Introduction

SimpliciTI is a Texas Instruments proprietary solution to easily add low cost, low power wireless capability to microcontroller applications. In this module, we’ll learn about and experiment with the protocol.

Learning Objectives

In the module we’ll be covering:

- Objects and Topologies
- SimpliciTI architecture
- Threading and Callback
- Configuration
- AP as Data hub
- Broadcast messages
- Frequency Agility
- Current Consumption
- Upcoming Features
*** ThisPageIntentionallyLeftBlank.com (no kidding) ***
Module Topics

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpliciTI</td>
<td>5-1</td>
</tr>
<tr>
<td>Module Topics</td>
<td>5-3</td>
</tr>
<tr>
<td>SimpliciTI</td>
<td>5-5</td>
</tr>
<tr>
<td>SimpliciTI Objects and Topologies</td>
<td>5-6</td>
</tr>
<tr>
<td>Example Application and Comparison</td>
<td>5-7</td>
</tr>
<tr>
<td>Architecture and Details</td>
<td>5-8</td>
</tr>
<tr>
<td>APIs, Threading and CallBack</td>
<td>5-9</td>
</tr>
<tr>
<td>Configuration Files</td>
<td>5-10</td>
</tr>
<tr>
<td>Configuration</td>
<td>5-11</td>
</tr>
<tr>
<td>Frame Format and Radio Configuration</td>
<td>5-12</td>
</tr>
<tr>
<td>Tokens and Packet Sniffer</td>
<td>5-13</td>
</tr>
<tr>
<td>Hardware and BSP</td>
<td>5-14</td>
</tr>
<tr>
<td>Lab5a – Two Device Peer to Peer Network</td>
<td>5-17</td>
</tr>
<tr>
<td>Description</td>
<td>5-17</td>
</tr>
<tr>
<td>Hardware list</td>
<td>5-18</td>
</tr>
<tr>
<td>Software list</td>
<td>5-18</td>
</tr>
<tr>
<td>Procedure</td>
<td>5-19</td>
</tr>
<tr>
<td>Data Hub and Multiple Links</td>
<td>5-29</td>
</tr>
<tr>
<td>Storing, Forwarding and Polling</td>
<td>5-30</td>
</tr>
<tr>
<td>App Level Acks and Broadcast Messages</td>
<td>5-31</td>
</tr>
<tr>
<td>Lab5b – AP as Data Hub Network</td>
<td>5-33</td>
</tr>
<tr>
<td>Description</td>
<td>5-33</td>
</tr>
<tr>
<td>Hardware list</td>
<td>5-34</td>
</tr>
<tr>
<td>Software list</td>
<td>5-34</td>
</tr>
<tr>
<td>Procedure</td>
<td>5-35</td>
</tr>
<tr>
<td>Frequency Agility and Current Consumption</td>
<td>5-41</td>
</tr>
<tr>
<td>Upcoming Features and Demo</td>
<td>5-42</td>
</tr>
<tr>
<td>Lab5c – Adding SimpliciTI to an Existing Application</td>
<td>5-43</td>
</tr>
<tr>
<td>Description</td>
<td>5-43</td>
</tr>
<tr>
<td>Hardware list</td>
<td>5-44</td>
</tr>
<tr>
<td>Software list</td>
<td>5-44</td>
</tr>
<tr>
<td>Procedure</td>
<td>5-45</td>
</tr>
</tbody>
</table>
*** a page is a terrible thing to waste ***
SimpliciTI

**SimpliciTI is ...**

- **Low Power:** Supports sleeping devices for low power consumption
- **Low Cost:** uses < 8K FLASH and < 1K RAM depending on the platform
- **Flexible:** Simple star with extender and/or peer to peer communication
- **Simple:** Utilizes 6 very basic core instructions
- **Versatile:** MSP430+CC1100/2500, CC111x/251x, CC2430/31 and CC2530(SoC)

SimpliciTI targets quick time-to-market wireless solutions for low power, low cost, and low data rate networks without the need to know the details of the network support.

---

**Example Applications**

- **Alarm & Security:** Occupancy sensors, light sensors, carbon monoxide sensors, glass-breakage detectors
- **Smoke Detectors**
- **Automatic Meter Reading (AMR):** Gas meters, water meters, electrical meters
- **Home Automation:** Garage door openers, appliances, environmental devices

---

Objects ...
SimpliciTI Objects and Topologies

SimpliciTI Objects

- **Access Point (AP) (ex. HVAC central)**
  - Acts as Star hub
  - One AP per network allowed
  - Promiscuous mode
  - Always on
- **Repeater (RE) (ex. Lamp)**
  - Range Extender
  - 4 RE per network recommended
  - Always on
  - Can not talk to other RE’s
  - Promiscuous mode
- **Device (D) (ex. Temperature sensor)**
  - Can sleep (SD)
  - Can transmit only (TD)
- **Multiple peer to peer links can exist on a single hardware platform**

Network Topologies – Star and P2P

- **Direct peer to peer**
- **Direct peer to peer through RE**
- **Store and forward peer to peer through AP**
- **Store and forward peer to peer through RE and AP**

Logical path
Data path
Network Mgmt
Example Application and Comparison

Wireless Sensing Application

Range can be extended through repeaters. The circles represent the RF range of the gateways and the extended RF range of repeaters.

SimpliciTI vs. Zigbee

<table>
<thead>
<tr>
<th>Network properties</th>
<th>SimpliciTI</th>
<th>ZigBee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mesh network</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Typical number of nodes</td>
<td>from 2 to ~30</td>
<td>from 2 to hundreds</td>
</tr>
<tr>
<td>Peer to Peer and Star network</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Hardware and software</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td>Any MSP430 + ChipCon TRX or 8051 SoC</td>
<td>MSP430F2418 + CC2420, CC2430</td>
</tr>
<tr>
<td>Frequency &amp; modulation</td>
<td>Any TI radio: &lt;1GHz, 2.4GHz, standard or proprietary</td>
<td>IEEE 802.15.4 DSSS, 2.4GHz</td>
</tr>
<tr>
<td>S/W object distribution</td>
<td>Free download</td>
<td>Free download</td>
</tr>
<tr>
<td>S/W source code</td>
<td>Free download</td>
<td>Not required for development.</td>
</tr>
<tr>
<td>Compiled code size on MSP430</td>
<td>~4k depending on configuration</td>
<td>50-60k depending on configuration</td>
</tr>
<tr>
<td>Interoperability between vendors</td>
<td>No</td>
<td>Optional</td>
</tr>
<tr>
<td>Encryption</td>
<td>Yes, 128bit AES on enabled HW devices, other in software.</td>
<td>Yes, 128bit AES</td>
</tr>
</tbody>
</table>
A SimpliciTI network address consists of the device address (set at compile time) + the assigned port.

- Utilizes a single MCU timer in software. SPI and I/O pins talk to radio (non SoC targets)
- Operates on 802.15.4 target (CC2430) or non-802.15.4 capable targets (CC251x/CC2500/CC111x/CC1100)
- Minimal HW abstraction ... no driver support (UART, SPI, LCD, Timers)
- No heap utilization (runtime allocation of memory)
- No runtime (nwk) context storage
- Single thread (app), no tasks or scheduling
- Network callback – app can provide callback
- Retries and acknowledges must be managed by the application
- Utilizes CCA (Clear Channel Assessment) “listen-before-talk” methodology for transmit

Details...

APIs...
APIs, Threading and CallBack

6 Simple APIs

- Initialization
  - `smplStatus_t SMPL_Init(uint8 (*pCB)(linkID));`

- Linking (bi-directional by default)
  - `smplStatus_t SMPL_Link(linkID_t *linkID);`
  - `smplStatus_t SMPL_LinkListen(linkID_t *linkID);`

- Peer-to-peer messaging
  - `smplStatus_t SMPL_Send(linkID_t lid, *msg, uint8 len);`
  - `smplStatus_t SMPL_Receive(linkID_t lid, *msg, *uint8 len);`

- Run time configuration
  - `smplStatus_t SMPL_Ioctl(Object_t object, ioctlAction_t action, void *val);`

Threading Model and Callback

- SimpliciTI is designed to operate along with your application and does not require its own context.

- Requires no access to a scheduler or OS (although its use isn’t precluded).

- You may register for a callback in the initialization call.

- This callback is invoked in the receive ISR thread when a valid application destination address has been received.

`smplStatus_t SMPL_Init(uint8 (*pCB)(linkID))`

Pointer to your received frame handler

A CallBack allows lower level code, like an ISR, to call upper level code, like your receive application. You must register for a callback.
Two Configuration Files

- **smpl_config.dat**
  - General network settings
  - Maximum number of hops
  - Maximum payload size
  - Link and Join tokens

- **smpl_nwk_config.dat**
  - Settings for each device
  - Queue sizes
  - Device address
  - etc

Build Time Configurations - General

<table>
<thead>
<tr>
<th>Item</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_HOPS</td>
<td>3</td>
<td>Maximum number of times a frame is re-sent before the frame is dropped. Each RE and the AP decrement the hop count before re-sending the frame.</td>
</tr>
<tr>
<td>MAX_HOPS_FROM_AP</td>
<td>1</td>
<td>Maximum distance an polling ED can be from the AP. To reduce broadcast storm.</td>
</tr>
<tr>
<td>NUM_CONNECTIONS</td>
<td>4</td>
<td>Number of links supported (Both Link and LinkListen). Should be 0 if the device supports no ED objects (APs or REs).</td>
</tr>
<tr>
<td>MAX_APP_PAYLOAD</td>
<td>10</td>
<td>Maximum number of bytes in the application payload.</td>
</tr>
<tr>
<td>SIZE_INFRAME_Q</td>
<td>2</td>
<td>Number of frames held in the RX frame queue. Can be 0 for Tx-only devices, or for devices that never receive frames.</td>
</tr>
<tr>
<td>SIZE_OUTFRAME_Q</td>
<td>2</td>
<td>Number of frames held in the TX frame queue. Some NWK applications keep TX frame around to find correct replies.</td>
</tr>
<tr>
<td>DEFAULT_JOIN_TOKEN</td>
<td>0x01020304</td>
<td>Joining a network requires this value to match on all devices (D, SD, RE, and AP).</td>
</tr>
<tr>
<td>DEFAULT_LINK_TOKEN</td>
<td>0x00080708</td>
<td>Obtaining a link access to a network device requires this value to match on all devices.</td>
</tr>
<tr>
<td>THIS_DEVICE_ADDRESS</td>
<td>0x12345678</td>
<td>Each device address should be unique.</td>
</tr>
</tbody>
</table>

Most build time configuration parameters will affect the memory requirements and should be kept as low as possible.
Configuration

Build Time Configurations – Device Specific

<table>
<thead>
<tr>
<th>Access Point Devices</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS_POINT</td>
<td>Defined</td>
<td></td>
</tr>
<tr>
<td>NUM_STORE_AND_FWD_CLIENTS</td>
<td>10</td>
<td>Number of polling End Devices supported.</td>
</tr>
<tr>
<td>AP_IS_DATA_HUB</td>
<td>Not Defined</td>
<td>If this macro is defined the AP automatically listens for a link message from each distinct device that joins and supports End Device objects. The ED joining must link immediately after it receives the Join reply.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Range Extender Devices</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RANGE_EXTENDER</td>
<td>Defined</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>End Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>END_DEVICE</td>
</tr>
<tr>
<td>RX_POLLS</td>
</tr>
</tbody>
</table>

Runtime Configuration

- Application access to frame header
- Application access to radio controls
- Access Point network management control

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOCTL_OBJ_FREQ</td>
<td>Get/Set radio frequency</td>
<td>Frequency agility. May be used by APP or NWK.</td>
</tr>
<tr>
<td>IOCTL_OBJ_CRYPTKEY</td>
<td>Set encryption key</td>
<td>Customer may provide external means for user to set a non-default key. Requires reset to take effect.</td>
</tr>
<tr>
<td>IOCTL_OBJ_RAW_ID</td>
<td>Application layer access to the frame header to directly send or receive a frame.</td>
<td>This object is used for example to ping another device where the network address of the target device is supplied directly and not done through the connection table.</td>
</tr>
<tr>
<td>IOCTL_OBJ_RADIO</td>
<td>Application layer access to some radio controls.</td>
<td>Limited access to radio directly. For example, sleeping and waking the radio and getting signal strength information.</td>
</tr>
<tr>
<td>IOCTL_OBJ_AP_JOIN</td>
<td>Access Point join-allow context</td>
<td>Interface to control whether Access Point will allow devices to join or not.</td>
</tr>
<tr>
<td>IOCTL_OBJ_ADDR</td>
<td>Get/Set device address</td>
<td>Permits run-time address generation for a device. Set function must be done before the SMPL_Init() call.</td>
</tr>
<tr>
<td>IOCTL_OBJ_CONNOBJ</td>
<td>Connection object</td>
<td>Delete a connection entry. Accessed with LinkID. Affects this device only – does not gracefully tear down connection.</td>
</tr>
<tr>
<td>IOCTL_OBJ_FWVER</td>
<td>Firmware and protocol versions.</td>
<td>Get only. FW version a byte array of length SMPL_FWVERSION_SIZE. Protocol version a uint8_t</td>
</tr>
</tbody>
</table>

Frame format ...
Frame Format and Radio Configuration

**SimpliciTI Frame Format**

<table>
<thead>
<tr>
<th>Field</th>
<th>Definition</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREAMBLE</td>
<td>Radio synchronization</td>
<td>Inserted by Radio HW</td>
</tr>
<tr>
<td>SYNC</td>
<td>Radio synchronization</td>
<td>Inserted by Radio HW</td>
</tr>
<tr>
<td>LENGTH</td>
<td>Length of remaining frame in bytes</td>
<td>Inserted by FW on Tx, Partially filterable on Rx.</td>
</tr>
<tr>
<td>MISC</td>
<td>Radio dependent (needed for future IEEE radio support)</td>
<td>Current set to 0.</td>
</tr>
<tr>
<td>DSTADDR</td>
<td>Destination address</td>
<td>Inserted by FW. LSB filterable. 0x00 and 0xFF LSB values reserved for broadcast. LSB:MSB formatted.</td>
</tr>
<tr>
<td>SRCADDR</td>
<td>Source address</td>
<td>Inserted by FW.</td>
</tr>
<tr>
<td>PORT</td>
<td>Application port number (bits 5-0)</td>
<td>Inserted by FW. Port 0x20,0x21 for customer applications, Port 0x00,0x01 for NWK applications.</td>
</tr>
<tr>
<td>DEVICE INFO</td>
<td>Receiver type (bit 7-6), Sender Type (5-4) &amp; Hop count (2-0)</td>
<td>Inserted by FW.</td>
</tr>
<tr>
<td>TRACTID</td>
<td>Transaction ID</td>
<td>Inserted by FW. Discipline depends on context.</td>
</tr>
<tr>
<td>APP PAYLOAD</td>
<td>Application data</td>
<td>0 ≤ n ≤ 52 (50 if FCS)</td>
</tr>
<tr>
<td>FCS</td>
<td>Radio append bytes</td>
<td>CRC checksum (Tx), RSSI, LQI and CRC status (Rx)</td>
</tr>
</tbody>
</table>

* RD = Radio-Dependent. Populate by MRFI or handled by the radio hardware.

**Radio Configuration**

- SmartRF Studio takes your settings and creates code for the project ...

- It can also create a chip-specific configuration file for the Packet Sniffer
Tokens and Packet Sniffer

Join & Link Tokens

- In networks with an AP, only devices with a matching Join token can join the network.
- Devices not matching the join token fail the join request. Their default link token (defined at build) will be used.
- Only devices with identical Link tokens can link together. In networks with an AP, the link token is distributed by the AP.

Networks with no AP do not join, they link only.

TI Packet Sniffer

- Sniffer hardware = SmartRF04EB + CC2510EM
- Displays all OTA messages within reception range of Sniffer hardware
- Filtering and recording ability
Hardware and BSP

SimpliciTI eZ430-RF2500 Hardware

$49

USB 2.0

- Extra target boards available at www.ti.com/ez430-rf
- Workshop hardware = 2 eZ430-RF2500 kits/workstation

Target Board

MSP430F2274

- 16-bit, 16MHz RISC processor
- 32KB Flash, 1KB RAM
- Two 16-bit timers
- USCI (UART/IrDA/SPI/I2C)
- 10-bit, 200ksps ADC
- Two configurable Op-Amps

CC2500

- 2.4GHz transceiver
- Separate 64-byte RX & TX FIFOs
- SPI interface
- 13.3mA current consumption in RX
- OOK, 2-FSK, GFSK & MSK
- -104dBm sensitivity at 2.4kBaud
Simple Board Support Library

Minimal hardware abstraction:
- LEDs (on, off, toggle)
- Buttons (read)
- Init (port setup)
- MRFI (radio interface)

Alter BSP files \( (\text{Components}\backslash\text{bsp}) \) for your target system

Lab time ...
*** The mind can only absorb what the bottom can endure ***
Lab5a – Two Device Peer to Peer Network

Description:

This lab utilizes one of the sample applications that is delivered with the SimpliciTI download. This network is a two device peer to peer application, with both devices implemented as SimpliciTI End Devices. One device will act as a talker and the other as a listener, but the designation is arbitrary, since the link is bi-directional. The talker sends a 2-byte payload every 1 to 4 seconds containing the LED to toggle and a incremented transaction ID.

The listener waits for a message and, using the Receive callback feature, handles the message and toggles the specified LED.

Use SmartRF Studio to program the base frequency and channel spacing
Browse the code
Program your assigned channel number
Build/Load/Run
Observe Sniffer

<table>
<thead>
<tr>
<th>Group #</th>
<th>Channel #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>10 or 0x0A</td>
</tr>
<tr>
<td>7</td>
<td>12 or 0x0C</td>
</tr>
</tbody>
</table>

Base frequency: 2.47 GHz
Channel Spacing: 400 kHz

AP as Data Hub...
Lab5a – Two Device Peer to Peer Network

Hardware list:

✓ 2 eZ430-RF2500 Target Boards
✓ 1 Battery Module
✓ 2 AAA Batteries
✓ 1 eZ430-RF2500 Emulator Board
✓ 1 SmartRF04EB Board (firmware revision 28 or later)
✓ 1 CC2510EM Board
✓ 1 Antenna
✓ 1 USB A/B Cable
✓ 1 USB Extender Cable

Software list:

✓ IAR Embedded Workbench for MSP430 version 4.11D
✓ TI Packet Sniffer version 2.10.1
✓ SmartRF Studio version 6.10.2.0

(You will find shortcuts for the above applications on the desktop)

✓ SimpliciTI version 1.0.6
**Procedure**

**Select the Frequency and Channel**

To make sure that each workstation is using a different frequency (and prevent a ton of confusion), we need to set up the configuration of the radio. SimpliciTI uses SmartRF Studio to export a configuration file into a format which the software will understand. There is also a structure (mrfiLogicalChanTable) where the channel is changed at run-time. We’ll utilize both methods to set the appropriate frequency.

SimpliciTI can utilize either 801.15.4 or non-802.15.4 radios for RF communication. On the non-802.15.4 radios that we’ll be using in this lab, we can specify channels anywhere within the ability of the radio to tune. Channel spacing can be as close as 25KHz or as wide as 400KHz. We will, though, want to avoid active Wi-Fi channels. For more information, look at SimpliciTI Channel Table Information.pdf in C:\Texas Instruments\SimpliciTI-1.0.6\Documents.

With this in mind, we’ll set our CC2500 radio to operate at a base frequency of 2470MHz with 400KHz channel spacing. 2470MHz is well above Wi-Fi channel 11.

1. **Open SmartRF Studio and Configuration File**

   Open SmartRF Studio, select the SmartRF04DK tab and then select Calculation Window – CC2500. Click Start.

2. **Open Existing File and Select Preferred Settings**

   From the menu bar, select File → Open configuration. Navigate to C:\Texas Instruments\SimpliciTI-1.0.6\Components\mrfi\smartrf\CC2500 and open the rfstudio.srfs2500 file.

   Look in the large Preferred settings box below and select the line that looks like:

   ![Preferred settings table](image)

   Why are we selecting these preferred settings? You’ll see in a moment.
3. **Set the Base Frequency and Channel Spacing**

Find the RF frequency box, enter 2470 into it and press **Enter**. Then enter the channel spacing of 400 in the **Channel** box. An information box will pop up and inform you that the closest value is 399.902344. That’s a calculated value from the closest register setting and is fine. The other selections were already made from the existing configuration file. Press **Enter**.

4. **Save Configuration**

Your instructor assigned each workgroup a **channel number** for this lab. You might think that the Channel number box is the spot to put this, but you’d be **wrong**. The channel number is set in software, and we’ll do that in a moment. Save your work by selecting **File ➔ Save configuration** from the menu bar, then select rfstudio.srfs2500 and **Save**. If you’re prompted to replace the file, click **Yes**.

5. **Export Settings for SimpliciTI**

Export the new settings to code that will be included by SimpliciTI, choose **File ➔ Export CC2500 Code**. Double click **SimpliciTI settings** in the bottom list of export targets and click the **Write to file** button. Note that you need to use the default filename that appears, unless you change your IAR project options. Click **Save** and **Yes** to replace.

6. **Export Settings for the Packet Sniffer**

In order for the TI Packet Sniffer to tune in to your message traffic, it needs to operate with the same settings as your Tx/Rx radios. If you were to export those settings at this point, they would contain all the settings needed for a CC2500 radio. Unfortunately, the sniffer board uses a CC2510, so we’ll have to re-make the settings starting with the chip selection. Why can’t you use a CC2500 as a sniffer? Because it does not have an on board microcontroller.

Close the Export and Calculation windows, and in the initial SmartRF Studio window, select **Calculation Window – CC2510** and click **Start**. Select the **Preferred settings** shown below:

<table>
<thead>
<tr>
<th>Preferred settings:</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Datarate</td>
<td>Deviation</td>
<td>Modulation</td>
<td>RX filter bandwidth</td>
<td>Optimization</td>
</tr>
<tr>
<td>2.4 kBaud</td>
<td>38 kHz</td>
<td>2-FSK</td>
<td>203 kHz</td>
<td>Sensitivity</td>
</tr>
<tr>
<td>2.4 kBaud</td>
<td>38 kHz</td>
<td>2-FSK</td>
<td>203 kHz</td>
<td>Current</td>
</tr>
<tr>
<td>10 kBaud</td>
<td>38 kHz</td>
<td>2-FSK</td>
<td>232 kHz</td>
<td>Sensitivity</td>
</tr>
<tr>
<td>10 kBaud</td>
<td>38 kHz</td>
<td>2-FSK</td>
<td>232 kHz</td>
<td>Current</td>
</tr>
</tbody>
</table>

You may notice that this setting is one of few that matches those for the CC2500, which is why we selected it before. Like before, find the RF frequency box, enter 2470 into it and press **Enter**. Then enter the channel spacing of 400 in the **Channel** box and press **Enter**.
From the menu bar, choose **File → Export CC2510 Code**. Double click on **Packet sniffer settings** in the list of export targets, click **Write to file**, name the file **MyPktSnfSettings.prs** and click **Save**. Make sure that you save the file to the **C:\Texas Instruments\SimpliciTI-1.0.6\Components\mrfi\smartrf\CC2510** folder. **Close** all of the SmartRF Studio windows.

**Install and Checkout**

7. **Install the Hardware**

Plug the USB Extension Cable into an open USB port on your PC and then insert the Emulator/Target board into the other end of it. If the Emulator and Target Boards aren’t already connected, note that the chips on the target board should be on the same side as the emulator connector when the two boards are connected. If the board gets hot, it’s backwards! It’s easy to misalign the Emulator/Target board connector. Be careful, don’t force it or you’ll break it! You should hear (ba-bump) when Windows recognizes the board.

8. **Open IAR Workbench and the Project**

Open IAR Embedded Workbench for the MSP430 by clicking on the shortcut on your desktop. When the **StartUp** window appears, click on **Open existing workspace**.

Navigate to: **C:\Texas Instruments\SimpliciTI-1.0.6\Projects\Examples\Peer applications\eZ430RF\2 simple End Devices with bi-di**, select **SimpliciTI eZ P2P.eww** and click **Open**. If you see a pop-up telling you that the project needs to be converted to the new format, click **Yes**.

9. **Create Project Configurations**

In the **Workspace** window, click on the + next to **peer applications** and again on the + next to **application**. The **main_LinkTo.c** file supports the **Talker** and the **main_LinkListen.c** file supports the **Listener**. We could manually include the proper file at build time, but chance are, we’d make a mistake. Let create two build configurations to include the correct file for us.

On the menu bar, click **Project → Edit Configurations** and click the **New** button. Type **Talker** into the **Name** window and click **OK**. Then, create another configuration called **Listener**. Click **OK**.
10. **Include/Un-include Source Files**

Both the **Talker** and **Listener** configurations currently have the same build source files as the original configuration. That is, `main_LinkTo.c` is part of the build. That’s OK for the **Talker**, but not the **Listener**.

In the Workspace Configuration pull-down menu, select Listener (like below):

![Workspace Configuration](image)

Right-click on `main_LinkTo.c` → **Options** and check the **Exclude from build** checkbox. Click **OK**.

![Options Configuration](image)

Right-click on `main_LinkListen.c` → **Options** and uncheck the **Exclude from build** checkbox. Click **OK**.

It’s that simple … now you can easily switch between the two builds.
11. **Select your Channel**

Click **Open** from the menu bar and navigate to: \Texas Instruments\SimpliciTI-1.0.6\Components\mrfi\radios\common. Open **mrfi_f1f2.c** for editing (This file isn’t part of the project, it’s referenced in another file) About 1/3 of the way down in the file you’ll find the following lines (make sure you’re not looking at the CC1100 table):

```c
#if defined( MRFI_CC2500 ) || defined( MRFI_CC2510 ) || defined( MRFI_CC2511 )
static const uint8_t mrfiLogicalChanTable[] =
{
    3,
    103,
    202,
    212
};
```

This table defines the available channels when Frequency Agility is enabled, but the first entry defines the operating channel when it is not. Replace **3** with the **assigned channel** number for your group, then save and close the file.

12. **Check out the Code**

Take a moment to open **main_LinkTo.c** and **main_LinkListen.c** and check out the code. Double-click on the filename to open the file in the editor window. Note the logical code dealing with the buttons and LEDs, but concentrate on the SimpliciTI calls … there aren’t many of them!

13. **Build/Load the Listener**

Select the **Listener** configuration. On the menu bar, click **Project ➔ Clean**. and then click the **Debug** button. If IAR is properly configured (and it should be) the code will build with no errors and download to the target board. Click the **Stop Debugging** button on the menu bar. **Remove** the target board from the Emulator and **connect** it (carefully) to the **battery module** (chips facing up). The battery module has a jumper on it to disconnect power; remove the jumper and **replace** it over one of the pins for safekeeping. If you’re forgetful, you can use a Post-It™ note to label the board as **Listener 79**.
14. A New Address for the Talker

Every SimpliciTI device requires its own unique address, and if we just rebuild as the **Talker**, both devices will have the same address. Change the configuration to **Talker**, and open the `smpl_config.dat` source file for editing from the **Workspace, peer applications → Configuration** folder.

Near the bottom of the file, you’ll find the line with:

```
-DTHIS_DEVICE_ADDRESS="\{0x79, 0x56, 0x34, 0x12}\"
```

change it to:

```
-DTHIS_DEVICE_ADDRESS="\{0x80, 0x56, 0x34, 0x12}\"
```

15. Build/Load the Talker

*Connect* a target board to the Emulator, *label* it **Talker 80** if you like and click the **Debug** button on the menu bar. Don’t run the software yet, but you can shut down **IAR Embedded Workbench** and save anything necessary.

Here’s a diagram of the program flow, but we’ll walk through it momentarily:
16. **Set up the Sniffer Hardware**

Make sure the **CC2510EM** board is connected properly to the top of the **SmartRF04EB** board and that the antenna is connected to the top of the **EM** board. Check the two jumpers on P3 (middle of the board, near the bottom) and assure that they are oriented vertically.

Using a USB A/B cable, **plug** the assembly into an open USB port on the PC and **switch** the **EB board** **on** (move the switch towards the **EM** board).

If necessary, install the driver from the location listed earlier in the workshop.

17. **Start the TI Sniffer**

Using the desktop shortcut, **start** the TI Sniffer software. Select the Protocol as **SimpliciTI (CC2510 or CC1110)** and click **Start**. In the **Connected** devices box, you should see your sniffer hardware. If you don’t see it, it’s not connected or not powered.

Click the **Radio Setting** tab, then the **Browse** button and navigate to **C:\Texas Instruments\SimpliciTI-1.0.6\Components\mrfi\smartrf\CC2510\MyPktSnfSettings.prs**. This is the file you exported from SmartRF Studio earlier.

Click the **Setup** tab and enter your workgroup **channel** into the **Select channel** box:

Select **SimplicTI v1.0.6** in the menu bar pull-down:

and then click **Start**.
If you see the following on your sniffer display …

![Sniffer Display](image)

… your P3 jumpers are probably placed incorrectly. Turn the board off and make sure they are positioned as shown in lab step 16, then switch the board on.

18. **Start your Network**

Remove/Replace your Talker 80 target board into the emulator connector. This will cycle power on the board and start the software you downloaded. Both LED’s on the board should light and you should see a Join message on the Sniffer display. If you see more than one message, you probably had too much coffee and your hands were shaking when you inserted the board.

Connect the jumper on the battery module of your Listener 79 target board. Both LED’s on the board should light and you should see another Join message on the Sniffer display.

19. **Linking**

Initiate linking by pressing the button on the Talker 80 board. The red LED will turn off and the board is now ready to hear the Link message from the Listener. Press the button on the Listener 79 board. The green LED will turn off and you should see Link and Link Reply messages on the sniffer. We could have done this process the other way, but there would have been more messages on the sniffer as the listener waited for us to press the talker button.

After this, the talker will send a 2-byte message to the listener every 4 seconds or so, and the listener will send it back. Note the source and destination addresses as well as the incrementing transaction value in the Application Payload.
Note that the **Time Stamp us** field has been removed from the display. Look under the **Select fields** tab.
20. **Shut Down**

Remove the boards from the emulator and battery module. Remove the battery module jumper and place it over one of the pins. Shut down all open windows on your desktop.

You’re done
**Data Hub and Multiple Links**

**AP as Data Hub**
- The peer for each End Device is on the Access Point
- AP can then act as a data collector or gateway

**Multiple Links**

Because applications have separate port addresses, multiple peer to peer connections can exist on a hardware platform

- Application port addresses are 0x20 – 0x3D
- LinkTo ports start at 0x20 and increment
- LinkListen ports start at 0x3D and decrement
- A SimpliciTI network address consists of the device address (set at compile time) + the assigned port.

Store and Forward ...
Storing, Forwarding and Polling

**Storing and Forwarding Messages**

- The AP can store and forward messages on behalf of sleeping end devices.
- The number of messages the AP is allowed to hold is configurable with `NUM_STORE_AND_FWD_CLIENTS`.
- Sleeping devices must poll the AP upon wakeup.

**Polling Devices**

- End Device can be configured to poll with `RX_POLLS` in `smpl_config.dat`.
- Linked AP is always on.
- When a polling device joins the network, the AP reserves resources to support it.
  - Messages addressed to sleeping device are held by AP.
  - Broadcast messages are not held.
- Each port must be polled separately until no more messages exist for that port.

App level acknowledgements ...
Application Level Acknowledgements

- No acknowledgement mechanism is built into SimpliciTI
- Acknowledgements must be implemented at the application level
- Re-sends, etc are completely at the discretion of the programmer
- NWK level acknowledgements are a proposed feature for future SimpliciTI releases

Broadcast Messages

- The broadcast message type is used to transmit to every node in the network, regardless of address
- The AP and RE will repeat, but not store and forward this message
*** I’m paid by the page ***
Lab5b – AP as Data Hub Network

Description:
In this lab, we’ll utilize another of the example applications to set up a AP-as-data-hub network. Each ED will have an equivalent ED peer on the AP. This example can also demonstrate frequency agility, but we’ll disable that and demo it later. With the channels changing, it would be hard to use the packet sniffer to visualize the message traffic.

<table>
<thead>
<tr>
<th>Group #</th>
<th>Channel #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>10 or 0x0A</td>
</tr>
<tr>
<td>7</td>
<td>12 or 0x0C</td>
</tr>
</tbody>
</table>

Base frequency: 2.47 GHz
Channel Spacing: 400 kHz
Hardware list:

- 3 eZ430-RF2500 Target Boards
- 2 Battery Modules
- 4 AAA Batteries
- 1 eZ430-RF2500 Emulator Board
- 1 SmartRF04EB Board (firmware revision 28 or later)
- 1 CC2510EM Board
- 1 Antenna
- 1 USB A/B Cable
- 1 USB Extender Cable

Software list:

- IAR Embedded Workbench for MSP430 version 4.11D
- TI Packet Sniffer version 2.10.1
- SmartRF Studio version 6.10.2.0

(You will find shortcuts for the above applications on the desktop)

- SimpliciTI version 1.0.6
Procedure

1. Hardware

The USB extender cable and emulator should still be connected from the previous lab. Connect one of the target boards to the emulator and we’re ready to go.

2. Start IAR Embedded Workbench

Start IAR Embedded Workbench and open the existing workspace:
C:\Texas Instruments\SimpliciTI-1.0.6\Projects\Examples\Peer applications\eZ430RF\AP as data hub\SimpliciTI AP Data Hub.eww

If you are prompted to update the project(s), go ahead.

This workspace consists of three projects; an End Device project, and Access Point project and a Range Extender project. We won’t be using the RE in this lab.

3. Build the End Devices First

Click on SimpliciTI End Device – Release in the Workspace Configuration pull-down:

That will set the ED project as the active project. Expand the project out so that we can see its contents, like below:
4. **Frequency/Channel**

   The entire SimpliciTI download shares the same mrfi folder that you programmed using SmartRF Studio in the last lab. So we’ll be operating with those settings.

5. **Frequency Agility**

   In order to use the packet sniffer easily, we won’t be using Frequency Agility in this lab, so we need to disable it. Open `smpl_nwk_config.dat` in the SimpliciTI End Device project, peer applications group, Configuration group. Comment out the bottom line:

   ```
   // -DFREQUENCY_AGILITY
   ```

6. **Unique Addresses for the EDs**

   Every SimpliciTI node needs a unique address, so open up `smpl_config.dat` in the Configuration folder of the SimpliciTI End Device – Release project. This first device will have address: 0x78, 0x56, 0x34, 0x12. Go ahead and Build/Load the program to this target board. When complete, disconnect the board, connect it to an un-powered battery module and label it ED 78. Click the Stop Debugging button.

   Connect another target board to the emulator and change the address in `smpl_config.dat` to: 0x79, 0x56, 0x34, 0x12. On the menu bar, click Project → Clean to remove files created by the last build. Build/Load to the new board, disconnect it from the emulator, connect it to the other un-powered battery module and label it ED 79. Click the Stop Debugging button.
7. Program the AP

Using the Workspace Configuration pull-down, select SimpliciTI Access Point – Release. Note that the project is BOLD, indicating that it is the active project.

Collapse the SimpliciTI End Device – Release project and expand the SimpliciTI Access Point project so that we can see its contents. Close all open editor windows (before you get confused). Let’s make sure that all the ED output files are out of the way before we build the AP. On the menu bar, click Project ➔ Clean.
8. Frequency Agility

Open `smpl_nwk_config.dat` in the SimpliciTI AP Data Hub project, peer applications group, Configuration group. Make sure the line below is commented out.

```c
// -DFREQUENCY_AGILITY
```

Open main_AP_Async_listen.c in the SimpliciTI AP Data Hub project, peer applications group, application group. About 2/3rds down in the file, find the following line of code:

```c
else
{
    //      checkChangeChannel();
}
```

Comment out the call to `checkChangeChannel();`

9. Unique Address for the AP

Open up `smpl_config.dat` in the Configuration folder of the SimpliciTI Access Point – Release project. Change the first byte of the address to `0x90`. Connect the last target board to the emulator, Build/Load the program to the board and it AP 90.

10. Start the Sniffer

Make sure the sniffer hardware is powered and start the TI sniffer software. Select the protocol/chip type as SimpliciTI (CC2510 or CC1110) and click Start.

Under the Radio Settings tab, select the CC2510 packet sniffer settings file that we created earlier. Then, back under the Setup tab, select your workgroups’ channel. Click the Start button.

11. Start the AP board

Click the Go button in IAR Embedded Workbench, then minimize it. Both LEDs on the board should light, but there should be no packets on the sniffer display.

12. Power ED 78 board

Connect the power jumper on the ED 78 battery module. If your hands aren’t too shaky, you should see a Join request (ED78), a Join response (AP90), a Link request (ED78) and a Link response (AP 90).
13. **Power ED 79 board**

   Connect the **power jumper** on the **ED 79** battery module. You should see a **Join** request (ED79), a **Join** response (AP90), a **Link** request (ED79) and a **Link** response (AP 90).

   Now both ED’s are linked to their peer devices on the AP.

14. **Data**

   Press the **button** on either board. The red LED on the AP 90 board should toggle. This application gives both remote ED boards control over a peripheral on the AP. Observe the packet traffic and payloads.

15. **Shut Down**

   Close all open windows on your desktop, remove the AP 90 target board from the emulator and power off both battery modules.

   ![You’re done](stop.png)
*** sometimes you feel like a nut, sometimes you don’t ***
Frequency Agility and Current Consumption

**Frequency Agility**

- 4 available channels to avoid noise
- Channels are defined in `mrfi_f1f2.c` or `mrfi_radio.c` for 802.15.4 radios
- Channel migration is initiated by the AP by:
  - Algorithmic detection of a noisy channel via RSSI
  - User initiation
- AP sends a broadcast message containing the channel number to which to change
  - Sleeping devices may miss this message and need to initiate channel discovery

```c
#if defined( MRFI_CC2500 ) || defined( MRFI_CC2510 ) || defined( MRFI_CC2511 )
static const uint8_t mrfiLogicalChanTable[] =
{ 3, 103, 202, 212 }
```

**Current Consumption**

- Osc startup
  - 2.7mA
- Ripple counter timeout
  - 1.75mA
- PLL calibration
  - 7.5mA
- Receive
  - 18.8mA
- Transmit
  - 21.3mA

MSP430 active: 2.7mA
MSP430 LPM3: 1.3uA
Upcoming Features and Demo

Upcoming Features

- Security
- NV storage of link tables
  - Network re-start from power-up without re-linking
- Listen Before Talk
  - Improved CCA algorithm
- Improved documentation
- Unlink
  - To remove a device from the network
- Auto acknowledge
  - In place of the current application level ack
- Code Composer Studio support

Demo

- Frequency Agility

Lab time...
Lab5c – Adding SimpliciTI to an Existing Application

Description:
In this lab, we’ll take an existing application running on the MSP430 and add wireless capability to it. This probably matches what most of you will be doing … it lays out all the steps, directories, includes and all the little things that can take so much time to fix.

Lab5c – Add SimpliciTI to Existing App

- Take an existing application (a temperature sensor)
- Add SimpliciTI radio support to the project
- Test

<table>
<thead>
<tr>
<th>Group #</th>
<th>Channel #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>10 or 0x0A</td>
</tr>
<tr>
<td>7</td>
<td>12 or 0x0C</td>
</tr>
</tbody>
</table>

Base frequency: 2.47 GHz
Channel Spacing: 400 kHz
Lab5c – Adding SimpliciTI to an Existing Application

**Hardware list:**

- 2 eZ430-RF2500 Target Boards
- 1 Battery Modules
- 2 AAA Batteries
- 1 eZ430-RF2500 Emulator Board
- 1 SmartRF04EB Board (firmware revision 28 or later)
- 1 CC2510EM Board
- 1 Antenna
- 1 USB A/B Cable
- 1 USB Extender Cable

**Software list:**

- IAR Embedded Workbench for MSP430 version 4.11D
- TI Packet Sniffer version 2.10.1
- SmartRF Studio version 6.10.2.0

(You will find shortcuts for the above applications on the desktop)

- SimpliciTI version 1.0.6
Lab5c – Adding SimpliciTI to an Existing Application

**Procedure**

1. **Install the Hardware**

   Insert an Emulator/Target board into the end of the USB extension cable.

2. **Build Lab5a Application**

   Open IAR Embedded Workbench for the MSP430 by clicking on the shortcut on your desktop. When the StartUp window appears, click on **Create new project in current workspace**. Select the **MSP430 Tool chain** and click **OK**.

   Navigate to: `C:\Texas Instruments\SimpliciTI-1.0.6\Projects\Examples\Peer applications\eZ430RF\Lab5c`, name the project **Lab5c** and click **Save**.

3. **New Groups and Files**

   Add a new **Group** in the **Workspace** window called **Source**. Add `main.c`, `VLO_Library.h` and `VLO_Library.s43` from `C:\Texas Instruments\SimpliciTI-1.0.6\Projects\Examples\Peer applications\eZ430RF\Lab5c` to this group.

   Add another **Group** called **Components** and add a **Group** to that called **bsp**. Add `bsp.c`, `bsp.h` and `bsp_macros.h` from `C:\Texas Instruments\SimpliciTI-1.0.6\Components\bsp` to this group.
4. Project Settings

If main.c isn’t open in the editor window already, look in the Source group in the Workspace window and double-click on main.c. To construct this project, we leaned heavily on the on-line code examples available at www.msp430.com. Note at the top of main.c that the board support header files are added. In order to use the board support library, we had to include/add bsp.c, bsp.h and bsp_macros.h. To make the workspace window a little cleaner, we created the Components/bsp and Source folders. All three source files were simply added to the project.

Right-click on Lab5c – Debug at the top of the workspace window and select Options.

In the General Options category, Target tab, configure the target device as a MSP430F2274. Click on the Stack/Heap tab and check the Override default checkbox. Change the Heap size to 0. We won’t be using any dynamic memory.

Select the C/C++ compiler category and click the Preprocessor tab. Add the paths from the top of the file Paths.txt in the Lab5c folder.

C:\\Program Files\\IAR Systems\\Embedded Workbench 5.0\\430\\inc  
C:\\Texas Instruments\\SimpliciTI-1.0.6\\Components\\bsp  
C:\\Texas Instruments\\SimpliciTI-1.0.6\\Components\\bsp\\drivers  
C:\\Texas Instruments\\SimpliciTI-1.0.6\\Components\\bsp\\boards\\EZ430RF

Click on the Linker category and select the List tab. Check the Generate linker listing checkbox so that a map file will be generated. That way we’ll be able to see what our code size is easily.

It’s very important to tell IAR Embedded Workbench to use the Debugger; otherwise it will default to the Simulator (this is both confusing and infuriating).

Click on the Debugger category and see that FET Debugger is selected as the driver. Select the FET Debugger category and see that the Texas Instruments USB-IF connection is selected. Click on the Breakpoints tab and make sure that the Use software breakpoints checkbox is selected. The emulator hardware provides only a single hardware breakpoint, and this will emulate more breakpoints in software for us.

Click OK to close out the Options window.

5. Build/Load

Build/load the program onto the eZRF430-2500 by clicking the Debug button. Fix any problems that arise (there should be none). When prompted, save the Workspace as Lab5c. The code will be automatically programmed into the F2274 Flash memory. It might be a good idea to position the board away from blowing fans, hot coffee or cold drinks at this time.
6. **Set a BreakPoint**

   In `main.c`, set a breakpoint on the line: `if (IntDegC>32)`. Right-click on the `IntDegC` variable in the code and select **Add to Watch**. The Quick Watch window will appear on the right of the screen.

7. **Run**

   Run the code a few time by clicking the **Go** button in the toolbar and observe `IntDegC` in the Quick Watch window. The temperature sensor hasn’t been calibrated, so the actual temperature may be off by a few degrees, but we’re only interested in a relative temperature change, so the accuracy isn’t important for this lab.

8. **Program Set Point**

   Add two degrees to the measured value and enter it into the two `if` statements in the code. For example, if you measured 36 degrees (boy, it’s HOT), your code would look like:

   ```c
   if (IntDegC>=38)
   {
     BSP_TURN_OFF_LED2();
     BSP_TURN_ON_LED1();
   }
   if (IntDegC<38)
   {
     BSP_TURN_OFF_LED1();
     BSP_TURN_ON_LED2();
   }
   ```

9. **Rebuild/Reload/Run**

   Click the **button to Stop Debugging**, and then remove the breakpoint from the code by double-clicking on the red dot. Click the **Debug** button to rebuild/reload the program. Then click **Go**.

10. **Operation**

    You should see the green LED light up. Place your thumb on the chips to heat them up (the largest one on the side with the LEDs is the MSP430F2274 with the internal temperature sensor). If your hands are cold, just rub your thumb on your pants leg to warm it up.

    The red LED should light after a few seconds. Sit the board down and, after a short cool down, the green LED should light. If none of this happens, you probably used the wrong set point. You’ll have to go back and re-set it.
11. Code Size

Open up the map file so that we can see our code size. Click the Open button on the menu bar and navigate to C:\Texas Instruments\SimpliciTI-1.0.6\Projects\Examples\Peer applications\eZ430RF\Lab5c\Debug\List. Select Lab5c.map and click Open.

Bearing in mind that the numbers are in hex, add up all the DATA16 and CSTACK section sizes and enter the result in the table below in the RAM row, w/o SimpliciTI column (I got 74h). Add the CSTART and CODE sizes together, and enter the result into the FLASH row, w/o SimpliciTI column (I got 250h).

<table>
<thead>
<tr>
<th></th>
<th>w/o SimpliciTI</th>
<th>TX code w/ SimpliciTI</th>
<th>RX code w/ SimpliciTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLASH</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Now that we have a working piece of code, let’s make it wireless! Let’s have the temperature sensor display its results on a remote device. The first thing we need to decide on is what topology we want to use ... let’s start out simple and use a Peer to Peer topology. Let’s assume that you’ve copied the eZ430-RF2500 target schematic and that your MSP430F2274 is connected via the SPI port to the CC2500.

12. Adding SimpliciTI

Let’s add the SimpliciTI components to the IAR project. Right-click on the Components folder, select Add -> Add Group and create a nwk group. Then right-click on the nwk group and select Add -> Add Files. Navigate to C:\Texas Instruments\SimpliciTI-1.0.6\Components\simpliciti\nwk, right-click in the window and change Arrange Icons By -> Type. Select all five .c files and click Open.

Right-click on the Components folder, select Add -> Add Group and create a network applications group. Add all seven .c files from C:\Texas Instruments\SimpliciTI-1.0.6\Components\simpliciti\Network applications to the group.

Right-click on the Components folder, select Add -> Add Group and create a mrfi group. Add mrfi.c from C:\Texas Instruments\SimpliciTI-1.0.6\Components\mrfi to the group.
13. Add a Folder and Add Files

Using Windows Explorer, navigate to C:\Texas Instruments\SimpliciTI-1.0.6\Projects\Examples\Peer applications\eZ430RF\Lab5c. Create a new folder called Configuration and a subfolder under Configuration called End Device.

Look in C:\Texas Instruments\SimpliciTI-1.0.6\Projects\Examples\Peer applications\eZ430RF\Simple polling with AP\Configuration and copy the smpl_nwk_config.dat file into your Configuration folder. Then look in the other project’s End Device folder and copy smpl_config.dat into your End Device folder. You can close Windows Explorer.

14. Another Group

Go back to IAR Embedded Workbench, right-click somewhere blank in the Workspace window and add a group named Configuration. Right-click on that group and add the .dat files from the previous lab step (the ones in our project folders). Note: You’ll have to change the Files of Type to make them visible.
15. Project Options

Let’s modify the Project options to support SimpliciTI. Right-click on Lab5c - Debug in the Workspace window and select the following categories:

**General Options → Stack/heap tab:**
Override the default and set the Stack size to 200.

**C/C++ compiler → Preprocessor tab:**
Add the following include paths from the middle of Paths.txt in the Lab5c folder.

- C:\Texas Instruments\SimpliciTI-1.0.6\Components\mrfi
- C:\Texas Instruments\SimpliciTI-1.0.6\Components\simpliciti\nwk
- C:\Texas Instruments\SimpliciTI-1.0.6\Components\simpliciti\Network applications

Add MRFI_CC2500 in the Defined symbols box to select the CC2500 radio.

**Extra Options tab: (on the far right using the arrow button)**
Check the Use command line options checkbox and add the following command line options in order to include the configuration file directives. Again, you can find these at the bottom of file Paths.txt in the Lab5c folder.

- `–f "C:\Texas Instruments\SimpliciTI-1.0.6\Projects\Examples\Peer applications\eZ430RF\Lab5c\Configuration\End Device\smpl_config.dat"`
- `–f "C:\Texas Instruments\SimpliciTI-1.0.6\Projects\Examples\Peer applications\eZ430RF\Lab5c\Configuration\smpl_nwk_config.dat"

These files contain a number of compiler required defines for the project. They need to be included prior to the compilation in order for the defines to have their intended function.

Finally, with a grand flourish, click OK.

16. Sanity Check

As a quick sanity check, you should be able to build the project with no errors or warnings at this time. From the top menu bar, click Project → Rebuild All. Correct any problems you may find. We haven’t changed the function of the code, so don’t bother to run it.
17. Create Configurations

We’re now ready to modify the code to use the radio, but we’d like to create two versions of it first; one for the transmitter (Tx) and one for the receiver (Rx). Let’s take the next few steps to do that.

Close main.c in the editor window, then right-click on main.c in the Workspace window and remove it from the project. When prompted, click Yes.

Then, using Windows Explorer, open the C:\Texas Instruments\SimpliciTI-1.0.6\Projects\Examples\Peer applications\eZ430RF\Lab5c folder, rename main.c to main_tx.c. Make a copy of main_tx.c and name it main_rx.c.

Back in IAR Embedded Workbench, right click on the Source group in the Workspace window and add both main_tx.c and main_rx.c to your project.

On the menu bar, select Project → Edit Configurations → New and enter Transmitter in the Name box. Also add a Receiver configuration.

Select Receiver in the Configuration dropdown box in the Workspace window. Right click on main Tx.c → Options, check the Exclude from build checkbox and click OK. Follow a similar procedure to exclude main_rx.c from the Transmitter project. If you look closely at the symbol to the left of the filename, you’ll see which file is excluded. Add something to the header comments of the transmitter file so you won’t get confused. We’ll deal with the receiver later.

18. Modify the Transmitter Code

Let’s start out with the transmitter code. Change the Configuration to Transmitter and double click on main_tx.c to open it in the editor.

Add the following header file includes just under the bsp includes:

```c
#include "bsp_buttons.h"  // button routines
#include "mrfi.h"                   // SimpliciTI header files
#include "nwk_types.h"
#include "nwk_api.h"
```

19. Add the following variable definitions in the Global variables area:

```c
linkID_t  linkIDTemp;
uint8_t   msg[1];
```

20. Right after BSP_Init();, add the following lines:

```c
// Initialize SimpliciTI
SMPL_Init(0);  // no callback supplied
```
21. Just before **Global Interrupts** `_EINT();` are enabled, add the following code. You can cut/paste from `Code.txt` in the **Lab5c** folder if you like.

```c
// wait for a button press...
do {
  if (BSP_BUTTON1())
    break;
} while (1);

// Link to receiver
while (SMPL_SUCCESS != SMPL_Link(&linkIDTemp))
{
  BSP_TOGGLE_LED1(); // Blink red LED until link success
}
BSP_TURN_OFF_LED1(); // Then indicate success by lighting
BSP_TURN_ON_LED2(); // green LED for 2 seconds
NWK_DELAY(2);
BSP_TURN_OFF_LED2();
```

We’re adding a button press here to initiate the link process. This will make the Sniffer results easier to interpret later on. The red LED will blink until linking is successful, at which point the green LED will light for 2 seconds.

22. Look in the main `while(1)` loop at the LED control statements. Just below the first pair, insert:

```c
msg[0] = 0xF; // Overtemp message
```

Just below the second pair, insert:

```c
msg[0] = 0x0; // Undertemp message
```

These are the messages that will be transmitted to indicate over or under temperature to the receiver.

23. Below the last line you inserted, there are three closing braces. Just below the first one, insert following line of code. If you’re a slacker, you can copy/paste it from `Code.txt` in the **Lab5c** folder.

```c
SMPL_Send(linkIDTemp, msg, sizeof(msg)); // Send message to receiver
```

This is the SimpliciTI command that actually sends the message over the air.
24. Radio Settings

We’ll re-use the settings that we previously made using SmartRF Studio, but open `C:\Texas Instruments\SimpliciTI-1.0.6\Components\mrfi\radios\common\mrfi_f1f2.c` and verify that the first entry in the CC2500 structure is your workgroups’ channel.

25. Test the Transmitter

**Build/Load** the Transmitter program to the target board and use a Post-it™ note to label it TX.

Start the **Packet Sniffer** for SimpliciTI (CC2510 or CC1110), make sure that your sniffer settings file is loaded and select your channel. **Start capturing packets.**

Return to IAR Embedded Workbench, and click the **Go** button.

A single **Join** message should appear in the sniffer. Since there is no AP present, this request will go unanswered. **Press** the button on the target board and you should see **Link** requests appear and the red LED on the target board should be blinking. The TX will continue to send out **Link** requests until it gets a response, which will be never unless we get busy constructing the receiver.

**Stop** the debugger, disconnect the TX target board and move it aside. Label another target board RX and **connect** it to the emulator.

26. Check the Code Size

Following the procedure outlined in **step 11**, fill in the **RAM/FLASH** sizes for the **TX code w/ SimpliciTI column**. Remember to look in the Transmitter folder.

I got: **RAM = 25Eh and FLASH = 1D52h**

27. Modify the Receiver Code

Switch the Workspace **Configuration** to **Receiver** and open **main_rx.c** for editing. Select **everything** and **delete** the contents. Copy the entire contents of **main_tx.c** into **main_rx.c**. Close **main_tx.c**. Make a change to the **main_rx.c** header so you can quickly see that this is the receiver.
28. In `main_rx.c`, delete the following lines of code. They aren’t needed on the receiver.

```c
#include "VLO_Library.h"
unsigned int dco_delta;
volatile unsigned long IntDegC;
void Setup_ACLK(void); and the Setup_ACLK subroutine
void Setup_ADC10(void); and the Setup_ADC10 subroutine
void Setup_TIMERB(void); and the Setup_TIMERB subroutine
Setup_ACLK();
Setup_ADC10();
Setup_TIMERB();
LPM3;
ADC10CTL0 |= ENC + ADC10SC;
LPM0;
temp = ADC10MEM;
IntDegC = ((temp - 673) * 423) / 1024;
msg[0] = 0xF;
msg[0] = 0x0;
SMPL_Send(linkIDTemp, msg, sizeof(msg));
```

Also, delete both interrupt service routines at the bottom of the code.

29. **Length, Flag and Callback**

We’ll need to add a length variable (for the message length) and we’re going to need to create a callback that will occur when SimpliciTI receives a frame. In the **Global variables** area, **add** the following:

```c
uint8_t len;
uint8_t flag;

// callback handler
static uint8_t sRxCallback(linkID_t);
```

Change the `SMPL_Init(0);` line to:

```c
SMPL_Init(sRxCallback); // Callback function
```

30. **SMPL_LinkListen**

Change:

```c
// Link to receiver
while (SMPL_SUCCESS != SMPL_Link(&linkIDTemp))
    - to -
// Link to transmitter
while (SMPL_SUCCESS != SMPL_LinkListen(&linkIDTemp))
```
31. Turn the Radio On

The default is for the radio to be in idle, so we’ve got to turn the receiver on. Just above the while(1) statement, insert the following:

```c
// Turn the radio receiver on
SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXON, 0);
```

32. Overtemp, Undertemp

The message we’re sending to the receiver is 0xF for Overtemp and 0x0 for Undertemp. Change the if() statements in the while() loop so they look like the following:

```c
if (temp == 0xF)
{
    BSP_TURN_OFF_LED2(); // Turn off green LED
    BSP_TURN_ON_LED1();  // Turn on red LED
}
if (temp == 0x0)
{
    BSP_TURN_OFF_LED1(); // Turn off red LED
    BSP_TURN_ON_LED2();  // Turn on green LED
}
```

33. Callback Function

We need to add the function that will be called when the radio receives a frame. We already called this function sRxCallback in the definitions. The function is invoked in the frame-receive ISR thread so it runs within the interrupt context. There’s only a single message from a single transmitter, so the code is pretty simple. You can type this code in, or you can cut/paste from Code.txt in the Lab5c folder.

```c
//****************************
// Received Message Handler
//****************************
static uint8_t sRxCallback(linkID_t linkIDTemp)
{
    if (SMPL_SUCCESS == SMPL_Receive(linkIDTemp, msg, &len))
    {
        temp = msg[0];
    }
    return 0;
}
```

34. Unique Address

Every SimpliciTI node need a unique address, right? We’ve already built the TX, so open `smpl_config.dat` and change the address slightly.
35. **Build/Load/Fix**

Build and Load the program to the Rx target board. Fix any problems or typos that you encounter. Click the **Stop Debugging** button.

36. **Check the Code Size**

Following the procedure outlined in step 11, fill in the RAM/FLASH sizes for the **RX code w/ SimpliciTI** column. Remember to look in the Receiver folder.

I got: RAM = 24Ah and FLASH = 1B48h. That makes my table look like this:

<table>
<thead>
<tr>
<th></th>
<th>w/o SimpliciTI</th>
<th>TX code w/ SimpliciTI</th>
<th>RX code w/ SimpliciTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>72h</td>
<td>25Eh</td>
<td>24Ah</td>
</tr>
<tr>
<td>FLASH</td>
<td>250h</td>
<td>1D52h</td>
<td>1B48h</td>
</tr>
</tbody>
</table>

Bearing in mind the differences between the application code sets, we can see that adding SimpliciTI costs about 500 words of RAM and about 6.9K of FLASH. Your code sizes should compare favorably.

37. **Packet Sniffer**

Make sure that your packet sniffer hardware is connected and powered, then start the packet sniffer software. Make sure you’ve selected **SimpliciTI (CC2510 or CC1110)**.

Check under the **Radio Settings** tab and see that your packet sniffer **settings file** is selected. Go back to the **Setup** tab and make sure your workgroups’ channel is entered. The boards are un-powered, right? Click the **start capture** button.
38. Operation

Connect the Tx board to a battery module and make sure the power jumper is on. If you’re lucky and your hands aren’t shaky, you’ll see a single Join message.

Now power the Rx board. Again, you should see a single Join message. Both of these Join requests fall on deaf ears, since there is no AP to answer them.

Press the button on the Rx board to execute SMPL_LinkListen(). The red LED will blink very slowly and you’ll see no sniffer activity.

Press the button on the Tx board and you should see a link request and reply appear. The green LEDs will light, and then the temperature sensing code has begun execution.

At this point the sniffer should be capturing about one message per second from the Tx indicating whether the temperature is over or under-temp. Both boards should have the red or green LED lit, depending on the temperature. If both are green, use your warm finger to cause an overtemp condition. Both boards should light red LEDs. If your LED’s start out red, you might want to use a cold can of soda to cool your finger down.
39. Lab Completed

I can think of several ways to improve on the code we’ve just written ... the power consumption of the receiver is a particular issue. But frankly, we’re out of time and need to move on. If you’d like to make some changes to the code, send them to me via this workshops’ Wiki site and if I add your changes to the course, you will get credit in the source files. Long live open source!

Congratulations! We’ll be changing out the hardware for the next lab, so power down and disconnect all the hardware but the **SmartRF04EB**. Power that board down and carefully remove the **CC2510EM** board and place it with the rest of the disconnected hardware. Make sure your **power jumpers** are over one of the pins. Shut down any open programs on your workstation.

You’re done