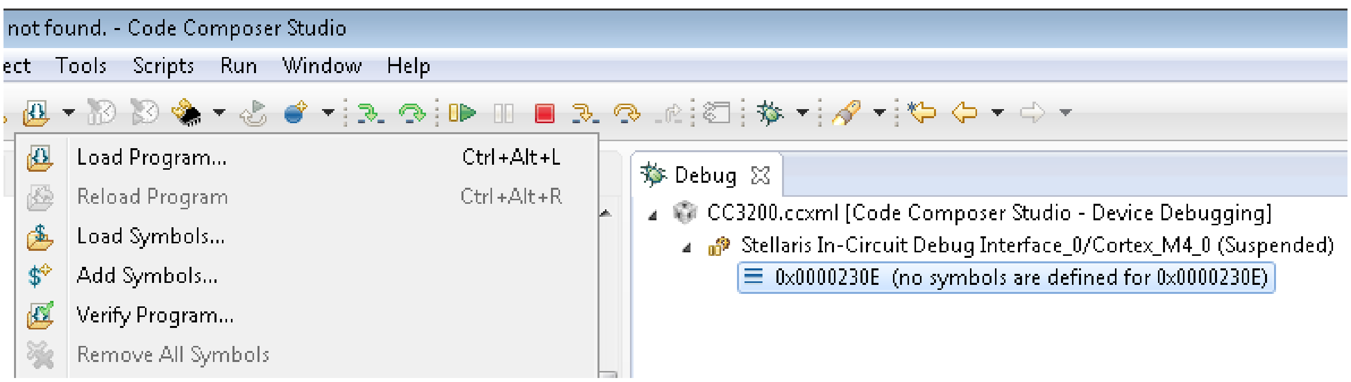
In the Debug window, right click on "Connect Target."

Figure 35. CC3200 CCS Executing 4



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

Figure 36. CC3200 CCS Executing 5



The execution will stop 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 Development Environment – Open Source [GCC/GDB]

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 like SimpleLink library, peripheral driver library and so forth).

5.3.3.1 Cygwin Installation (Windows)

Cygwin can be installed by downloading setup-x86.exe from http://cygwin.com/install.html and running it locally. Set the proxy setting in Cygwin installation window and proceed further. CC3200 SDK is ported and tested with the Cygwin 32-bit version only.

Include the following packages in the Cygwin installation in addition to the "base" installation. The latest versions of all packages should be acceptable.

1. Archive/unzip
2. Archive/zip
3. Devel/autoconf
4. Devel/automake
5. Devel/libtool
6. Devel/make
7. Devel/subversion (Note: if you plan to use TortoiseSVN/Windows7, skip this)
8. Devel/gcc-core
9. Devel/gcc-g++
10. Devel/gcc-mingw-core
11. Devel/gcc-mingw-g++
12. Devel/mingw-runtime

Note: After a successful Cygwin installation, add its path (*c:\cygwin\bin*\) to the Windows environment variable.

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*).

5.3.3.3 Open On-Chip Debugger (OpenOCD)

Open on-chip debugger (OpenOCD) can be downloaded in source form from http://sourceforge.net/projects/openocd/files/openocd/0.7.0/ and complied locally.

To build OpenOCD for FTDI interface, the user needs to download the FTDI driver library (x86 [32-bit] zip version) from http://www.ftdichip.com/Drivers/D2XX.htm.

Steps to compile OpenOCD with FTDI support (Cygwin bash shell):

1. Extract OpenOCD source into the Cygwin directory (*c:\cygwin*). This will create a directory called *openocd-<version>* under the Cygwin directory which contains all OpenOCD source contents.
2. Extract FTDI source into the *openocd-<version>* directory. This creates a directory called: "*CDM 2.04.06 WHQL Certified*", rename it to *ftd2xx*.
3. Change the director to *openocd-<version>*.
4. Run the following command at prompt:

```
./configure --enable-maintainer-mode --disable-werror --disable-shared --enable-
ft2232_ftd2xx --with-ftd2xx-win32-zipdir=ftd2xx
```

5. This should successfully configure OpenOCD for building.
6. Run the '**make**' command followed by '**make install**'
7. After the command runs successfully, check that openocd.exe is generated at '*C:\cygwin\usr\local\bin*'. Add the same path to the environment variable.

5.3.3.4 Compile the GCC SDK project

Go to *<cc3200-sdk>\example\getting_started_with_wlan_station \gcc* in the command prompt and run the following command at prompt:

```
make –f  Makefile
```

This will generate a *wlan_station.axf* file under the *gcc\exe* folder

5.3.3.5 Target Connection and Debugging (GDB)

The OpenOCD configuration file for FTDI is under *cc3200-sdk\ tools\gcc_scripts* folder (cc3200.cfg). To test the connection to the CC3200 FTDI LaunchPad, go to the *<cc3200-sdk>\tools\gcc_scripts* folder, run the following command at the Cygwin prompt window and check the output to ensure the connection happened properly.

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

See Figure 37for the connection output screen while the CC3200 device is connected through GDB.

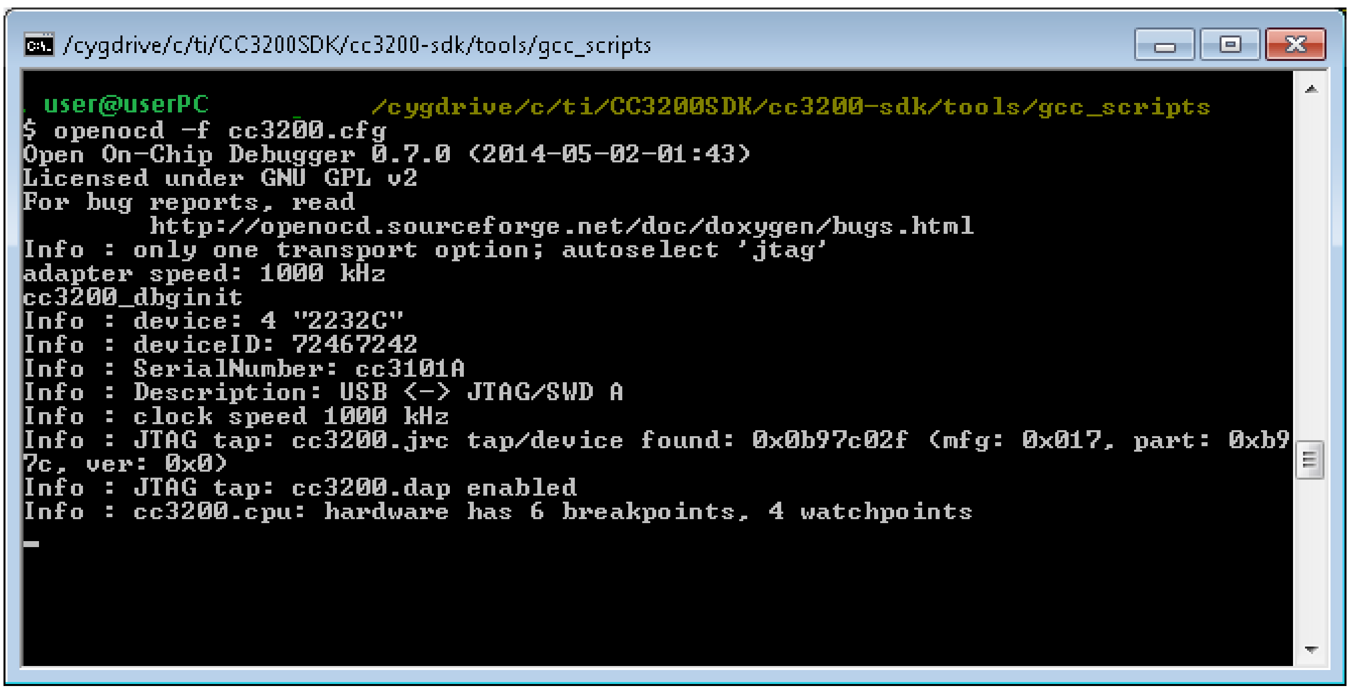


Figure 37. Connection Output Screen

To start debugging using GDB on CC3200, go to *<cc3200-sdk>\tools\gcc_scripts* and run the following command at the Cygwin prompt:

```
arm-none-eabi-gdb -x gdbinit<app.axf>
```

See Figure 38 for the debugging result of a blinky application from GCC.

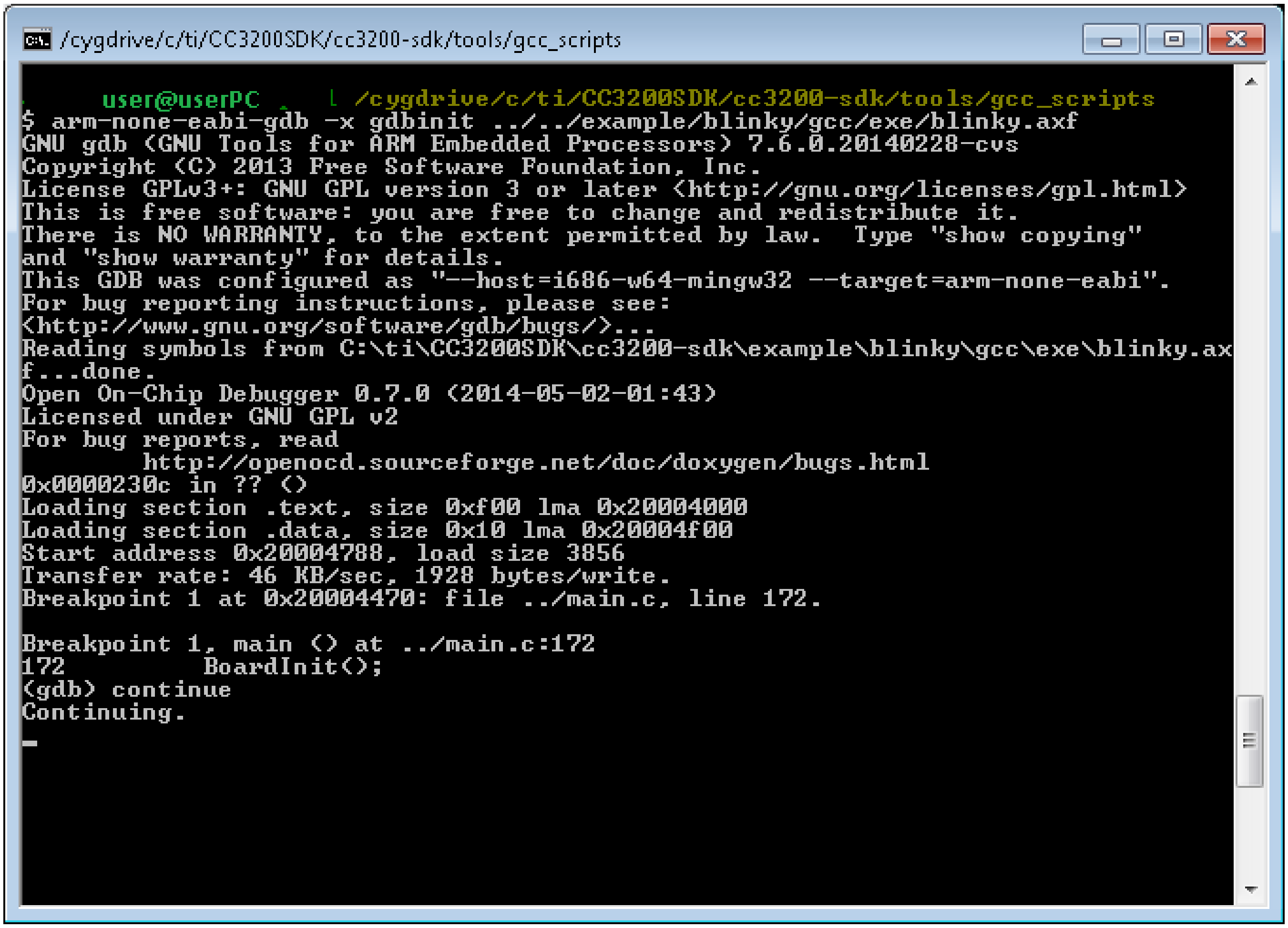


Figure 38. Blinky GCC Application

On the (gdb) prompt give a 'continue' command.

GDB Quick Guide - http://users.ece.utexas.edu/~adnan/gdb-refcard.pdf

5.3.4 Setup the Terminal Application

To view the UART-based application’s output on the terminal screen, the user should setup the terminal application (HyperTerminal, TeraTerm and so forth). Serial port settings:

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

5.4 Flashing and Running the .bin using Uniflash Tool

Once finalized, the binary images can be flashed onto the non-volatile SerialFlash (SFlash) of the LaunchPad. The application will start 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 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).

> **CAUTION**
>
> Serial Flash also hosts images corresponding to TI proprietary components. These images are not shared with the SDK. Please contact your TI representative in case the serial flash on your launchpad gets erased.

6 CC3200 ROM Services

The CC3200 ROM hosts the boot loader 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 provide an opportunity to the developer to reduce his application's RAM footprint.

Boot loader 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 Boot Loader

The CC3200 boot loader resides in the ROM of the application processor.

- **Update/Download** – The boot loader is used to download an application image from the PC to the CC3200 device. The Bootloader-DNLD functionality can be triggered only when the board is in UARTLOAD Sense On Power (SOP) mode.
- **Bootstrap** – Boot loader is responsible for scanning a valid application image (binary) in the Serial Flash (for CC3200R device). Subsequently the image is loaded to internal memory and execution control is passed on to the user program.

6.1.1 Boot Loader 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. In the context of boot loader there are two modes:

- A setting corresponding to SOP[2:0] = 0b100, makes the boot loader enter the DOWNLOAD mode and in this mode it would expect external intervention to trigger an operation – for example a "break" signal on UART from the SimpleLink programming application, which would be followed by a sequence to push the application image to device serial flash.
- A setting corresponding to SOP[2:0] = 0b000, would instruct the boot loader to load the application image from the SFLASH to internal MCU RAM.

6.1.2 Boot Loader / User Application – Sharing MCU RAM

In the DOWNLOAD mode, boot loader requires memory resources. These are acquired from the MCU RAM. The amount of RAM used by boot loader is 16 KB. This implies that for the production CC3200 R device, user application image needs to be restricted to 240KB for the 256KB MCU RAM variant of CC3200. There are several key points needing the developer's attention:

- MCU RAM address range 0x20000000 - 0x20003FFF: This area is shared between the application and the boot loader. The developer can only locate application data sections, as data sections are not part of application image; this ensures that when the boot loader is loading the application image from serial flash to RAM, this memory region is made exclusive to the boot loader. Once the boot loader 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.

Table 3. End of RAM

END of RAM Exclusively for application. Application should be part of this region and start at 0x20004000 [0x20004000]
16 KB Shared between boot loader 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/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 that helps 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_UDMATABLE
[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 0x0000042C. 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[5].

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

6.2.2 Linking User Application with ROM APIs

Using the ROM driver lib APIs for devices before ES 1.32 is not recommended, as the number of APIs that would need to be patched would be higher. Nevertheless, below are the steps to use ROM driver lib APIs instead of the RAM APIs. These could be used with the production version of a CC3200 device.

1. These steps apply to all relevant source and project files that use driver lib APIs such as the SimpleLink library.
2. All the .c files which use driver lib APIs should include these header files in order:
   - #include "rom.h"
   - #include "rom_map.h"
3. All the project files should add global preprocessor define “TARGET_IS_CC3200.”
4. 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.
5. 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

Changes from Original (June 2014) to A Revision **Page**

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