TEXAS INSTRUMENTS

SimpliciTI
Sample Application
User's Guide

Document Number: SWRA243

Texas Instruments, Inc.
San Diego, California USA

Version	Description	Date
1.0	Initial release.	03/20/2009

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

1.1 Purpose

This document provides basic information necessary for a first-time SimpliciTI™ user to try the sample applications that are included in Texas Instruments' SimpliciTI package. This includes a guide to programming the development kit hardware, a description of what each application does, and a step-by-step procedure to run the application.

1.2 Contents

This document is organized in two main sections. Section 2, *Development Environment*, introduces the available hardware and software tool options for working with SimpliciTI. Section 3, *Sample Applications*, discusses four easy-to-use sample applications that demonstrate some of the capabilities of SimpliciTI.

1.3 Audience

This document is intended for use by engineers, managers, and students who are new to SimpliciTI and would like to evaluate some of its capabilities. The user is expected to be capable of obtaining and installing Windows-based tools and software on their PC and then following instructions on building, loading, and running sample applications on the hardware included with their development kit.

1.4 Font usage

This document uses different text fonts to provide emphasis of important topics:

`Bold Fixed pitch text`	Used for file names, symbols, code snippets, and code examples.
<u>Underlined blue normal text</u>	Document cross reference hyperlink
Bold→italicized→text	Navigation through Windows program menu options

Table 1: Font Usage

1.5 Abbreviations and Acronyms

This document uses the following acronyms to abbreviate commonly used terms:

API	*A*pplication *P*rogramming *I*nterface
CCE	*C*ode *C*omposer *E*ssentials (<u>http://www.ti.com/cce</u>)
EW	*E*mbedded *W*orkbench (toolset from IAR Systems)
IAR	IAR Systems (<u>http://www.iar.com/</u>)
IDE	*I*ntegrated *D*evelopment *E*nvironment
ISR	*I*nterrupt *S*ervice *R*outine
SoC	*S*ystem *o*n *C*hip (MCU and radio are on the same integrated circuit)
SRF04	*S*mart*RF04* Evaluation Board
SRF05	*S*mart*RF05* Evaluation Board (Rev 1.7 or later)

Table 2: Abbreviations and Acronyms

1.6 References

The following documents are located within the **Documents** folder of your SimpliciTI installation:

1. *SimpliciTI Specification Version*
2. *SimpliciTI Developers Notes*
3. *SimpliciTI Channel Table Information*
4. *Application Note on SimpliciTI Frequency Agility Description*

2. Development Environment

In order to evaluate the SimpliciTI sample applications, you will need to obtain some tools to build and download the software and some hardware to run it on. Currently, the SimpliciTI package has support for 14 different hardware development kit configurations, which use various combinations of Texas Instruments processors and radios. For each hardware configuration, there are software development projects to build each of the four sample applications. All of these projects are supported by IAR Systems Embedded Workbench (EW) development tools. Projects based on the MSP430 processor are also supported by Texas Instruments' Code Composer Essentials (CCE) development tools. Both toolsets include an IDE, compiler, assembler, linker, downloader, debugger, and documentation. The following two tables show which CCE or IAR toolset you will need and which SimpliciTI project you will use, based on the development kit hardware that you have:

System-On-Chip Development Hardware	Frequency	Project	CCE Toolset	IAR Toolset
SmartRF04EB + CC1110EM	< 1 GHz	SRF04	N/A	EW8051
CC1111EM USB Dongle	< 1 GHz	RFUSB	N/A	EW8051
CC2430DB	2.4 GHz	CC2430DB	N/A	EW8051
SmartRF04EB + CC2430EM	2.4 GHz	SRF04	N/A	EW8051
SmartRF04EB + CC2431EM	2.4 GHz	SRF04	N/A	EW8051
SmartRF04EB + CC2510EM	2.4 GHz	SRF04	N/A	EW8051
SmartRF05EB + CC2530EM	2.4 GHz	SRF05	N/A	EW8051
CC2511EM USB Dongle	2.4 GHz	RFUSB	N/A	EW8051

Table 3: SoC Development Hardware and Tool Options

Dual-Chip Development Hardware	Frequency	Project	CCE Toolset	IAR Toolset
EXP430FG4618 + CC1100:433 or CC1100:868 + USB Debug IF	< 1 GHz	EXP461x	CCEv3.1	EW430
EXP430FG4618 + CC1101:433 or CC1101:868 + USB Debug IF	< 1 GHz	EXP461x	CCEv3.1	EW430
EXP430FG4618 + CC1100E:470 or CC1100E:950 + USB Debug IF	< 1 GHz	EXP461x	CCEv3.1	EW430
EXP430FG4618 + CC2500 + USB Debug IF	2.4 GHz	EXP461x	CCEv3.1	EW430
eZ430 + RF2500	2.4 GHz	eZ430RF	CCEv3.1	EW430
SmartRF05EB + MSP430F2618 + CC2520	2.4 GHz	SRF05	CCEv3.1	EW430

Table 4: Dual-Chip Development Hardware and Tool Options

As shown in the tables above, SimpliciTI is available for a wide variety development kits. Target hardware for this release is shown in the following table. The 'Platform' column corresponds to the project folder names in the installed SimpliciTI directory structure. An entry of *SoC* in the 'Radio' column indicates a single-chip platform. The 'Radio Version' and the 'EM Revision' indicate the hardware level with which the code was tested. Some platforms, such as the EXP461x, RFUSB, SRF04, and SRF05 support multiple radios. The target radio for these platforms is selected during the build by defining the appropriate pre-processor value in the project file.

Platform	MCU	Radio	Radio Version	EM Revision	Comments
CC2430DB	CC2430	SoC	E	1.3	
EXP461x	MSP430FG4618	CC1100	F	3.2	Radio on EM daughter board
EXP461x	MSP430FG4618	CC1101	A	1.0	Radio on EM daughter board
EXP461x	MSP430FG4618	CC1100E	H	1.0	Radio on EM daughter board
EXP461x	MSP430FG4618	CC2500	F	2.2	Radio on EM daughter board
eZ430RF	MSP430F2274	CC2500	F	1.0	Debug via USB
RFUSB	CC1111	SoC	D	1.2	Debug via SmartRF04
RFUSB	CC2511	SoC	E	1.3	Debug via SmartRF04
SRF04	CC1110	SoC	D	2.0	SoC on EM daughter board
SRF04	CC2430	SoC	E	1.2	SoC on EM daughter board
SRF04	CC2431	SoC	E	1.3	SoC on EM daughter board
SRF04	CC2510	SoC	E	2.0	SoC on EM daughter board
SRF05	MSP430F2618	CC2520	-	2.1	Chip ID 0x84 to validate, not radio version
SRF05	CC2530	SoC	2.1	1.0	SoC on EM daughter board

Table 5: Radio and Evaluation Module Versions

2.1 Getting the Tools You Need

The best way to evaluate SimpliciTI is to obtain a development kit from Texas Instruments. The tables in the previous section provide links to the available hardware development kits and the associated software development toolsets. Note that some of the development kits have more than one link, indicating that you need to obtain more than one hardware package to proceed. Typically, this means that the "motherboard", "radio", and "debugger" modules are not bundled in the same package. Also note that some radio modules are offered with different frequency ranges – make sure that you have the correct one for your evaluation purposes.

Once the hardware development kit is selected, the software development toolset can be obtained. As shown above, kits based on the MSP430 are supported by both toolsets. To get started with Code Composer Essentials (CCE), you can download the Core edition, which allows unlimited use on programs up to 16KB in size. Later, if your program grows to exceed 16KB, you can get the Professional version of CCE. To begin with IAR Embedded Workbench (EW), you can download the 30-day evaluation edition (extended to 60 days when a development kit is purchased), which is a full-featured toolset. After the evaluation period expires, you can get a permanent version of EW. Note that SimpliciTI is not configured to work with the EW KickStart editions.

The RFUSB-CC1111 and RFUSB-CC2511 are programmed using the SmartRF04 board. If there are problems with programming the CC1111/CC2511 target, it is possible that the firmware on the SmartRF04 requires an upgrade. This upgrade can be installed using the SmartRF04 Flash Programmer. For instructions on programming the RFUSB-CC2511 (also applicable to RFUSB-CC1111), download the document "CC2511 USB Dongle User Manual (swru082.pdf)": http://focus.ti.com/docs/toolsw/folders/print/cc2510-cc2511dk.html

2.2 Installing SimpliciTI and Development Tools

SimpliciTI and the software development tools are provided as Microsoft Windows-based product installers. These can be installed on a computer that is running Windows XP (or later). Simply double-click on the product installer file and follow the instructions for loading and registering the products.

After installing SimpliciTI on your computer, you can use the Windows browser to examine the folders and files that were placed in the ***C:\Texas Instruments\SimpliciTI*** directory. In the ***Projects\Examples*** folder there are sub-folders for each of the available development kits that contain project files to build the four SimpliciTI sample applications. The examples in this tutorial are based on the `eZ430RF` and `SRF04` platforms (see Table 3), as shown below:
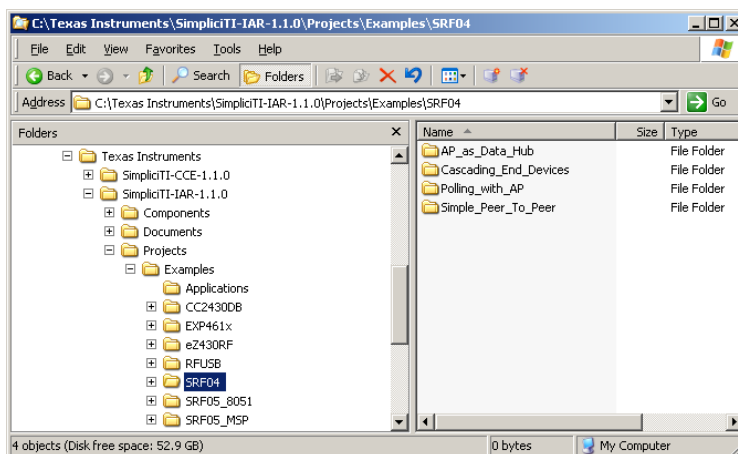

Figure 1: Windows Folder View of SimpliciTI Installation

2.3 Building a Sample Application

SimpliciTI has four different sample applications for you to try. To demonstrate the process of building, loading, and running an application, we'll use the "Simple_Peer_To_Peer" example, in conjunction with the `eZ430RF` (CCE tools) or `SRF04` (IAR tools) platform. To build other applications, just select the desired project file in the software development IDE, and follow the steps provided in section 2.3.1 if you're using the IAR tools, or section 2.3.2 if you're using the CCE tools.

2.3.1 Using IAR Embedded Workbench

To begin working with a SimpliciTI sample application, start the IDE -- double-click on either (1) the IAR icon on the PC desktop, or (2) the IDE program file located in the installed IAR tools folder, such as: "C:\Program Files\IAR Systems\Embedded Workbench 5.0\common\bin\IarIdePm.exe". Prepare to open a project workspace by clicking through the ***File→Open→Workspace…***options, as shown below:
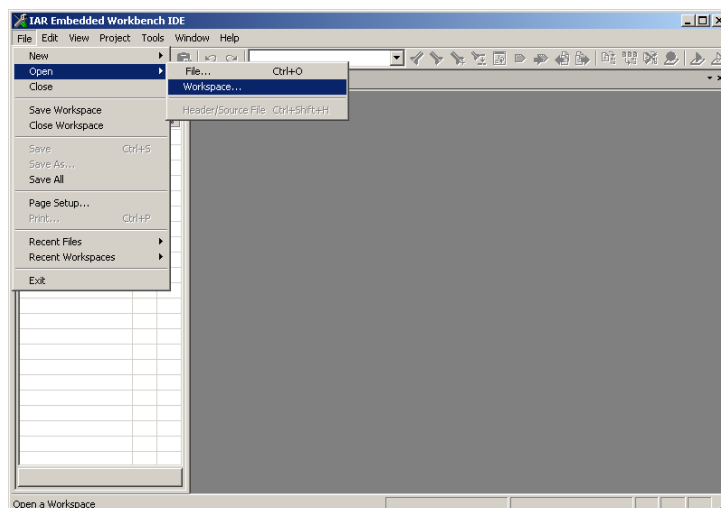

Figure 2: Opening an IAR Project Workspace

At this point, find the folder that corresponds to the development kit that you have, by navigating through several levels to the *C:→Texas Instruments→SimpliciTI-IAR-1.1.0→Projects→Examples* folder. Here you see one folder for the Applications source files and many others for the available development kits. In this example, we use a kit for a SmartRF04EB motherboard and CC2430 radio module – so, double-click on the `SRF04` folder:
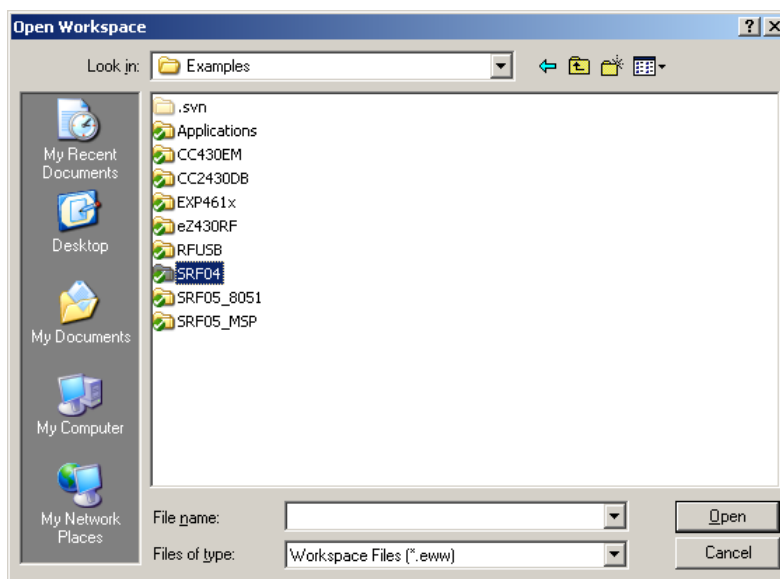

Figure 3: Selecting an IAR Workspace for Development Kit

For each development kit, there are four folders which contain IAR project files for the sample applications. In this demonstration, we use "Simple Peer to Peer", so double-click on the `Simple_Peer_To_Peer` folder:
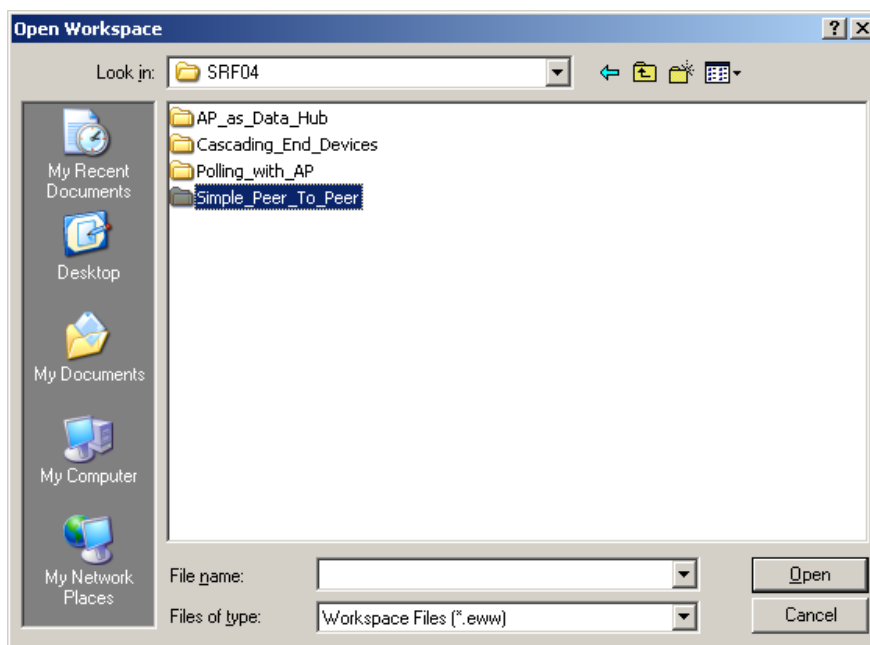

Figure 4: Selecting IAR Workspace for Sample Application

Within each of the sample application folders, there is an IAR folder which contains an Embedded Workbench "workspace" project file (with filename extension of .eww). To open the demonstration project, double-click on the **Simple_Peer_To_Peer.eww** file:
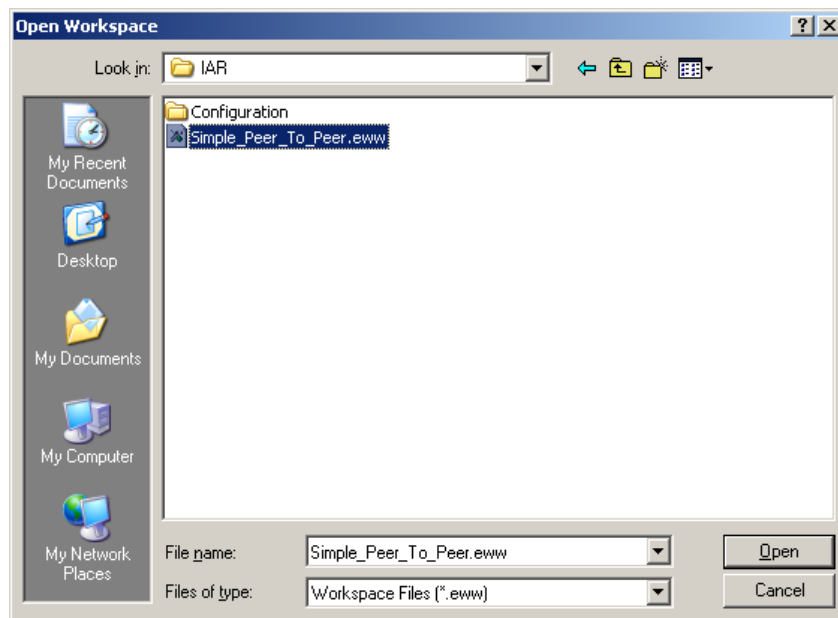
Figure 5: Opening IAR Workspace for Sample Application

Looking back at Table 3, you'll see that three different development kits are supported by the SRF04 project. In addition, the Simple_Peer_To_Peer example uses two different devices, *LinkTo* and *LinkListen*. To select the target device for the kit that you are using, pull down the menu below the *"Workspace"* label and click on the proper device. This demonstration uses the **CC2430-LinkTo** target, as shown below:
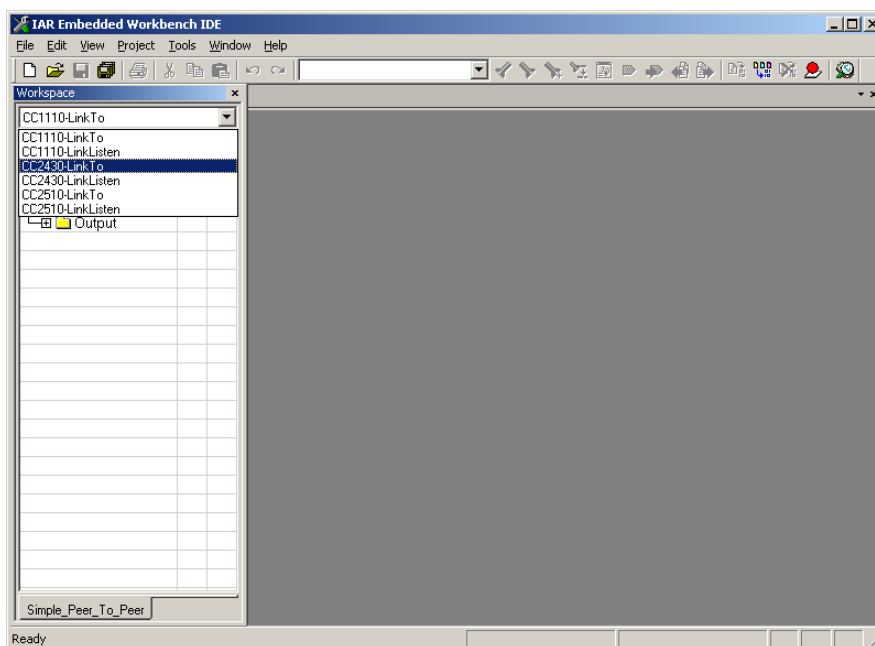
Figure 6: Selecting Target Device for IAR Sample Application

Before building a sample application program, it's best to clean up any residual files from previous builds. From the IDE's menu bar, click through **Project→Clean** in the pull-down menu to remove old files:
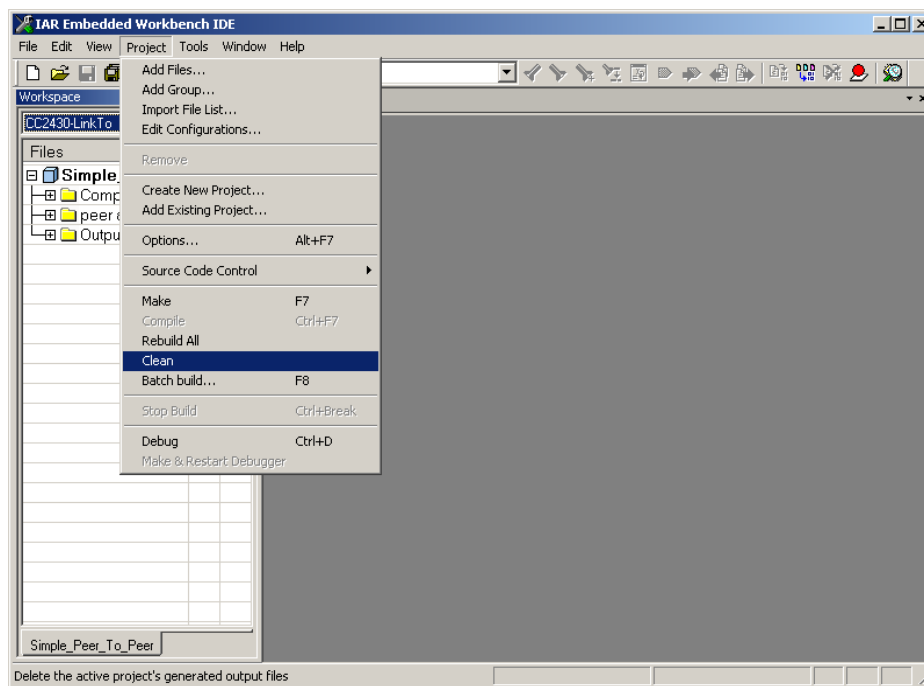
Figure 7: Cleaning Up Workspace Before Building IAR Sample Application

Now, start the build of the sample application by clicking through **Project→Rebuild All** in the pull-down menu:
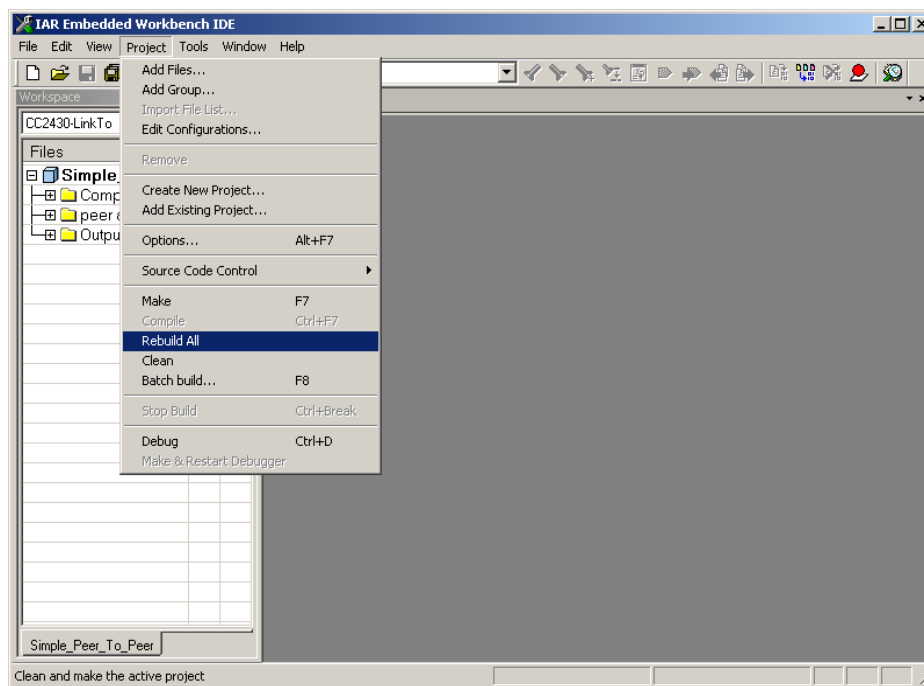
Figure 8: Building IAR Sample Application

During the building process, the status of compiling and linking the sample application program is displayed in a window at the bottom of the IDE. When the linking phase is complete, the total number of errors and warnings is indicated – normally there should not be any errors or warnings, as shown below:
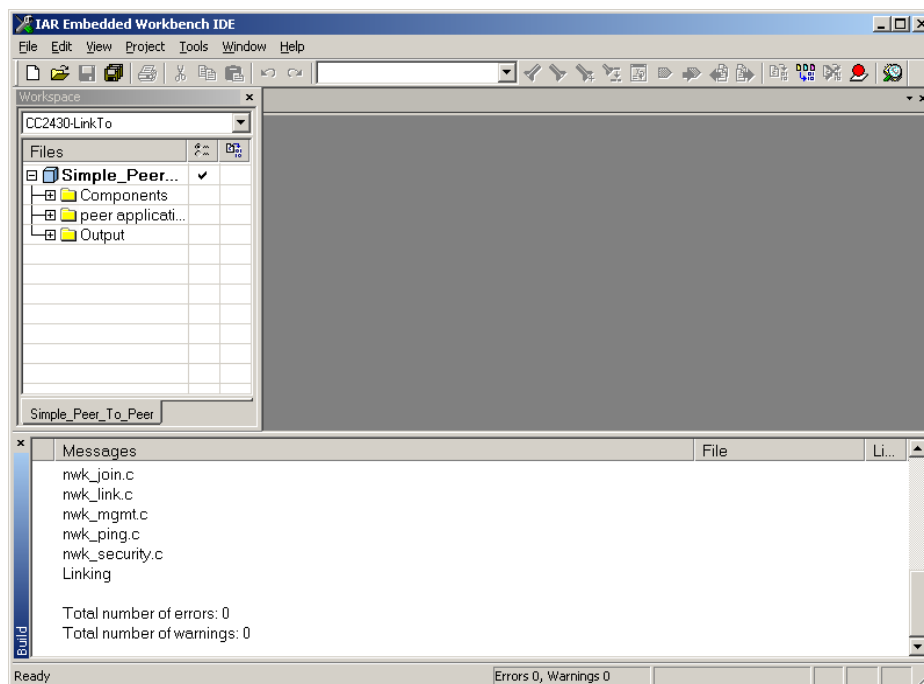
Figure 9: Successful Build of IAR Sample Application

Connect the **SRF04** board to your computer (with a USB cable) and turn on the power. If Windows prompts you to install a driver for new hardware, don't let Windows look for it automatically:

Figure 10: Installing the USB Driver for IAR – Step 1

Select the option to install the driver from a list or specific location:
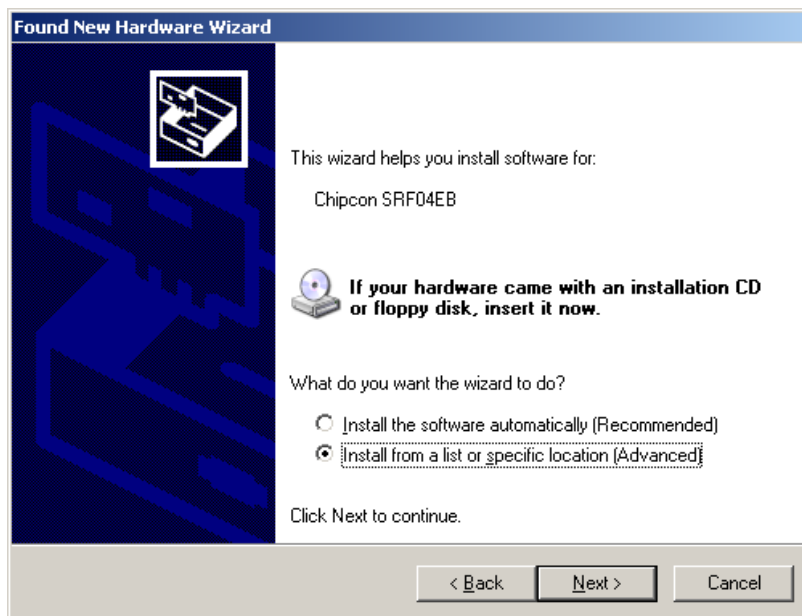
Figure 11: Installing the USB Driver for SRF04 – Step 2

Browse to drivers, such as: "C:\Program Files\IAR Systems\ Embedded Workbench 5.0\8051\drivers\Chipcon":
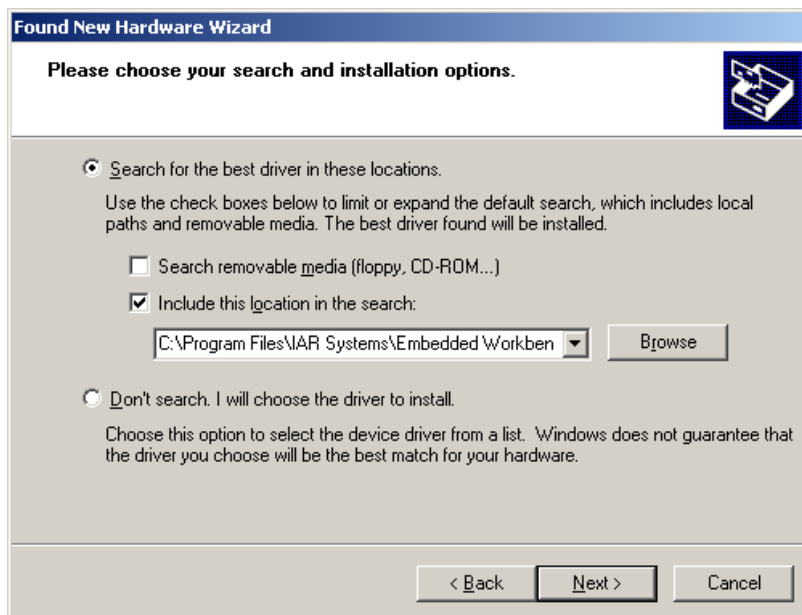
Figure 12: Installing the USB Driver for SRF04 – Step 3

Start the download of the application to the **SRF04** by clicking through *Project→Debug* in the pull-down menu:
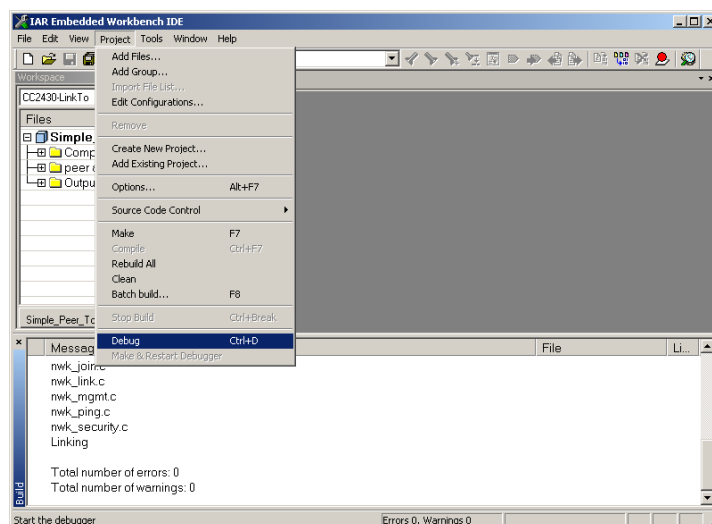
Figure 13: Starting Download of an IAR Sample Application

If asked "Do you want to Make before debugging", click the *No* button:
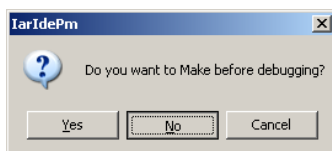
Figure 14: Continue Download

When download is complete, the IDE will enter debug mode, with the next line of code to run highlighted in green:
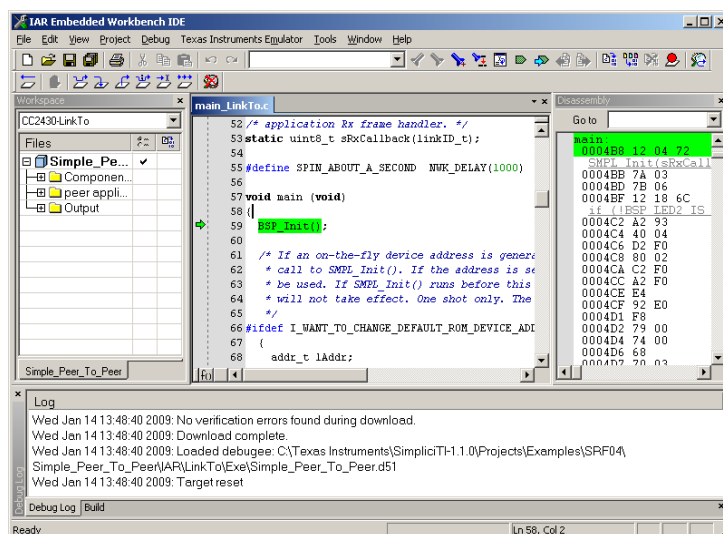
Figure 15: Completed Download to SRF04+CC2430

Up to this point, no changes have been needed to any SimpliciTI source code or configuration files. The first device was built and programmed with "out-of-the-box" settings. In order to build another *LinkTo* device for this sample application, one change must be made to the **smpl_config.dat** file. Each SimpliciTI device must have a unique address – the address shown below is the one that was used for the first *LinkTo* device. Before building another device, change the value of THIS_DEVICE_ADDRESS – the first byte is suggested, like, from 0x79 to 0x97.
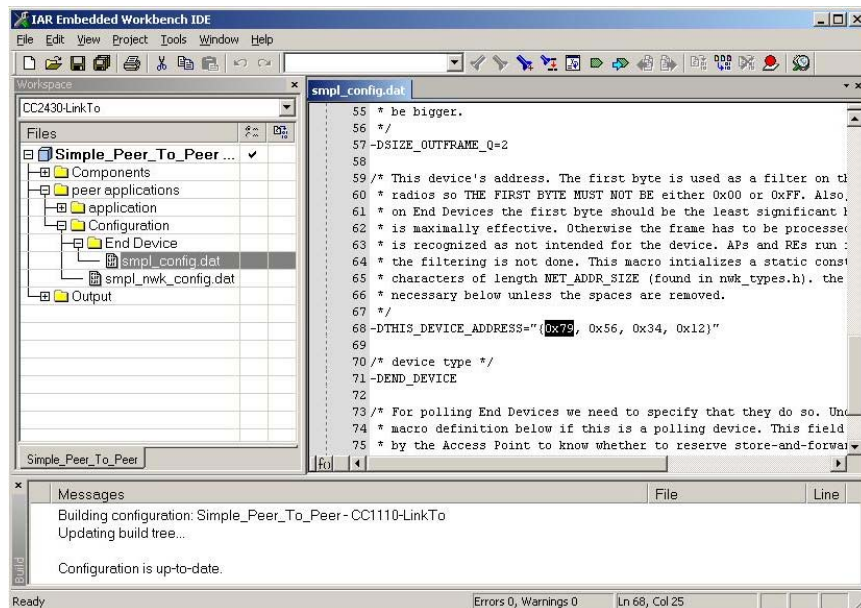
Figure 16: Changing the SimpliciTI Device Address

Repeat the steps shown in Figure 6 through Figure 16 to build and program your second SimpliciTI device. Proceed to section 3.1 for instructions on running the Simple Peer-To-Peer sample application.

2.3.2 Using Code Composer Essentials

To begin working with a SimpliciTI sample application, start the IDE -- double-click on either (1) the CCE icon on the PC desktop, or (2) the program file located in the installed CCE tools folder, such as: "C:\Program Files\Texas Instruments\CC Essentials v3.1\eclipse\eclipse.exe". This will display the product splash-screen:

Figure 17: Code Composer Essentials Splash Screen

For each development kit, there are four folders which contain CCE project files for the sample applications. Each CCE project for SimpliciTI is located in a CCE "workspace" folder associated with each sample application. In this demonstration, we use "Simple Peer to Peer" in conjunction with the **`eZ430RF`** development kit. Use the ***Browse*** button to select the *Workspace* that we will use, and hit the ***OK*** button:
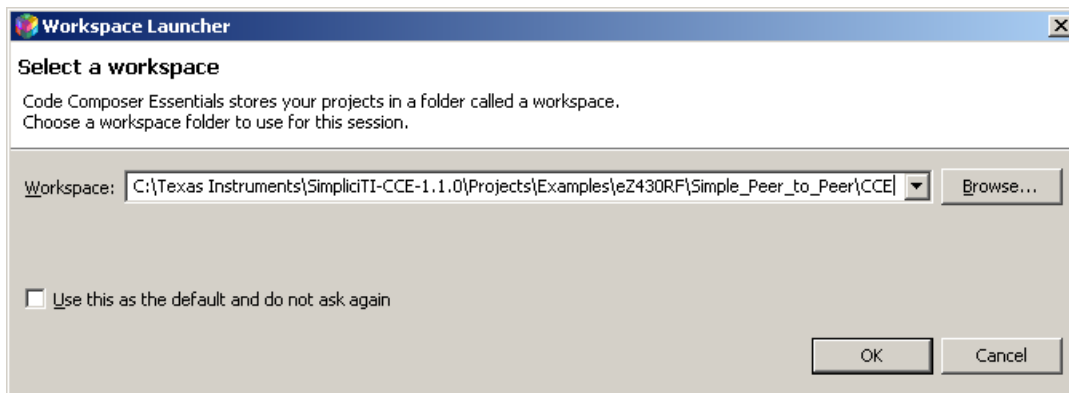
Figure 18: Selecting CCE Workspace for a Sample Application

If this is the first time that the project has been accessed by Code Composer Essentials, the "Welcome" screen will be displayed. Click on the yellow diamond to exit and launch the Code Composer Workbench:

Figure 19: CCE Welcome Screen

SimpliciTI projects are designed to be portable, allowing the "root" folder to be moved or renamed if desired. The default root folder for this SimpliciTI installation in this example is *C:\Texas Instruments\SimpliciTI-CCE-1.1.0*, as shown when selecting the CCE workspace in Figure 18. The root folder location, referred to in the project files as `DEV_ROOT`, needs to be entered into each new CCE workspace, using the process shown in the next 4 screenshots.

Select the *Preferences* item from the *Window* pull-down menu:
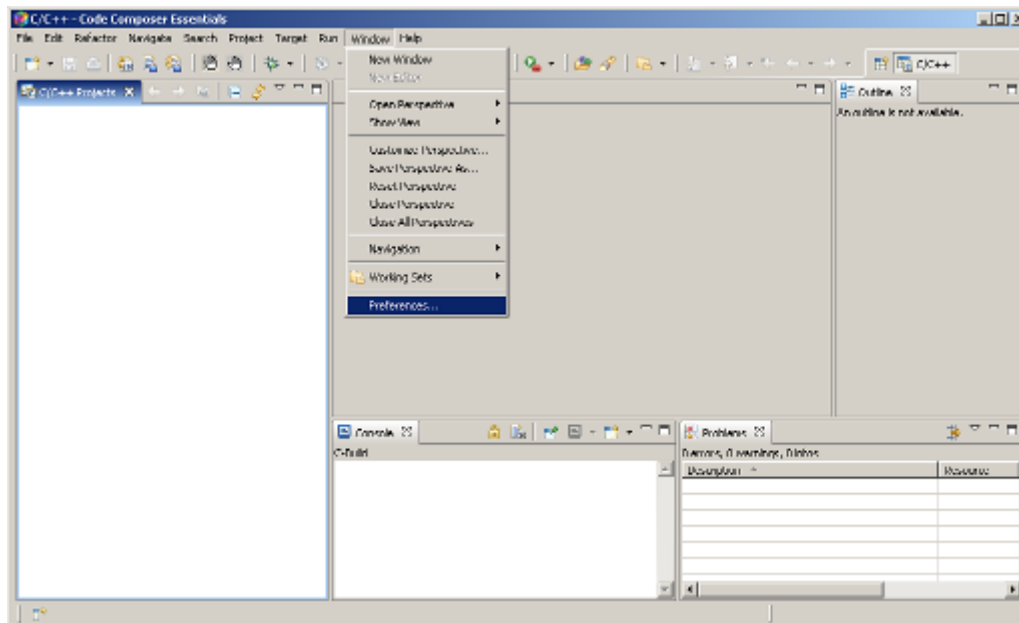
Figure 20: Setting the Root Folder Location for a CCE Workspace – Step 1

Select the *Linked Resources* item from the *General→Workspace* settings, ensure that the *"Enable linked resources"* box is checked, and hit the *New…* button:
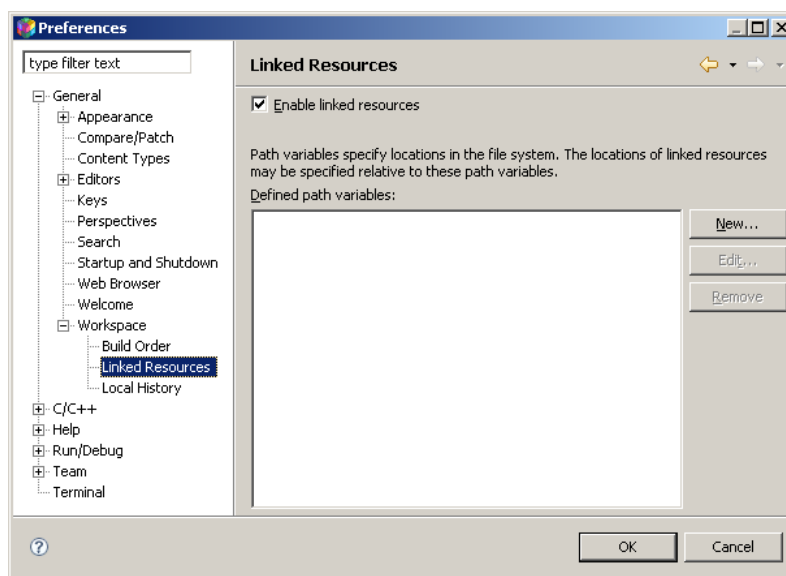
Figure 21: Setting the Root Folder Location for a CCE Workspace – Step 2

All source and configuration files in the CCE sample application projects are linked to the SimpliciTI root folder by the path variable known as **DEV_ROOT**. Each time a new CCE workspace is set up, the **DEV_ROOT** path variable must be defined. Enter **DEV_ROOT** into the *Name:* box and the root folder location of your SimpliciTI installation into the *Location:* box (you can type it directly or browse using the *Folder…* button). Hit the ***OK*** button to proceed:
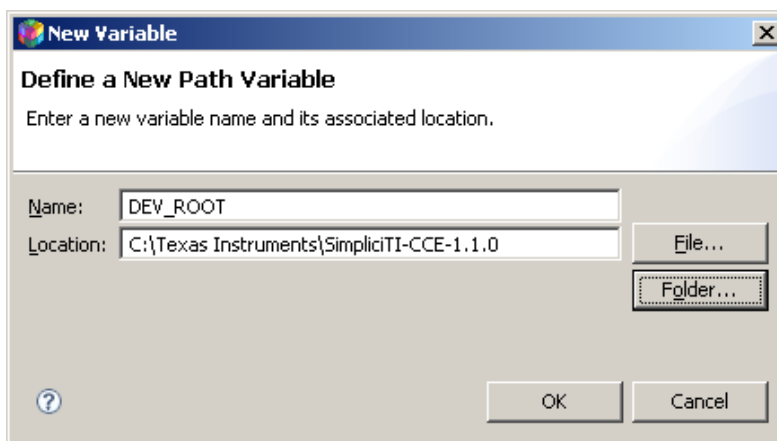
Figure 22: Setting the Root Folder Location for a CCE Workspace – Step 3

Verify that the *Defined path variable:* for **DEV_ROOT** is correct and hit the ***OK*** button to proceed:
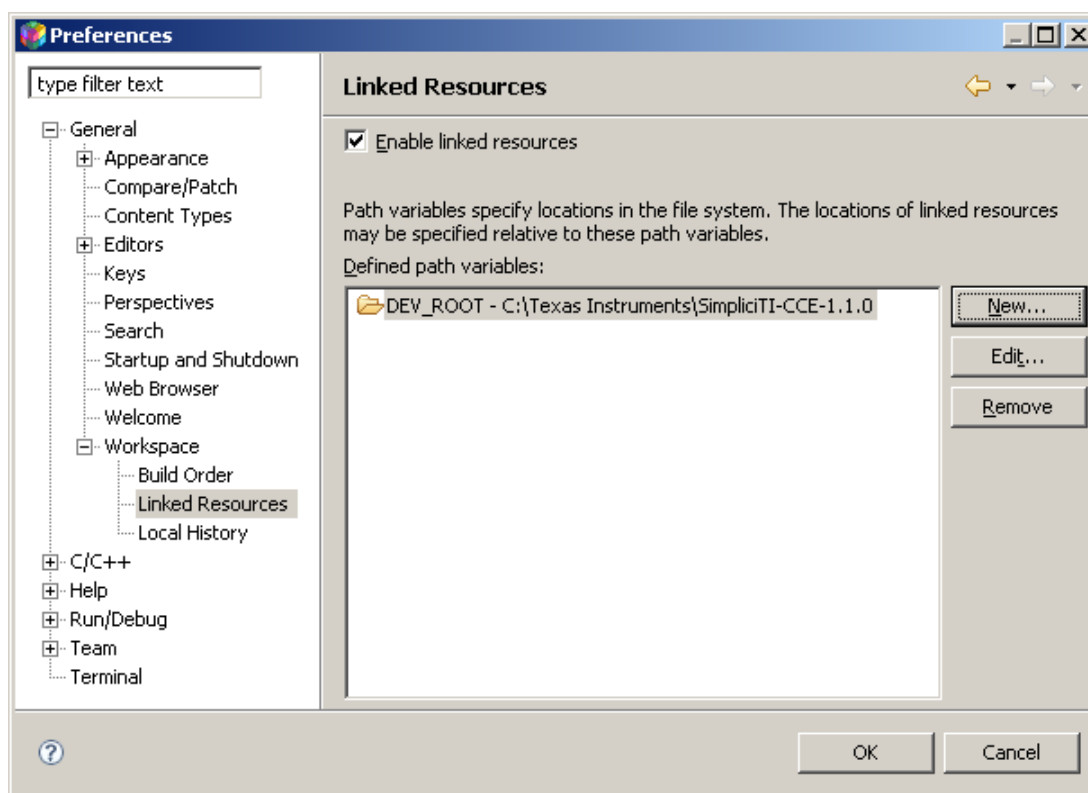
Figure 23: Setting the Root Folder Location for a CCE Workspace – Step 4

To open a sample application project, elect the **Open Existing Project** item from the *Window* pull-down menu:
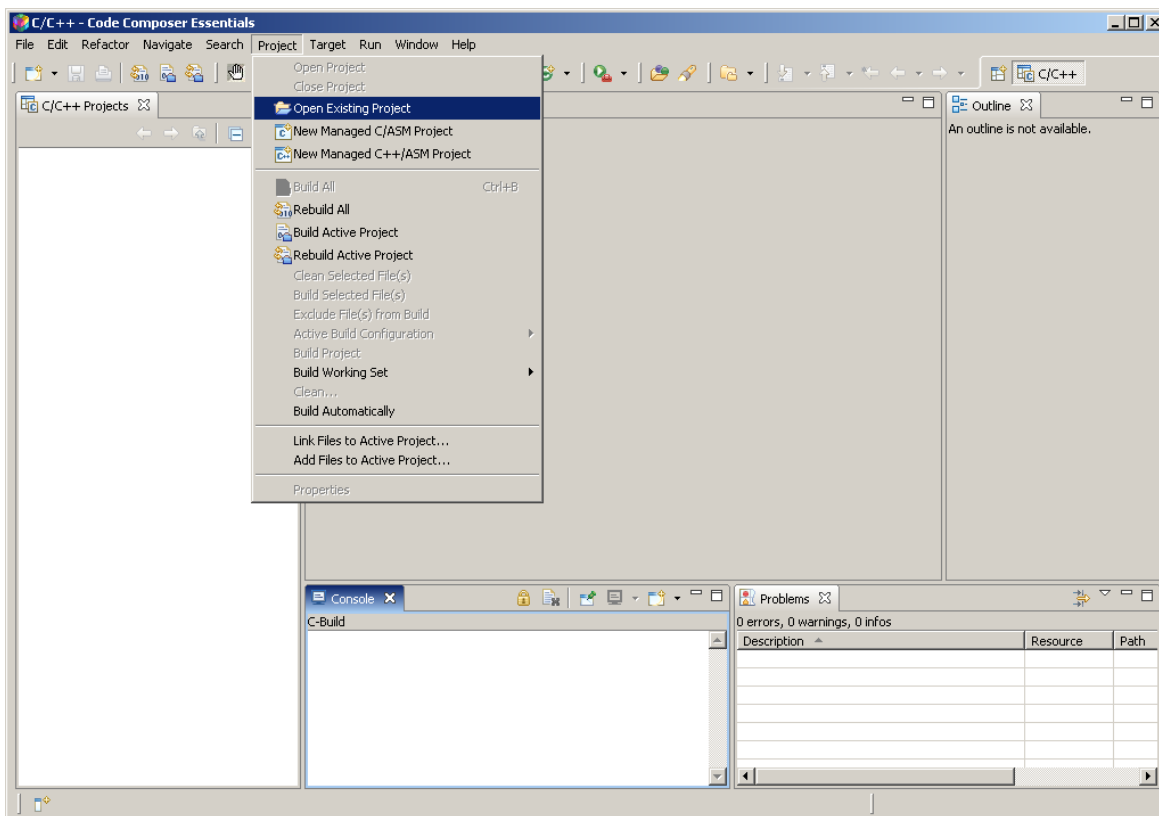
Figure 24: Opening a CCE Project – Step 1

Hit the **B*r*owse…** button to fill the *Select root directory:* box:
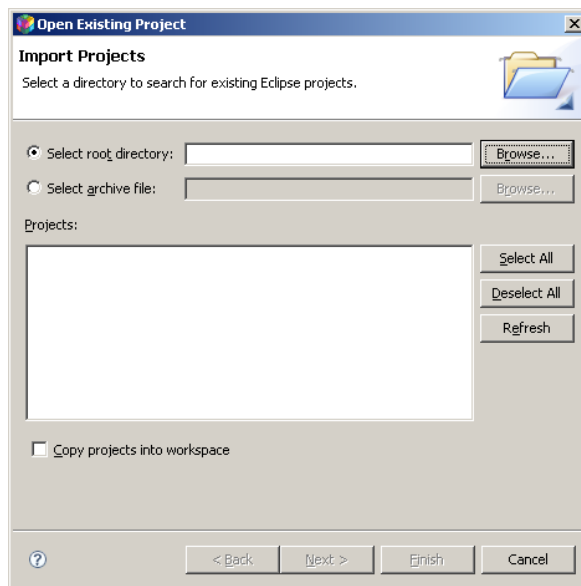
Figure 25: Opening a CCE Project – Step 2

Select the CCE folder for the project to be used (**Simple_Peer_to_Peer**). As shown below, CCE defaults to the folder that was selected for the workspace (see Figure 18). Hi the *OK* button to continue:
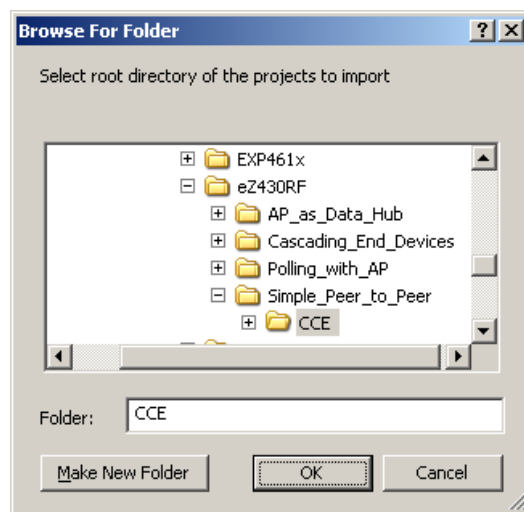
Figure 26: Opening a CCE Project – Step 3

At this point, CCE searches the selected folder for project files and displays them in the *Projects* box. SimpliciTI provides a folder named **Project** for each sample application. Hit the *Finish* button to open the project:
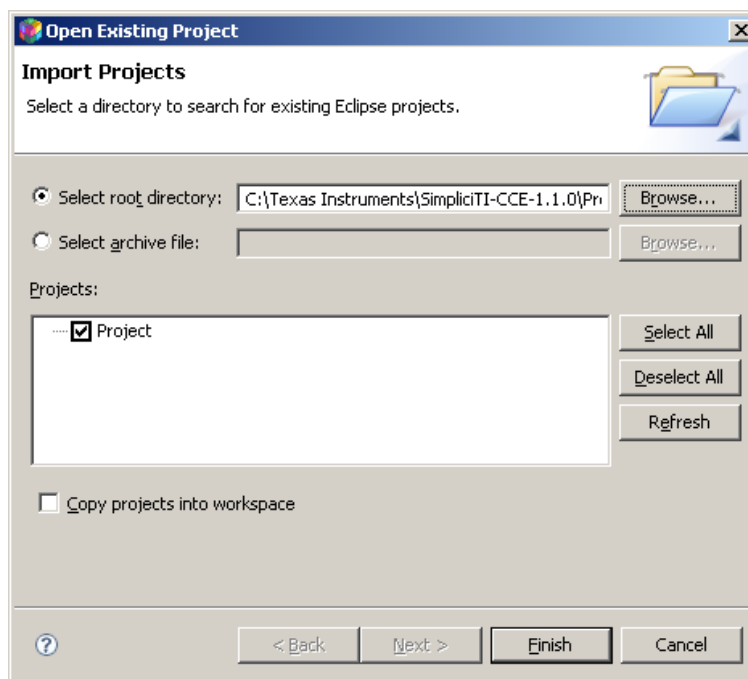
Figure 27: Opening a CCE Project – Step 4

The first time that CCE opens a specific project, it will display the basic project development window, and then will take a few seconds to generate workspace and project related files. There is a status display area located in the lower

right corner that indicates various actions during this time (the screen capture shows "Refreshing workspace"). When complete, the *C/C++ Projects* panel on the left side of the window will display the default device configuration. For this example, the `CC2500-LinkTo` configuration is active:
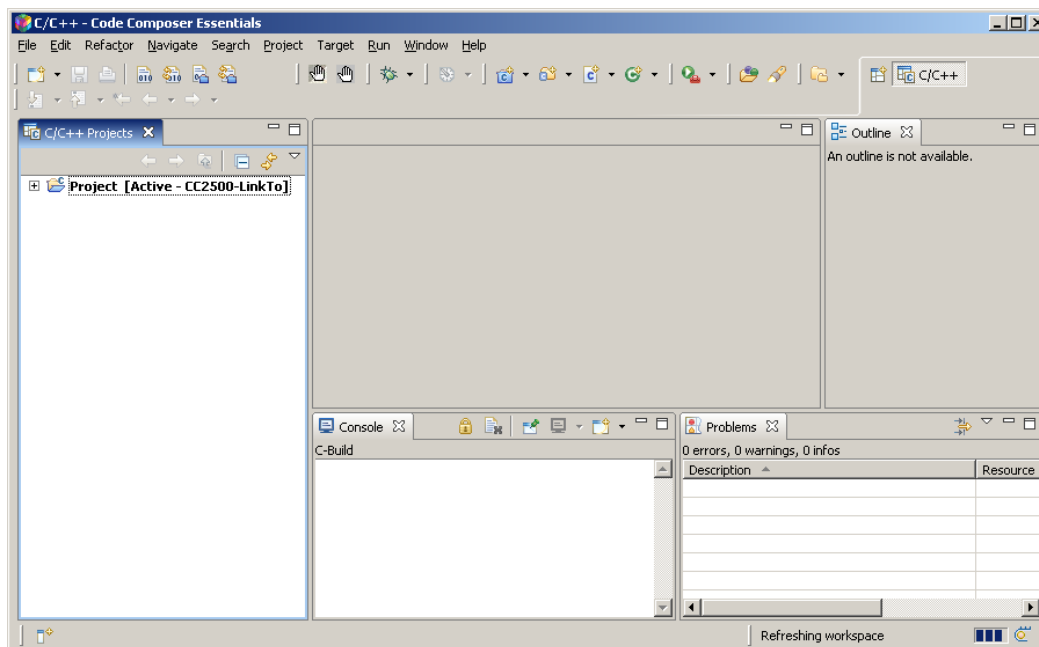
Figure 28: Opening a CCE Project – Step 5

To expand the Project folder in the *C/C++ Projects* pane, click on the □ next to the ***Project*** folder icon. Before building a sample application, it's best to clean up any residual files from previous builds. From the IDE's menu bar, click through ***Project→Clean*** in the pull-down menu to remove old files:
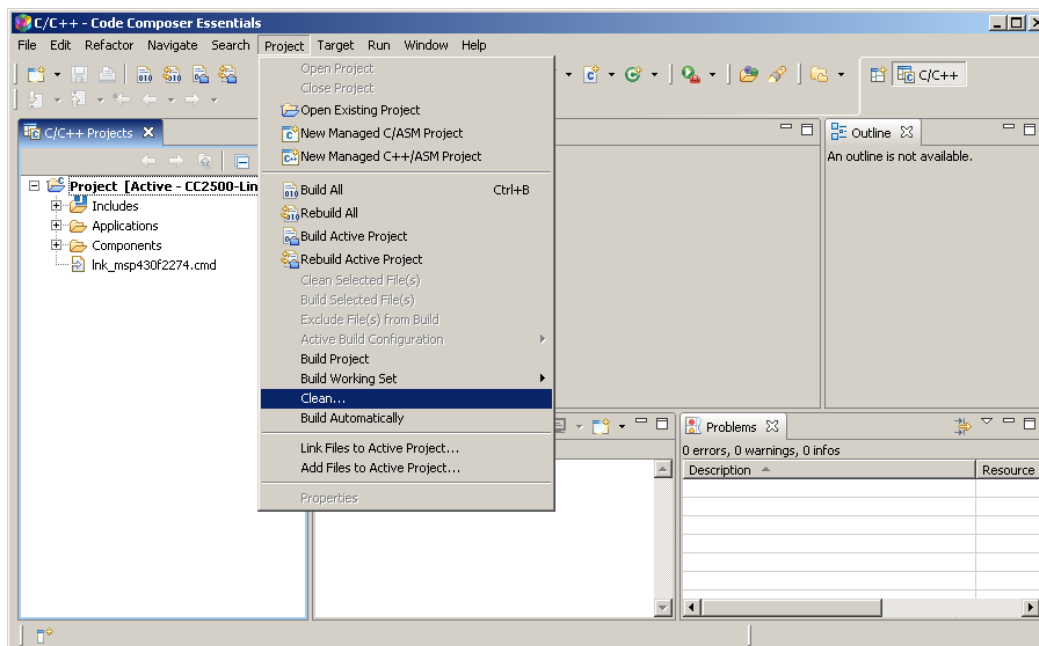
Figure 29: Cleaning Up Workspace Before Building a CCE Project

Now, start the build of the sample application by clicking through ***Project→Build Projectl*** in the pull-down menu:

Figure 30: Building a CCE Project

During the building process, the status of compiling and linking the sample application program is displayed in the *Console* pane at the bottom/center of the IDE. When the linking phase is complete, the total number of errors and warnings is indicated in the *Problems* pane at the bottom/right of the IDE – normally there should not be any errors or warnings, as shown below:
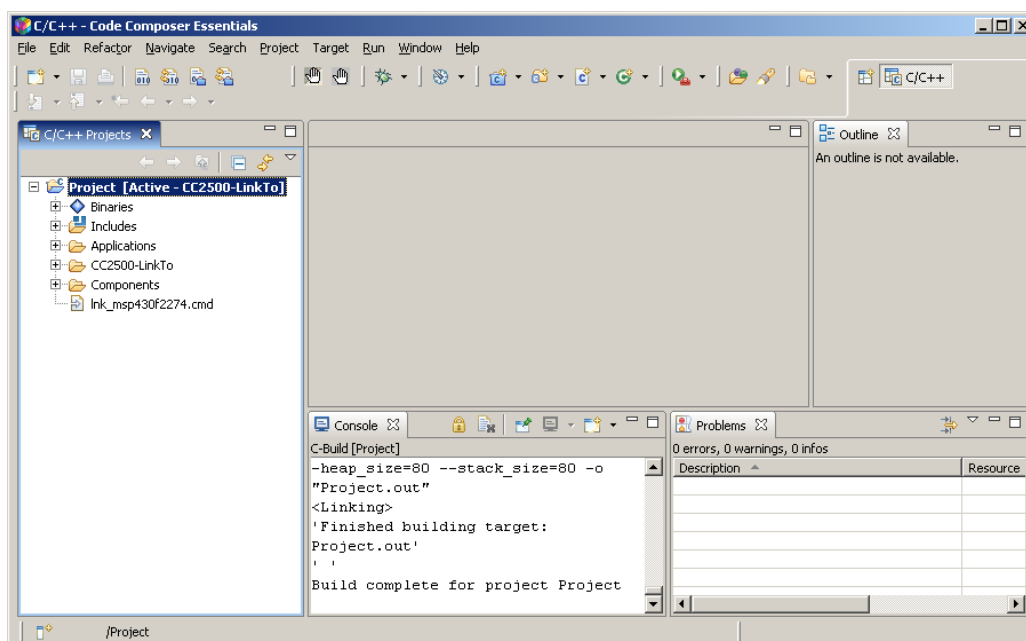
Figure 31: Successful Build of a CCE Project

Plug an `eZ430RF` board into a USB port on your computer. Start the download of the sample application to the `eZ430RF` by clicking through ***Run→Debug Active Project*** in the pull-down menu:
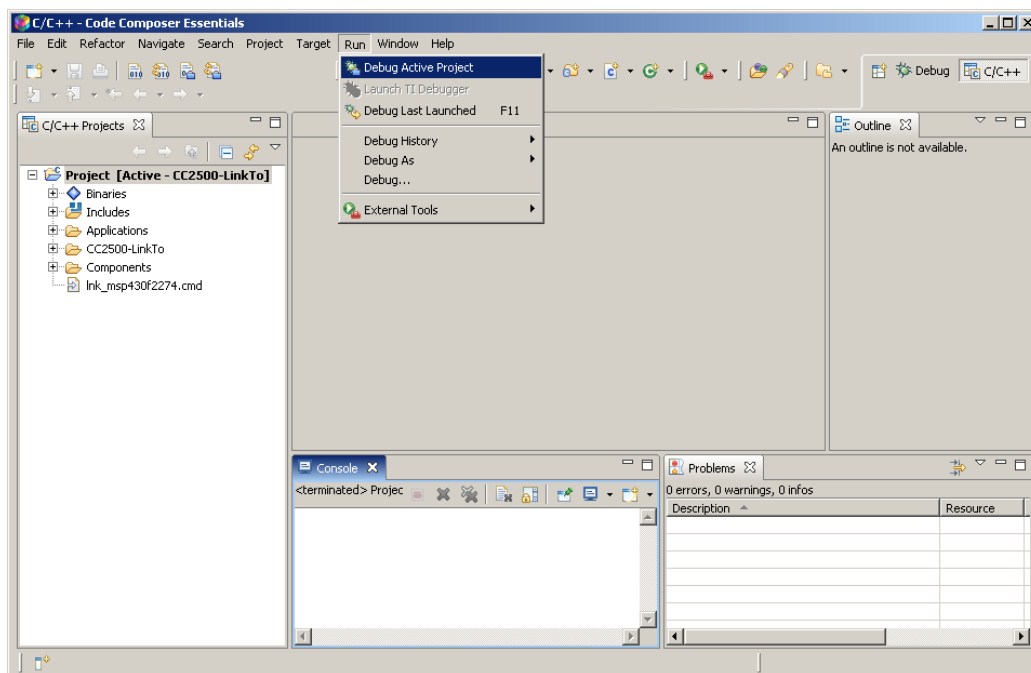
Figure 32: Downloading a CCE Sample Application

When the download is complete, the IDE enters debug mode, with the next line of code to run highlighted in green:
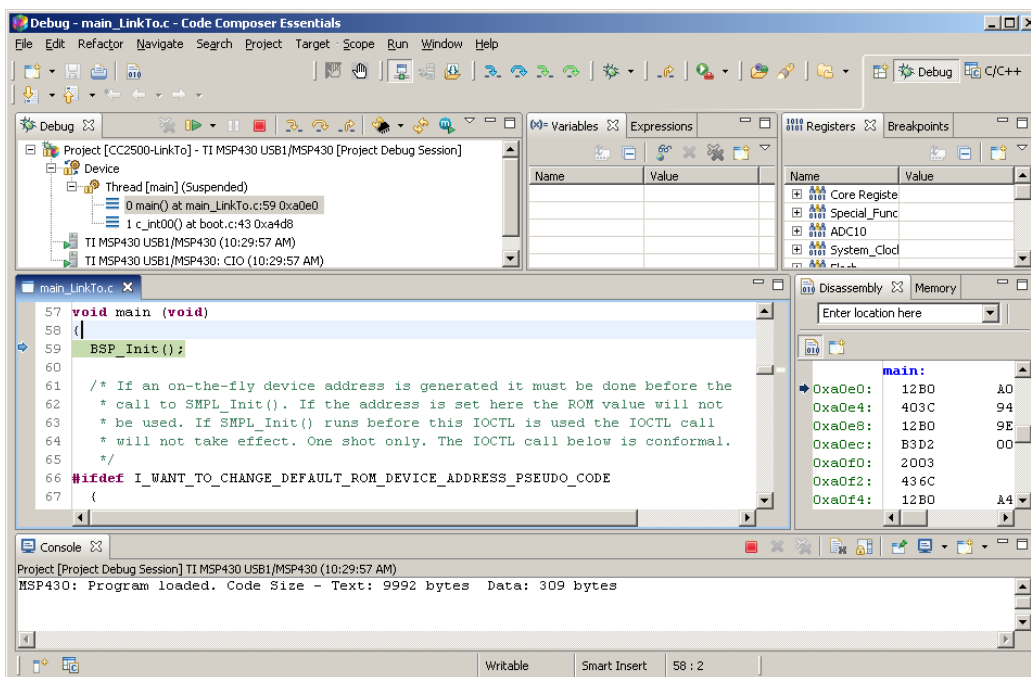
Figure 33: Completed Download to eZ430RF+CC2500

Up to this point, no changes have been needed to any SimpliciTI source code or configuration files. The first device was built and programmed with "out-of-the-box" settings. In order to build another *LinkTo* device for this sample application, one change must be made to the **smpl_config.dat** file. Each SimpliciTI device must have a unique address – the address shown below is the one that was used for the first *LinkTo* device. Before building another device, change the value of THIS_DEVICE_ADDRESS – the first byte is suggested, like, from 0x78 to 0x87.
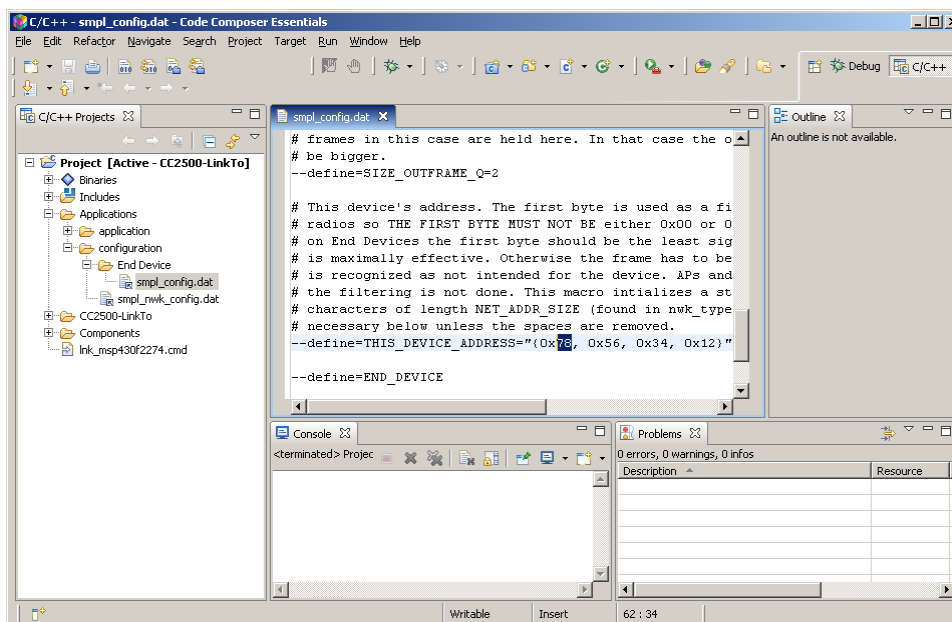
Figure 34: Changing the SimpliciTI End-Device Address

This sample application provides for two different device configurations, *LinkTo* and *LinkListen*. To change to another device, click through **Project→Active Build Configuration** to show the pull-down menu of devices:
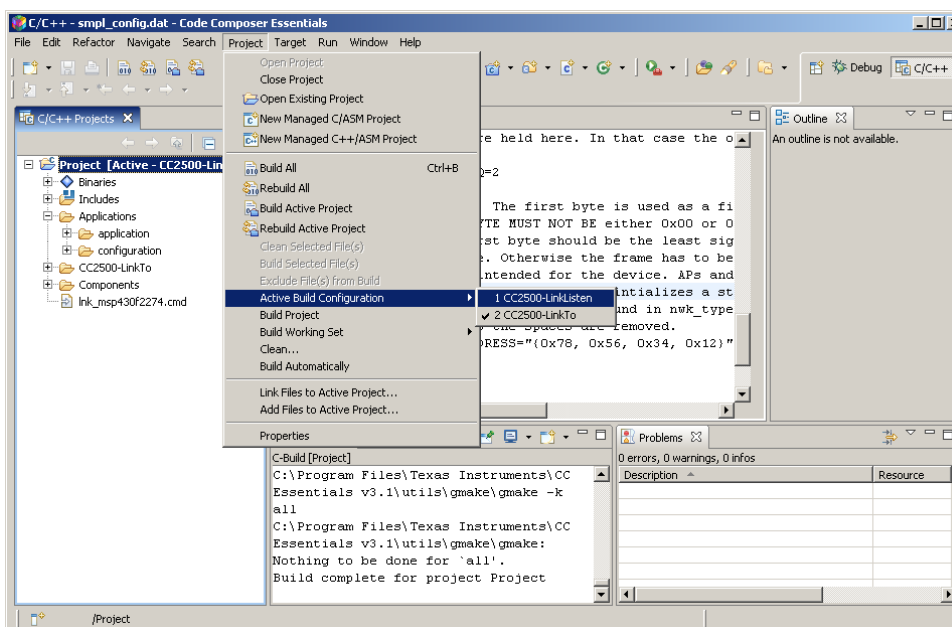
Figure 35: Changing the SimpliciTI Device Configuration

3. Sample Applications

This section introduces 4 simple sample applications to demonstrate various features and capabilities of SimpliciTI. Each example has an explanation what the application does, a detailed procedure for running the application, and a discussion of specific features of SimpliciTI that are used. These four sample applications are presented in order of increasing complexity:

- *Simple Peer-To-Peer* – two linked End-Devices communicate directly with each other
- *Polling with AP* – one End-Device sends data to another sleeping/polling End-Device via an Access Point
- *Cascading End Devices* – three un-linked End-Devices continually broadcast an alarm
- *Access Point as Data Hub* – two End-Devices send data to AP Data Hub, with Frequency Agility

All of the sample applications involve End-Devices sending data to other End-Devices or to an Access Point, using different topologies. For each application, there is a simple diagram that represents the topology, as supported by SimpliciTI. The following legend applies to the line formats in the topology diagrams:
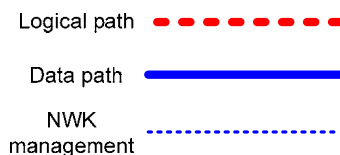
Figure 36: Legend for Topology Block Diagrams

Each of the sample applications require some form of user input, via two "logical" buttons and provide status display via two "logical" LEDs. SimpliciTI runs on various development kits, each having a unique combination of button and LED hardware on board. Most provide at least two button/switch inputs and at least two LEDs, but there are some exceptions (ez430RF and RFUSB). The mapping of physical to logical buttons and LEDs for each one of the development kit platforms is as follows:

Platform	Button1	Button2	LED1	LED2
CC2430DB	S1 Pushbutton	Joystick Push	Green (D1)	Red (D2)
EXP461x	S1 Pushbutton	S2 Pushbutton	Green (LED1)	Red (LED2)
eZ430RF	TS1 Pushbutton (see Note below)	None (see Note below)	Red (D1)	Green (D2)
RFUSB	S1 Pushbutton (see Note below)	None (see Note below)	Green (D2)	None
SRF04	S1 Pushbutton	Joystick Push	Green (LED1)	Yellow (LED3)
SRF05-8051	S1 Pushbutton	Joystick Push	Green (LED1)	Red (LED2)
SRF05-MSP	S1 Pushbutton	S2 Pushbutton	Green (LED1)	Red (LED2)

Table 6: Sample Application Buttons and LEDs

Note: These boards only support one button, which acts as Button1 except in the Polling with AP sample application (see Section 3.2).

3.1 Simple Peer-to-Peer

In this example, there are two End-Devices, a *Listener* and a *Talker*, that establish a direct peer-to-peer connection. Initially, the Listener (ED2 in the diagram below) waits for a link message, and the Talker (ED1) sends the link message. After a connection has been established, the Talker periodically sends a 2-byte message to the Listener, which then sends a 2-byte reply to the Talker. The connection is actually bi-directional but the initial connection negotiation assigns the roles of Talker and Listener.

Figure 37: Network Topology for Simple Peer-To-Peer

The observable functionality of this application is periodic toggling of an LED on each device. The Listener toggles an LED specified in messages received from the Talker. Then the Talker toggles an LED specified in replies received from the Listener. The Talker's message and the Listener's reply payload formats are identical:
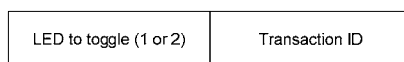
Figure 38: Message Payload for Simple Peer-To-Peer

Both devices in this example demonstrate use of the Rx Callback feature of SimpliciTI. In the Listener, incoming messages invoke the Rx Callback, where the specified LED is toggled and a semaphore gets posted to the main application before returning to the Rx ISR. In the Talker, incoming replies invoke the Rx Callback, which toggles the specified LED and returns. Note that execution of the Rx Callback is performed in the Rx ISR context – proper callback design dictates minimal code execution and return to the ISR as quickly as possible.

To run this sample application, follow these steps:

1. Using the process explained in Section 2, program and download a Talker (LinkTo) device.
2. Using the process explained in Section 2, program and download a Listener (LinkListen) device.
3. Power up both devices – LED1 and LED2 should be lit on both devices. Note that some platforms only have one LED (for example, CC1111 and CC2511 dongles).
4. Press a button on the Listener to listen for a link message - LED2 should turn on.
5. Press a button on the Talker to send a link message. Both LEDs should turn off if linking was successful. Both LEDs will blink if linking failed – if so, reset both devices and return to step 4.
6. At this point, the devices have established a connection and will continually perform the following actions:
    a. The Talker sends a message with a 2-byte payload to the Listener. The message indicates an LED for the Listener to toggle and a transaction ID. The transaction ID is incremented for each new message. It is treated as an unsigned number and wraps to zero after reaching the maximum value.
    b. The Listener receives the 2-byte message, immediately toggles the indicated LED, posts a semaphore to its main application, and returns to complete Rx interrupt processing.
    c. The main application on the Listener device eventually runs and detects that the semaphore has been posted. This allows it to send a 2-byte reply to the Talker, indicating an LED for the Talker to toggle and the received transaction ID.

d.   The Talker receives the 2-byte reply, immediately toggles the indicated LED, and returns.

e.   After a variable time interval on the Talker device, it returns to step (a) to do it all over again.

The following diagram illustrates the sequences followed by the Talker and the Listener devices:
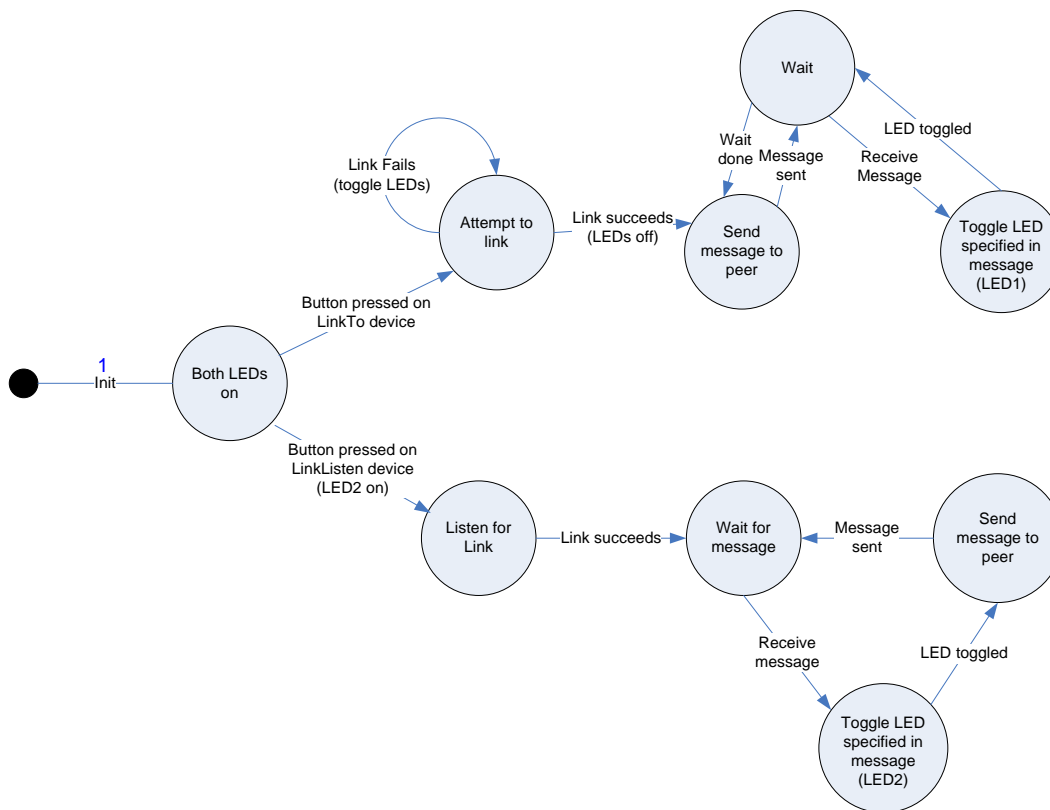
Figure 39: Sequence Diagram for Simple Peer-To-Peer

3.2 Polling with AP

In this example, two End-Devices, a *Receiver* and a *Sender*, establish a unidirectional peer-to-peer connection. The Receiver (ED2 in the diagram below) is a polling device, so the indirect connection to the Sender (ED2) requires an *Access Point* to support store-and-forward functionality. In this type of network, the Sender does not need to know or care that the Receiver is a polling device.

Initially, the Receiver joins the network and waits for a link message, and the Sender joins the network and sends a link message. After a connection has been established, the Sender periodically sends a 2-byte message to the Listener, which is stored by the Access Point, and later forwarded to the Receiver when it polls for a message.
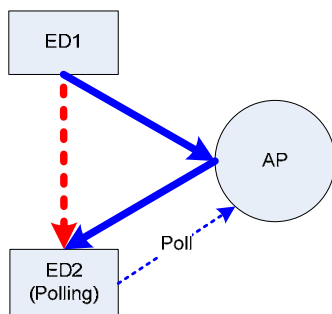

Figure 40: Network Topology for Simple Peer-To-Peer with Polling

The observable functionality of this application is periodic toggling of LEDs on the End Devices. The Sender toggles an LED each time it sends the 2-byte message. The Receiver toggles an LED specified in the message received from the Access Point. The Sender's message payload consists of an LED number and a transaction ID:
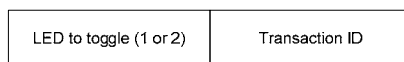
Figure 41: Message Payload for Simple Peer-To-Peer with Polling

To run this sample application, follow these steps:

1. Using the process explained in Section 2, program and download a Sender (LinkTo) device.
2. Using the process explained in Section 2, program and download a Receiver (LinkListen) device.
3. Using the process explained in Section 2, program and download an Access Point device.
4. Power up the Access Point. LED1 and LED2 should be on and not change while running the application. This Access Point will supply subsequent joining devices with the network's link token. In addition, it will determine which devices need store-and-forward support. It will then provide that support for any messages that are sent to the polling device.
5. Power up the Receiver. LED1 and LED2 should be on. Press button 2 to join the network and receive the link token from the Access Point. Upon receipt of this frame, the device can glean the address of the Access Point, which it will use later to send the polling requests. Only LED1 should be on, and the device now listens for a link message.
6. Power up the Sender. LED1 and LED2 should be on. Press button 1 to join the network, receive the link token from the Access Point, and send a link message to the Receiver. LED1 and LED2 should turn off at both the Sender and Receiver if linking is successful. Both LEDs will blink on the Sender if linking fails – if so, power down all three devices and return to step 4.
7. At this point, the devices have established a connection and will continually perform the following actions:

a. At 3-6 second intervals, the Sender sends a message with a 2-byte payload to the Receiver, and toggles its LED1. The message contains an LED number for the Receiver device to toggle and a transaction ID. The transaction ID is incremented for each new message. It is treated as an unsigned number and wraps to zero after reaching the maximum value. Every 8th message, the LED number is set to LED1, otherwise it is set to LED2.

b. At approximately 1 second intervals, the Receiver polls the Access Point. If the reply message from the Access Point has a payload, the Receiver checks the LED number and the transaction ID. If they both appear to be valid, the Receiver toggles the indicated LED.

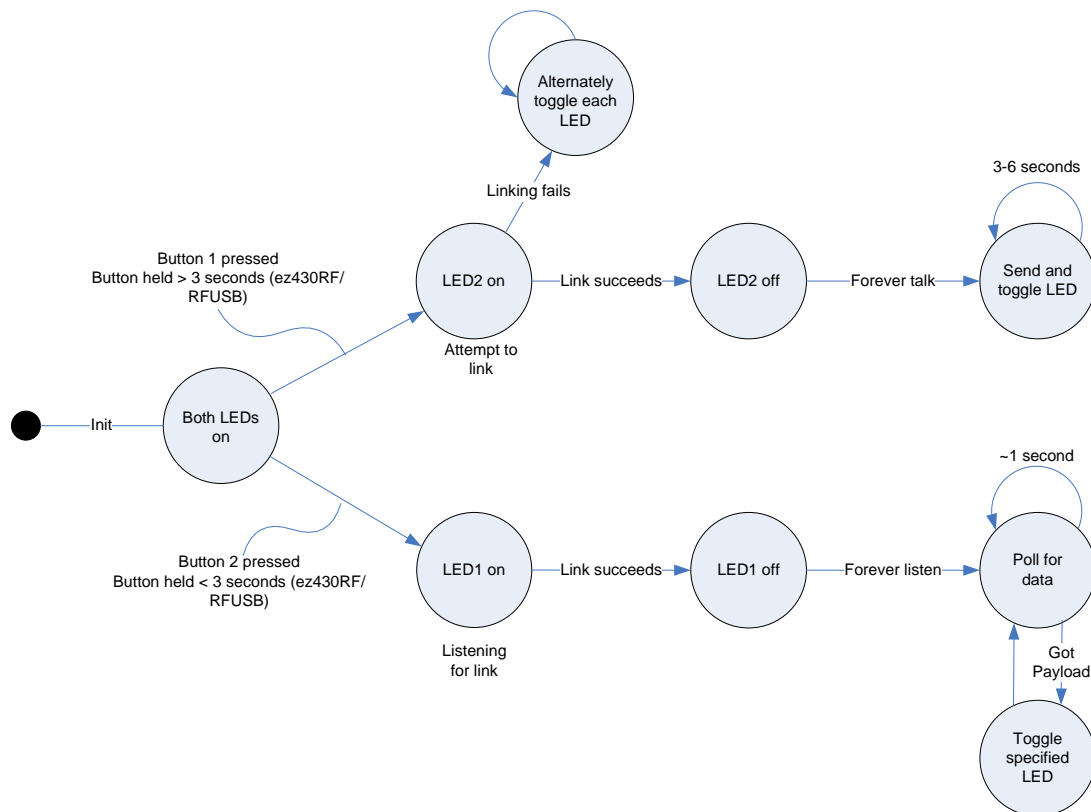The following diagram illustrates the sequences followed by the Sender and the Receiver devices:

Figure 42: Sequence Diagram for Simple Peer-To-Peer, with Polling

3.3 Cascading End Devices

In this example, there are two or more End Devices, forming a broadcast network in which explicit links are not established. Each device can talk to any other devices that are in radio range. The diagram below shows a network in which three End Devices can communicate with each of the others.

The idea is that messages will cascade through the network. For example, imagine an array of smoke alarms, each of which can communicate with at least one other device. When one detects a problem, it not only activates its alarm, but propagates the alert to other devices in the array, which also sound the alarm and continue to propagate the alert.
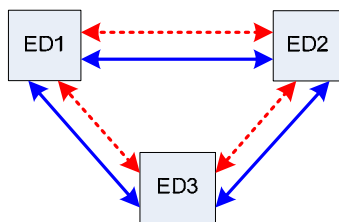

Figure 43: Network Topology for Cascading End Devices

The observable functionality of this application is periodic toggling of LEDs on each device to indicate current status. These devices emulate an array of smoke alarms or similar sensors that check their environment periodically. Initially, each device "sleeps" for a while (about 5 seconds) and then wakes up and checks its sensor (in the sample case, a button press). If the sensor is not activated, the device stays awake for a short time to see if it receives a message from another device. If no message is received, it toggles LED1 and goes back to sleep. If the device either detects that the sensor is activated or receives a "bad news" message, it sounds its alarm (toggles LED2) and then "babbles" a "bad news" message over air to alert other devices.

The key SimpliciTI feature that is demonstrated by this example is the operation of a broadcast network in which explicit links between devices are not established. The communication among End Device objects (applications) is via an unconnected datagram port mapped to a special Link ID. Each device may send to and receive from this Link ID without explicit linking as in the normal SimpliciTI scenario.

Connections between peers are not established. However, from the application perspective the send and receive API is the same. A special Link ID is used in place of a Link ID assigned by the stack during a normal SimpliciTI Link operation. Support for this unconnected user datagram capability consumes an entry in the Connection Table managed by the stack. This scenario does not prohibit either normal peer-to-peer connections or the presence and support of an Access Point. The Rx callback mechanism can also be used transparently.

In this sample application, devices "sleep" most of the time, waking up periodically to check for alert situations. In the sample code, sleep mode is actually implemented as a simple busy-wait loop, so this application is not suitable for power consumption evaluations. The user can substitute "real" sleep logic as needed.

To run this sample application, follow these steps:

1. Using the process outlined in Section 2, program and download an End-Device (ED1).
2. Using the process outlined in Section 2, program and download another End-Device (ED2).
3. If another device is available, using the process outlined in Section 2, program and download one more End-Device (ED3).
4. Power up all of the devices. LED1 and LED2 should be lit on all devices. Note that some platforms only have one LED (for example, CC1111 and CC2511 dongles).
5. Press a button on ED1. In response, its LEDs should turn off, indicating that the device has entered a cycle where it "sleeps" for about 5 seconds, "wakes up" to check for problems, and toggles LED1.

6. About 1 second later, press a button on ED2. Its LEDs should turn off, indicating that the device has entered a cycle where it "sleeps" for about 5 seconds, "wakes up" to check for problems, and toggles LED1.

7. If available ED3 is available, about 1 second later, press a button on it. Its LEDs should turn off, indicating that the device has entered a cycle where it "sleeps" for about 5 seconds, "wakes up" to check for problems, and toggles LED1.

8. In this state, each device periodically wakes up from "sleep" to check for a "problem":

    a. The device checks its sensor (in this example, a button press) to determine whether an alert situation has occurred. If so, go to step 10 to propagate the "bad news".

    b. The device turns on its radio for about ¼ second and checks for receipt of a "bad news" message. If a message is received with the "bad news" payload, go to step 10 to propagate the "bad news".

    c. The device toggles LED1 to indicate that no "problem" has been detected.

    d. The device "sleeps" for about 5 seconds.

    e. Return to step 8a to repeat the cycle.

9. After each device has performed several alert monitoring cycles in step 8, simulate a "problem" event by pressing and holding a button on one of the End Devices. Let go of the button when LED2 on that device begins to blink quickly (it has transitioned from step 8 to step 10). The other devices are expected to detect the "bad news" message, and also transition to step 10.

10. In this state, a device has encountered a "problem", so it continuously broadcasts the "bad news" message:

    a. The device sends a message with a 1-byte "bad news" payload via an unconnected datagram.

    b. The device toggles LED2 to indicate that a message has been sent.

    c. The device "sleeps" for about $1/10^{th}$ of a second.

    d. Return to step 10a to repeat the cycle.

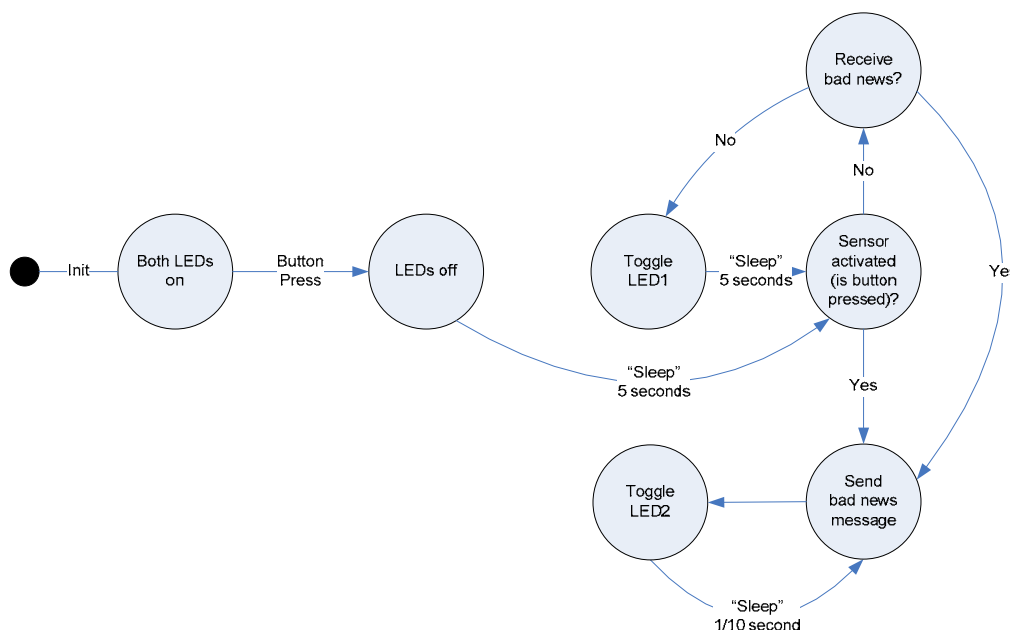The following diagram illustrates the sequence followed by all End Devices in this example:

Figure 44: Sequence Diagram for Cascading End Devices

3.4   Access Point as Data Hub

This example is a special case of a network with an Access point in which the peer for one or more End-Devices is on the Access Point. So, while each End-Device may have peers on other devices, it always has a peer on the Access Point. In this scenario the Access point may act as a data collector or gateway for information coming from and going to distributed sensors/actuators. In addition, this sample application can be used to demonstrate the Frequency Agility feature of SimpliciTI (selection of the number of channels and channel settings are available to the application developer and may be changed from the default settings to suit the application).

Initially, each End-Device joins the network and sends a link message to the Access Point. After a connection has been established, the Access Point will toggle a specified LED when a message is received from an End-Device (pressing button 1 toggles LED1 and pressing button 2 toggles LED2). In this example, two End-Devices (ED1 and ED2) are shown to control the LEDs on the Access Point. Note that this sample application is provided as a simple example of how an Access Point may be used. It is expected that the developer will make appropriate changes to utilize other design criteria that fit their specific application needs.
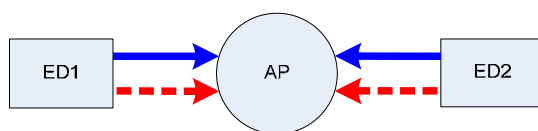
Figure 45: Network Topology for Access Point as Data Hub

The observable functionality of this application is toggling of LEDs on the Access Point, corresponding to button presses on the End-Devices. This implements a simulation of a multi-button RF remote control. The End-Device message payload consists of an LED number and a transaction ID:
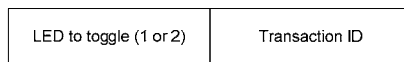
LED to toggle (1 or 2)	Transaction ID

Figure 46: Message Payload for Access Point as Data Hub

To run this sample application, follow these steps:
1. Using the process outlined in Section 2, program and download an End-Device (ED1).
2. Using the process outlined in Section 2, program and download another End-Device (ED2).
3. Using the process outlined in Section 2, program and download an Access Point device (AP).
4. Power up the Access Point. LED1 and LED2 should turn on. This Access Point will supply subsequent joining devices with the network's link token. In addition, it will listen for a link frame from a peer on each new joining End-Device. While running this application, LED1 and LED2 might start flashing together. This indicates that Frequency Agility noise detection caused an automatic channel change (see section 3.4.1). The LEDs continue to blink until a message is received from an End-Device on the new channel.
5. Power up the End-Devices (ED1 and ED2). LED1 and LED2 should flash once (indicating a successful join to the network) and then turn off (indicating a successful link with a peer on the Access Point).
6. Press button 1 on one of the End-Devices. LED1 on the Access Point should toggle, followed by LED1 toggling in this End-Device (indicates reply message received from the AP).
7. Press button 1 on the other End-Device. LED1 on the Access Point should toggle, followed by LED1 toggling in this End-Device (indicates reply message received from the AP).
8. Press button 2 on one of the End-Devices. LED2 on the Access Point should toggle, followed by LED1 toggling in this End-Device (indicates reply message received from the AP).

9. Press button 2 on the other End-Device. LED2 on the Access Point should toggle, followed by LED1 toggling in this End-Device (indicates reply message received from the AP).

10. Repeat steps 6 through 9 in other orders and verify expected toggling of LEDs. Since this example is using the Auto Acknowledge feature, the expectation is that LED1 will toggle on the End-Devices when the message ACK is received, otherwise LED2 will toggle.

11. Press button 1 on the Access Point. LED1 and LED2 on the AP should begin flashing, indicating that the Frequency Agility feature has executed a "forced" channel change (see section 3.4.1). When a button is pressed on an End-Device, LED1 and LED2 should stop blinking, indicating that the End-Device has found the new channel. Repeat steps 6 through 11 to test the Frequency Agility logic more times.

3.4.1 Frequency Agility

SimpliciTI has an optional feature, called Frequency Agility (refer to the *Application Note on SimpliciTI Frequency Agility Description* document), which can be used to increase a network's reliability by changing channels to avoid noise. This feature uses a small, fixed list of channels from which the devices can choose. The fixed channel set distributed with SimpliciTI varies depending on the radio (refer to the *SimpliciTI Channel Table Information* document). The size and content of this table can be changed. The Access Point can be stimulated to change channels in two ways, forced and automatic.

In this sample application, pressing button 1 on the Access Point will force it to move to the next channel in its list, wrapping when leaving the last logical channel. There is also an "auto-detect" algorithm implemented in this application, where the Access Point monitors the current channel for excess noise. If the channel is too noisy, the Access Point will change channels without user intervention. The channel monitoring algorithm is shown below. NOTE: The algorithm below is sensitive to the data rate used. If the data rate is too low the noise criterion shown below will actually be met by a valid frame sent by any device and the AP will change channels. It is recommended that for demonstration purposes the default data rate be used.
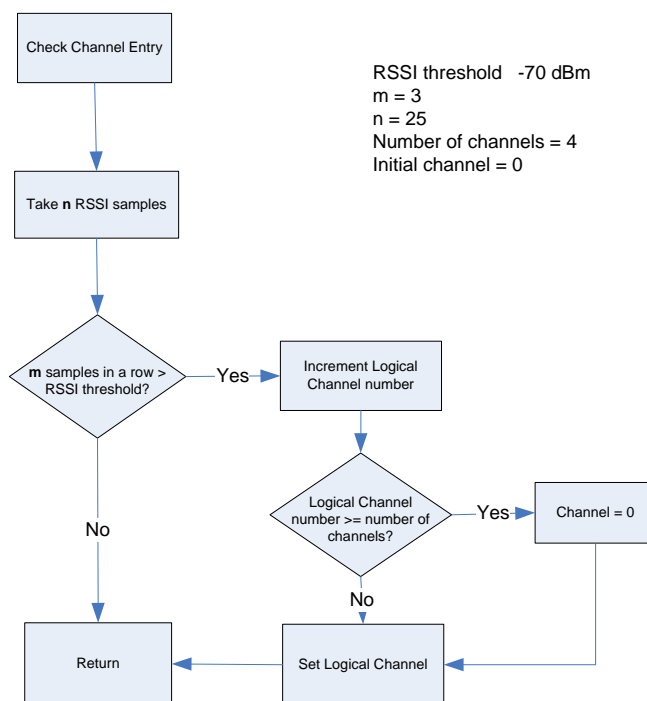

Figure 47: Frequency Agility Channel Change Algorithm

3.4.2 Channel Sniffer

When using Frequency Agility, it is useful to know what channel the network is currently on. Included with this sample application, is a project to build a channel Sniffer that will continually scan the channel list and display the logical channel number on its LEDs:

Logical Channel	LED1	LED2
0	Off	Off
1	Off	On
2	On	Off
3	On	On

Table 7: Channel Indication LEDs on a Sniffer Device

The Channel Sniffer is actually an End-Device with no peer application – it only scans for the active channel designated by the Access Point. The key configuration item (in **smpl_config.dat**) is to set the `NUM_CONNECTIONS` macro to **0**. The AP will not listen for a link frame if the joining device does not support a peer application. Normally this is true of Range Extenders but the scenario is also true for this sniffer device.

To run the Channel Sniffer device with this sample application, follow these steps:
1. Using the process outlined in Section 2, program and download a Sniffer device.
2. Run at least steps 1 through 4 of the process described in section 3.4 above.
3. Power up the Sniffer. LED1 and LED2 should toggle alternately for about 5 seconds while scanning the channels. When the toggling stops, the LEDs indicate the current channel, according to the table above.
4. Run steps 5 through 11 of the process described in section 3.4 above to demonstrate channel changes.