Wireless Sensor Monitor Using the eZ430-RF2500

Miguel Morales, Zachery Shivers *MSP430 Applications*

ABSTRACT

This application report documents the wireless temperature-sensor network demonstration application provided with the eZ430-RF2500 development tool. The application uses Texas Instruments SimpliciTI™ wireless communication protocol to set up a simple network in which end devices communicate sampled temperature and voltage data to a network access point. The access point communicates all collected data through an available UART to a PC COM port. This port is then used with an accompanying graphical user interface (GUI) to display the data in a user-friendly manner. This document is intended to act as a guide for the eZ430-RF2500 firmware only. It does not focus on the use of the accompanying Network Visualizer GUI or on the SimpliciTI network protocol. For more information on the Network Visualizer and SimpliciTI network protocol, see Appendix B and www.ti.com/simpliciti, respectively.

> **NOTE:** Due to a change in IAR compiler calling conventions, the Wireless Sensor Monitor v1.03 runs only with version 5.12 and later of the IAR Embedded Workbench KickStart™, available online at http://focus.ti.com/docs/toolsw/folders/print/iar-kickstart.html. The code does not compile with KickStart v3.42 and returns a linker error.

Project collateral and source code discussed in this application report can be downloaded from http://www.ti.com/lit/zip/slac139.

The eZ430-RF2500 is available for order from the TI eStore (http://ti.com/appnote-ad-slaa378-rf2500).

Contents

List of Figures

1　Wireless Sensor Monitor Network Overview

1.1　System Component Overview

This wireless temperature sensor network application was created as an example of a wireless application using the combination of MSP430 microcontroller, a CC2500 low-power wireless radio from Texas Instruments, and the SimpliciTI network protocol. Specifically, the Wireless Sensor Monitor leverages two existing solutions to implement the application:

1. eZ430-RF2500 design kit
2. SimpliciTI network protocol

1.1.1　eZ430-RF2500

The eZ430-RF2500 is a complete USB-based MSP430 wireless development tool that provides all of the hardware and software to evaluate the MSP430F2274 microcontroller and CC2500 2.4-GHz wireless transceiver. The eZ430-RF2500T target board is an out-of-the-box wireless system that may be used with the USB debugging interface, as a stand-alone system with or without external sensors, or incorporated into an existing design. The new USB debugging interface enables eZ430-RF2500 to remotely send and receive data from a PC using the MSP430 application UART, referred to as the application backchannel.

eZ430-RF2500 features include:

- USB debugging and programming interface featuring a driverless installation and application backchannel
- 21 available development pins
- Highly integrated, ultra-low-power MSP430 MCU with 16-MHz performance
- Two general-purpose input/output (GPIO) digital I/O pins connected to green and red LEDs for visual feedback
- Interruptible push button for user feedback [1]

The battery pack with the expansion board is used to remotely run firmware on an eZ430-RF2500T target board (see Figure 1). For more specific information on the eZ430-RF2500, visit the Texas Instruments website www.ti.com/ez430-rf or see the *eZ430-RF2500 User's Guide* (SLAU227). [1]

The eZ430-RF2500 is available for order from the TI eStore (http://ti.com/appnote-ad-slaa378-rf2500).
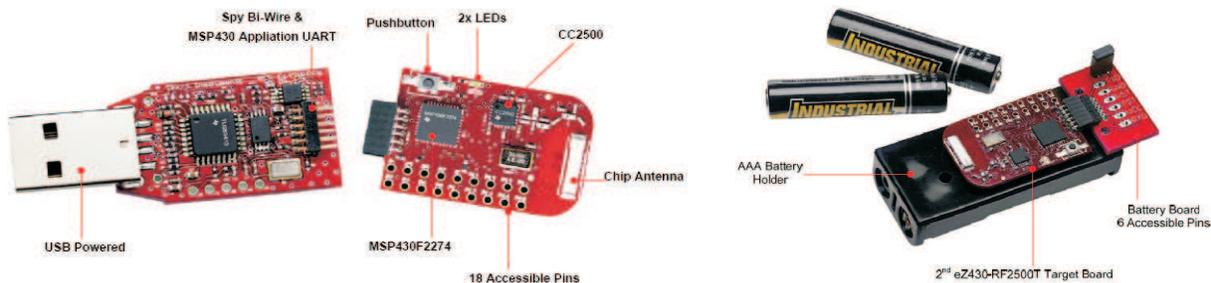

Figure 1. eZ430-RF2500 Development Kit Components

Submit Documentation Feedback

1.1.2 SimpliciTI Network Protocol

SimpliciTI is a proprietary, low-power radio-frequency (RF) protocol targeting simple, small RF networks. The SimpliciTI network protocol was designed for easy implementation with minimal microcontroller resource requirements. The protocol runs out of the box on TI's MSP430 ultra-low-power microcontrollers and multiple RF transceivers.

Small, low-power RF networks typically contain battery-operated devices, which require long battery life, low data rate, and low duty cycle, and have a limited number of nodes communicating directly with each other. With the SimpliciTI network protocol, MCU resource requirements are minimal, resulting in lower system cost for low-power RF networks.

Despite the modest resources required, the SimpliciTI network protocol supports end devices in a peer-to-peer network topology, the option to use an access point to store and forward messages, and range extenders to extend the range of the network.

The SimpliciTI network protocol can be used in a wide range of low-power applications including alarm and security (smoke detectors, glass breakage detectors, carbon monoxide sensors, and light sensors), automated meter reading (gas meters and water meters), home automation (appliances, garage door openers, and environmental devices), and active RFID.

The SimpliciTI network protocol is provided as source code under a free license without royalties. For information on compatibility, updates, and the latest version of the SimpliciTI protocol, visit www.ti.com/simpliciti. [2] Note that this application report has been written using SimpliciTI v1.1.1.

1.2 Network Overview

Although identical, the two target boards that come with the eZ430-RF2500 kit come preprogrammed with distinct firmware to exist as members of the Wireless Sensor Monitor network. A SimpliciTI Access Point (AP) manages the network and is always on, receiving sampled data from one or more SimpliciTI End Devices (ED) once per second. The EDs of the Wireless Sensor Network contain the sensors that implement the end-application for the network and spend most of their time in low power mode 3 (LPM3), waking up once a second to sample their ambient temperature and battery voltage and send the results to the network's AP. Upon receiving the data from any ED on the network, the AP sends it through its application UART to a COM port on the PC for presentation by the Network Visualizer GUI. To learn more about setting up the serial communications interface using the Network Visualizer GUI, see Appendix B.

1.2.1 Access Point (AP)

The first task an AP executes is the transmission of a startup splash to the COM port.

Figure 2. Application Splash Screen

The network AP is then initialized as the network hub. Upon completion of the initialization procedure, the AP transmits text notifying success:

Using the ADC10's internal temperature sensor, the AP begins to measure the ambient temperature once per second for transmission to the PC. In addition, the AP continuously listens for new EDs joining the network and for packages from EDs that have already joined. Using two indicator LEDs, an AP notifies the user of the two transactions in the network: a red LED indicates the transmission of the AP's measurements to the PC; a green LED indicates the receipt of a packet from one of the network EDs.

1.2.2 End Device (ED)

On startup, the ED immediately begins searching for an AP to which to connect. While searching, both the green and red LEDs toggle on/off. Upon discovery of an AP, the ED attempts a network link, flashing the red LED to indicate the link attempt. If it cannot link to the AP, it continues to flash the red LED. Once connected to the network AP, all LEDs are turned off, and the ED defaults to LPM3, toggling the green LED only when active.

1.3 Modes of Operation

There are two display options specified for the AP firmware's communication to the PC COM port, each of which has two possible modes of operation. The user must select one of the two modes per display option. Each mode can be entered by sending a character to the AP through the COM port connection. Characters are not case sensitive. The four characters are:

Temperature Display Option 1
>   C - Output all temperatures in degrees Celsius
>   F - Output all temperatures in degrees Fahrenheit

Data Format Display Option 2
>   V - Output all data in extended Verbose mode
>   M - Output all data in shortened Minimal mode

1.3.1 Verbose Mode

The following is an example of the output from an AP in Verbose mode:

```
Node:HUB0,Temp: 91.2F,Battery:3.5V,Strength:000%,RE:no
```

Node:	This is the device identifier for the hub; it is an assigned integer based on the order in which an ED has joined the network. The AP is given the identifier "HUB0".
Temp:	This is the temperature measured by the node. This can be in either degrees Celsius or Fahrenheit, as specified by the Temperature Display Option.
Battery:	This is the voltage of the power supply as measured by the ADC10 on the MSP430.
Strength:	This is the measured received signal strength indicator (RSSI) given by the CC2500 radio. It is output as a percentage for readability.

> **NOTE:** AP RSSI is always output as 000%.

RE:	This indicates whether a received packet has gone through a range extender (RE).

> **NOTE:** The AP firmware does not check for this, so its value is always No.

1.3.2 Minimal Mode

The following is an example of the output from an AP in minimal mode:

```
$HUB0, 89.6F,3.5,000,N#
```

This mode transmits the data with minimal formatting to reduce bandwidth usage. Its primary purpose is for parsing by the included PC Network Visualizer Application. The output in minimal mode contains the same information as in Verbose mode, in the same order. Data is comma separated and has a start character ($) and an end character (#).

2   Application Firmware

2.1   Access Point (AP)

```
                            BSP_Init()

                          Initialize MSP430

                          Transmit splash
                            screen and
                        "Initializing Network"

                         Initialize network
                          SMPL_Init(sCB)

                              sJoinSem,
                          sSelfMeasureSem,
        sJoinSem         or sPeerFrameSem?          sPeerFrameSem

                            sSelfMeasureSem

    Listen for a link       Read temperature        Define input
    SMPL_LinkListen()         (ADC10)              message buffer

    sNumCurrentPeers++;       Read voltage
                               (ADC10)

    Decrement join        Format ADC10 samples      Process all waiting
    semaphore               into a message            frames
                           for transmission

                         Transmit message to PC
```
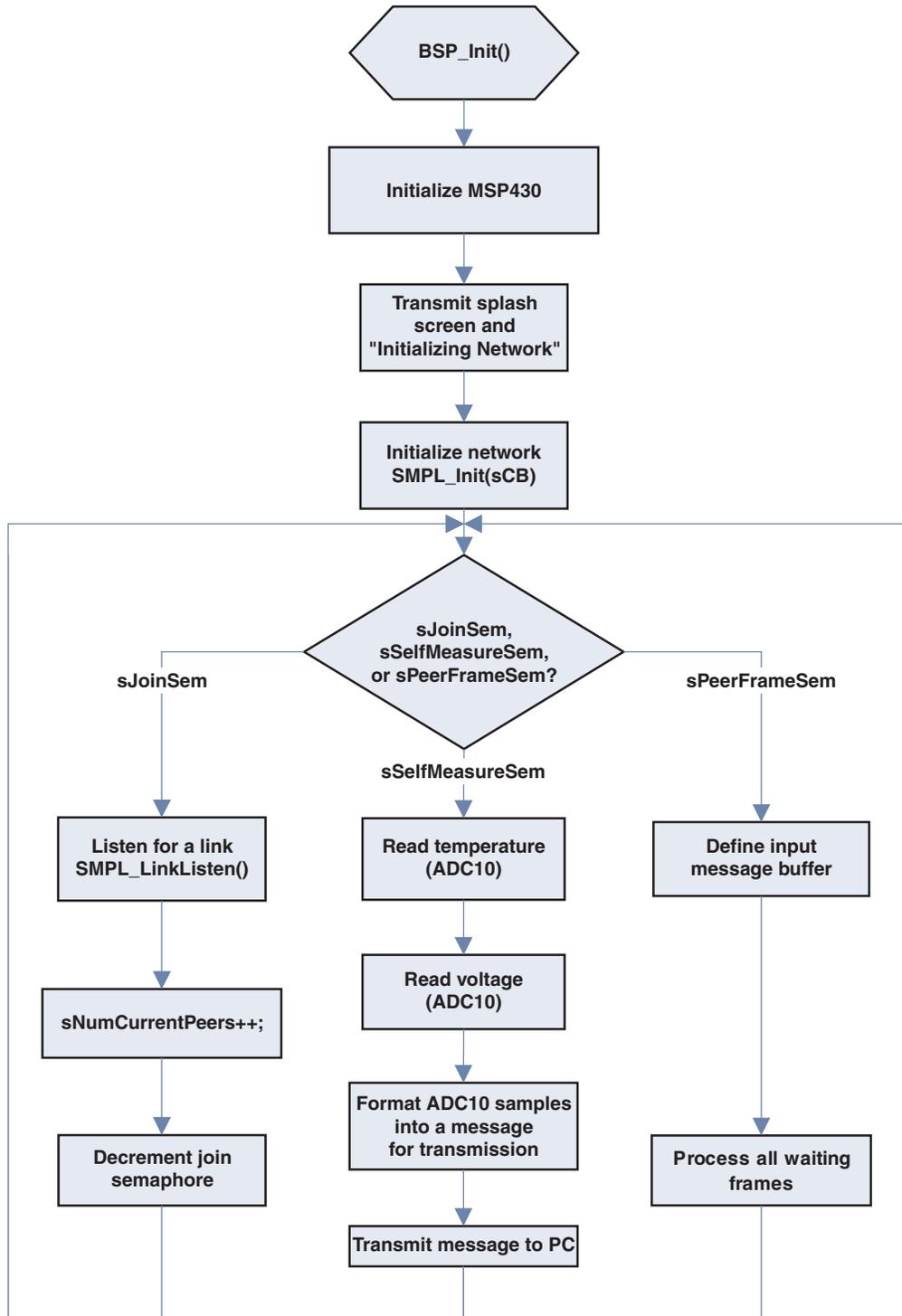
Figure 3. Flowchart for demo_AP.c

The BSP_Init() SimpliciTI API call initializes the board-specific hardware interfaces such as the LEDs/switches on the board that are to be used in the application.

The MCU_Init() function performs further MSP430-specific initializations for the application. These include:

- The DCO and MCLK are set to run at 8 MHz.
- Timer_A is set to trigger interrupts at 1-second intervals.
- The universal serial communication interface (USCI) UART is initialized to communicate with the PC COM port to 9600 Baud and RX/TX; interrupts enabled.

Once the hardware initialization is complete, the TI splash screen is transmitted to the COM port on the PC and the program calls the SMPL_Init(sCB) function to initialize the radio and the SimpliciTI network protocol. The sCB parameter is a function pointer to a callback function that is executed within the interrupt service routine (ISR) upon packet reception by the AP (see Example 1).

Example 1. sCB Callback Function

```
 /*------------------------------------------------------------------------*/
* Runs in ISR context. Reading the frame should be done in the
* application thread not in the ISR thread.
/*------------------------------------------------------------------------*/
static uint8_t sCB(linkID_t lid)
{
  if (lid)
  {
    sPeerFrameSem++;
  }
  else
  {
    sJoinSem++;
  }
  // leave frame to be read by application.
  return 0;
}
```

The sCB callback function filters the received packet according to its link ID to identify the source of the transmission and distinguish an ED join request from a data packet transmission from an ED that has already established a connection to the network. A link ID of 0 identifies a join request. Upon acceptance of an ED join request, the AP and the ED must then create a link for communication. Each link to a new device is enumerated so the AP assigns incremental link IDs as devices join the network. Due to application considerations, the Wireless Sensor Monitor allows a maximum number of eight devices to be linked to the AP.

According to the link ID, the sCB callback function identifies and increments either the sPeerFrameSem or sJoinSem semaphores. The AP code also defines a sSelfMeasureSem semaphore, set by the Timer A interrupt once a second, so that the AP knows to sample its own temperature and battery voltages for display. These three semaphores control the program flow after network initialization. Take particular note of the return value of the sCB function. A return value greater than zero indicates to the SimpliciTI protocol that the callback function has handled the received frame and releases the frames memory for reuse. The sCB callback function in demo_AP.c returns zero because the firmware leaves the received frame in the input buffer to be handled by the application to reduce the amount of time spent in the ISR context of the SimpliciTI protocol. When a device is expected to maintain a multitude of links to other applications/nodes in the network, it is especially important to keep code in ISRs small to minimize the risk of losing packet reception and notification.

2.1.1 sJoinSem Branch

The sJoinSem semaphore in demo_AP.c is set in the sCB callback function every time that an ED calls its SMPL_Init() function and, as a result, attempts to join the network. A join to the network is really a side-effect of initialization; there is not a SimpliciTI API call to request a network join. When the ED application calls SMPL_Init(), SimpliciTI sends a Join frame with a link ID of zero, informing the AP to increment the sJoinSem semaphore in the callback function.

When the AP callback function then returns to the main loop, and as long as the AP has made less than its maximum number of links (as specified in smpl_config.dat), the AP then initiates a SMPL_LinkListen(). The SMPL_LinkListen() function takes a pointer to the link ID that is be used to identify all messages from the node requesting the link. SMPL_LinkListen() is a blocking call with a timeout parameter (introduced in SimpliciTI 1.0.5), meaning it does not return until a successful link has been created or time has expired, so it is important that the SMPL_LinkListen() call be made only when the user knows that another device has made the link request using the SMPL_Link() function.

On a successful link creation, the sJoinSem branch increases the number of devices that the AP recognizes as part of the network and unlocks the sJoinSem semaphore for another device to set.

2.1.2 sPeerFrameSem Branch

The sPeerFrameSem semaphore is incremented in the sCB callback function every time that the AP receives a frame from a peer application. In the case that the sPeerFrameSem semaphore has been set or incremented, the AP first defines a message buffer in which to store the current frame being analyzed and then loops through its input queue searching for messages until it has processed all of the waiting frames.

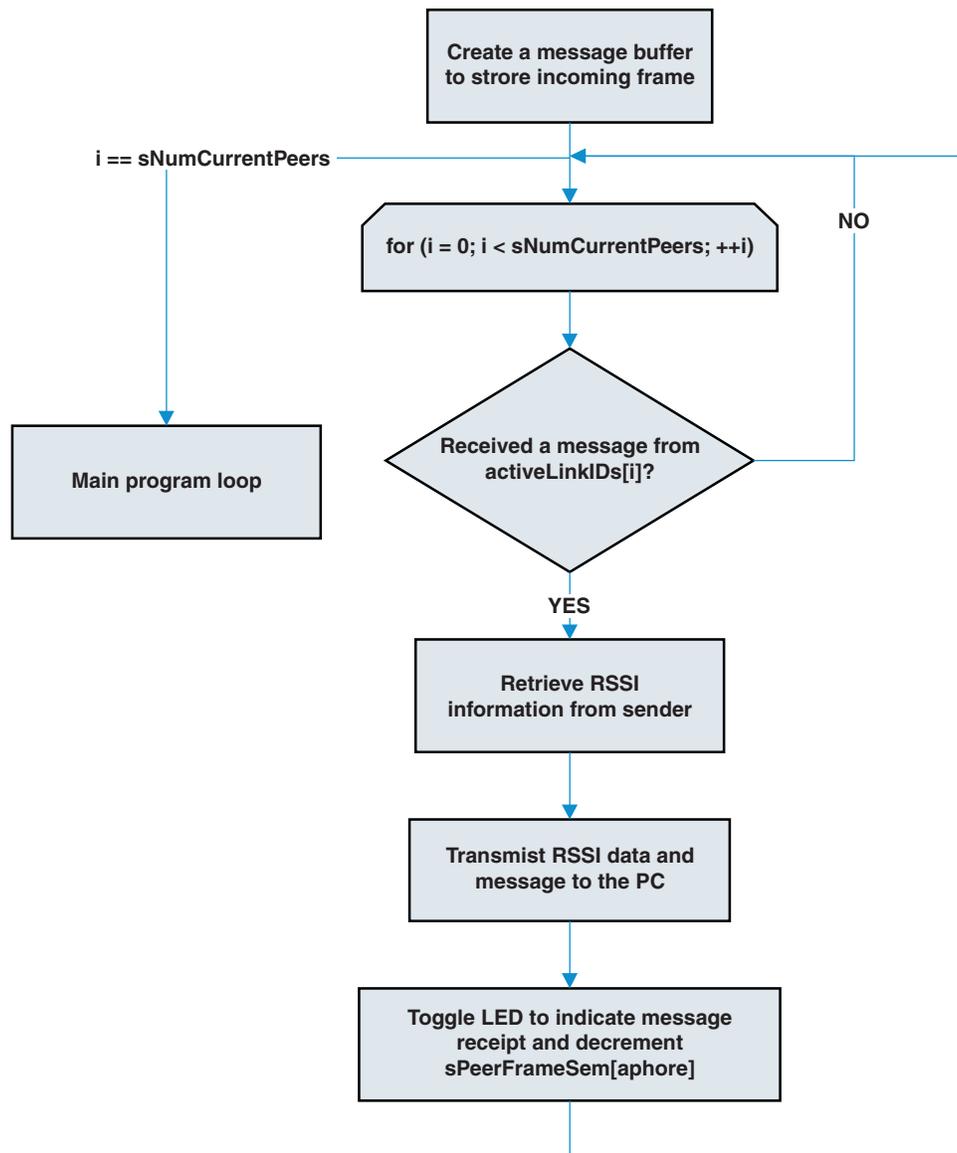The flowchart in Figure 4 shows how the application handles messages from its peers.

Figure 4. Flowchart for Peer Frame Handling

2.1.3 sSelfMeasureSem Branch

The sSelfMeasureSem semaphore is specific to this demo application in that it is set at a user-specified time interval to execute an application-layer routine. It is a prime example of how easy it is to combine an eZ430-RF2500 user application like temperature and voltage sampling with the SimpliciTI RF protocol by threading the requirements of both applications - ADC10 sampling at one-second intervals (application) and network management/peer-frame handling (SimpliciTI protocol) - into a very low level task manager.
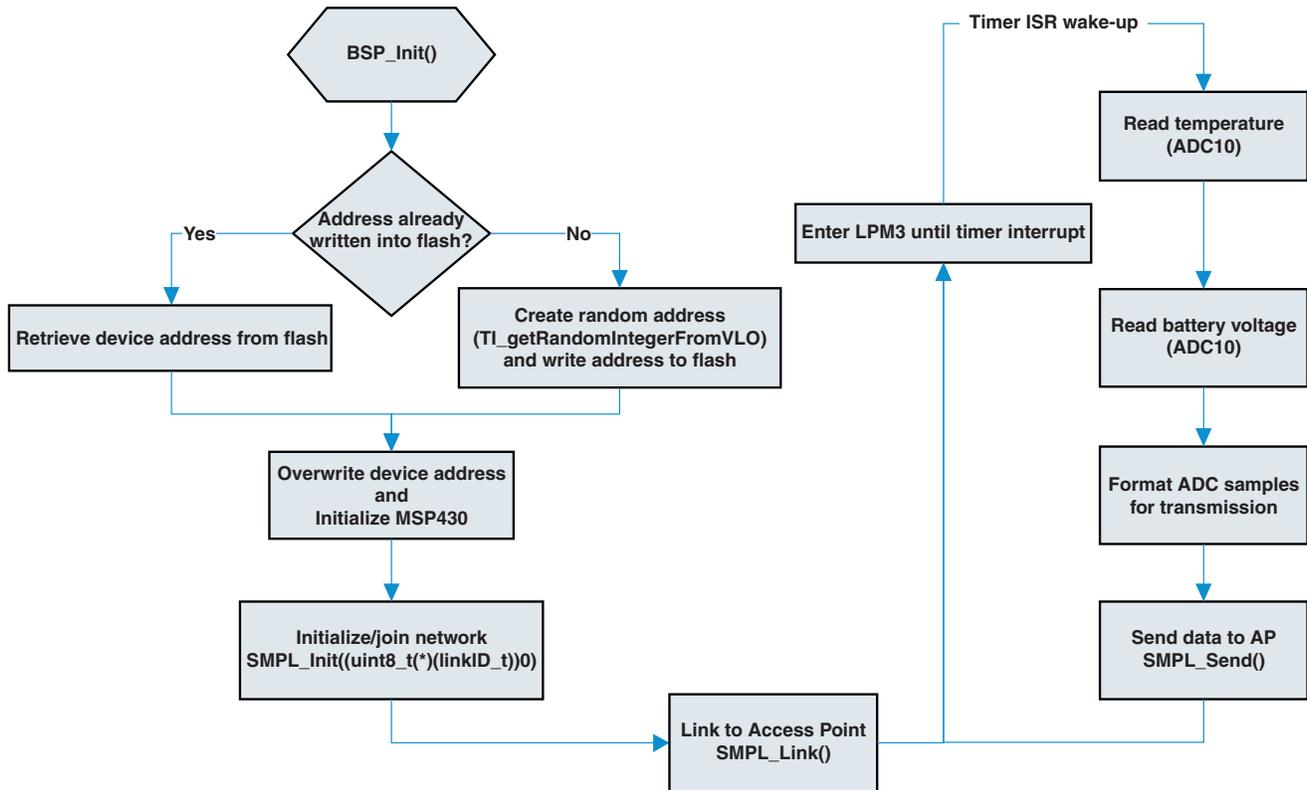
2.2 End Device (ED)

Figure 5. Flowchart for demo_ED.c

The initialization of the ED device is similar to the AP in that both devices use the SMPL_Init() call to initialize onto the network. The initialization procedure is different for the ED, however, in two important ways: the generation of the device address at runtime and the parameter passed to the SMPL_Init() function.

As part of the initialization procedure, EDs in this application create a random 4-byte address, write that address into flash memory for reuse on system reset, and then write over their default build-time device address using the SMPL_Ioctl() API. Because a SimpliciTI AP identifies new devices on the network by their device addresses, storing this randomly-generated address in flash and checking this predefined location at device initialization assures that an ED that has lost power or is reset is always recognized as the same device (i.e., is given the same link ID) by the AP.

The random address is created using the results from the TI_getRandomIntegerFromVLO function inside the vlo_rand.s43 library file provided with the project. This library uses the rising edges of the very low frequency oscillator clock found in 2xx devices to trigger samples of a system clock that are then interpreted into a 4-byte device address. By changing the frequency of the system clock between triggers, the randomization of resulting device addresses is increased, and the user can be confident that two devices will not create the same network address. For more information on the random number generation library, see *Random Number Generation Using the MSP430* (SLAA338). [6]

After writing over the build-time device address, the SMPL_Init() call is then made, initializing the ED onto the AP's network by checking for matching join tokens and exchanging network parameters such as the link token and whether or not the ED is a polling device. The ED joining the AP-managed network is, therefore, a side effect of the SMPL_Init( ) call; there is no such thing as a "join" API call.

The SMPL_Init( ) function does not take a callback function pointer – as in the case of the AP – because it does not need to be notified when a packet has arrived from the AP. The AP does not send packets to the ED. A callback function is used when the ED firmware wants to be informed that a packet has been received to immediately retrieve/process the message. It is also possible to handle messages from the AP without a callback function by using SMPL_Receive( ) to poll the input frame queue until the call returns a successful value. In both cases, SMPL_Receive( ) is used only to poll the input frame queue for the received message, but the latter solution is less reactive than the former, as the code spends its time polling the input frame queue rather than executing other application tasks. Therefore, the recommended program flow for a non-polling ED to receive messages from the AP is to use a callback function that sets a semaphore, informing the code that a message has been received and that it needs to call the SMPL_Receive( ) API to retrieve/process the message.

This brings to light an important difference between calling the SMPL_Receive( ) API when an ED is a polling versus non-polling device. An ED can be configured as a polling ED by defining the macro, RX_POLLS, in the smpl_config.dat file. Defining the RX_POLLS macro changes the behavior of the SMPL_Receive( ) call such that a call to the SMPL_Receive( ) transmits a query to the AP for pending messages – changing the state of the radio. Upon receiving the query from a polling ED, the AP responds with the pending message or, if no message is being held, a frame with no payload. This is called store-and-forward messaging and is not used in this example. A call to SMPL_Receive( ) when the device is non-polling does not change the state of the radio, but rather polls the local input buffer for a previously-received packet. The ED nodes in this application report are non-polling. For further information on device configurations, see the SimpliTI documentation. [2]

Once the SMPL_Init( ) call returns successfully – meaning the ED has verified its join token and is now allowed onto the AP's network – the ED calls the SMPL_Link( ) API to create the explicit link required to exchange packets of information. Only after the ED has both joined the network [SMPL_Init()] and created a link to the AP [SMP_Link( )] does it enter the main while loop, waking up once per second to sample and communicate its ambient temperature/battery voltage.

3 Performance Overview

The memory requirements of the built code are shown in Table 1. The build with IAR Embedded Workbench for MSP430 KickStart v5.10.1 used optimization settings set to Size → High. The build with Code Composer Studio v4.1.2.00027 used optimization: 3 (highest) and speed/size tradeoff: 0 (size).

Table 1. Memory Requirements for Wireless Sensor Monitor v1.04

Compiler	Version	ROM (bytes)	RAM (bytes)
Code Composer Studio	Access Point	10554	875
	End Device	8086	293
IAR	Access Point	11816	790
	End Device	9100	505

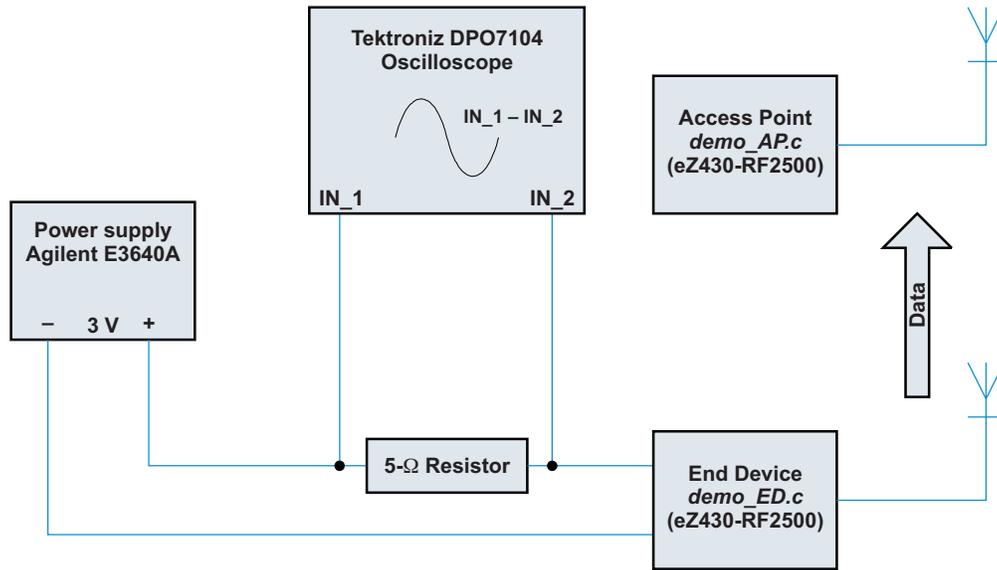To analyze the current profile of the application, the hardware setup in Figure 6 was used:

Figure 6. End Device Test Platform

The oscilloscope shot in Figure 7 shows the current profile of an ED over five seconds. EDs send data to the AP once per second. By decreasing the duty cycle for data transmission as much as possible, the radio and MCU are active for minute amounts of time, allowing an ED to run for long periods of time on the same batteries.
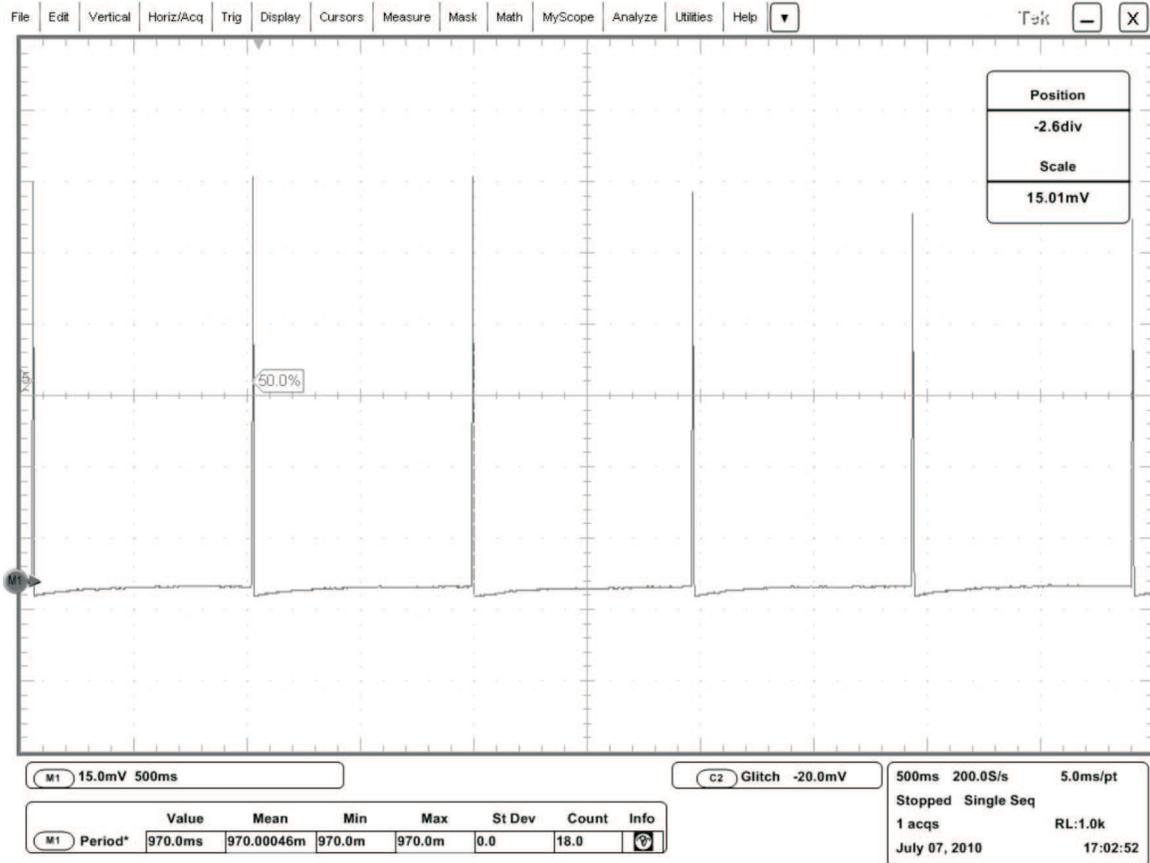

Figure 7. End Device Five-Second Current Profile

Figure 8 shows one of the spikes after decreasing the time step and sampling in high-resolution mode. At this time scale, the waveform can be more closely analyzed for specific events in hardware or software and their respective power contributions.
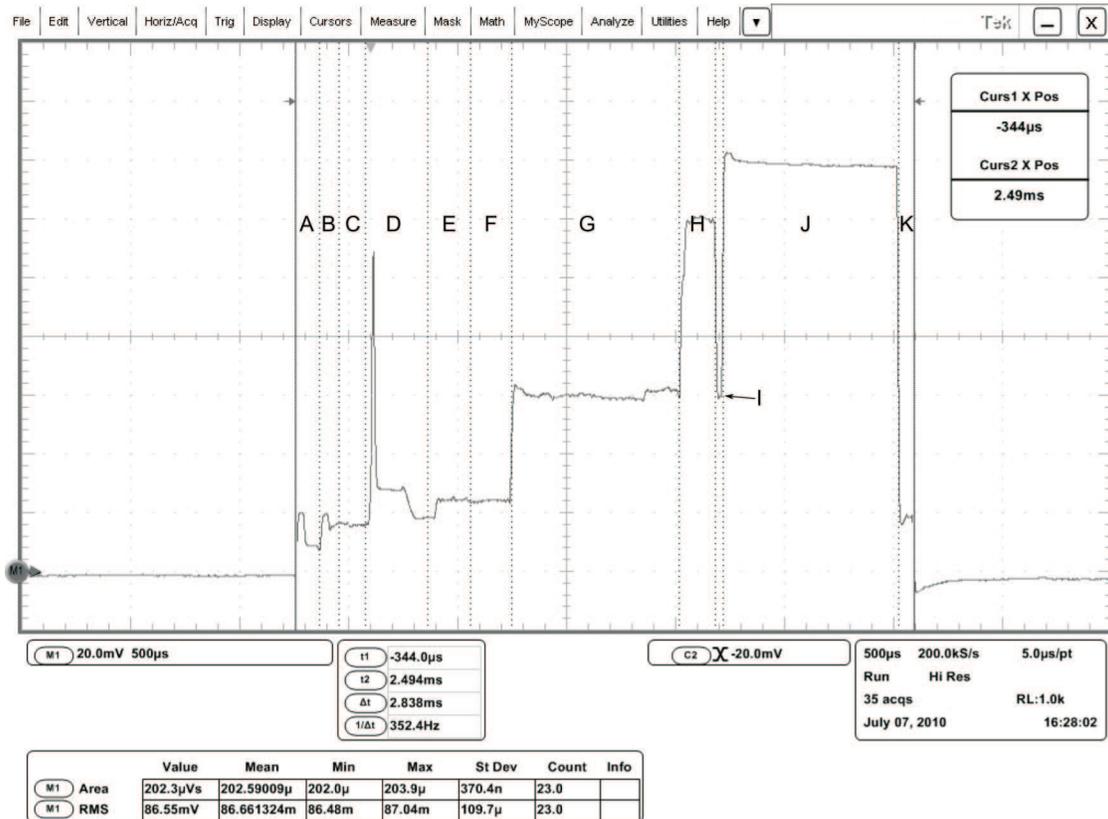

Figure 8. End Device Transmission Current Profile (AP ACK Disabled)

Figure 9 is the current profile when automatic access-point acknowledgments and frequency agility are enabled. With automatic acknowledgment enabled, the end device waits for an acknowledgment packet to be sent from the AP to confirm the data was successfully transferred. With frequency agility enabled, the end device automatically switches RF channels when data is not acknowledged. These features make data transfer more robust, especially when packet loss is critical, i.e., at very low transmit periods or in noisy environments. However, the current consumption is more than doubled due to increased time in RX mode on the CC2500.
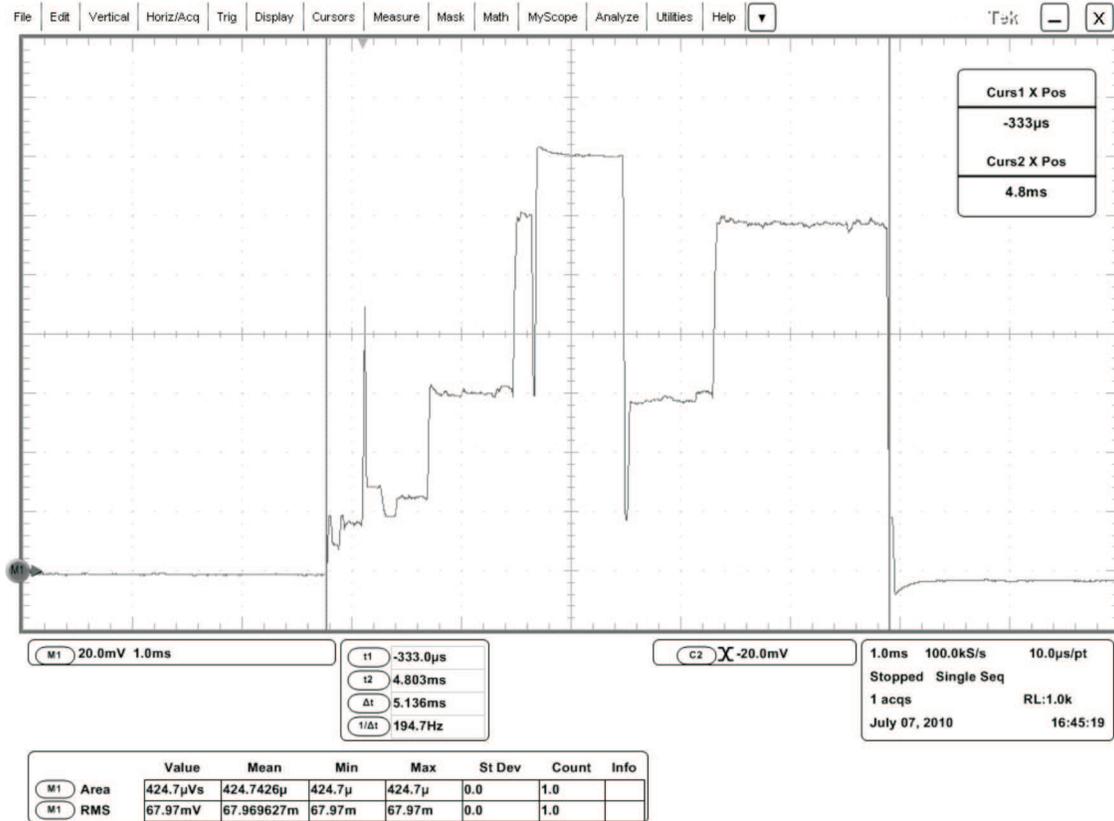

Figure 9. End Device Transmission Current Profile (AP ACK Enabled)

> **NOTE:** Both APP_AUTO_ACK and FREQUENCY_AGILITY defines are disabled (commented out) by default. To enable them, remove the comment in smpl_nwk_config.dat. To enable FREQUENCY_AGILITY, you must also enable APP_AUTO_ACK.

Table 2 and the following power consumption discussion describe the most significant contributors to power and program flow in both hardware and software. Other sources of current consumption, such as the USCI current consumption, do exist in the application but are not highlighted in its analysis due to their relatively small significance in the overall current consumption.

NOTE: The code to toggle the LED has been commented out for the power consumption calculation and that the typical value for the ADC10OSC of 5 MHz is used for calculation.

Table 2. Significant Current and Timing Contributions

Hardware	Current Value	Unit	Software	Time for Execution Value	Unit
Sleep Modes			**Oscillator (D in Figure 8)**		
MSP430 low-power mode 0 (LPM0) [4]	1.1	mA	XOSC startup time [1]	300	µs
MSP430 low-power mode 3 (LPM3) [4]	900	nA	**Ripple counter (E in Figure 8)**		
CC2500 sleep state [5]	400	nA	Timeout before CHP_RDY Hi→Lo [5]	150	µs
MSP430 Active Mode			**PLL (G in Figure 10)**		
8 MHz = DCO = SMCLK, 3 V [4]	2.7	mA	Calibration of radio frequency synthesizer [5]	809	µs
MSP430 ADC10			**Temperature sample (A in Figure 8)**		
$f_{ADC10CLK}$ = 5.0 MHz, ADC10ON = 1, REFON = 1, REFOUT = 0, ADC10DIV = 0x4 (ADC10CLK / 5) [2]	850	µA	ADC10 REFON + Delay for Stabilization [4]	30	µs
			LPM0 + Sync + Temp Sample (64 × ADC10CLKs) + Conversion (13 × ADC10CLKs) [7]	48	µs
CC2500 Active Modes			**V_{CC} sample (B in Figure 8)**		
Idle [5]	1.5	mA	ADC10 2.5-V REFON + delay for stabilization [4]	30	µs
PLL calibration [5]	7.4	mA	LPM0 + Sync + V_{CC} Sample (16 × ADC10CLKs) + Conversion (13 × ADC10CLKs) [5]	7	µs
Receive (RX) (weak input signal, DEM_DCFILT_FILT_OFF = 0, 250 kbps) [5]	18.8	mA	**Message handling overhead (C and F in Figure 8)**		
Transmit (TX) (250 kbps, 0-dB output power) [5]	21.2	mA	Calculate temperature and V_{CC} from ADC sample results (MSP430 active and radio idle) [1]	140	µs
			Transmit message to TX FIFO and prepare message TX (SMPL_Send) (MSP430 active and radio idle) [1]	140	µs
			RX/TX modes (H, I, and J in Figure 8)		
			RX mode (clear channel assessment) [1]	180	µs
			Switch between RX and TX mode [1]	30	µs
			TX mode (message transmission) [1]	800	µs
			Radio and MCU low-power modes (K in Figure 8)		
			CC2500 prepares for sleep; turns off 32-MHz crystal [2] (MSP430 active and radio idle)	70	µs

[1]   Measured using an oscilloscope

[2]   See the *CC2500 Low-Cost Low-Cost Low-Power 2.4 GHz RF Transceiver Data Sheet* (SWRS040): add I_{ADC10} to I_{REF+}.

Notice that the ADC10 sample times are significantly different. This is because the internal temperature sensor requires 30-µs sampling time, whereas the V_{CC} measurement requires only 1.4 µs. [4] Therefore, the ADC10 must be initialized differently for each measurement – different divisors and a different number of sampling cycles. To account for the ADC10's internal oscillator speed variance over temperature, these values must be sufficient to provide the required sample time at the ADC10OSC corner-case operating speed of 6.3 MHz. Hence, the sampling time in typical- and worst-case operating scenarios is longer than the minimum sampling time required for the measurement, as shown in Table 2. See the code for the recommended ADC10 initialization procedures.

The MSP430 contributions to program flow and power consumption are straightforward and can be traced through an analysis of the application-level firmware. The radio events, however, are abstracted from the user by design and often occur by default, executed by hardware and invisible to the programmer.

The radio events are denoted using the dotted lines in the chart, whereas MSP430 events are not explicitly shown. Events A and B occur every time that the radio is awakened from sleep into idle mode by a SMPL_Ioctl(IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_AWAKE, 0) call. Events C and D occur every time that the radio is put into a receive state from IDLE.

1. XOSC startup – D

   XOSC is the CC2500 oscillator used to source the chip's system clock.

2. Ripple Timer Timeout – E

   A setting inside the radio's configuration registers specifies how many times a ripple counter must timeout after a successful XOSC startup routine before signaling the CC2500 chip ready symbol (negative edge on CHP_RDY). In this case, the requirement is 64 timeouts, or 150 µs.

3. IDLE → RX Mode + PLL calibration – G

   Upon changing states from IDLE to either RX or TX modes, the PLL (the on-chip frequency synthesizer used for RX and TX (de-)modulation) is automatically calibrated according to a setting inside the radio's configuration registers. This frequency synthesizer must be calibrated regularly, and takes 809 µs to enter RX or TX mode from the IDLE radio state.

4. RX Mode – H

   Receive mode is necessary for a successful transmission so that a Clear Channel Assessment (CCA) can be completed before transmission. A CCA check is done to verify that the radio is not currently receiving a packet and that another signal on the channel is not registering an RSSI value over a certain threshold (in effect, it checks whether another radio is already transmitting on the channel of interest). If the channel is occupied, SimpliciTI backs off for a random time and perform CCA again. When the CCA check is complete, the radio can then switch to transmit (TX) mode and transmit its intended application payload. Through this procedure, SimpliciTI maintains a listen-before-talk, or Carrier Sense Multiple Access (CSMA) algorithm for communication between nodes.

   For more information on CC2500 settings, see the CC2500 Low-Power RF Transceiver data sheet [5].

Due to these four radio requirements, it is important to understand that the transmission of the data (time spent in TX mode) is only partly responsible for the majority of total current consumption drawn. Much of the total current consumption is tied directly to the overhead incurred from the CC radio and MSP430 initializing each transmission. It is, therefore, important to maximize the amount of data that can be sent per transmission and minimize the number of transmissions per unit of time.

Table 3 calculates the expected current consumption for the application. The radio and MSP430 components of the application have been separated.

Table 3. Expected Current Consumption

Radio Event	Current Consumed	Time Executed	Amps * Seconds Consumed
XOSC startup	2.7 mA[1]	300 µs [1]	810 nA*s
Ripple counter timeout	1.75 mA	150 µs	262 nA*s
Idle mode	1.5 mA	375 µs	563 nA*s
PLL calibration	7.5 mA	809 µs	6067 nA*s
RX mode	18.8 mA	180 µs [1]	3384 nA*s
TX mode	21.2 mA	800 µs [1]	16 960 nA*s
		Total	**28 047 nA*s**
MSP430 Event	**Current Consumed**	**Time Executed**	**Amps * Seconds Consumed**
MSP430 active current	2.7 mA	2.705 ms	7304 nA*s
MSP430 LPM0 current	1.1 mA	55 µs	61 nA*s
ADC10 active	850 µA	115 µs	98 nA*s
		Total	**7463 nA*s**
		Transmission Total	**35 508 nA*s**

[1] Estimated from oscilloscope measurement

The average current calculation must also take into account the sleep current of an ED. The less frequently that an ED transmits, the more significant that the sleep current's contribution to the overall average becomes.

$sleep\_current\_contrib = (MSP430\ idle\ current + CC2500\ idle\ current) \times (period\_of\_transmission - application\_execution\_time)$ (1)

= 1.3 [µA] * (1 [s] – 2.838 [ms])
= 1.296 µA*s

The average current consumption for the application can then be derived using the following equation:

$average\_current\_consumption = (sleep\_current\_contrib + transmission\_total) / period\_of\_transmission$ (2)

$average\_current\_consumption_{EXPECTED}$ = (1.296 [µA*s] + 35.508 [µA*s]) / 1 [s]
= 36.80 µA

For comparison, the second method used to calculate the average current consumption – integrating the curve using an oscilloscope – resulted in an area under the transmission waveform of 184 µV*s. Divided by the real resistor value of 5.92 Ω and added to the sleep_current_contrib from equation (1), the average measured current consumption is derived as:

$average\_current\_consumption_{MEASURED} = (measured\_voltage\_reading / 5\ \Omega) / period\_of\_transmission$ (3)

= (202 [µV*s] / 5.92 Ω) + sleep_current_contrib) / 1 [s]
= (34.12 [µA*s] + 1.296 [µA*s]) / 1 [s]
= 35.42 µA

Given the simplification of the current analysis and the accuracy of various measurement tools and the accuracy of various measurement tools, the slight difference between the two numbers is acceptable. To calculate the life expectancy of an ED on the network, and assuming two typical AAA batteries with a capacity of 1000 mA*hr under the hypothetical condition in which the batteries hold their voltage ideally until their capacity is exhausted:

$hours\_of\_operation = current\_rating / average\_current$ (4)

Calculated life expectancy:

= 1000 [mA*hrs] / 0.03680 [mA] = 27173 [hrs] / 24 [hrs/day] = 1132 [days] / 365 [days/yr] = 3.10 years
= 3 years, 1 month, and 6 days

Measured life expectancy:

= 1000 [mA*hrs] / 0.03542 [mA] = 28232 [hrs] / 24 [hrs/day] = 1176 [days] / 365 [days/yr] = 3.22 years

= 3 years, 2 months, and 20 days

Figure 10 charts the expected years of operation due to power consumption of an ED node transmitting the current application payload at different time intervals, verifying that to minimize the power consumption of an application, a programmer should always:

1. Minimize the number of transmissions
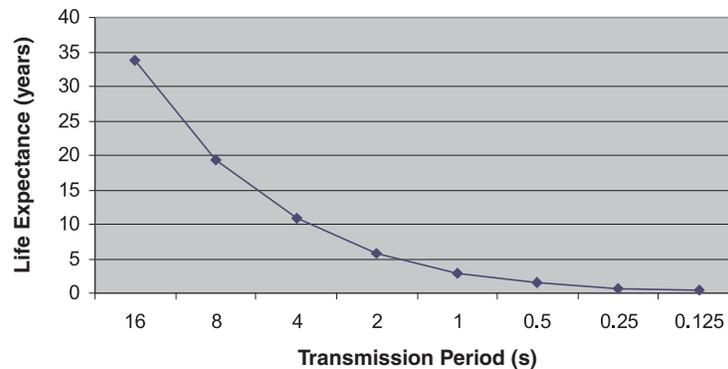2. Fit as many bytes into the transmission packet as is feasible for the application.

Figure 10. Years of Operation vs Transmission Interval

It should be noted that in the real world, the application life expectancy is lower due to battery leakage, battery voltage dissipation, and the requirements of the application itself (e.g., 2.5-V V_{CC} for the ADC10 reference). To mitigate these effects, one could select batteries that hold their specified voltage for as long as possible and implement a function that changes the ADC10 voltage reference down to 1.5 V once the battery voltage reaches 3 V. This function would extend the usable battery life by decreasing the required voltage for proper ADC10 operation down to the minimum ADC10 voltage of 2.2 V specified in the MSP430F2274 data sheet. [4]

Using this document, the supporting documentation, and the Wireless Sensor Monitor firmware, the reader now has all the tools needed to integrate a low-cost easy-to-use wireless solution into existing or developing applications. For more information and ideas on how to use the eZ430-RF2500 to evaluate or implement your wireless solution, see http://www.ti.com/ez430-rf.

4   References

1. *eZ430-RF2500 User's Guide* (SLAU227)
   The eZ430-RF2500 is available for order from the TI eStore (http://ti.com/appnote-ad-slaa378-rf2500).
2. SimpliciTI Developer's Notes located in *SimpliciTI-IAR-1.1.1* (SWRC099)
3. *Measuring Power Consumption With CC2430 and Z-Stack* (SWRA144)
4. *MSP430x22x2, MSP430x22x4 Mixed Signal Microcontroller Data Sheet* (SLAS504)
5. *CC2500 Low-Cost Low-Cost Low-Power 2.4 GHz RF Transceiver Data Sheet* (SWRS040)
6. *Random Number Generation Using the MSP430* (SLAA338)
7. *MSP430x2xx Family User's Guide* (SLAU144)

Appendix A Wireless Sensor Monitor FAQs

A.1 Which version of IAR is required to launch the eZ430-RF2500 Wireless Sensor Monitor v1.04 project?

IAR KickStart 5.12 or later is required. It includes support for the eZ430-RF2500 emulator and allows the inclusion of library files that exceed the KickStart 4-KB C-code limitation.

A.2 The project does not debug/run properly.

Before debugging and when switching between End Device and Access Point projects (or vice versa), clean the project by clicking Project → Clean.

A.3 Why does temperature vary among End Devices?

Calibration of the MSP430 ADC10 for temperature is application specific, and beyond the scope of this document. For this application, a simple 1 point ambient temperature calibration was deemed acceptable. Greater accuracy can be achieved using other methods such as 2-point calibrations in temperature controlled environments.

A.4 Why does my battery measure 3.5 V?

The battery measurement for this application is limited to 1 decimal point and rounds down. Therefore, if a 3.6-V battery is measured as 3.59 V, the End Device shows it as 3.5 V.

A.5 What is the maximum number of supported end devices?

Each Access Point can link to a maximum of eight end devices using the current project.

A.6 Can the network be extended beyond 30 end devices?

Yes, the device address for a device is 4 bytes long, so the hard-stop for possible devices on a SimpliciTI network is 2^{32} nodes. However, each node on the network can link to only 30 nodes. The network limitations are also very much dependent on the amount of RAM that a device contains. Each additional node on the network requires additional RAM for the input/output buffers and network management data structures. To allow plenty of room for further application development, this demonstration application was limited to eight nodes.

Appendix B Network Visualizer GUI

The user has the option of running the project from the CD that comes with the eZ430-RF2500 kit or downloading the project from the web at http://www.ti.com/lit/zip/slac139. The devices in an eZ430-RF2500 kit come preprogrammed with the Wireless Sensor Monitor v1.04 firmware and may be reprogrammed at any time. If the user experiences any issues on hardware or software installation, see the *eZ430-RF2500 User's Guide* for detailed explanations of project installs and instructions on how to update the application firmware.

An Access Point running the Wireless Sensor Monitor v1.04 firmware transfers its data to a terminal window through its backchannel UART at a fixed rate of 9600 bps. The user can open a terminal window to see the streaming data or can view a graphical representation of the network using the Network Visualizer GUI.

The Network Visualizer GUI reads the data coming in through the COM port and depicts the network in an easy-to-understand graphical format according to user settings. Figure 11 is a screenshot of the GUI using the two devices that come with the eZ430-RF2500 kit.
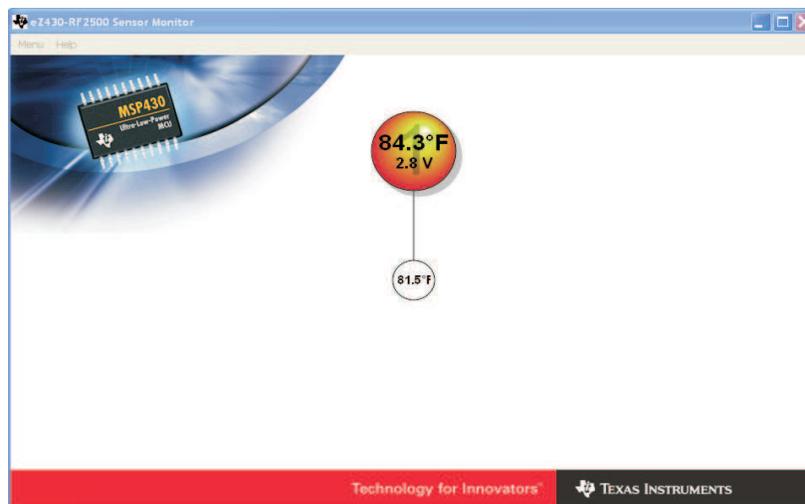

Figure 11. Network Visualizer Screen

The GUI displays the temperature and voltage readings that are transmit per device and can be run as-is by opening its executable file. It is possible to configure the GUI display by navigating to the Menu → Settings window, shown in Figure 12. In the Settings window, the user sets the port over which the AP is communicating, values for the color gradient of the EDs, temperature display units, and the resulting distance from the center node for the interpreted RSSI values.
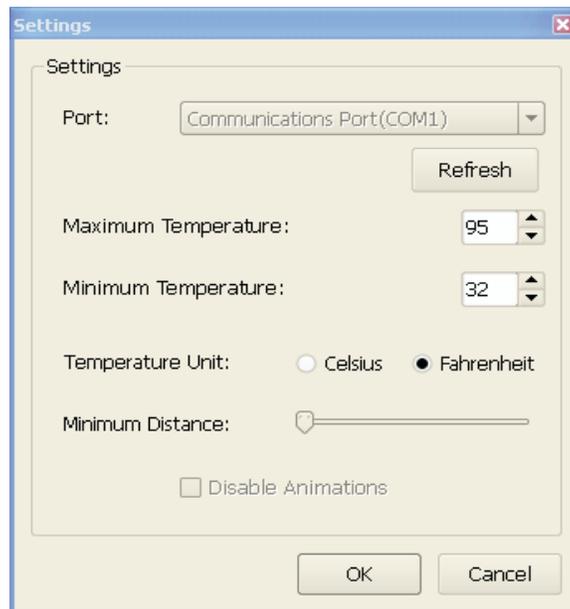
Figure 12. Network Visualizer Display Settings

It is also possible to see the data streaming through the selected COM port terminal by navigating to the Menu → Console window. An example console window is displayed in Figure 13. For further questions on the Network Visualizer GUI, see the Help menu.
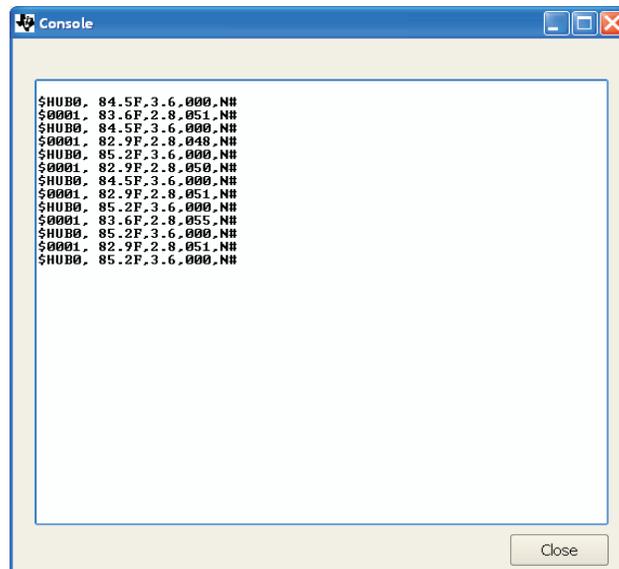

Figure 13. Network Visualizer Console Window

NOTE: Texas Instruments does not provide the source code for the Network Visualizer GUI.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

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

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

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

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

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Audio	www.ti.com/audio	Communications and Telecom	www.ti.com/communications
Amplifiers	amplifier.ti.com	Computers and Peripherals	www.ti.com/computers
Data Converters	dataconverter.ti.com	Consumer Electronics	www.ti.com/consumer-apps
DLP® Products	www.dlp.com	Energy and Lighting	www.ti.com/energy
DSP	dsp.ti.com	Industrial	www.ti.com/industrial
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Security	www.ti.com/security
Logic	logic.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Power Mgmt	power.ti.com	Transportation and Automotive	www.ti.com/automotive
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com	Wireless	www.ti.com/wireless-apps
RF/IF and ZigBee® Solutions	www.ti.com/lprf		

TI E2E Community Home Page   e2e.ti.com