

## Introduction

Z-Accel is a new packaging of our, now familiar Z-Stack into a more user-friendly format. If you need to implement a ZigBee solution quickly, Z-Accel could be your most promising alternative.

## Objectives

- Look at the Z-Accel architecture
- Learn about Simple API (SAPI)
- Look at Z-Accel solutions
- Do an introductory lab

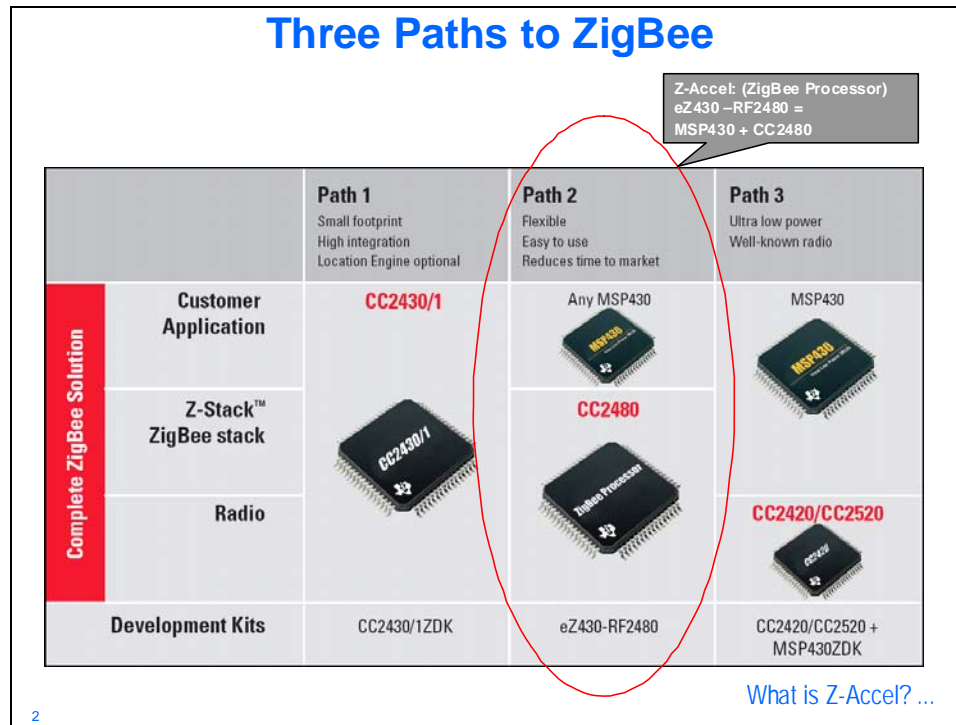
\*\*\* ... the greatest hazard in life is to risk nothing. (Dr. Grissom) \*\*\*

# Module Topics

<b>Z-Accel .....</b>	<b>9-1</b>
<i>Module Topics.....</i>	9-3
<i>Three Paths to ZigBee.....</i>	9-5
<i>Z-Accel.....</i>	9-6
<i>Simple API.....</i>	9-7
<i>Remote Procedure Calls .....</i>	9-8
<i>Architecture .....</i>	9-9
<i>App Model and Architecture .....</i>	9-10
<i>Profiles and Endpoints.....</i>	9-11
<i>PAN Formation and Discovery.....</i>	9-12
<i>Target Hardware .....</i>	9-13
<i>Lab9A – Z-Accel Sample Application (ZASA).....</i>	9-15
Hardware list: .....	9-16
Software list:.....	9-16
<i>Procedure.....</i>	9-17
Open the Project .....	9-17

\*\*\* do you know who ELI the ICEman is? \*\*\*

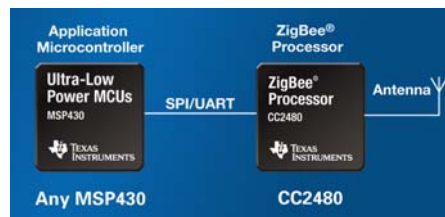
# Three Paths to ZigBee



## Z-Accel

### What is Z-Accel and the CC2480?

- ◆ Z-Accel is a ZigBee Network Processor that communicates with any MCU via an SPI or UART interface
- ◆ CC2480 is the first-generation ZigBee 2006-compliant network processor in the Z-Accel family
- ◆ Z-Accel allows customers to work with their favorite MCU
- ◆ Z-Accel provides complete ZigBee functionality without having to learn the complexities of a full ZigBee stack

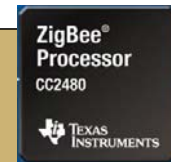


Key Features ...

3

### Z-Accel – Key Features

- ◆ **Reduce Development Complexity**
  - Preprogrammed / preconfigured ZigBee Node
- ◆ **Easy to Deploy**
  - Accelerate ZigBee Connectivity for existing solution
  - Easily added as serial peripheral
- ◆ **Easy to Use**
  - Split Application and Network processor
  - Application Processor running Application Framework & Profile
- ◆ **Standardized protocol**
  - Network Infrastructure reused by multiple applications
  - Manufacturer Specific Profiles to enable interoperability whenever possible
- ◆ **Z-Accel supports SimpleAPI**
  - SimpleAPI has only 10 API calls to learn, which drastically simplifies the development of ZigBee applications



SAPI ...

4

# Simple API

## Benefits of using SimpleAPI

- ◆ Shorten time from idea to launch
- ◆ Shorten learning curve
  - ◆ No need to read the ZigBee specification
  - ◆ SimpleAPI documentation is less than 30 pages in total (ZigBee2006 specification is 534 pages)
- ◆ Very little code space needed on the application MCU
- ◆ Small application microcontroller can be used (4kB+)
- ◆ Advanced API can still be used if needed
  - ◆ ~90% of all applications can be done with SimpleAPI

SimpleAPI implements:

- ◆ Device configuration
- ◆ Commissioning of networks
- ◆ Binding of devices
- ◆ Sending and receiving data



Functions & Callbacks ...

5

## SimpleAPI Functions and Callbacks

- ◆ **zb\_SystemReset**
  - ◆ Resets network
- ◆ **zb\_StartRequest**
  - ◆ Starts network
- ◆ **zb\_PermitJoiningRequest**
  - ◆ Allow nodes to join network
- ◆ **zb\_BindDevice**
  - ◆ Establish a binding (connection)
- ◆ **zb\_AllowBind**
  - ◆ Allow binding request
- ◆ **zb\_SendDataRequest**
  - ◆ Send data
- ◆ **zb\_ReadConfiguration**
  - ◆ Read configuration parameters
- ◆ **zb\_WriteConfiguration**
  - ◆ Write configuration parameters
- ◆ **zb\_GetDeviceInfo**
  - ◆ Get current address, PAN ID etc
- ◆ **zb\_FindDeviceRequest**
  - ◆ Search for a device on the network

- ◆ **zb\_StartConfirm**
  - ◆ Network start up callback
- ◆ **zb\_AllowBindConfirm**
  - ◆ Accepted bind request callback
- ◆ **zb\_SendDataConfirm**
  - ◆ Send data status callback
- ◆ **zb\_ReceiveDataIndication**
  - ◆ Incoming data callback
- ◆ **zb\_FindDeviceConfirm**
  - ◆ Search results callback
- ◆ **zb\_HandleKeys**
  - ◆ EVM key push callback
- ◆ **zb\_HandleOsalEvent**
  - ◆ Operating system callback

SPI RPC ...

6

## Remote Procedure Calls

### Z-Accel Transport Overview: Remote Procedure Calls (RPC)

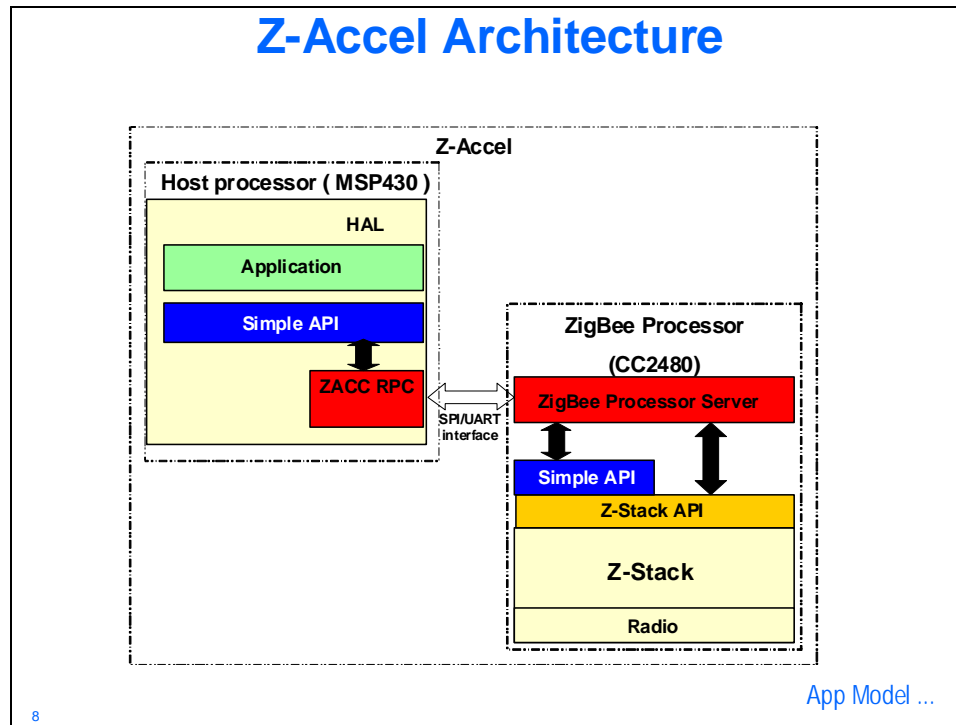
CC2480's 10 Simple API calls are abstracted to  
4 Physical Transport Layer Command Types

- ◆ **AREQ – Asynchronous Request**
  - ◆ Callback Events
  - ◆ Function call with a return value
- ◆ **POLL – Polling for Data**
  - ◆ Retrieve Queued Data
- ◆ **SREQ – Synchronous Request**
  - ◆ Immediate Response
- ◆ **SRSP – Synchronous Response**
  - ◆ Response to a SREQ command

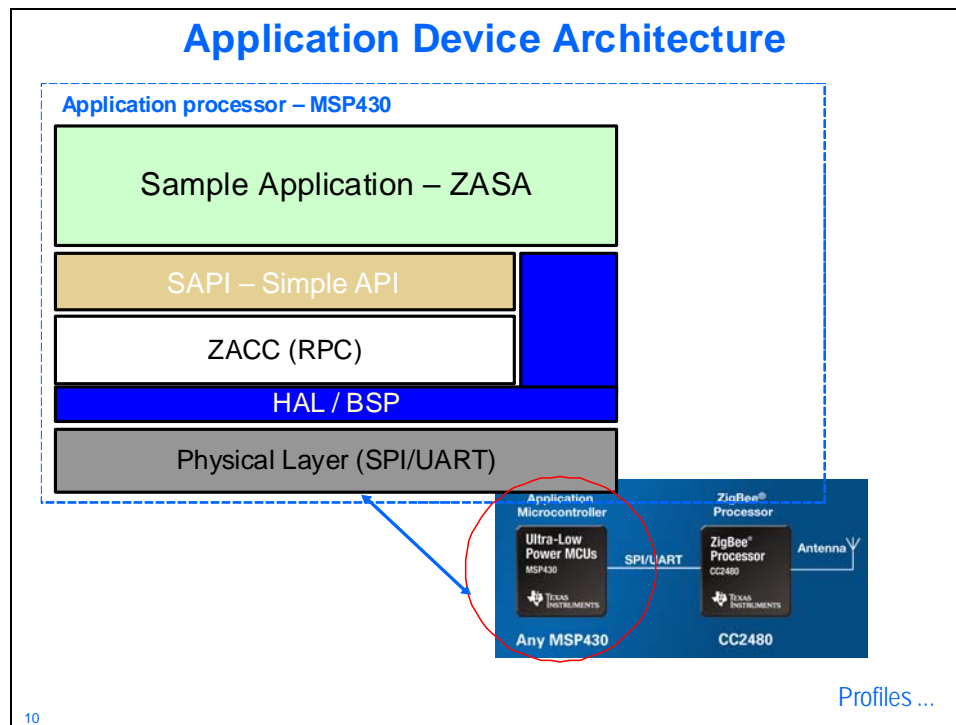
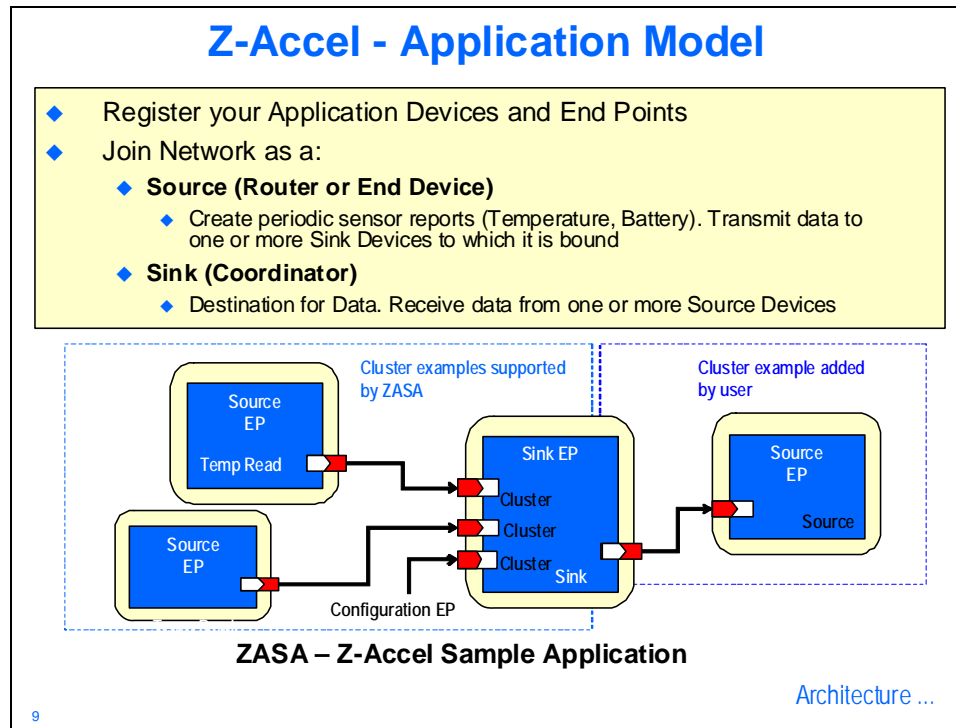
Architecture ...

7

# Architecture



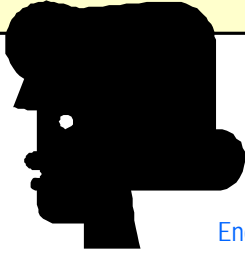
# App Model and Architecture



# Profiles and Endpoints

## Profiles

- CC2480 only supports Manufacturer Specific Profiles (Private Profile)
  - Privately developed by individual manufacturers
- CC2480 only supports ZigBee 2006
  - '06 devices can only join a PRO network as End Devices
  - ZigBee '06 and '07 network seamlessly
- A profile ID must be unique and use a ZigBee Alliance allocated profile identifier
- Commercial products developed using this profile must undergo network capable testing

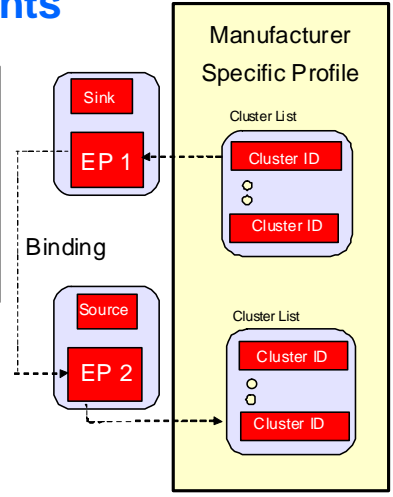


Endpoints ...

11

## Endpoints

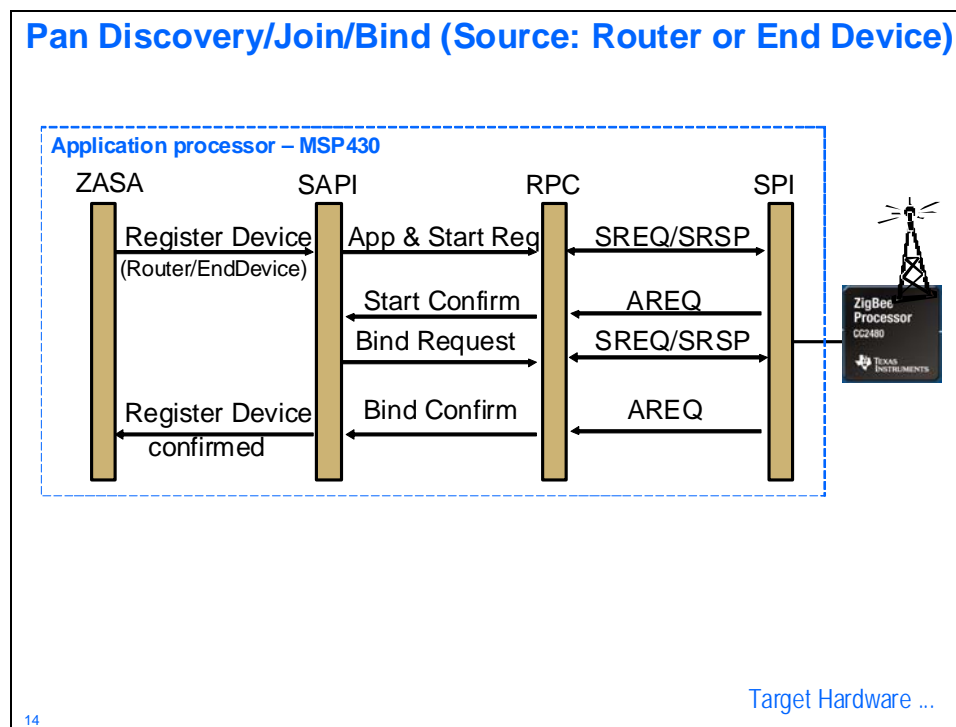
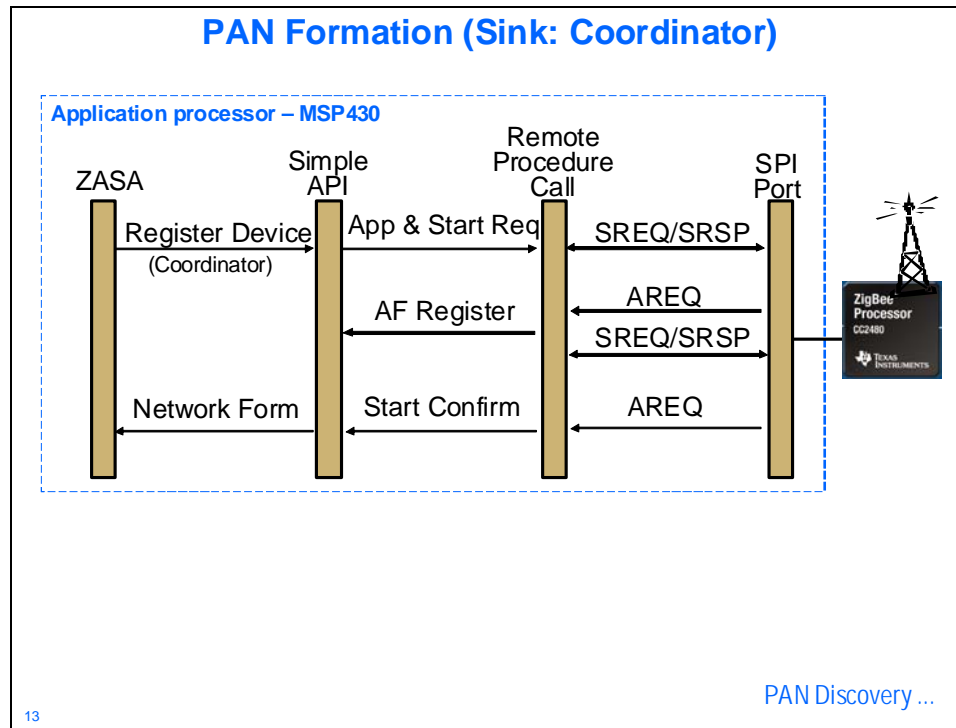
- Endpoint 0 (EP0) is the ZDO application running on the CC2480
- Additional Endpoints run on the application processor (MSP430)



PAN Formation ...

12

# PAN Formation and Discovery



## Target Hardware

### Z-Accel Lab Target Hardware



- ◆ MSP430F2274 processor
- ◆ SPI connection to CC2480
- ◆ 2 LEDs (1 red, 1 green)
- ◆ 1 Pushbutton
- ◆ Photo-resistor
- ◆ 5 exposed MSP430 GPIO (P2.2 – P2.4, P2.4 & P4.5)

#### MSP430F2274 features:

- ◆ 32K Flash / 1K RAM
- ◆ Two 16-bit timers
- ◆ USCI Comm Interface
- ◆ 10-Bit, 200Ksps ADC
- ◆ Two Op-Amps



Lab time ...

15

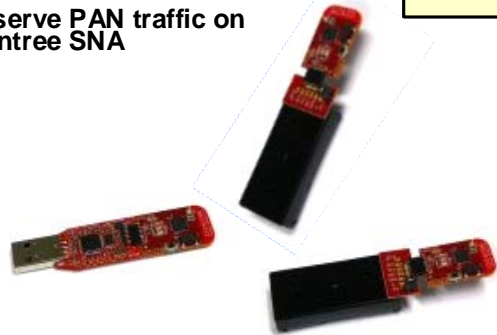
\*\*\* it's later than we ALL think \*\*\*

# Lab9A – Z-Accel Sample Application (ZASA)

## Lab 9A – ZASA Project

- ◆ Open ZASA project
- ◆ Update channel and PAN ID parameters
- ◆ Configure, build and load
- ◆ Start PC Demo Application
- ◆ Start Coordinator-Sink, then Router-Source then End Device-Source
- ◆ Observe PAN traffic on Daintree SNA

	<b>Channel</b>	<b>PAN ID</b>
1	0x0C (12)	0x0AAA
2	0x0D (13)	0x0BEE
3	0x0E (14)	0x0CEE
4	0x0F (15)	0x0DEE
5	0x10 (16)	0x0EEE
6	0x11 (17)	0x0EFF
7	0x12 (18)	0x0BAB
8	0x13 (19)	0x0ACE



Another lab ...

## Hardware list:

- ✓ 1 eZ430-RF2480 Emulator Board
- ✓ 3 eZ430-RF2480 Target Boards
- ✓ 2 Battery Modules
- ✓ 4 AA Batteries
- ✓ 1 USB Extender cable
- ✓ 1 SmartRF04EB board
- ✓ 1 CC2430EM board
- ✓ 1 antenna

## Software list:

- ✓ IAR Embedded Workbench for MSP430 version 4.11D
- ✓ TI Packet Sniffer version 2.10.1
- ✓ Daintree Sensor Network Analyzer version 2.3.0.8
- ✓ eZ430-RF2480 Wireless Sensor Monitor version 1.1.4 (swrc096c.zip)

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

- ✓ eZ430-RF2480 ZASA Installer (Rev.B) (swrc097b.zip)
- ✓ CC2480 Software Examples (swru169.zip)

---

# Procedure

## Open the Project

### 1. Open the ZASA project

Open C:\Texas Instruments\ez430-RF2480\ZASA\IAR\SampleApp.ewp with IAR Embedded Workbench.

### 2. Set your channel and PAN ID

In the **App** workspace group, open **sample\_app.cfg** for editing and select the channel and **PAN ID** that your workgroup was assigned. Make sure to only select a **single channel**.

### 3. Check the Options

In the options for the project, in the **FET Debugger** category, make sure that the **Connection port** is set to **Automatic**.

### 4. Build/Download

Connect the eZ430-RF **emulator** to an open USB port on your PC with the USB extender cable. Connect one of the eZ430-RF2480 **target** boards to the emulator and **build/download** the ZASA code to it. Do the same for the other two target boards. Leave the last target board attached to the emulator and connect the other two boards to the battery modules.

### 5. Operation of the ZASA Code

The following description is pretty much verbatim from section 6 of the **eZ430-RF2480 Demo Kit User's Guide** (in the eZ430-RF2480 folder). My additions, on exactly what actions to take, will be in **BOLD**.

The ZASA application is comprised of two types of devices, Sources and Sinks, operating in potentially all three ZigBee roles; Coordinator, Router, and End Device. Sources provide temperature and bus voltage readings collected by the MSP430 using its on chip temperature sensor and ADC converter. The ZASA application provides code for operating as either type of device, and allows for easy configuration via the button on the CC2480 target board.

After power up, the Sample Application (the host), is in the state *applniting* which is to say, it is waiting to initialize the CC2480. CC2480 will always restore its previous network state from NV memory, unless it was pre-configured not to do so by the host before the last reset. This Sample Application does not take advantage of the NV restore behavior so as to have a perfectly predictable out-of-the-box behavior. Whenever the CC2480 resets, it always sends a SYS API message to indicate that it has reset. Thus, the host is waiting for this indication when it is in the *applniting* state. Upon receiving the reset indication, the host pre-configures CC2480 to not perform NV restore on the next reset and transitions to the *appWaiting* state. When the host is in this *appWaiting* state, both LEDs will blink at 1-Hz.

**Connect the power jumpers on both battery modules and click the Go button in IAR Embedded Workbench now. The LED on each board should be blinking at 1-Hz.**

The user can configure the device as follows:

- One momentary button push in a two-second window configures the device to be an FFD (Full Function Device – a device capable of routing).
- Two momentary button pushes in a two-second window configures the device to be an RFD (Reduced Function Device – an End Device).

When the two-second window expires, the device is “configured” and transitions to the *appJoining* state. An FFD will try to join an existing network as a ZigBee Router for 10 seconds. If it fails to join, it will start a new network as a ZigBee Coordinator. An RFD will try to join an existing network as a ZigBee End Device for 10 seconds. If it fails to join, it will stop, the state will transition to *appJoinWaiting*, and it will enter low-power for 30 seconds, and try again – repeating the sequence until it succeeds in joining. While in the *appJoining* or *appJoinWaiting* state, the LEDs will be off.

**On the target board attached to the emulator, press the button once. After about 10 seconds, the red LED will light, indicating that the device has configured itself as the Coordinator.**

When an FFD starts a new network as a ZigBee Coordinator, it will set the red LED solid on and allow devices to bind – the state will transition to *appRunning* and its sub-state will be “allowing joins” as set by the flag *appPermittingF* in the variable *appFlags*. The device also maintains a tertiary state, that it is sinking data, as set by the flag *appSinkingF* in *appFlags*. Whenever a message is acknowledged, the device blinks the green LED once. The sub-state of allowing joining can be toggled on the Coordinator by a momentary button press. The sub-state transitions to not allowing joining by clearing the *appPermittingF* and the red LED starts blinking at 1-Hz.

**Pick one of the target boards attached to a battery module and press the button once. After a moment, the green LED will light, indicating that the device has configured itself as a Router. Take a Post-it flag and label the board “Router”.**

When an FFD successfully joins an existing network as a ZigBee Router, its state transitions to *appBinding*, and the device sets the green LED solid on. The device uses the SAPI to negotiate OTA and find the device that is sinking data. The FFD will continue to request a bind from the SAPI every 30 seconds until it is successful. Upon successful binding the state will transition to *appRunning* and its sub-state will be “allowing joins” as set by the flag *appPermittingF* in the variable *appFlags*. The sub-state of allowing joining can be toggled on the Router by a momentary button press. The sub-state transitions to not allowing joining by clearing the *appPermittingF* and the green LED starts blinking at 1-Hz. Every 10-seconds, the device sets a tertiary state, that it is sending data, as set by the flag *appSendingF* in *appFlags*. The device requests that the sinking device should acknowledge receipt of the data message OTA, so it retries sending the same data OTA up to 3 times every time that the SAPI feedback from the CC2480 indicates that the message has not been acknowledged. When the message is acknowledged or the retries have expired, the sub-state of sending data is cleared by clearing the *appSendingF* flag. Whenever a message is acknowledged, the device blinks the red LED once.

**On the last target board, press the button twice within a 2 second window. This will configure the device as an End Device. Data transmissions and acknowledgements will be indicated by the flashing LEDs. Take a Post-it flag and label the board “End Device”.**

When an RFD successfully joins an existing network as a ZigBee End Device, its state transitions to *appBinding*, and the device starts blinking the green LED at 1-Hz. The device uses the SAPI to negotiate OTA and find the device that is sinking data. The RFD will wait up to 10 seconds for a bind request to succeed and then transition the state to *appBindingWait* and enter low power for 30 seconds. The device will continue to alternate between the two binding states and low power every 30 seconds until it is successful. Upon successful binding the state will transition to *appRunning* and its sub-state will be “allowing low power” as set by the flag *appLowPwrF* in the variable *appFlags*. Every 10-seconds, the device sets a tertiary state, that it is sending data, as set by the flag *appSendingF* in *appFlags*. The device requests that the sinking device should acknowledge receipt of the data message OTA, so it retries sending the same data OTA up to 3 times every time that the SAPI feedback from the CC2480 indicates that the message has not been acknowledged. While awaiting the acknowledge, the device enters low power and awakes every 2 seconds by a timer in order to poll for the SAPI feedback from the CC2480. When the message is acknowledged or the retries have expired, the sub-state of sending data is cleared by clearing the *appSendingF* flag. Whenever a message is acknowledged, the device blinks the red LED once.

## 6. GUI for Sample Application

We have a PC GUI that displays the data received by the Coordinator (Z-Accel sink) via a UART connection through the USB port. **Stop Debugging** in IAR Embedded Workbench and minimize the tool. Power-off the Router and End Device boards.

**Start** the GUI by clicking on the **eZ430-RF2480 Sensor Monitor** shortcut on your desktop. Cycle power to **reset** the Coordinator by **unplugging/plugging** the USB extension cable attached to the emulator/target board. After a moment, the “**SINK RX**” bubble in the GUI should turn **red**, indicating that the GUI is connected **to** and reading **from** the device. **Press** the button on the target board once to configure the board as a **Coordinator-Sink**. After about 10 seconds, the red LED on the board will light.

**Connect** power for the Router, and then press the target button **once** to configure the device as a **Router-Source**. A blue bubble should appear on the PC GUI with data inside. The bubble will flash each time a data packet is received. Inside the bubble you’ll see four pieces of information:

- a. The temperature reported by the device’s MSP430 internal temperature sensor (don’t worry if this is incorrect; it has not been calibrated).
- b. The short address of the ZigBee device. (being the first Router to join the Coordinator, this should be 0x0001)
- c. The time that the last data was received (PC time)
- d. The battery voltage of the device.

**Connect** power for the End Device, and then press the target button **twice** within 2 seconds to configure the device as a **End Device-Source**. After a few seconds, the LEDs will flash occasionally, indicating data transmission and acknowledgement. A green-ish bubble should appear on the PC GUI with data inside. The bubble will flash each time a data packet is received. Note the data inside the bubble.

**Close** the Sensor Monitor GUI and **power off** all three Z-Accel boards.

## 7. Run the Packet Sniffer

Turn on your sniffer board, making sure that it is connected to the PC's USB port. Start the **Daintree Sensor Network Analyzer**. Don't forget to select your source and your workgroup's channel before clicking the **Start Capture** button.

Watch the action on the **SNA** as you follow the steps below. Make sure that the packets you see make sense in terms of the ZigBee network activity that you'd expect.

- Power on the Coordinator board and press the target button once.
- Power on the Router and press the target button once.
- Power on the End Device and press the target button twice within 2 seconds.

## 8. Power down

Close Daintree SNA and IAR Embedded Workbench. Power all the boards down.



You're done