Application Report
SLAA378 – November 2007

A Wireless Sensor Monitor Using the eZ430-RF2500

Miguel Morales 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 the SimpliciTI documentation on the web, respectively.

Contents

Figures

Tables

NOTE: Due to a change in IAR compiler calling conventions, the Wireless Sensor Monitor v1.0.2 only runs with the latest version of IAR Embedded Workbench KickStart™ v5.10, available online at http://focus.ti.com/docs/toolsw/folders/print/iar-kickstart.html. The code will NOT compile with KickStart v3.42 and will return a linker error.

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 SimpliciTI. 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 providing all 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 may be 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.

The eZ430-RF2500 features:

- 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 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 2). 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]

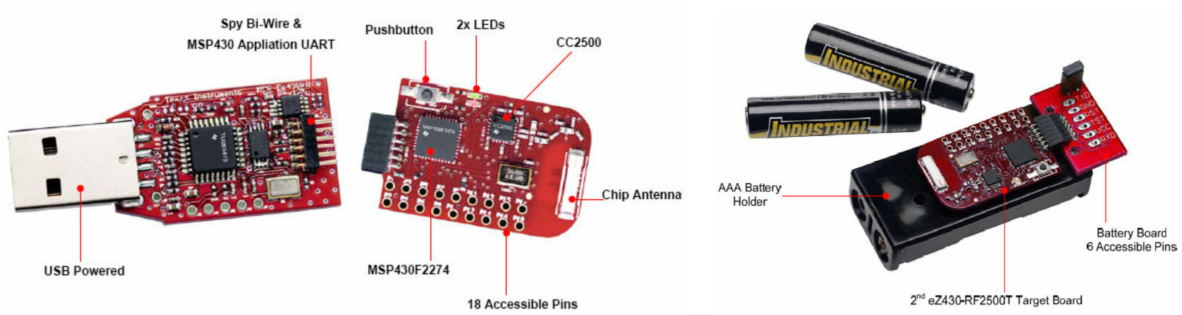


Figure 1. The eZ430-RF2500 Development Kit Components

1.1.2 SimpliciTI Network Protocol

SimpliciTI is a proprietary, low-power radio-frequency (RF) protocol targeting simple, small RF networks (<100 nodes). 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 talking directly to each other. With the SimpliciTI network protocol, MCU resource requirements are minimal, resulting in lower system cost for low-power RF networks. More complex mesh networks that need routing typically require 10× the program memory and RAM to implement.

Despite the modest resources required, SimpliciTI network protocol supports End Devices (EDs) in a peer-to-peer network topology, the option to use an Access Point (AP) to store and forward messages, and Range Extenders (REs) to extend the range of the network up to four hops. Future releases will add more sophisticated features such as frequency agility, an ETSI-compliant listen-before-talk discipline, and a software security routine for message encryption.

The SimpliciTI network protocol supports 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.

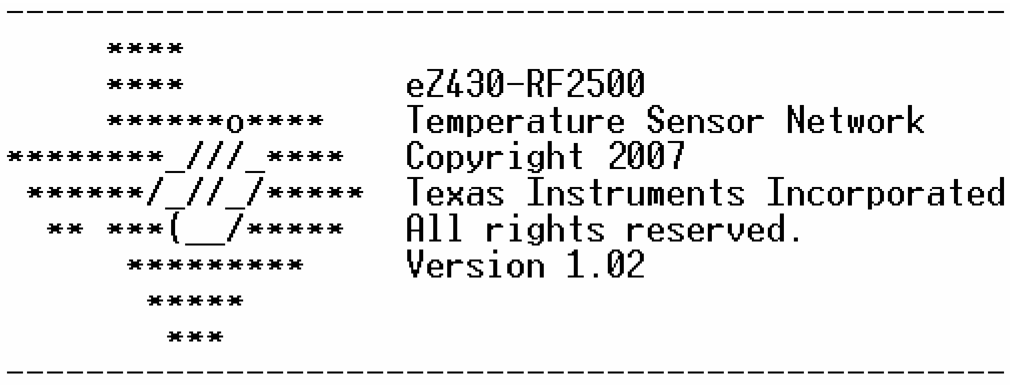
The SimpliciTI network protocol is provided as source code under a free license without royalties. Developers are encouraged to adapt the protocol to their own specific application needs. For information on compatibility, updates, and the latest version of the SimpliciTI protocol, visit www.ti.com/simpliciti. [4]

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

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

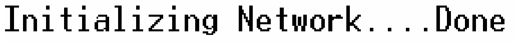


```
--------------------------------------------------

     ****
     ****           eZ430-RF2500
     ******o****    Temperature Sensor Network
********_///_****   Copyright 2007
 ******/_//_/*****  Texas Instruments Incorporated
  ** ***(__/*****   All rights reserved.
      *********     Version 1.02
       *****
        ***
--------------------------------------------------
```

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:



```
Initializing Network....Done
```

Using the ADC10's internal temperature sensor, the AP then 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 are 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's EDs.

1.2.2 End Device

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's 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

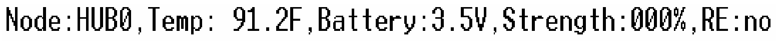
C - Output all temperatures in degrees Celsius
F - Output all temperatures in degrees Fahrenheit

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.

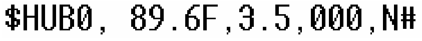
NOTE: AP RSSI is always output as 000%.

RE: This indicates whether a received packet has gone through a Range Extender (RE).

NOTE: Version 1.02 of the AP firmware does not check for this, so its value is always "No".

1.3.2 Minimal Mode

Below 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 in order 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, has a start character ($) and end character (#).

2 Application Firmware

2.1 Access Point

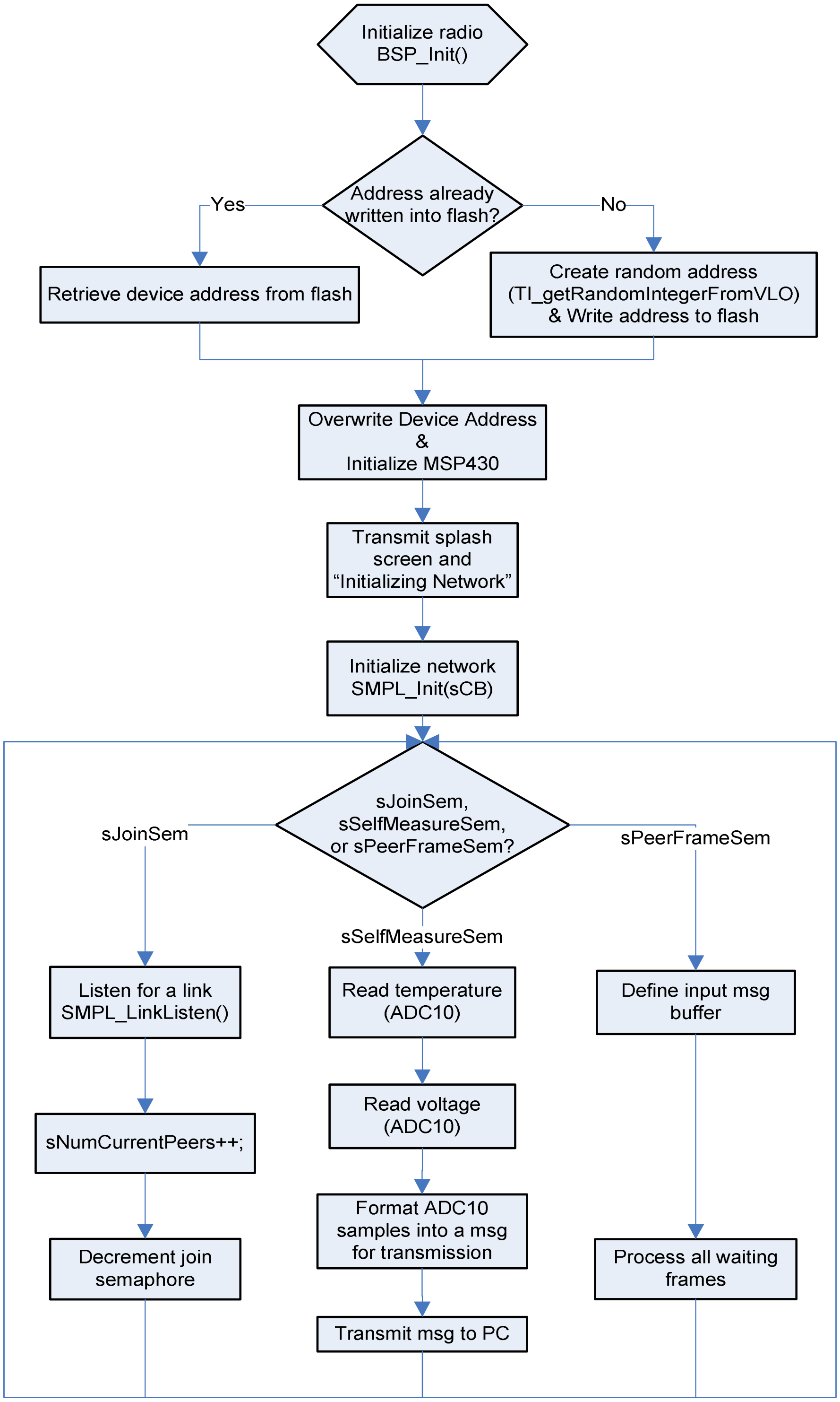

Initialize radio
BSP_Init()
Address already written into flash?
Yes
No
Retrieve device address from flash
Create random address (TI_getRandomIntegerFromVLO) & Write address to flash
Overwrite Device Address & Initialize MSP430
Transmit splash screen and “Initializing Network”
Initialize network SMPL_Init(sCB)
sJoinSem, sSelfMeasureSem, or sPeerFrameSem?
sJoinSem
sSelfMeasureSem
sPeerFrameSem
Listen for a link SMPL_LinkListen()
sNumCurrentPeers++;
Decrement join semaphore
Read temperature (ADC10)
Read voltage (ADC10)
Format ADC10 samples into a msg for transmission
Transmit msg to PC
Define input msg buffer
Process all waiting frames

Figure 3. Flowchart for demo_AP.c

Demo_AP.c contains the firmware built into the AP for the Wireless Sensor Monitor v1.02. The code execution begins with a system initialization that is almost identical for APs and EDs in the network. The following code shows the AP's system initialization procedure.

```
BSP_Init();

if( Flash_Addr[0] == 0xFF &&
      Flash_Addr[1] == 0xFF &&
      Flash_Addr[2] == 0xFF &&
      Flash_Addr[3] == 0xFF )
{
  createRandomAddress();            // set Random device address at initial startup
}
lAddr.addr[0]=Flash_Addr[0];
lAddr.addr[1]=Flash_Addr[1];
lAddr.addr[2]=Flash_Addr[2];
lAddr.addr[3]=Flash_Addr[3];

SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, &lAddr);

MCU_Init();

//Transmit splash screen and network init notification
TXString( (char*)splash, sizeof splash);
TXString( "\r\nInitializing Network....", 26 );

SMPL_Init(sCB);      // Access Point specific function parameter

// network initialized
TXString( "Done\r\n", 6);
```

Figure 4. System Initialization

The BSP_Init() SimpliciTI API call initializes both the communication between the MSP430 and the CC2500 radio and the LEDs/switches on the board that are to be used in the application.

After hardware initialization, APs and EDs in the wireless sensor network 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. Since 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 gets reset will always be recognized as the same device (will be given the same link ID) by the AP and that if the AP goes down itself, any ED that used the AP address to identify their respective SimpliciTI network will see the same AP on network reset. 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 do not create the same network address. For more information on the random number generation library, please reference the application report *Random Number Generation Using the MSP430* (SLAA338). [7]

The MCU_Init() function performs further MSP430-specific initializations that are necessary 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) network initialization function. 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 Figure 6).

```
/*------------------------------------------------------------------------------
* 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;
}
```

Figure 5. sCB Callback Function

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 enumerates new members to the network, assigning incremental link IDs from 0x01 to 0x1D. A link ID from 0x01 to 0x1D identifies the reception of a packet from one of the network's EDs. The possible enumeration values from 0x01 to 0x1D are a designed network constraint for the SimpliciTI protocol that allows up to 30 devices to be linked to the AP. Due to application considerations, however, the Wireless Sensor Monitor v1.02 allows a maximum number of only 8 devices to be linked to the AP.

According to the link ID, the sCB callback function identifies and increments the respective sPeerFrameSem[aphore] or sJoinSem[aphore] for handling in the program's main loop. The AP code also defines a sSelfMeasureSem[aphore], set by the Timer A interrupt once a second, so that the AP knows to sample its own temperature and battery voltages to be displayed. It is these three semaphores that 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 frame's 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, as 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 when an ED calls its SMPL_Init() function. A join to the network is actually a side-effect of initialization, as there is never an actual call made that requests a network join. There are no SMPL_Join() or SMPL_JoinListen() API calls. When the sJoinSem[aphore] has been set in the AP's callback function, and as long as the AP has made less than its maximum number of links (as specified in smpl_config.dat), the AP will call its SMPL_LinkListen() function. As the function parameter, the link listen function takes a pointer to the link ID that will be used to communicate with the linked application. SMPL_LinkListen() is a blocking call, meaning it will not return until a successful link has been created, 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 SMPL_Link().

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[aphore] for another device to set.

2.1.2 sPeerFrameSem Branch

The sPeerFrameSem semaphore is incremented in the callback function every time that the AP receives a frame from a peer application. In the case that the sPeerFrameSem[aphore] has been set or incremented, the AP will first define a message buffer in which to store the current frame being analyzed and then loop through its input queue searching for messages until it has processed all of the waiting frames. The flowchart in Figure 7 shows how the application handles messages from its peers.

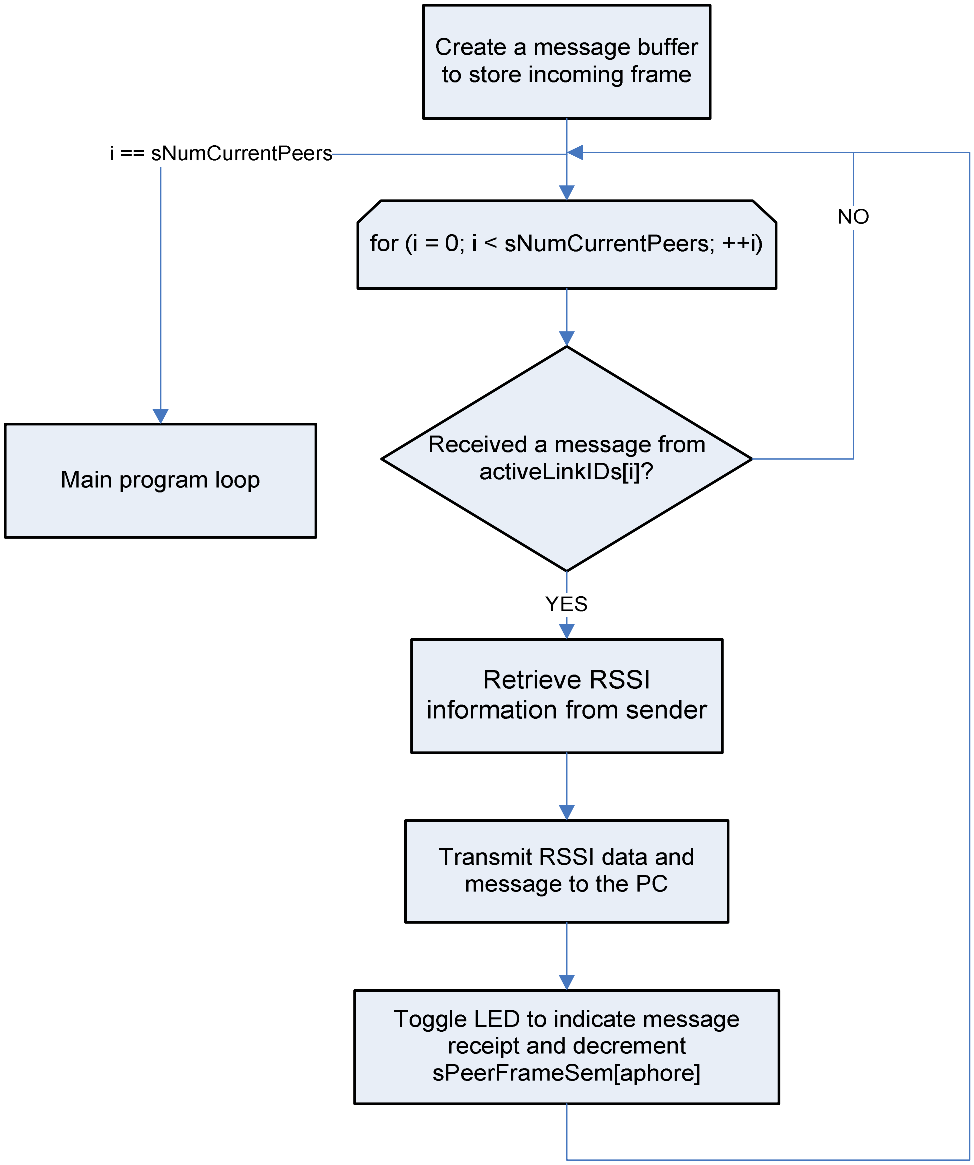

Create a message buffer to store incoming frame
i == sNumCurrentPeers
for (i = 0; i < sNumCurrentPeers; ++i)
NO
Main program loop
Received a message from activeLinkIDs[i]?
YES
Retrieve RSSI information from sender
Transmit RSSI data and message to the PC
Toggle LED to indicate message receipt and decrement sPeerFrameSem[aphore]

Figure 6. Flowchart for Peer Frame Handling

2.1.3 sSelfMeasureSem Branch

The sSelfMeasureSem[aphore] is the semaphore 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 simple low-level operating system.

2.2 End Device

End devices on a SimpliciTI network exist purely to instantiate the network's application or intended function. In this instance, EDs initialize onto the network, then wake up once a second to sample and communicate their ambient temperature/battery voltage. A notable difference exists, however, in their method of initialization. The parameter an ED passes to its SMPL_Init() function is a void pointer to the nonexistent callback function it would use to receive messages from peers.

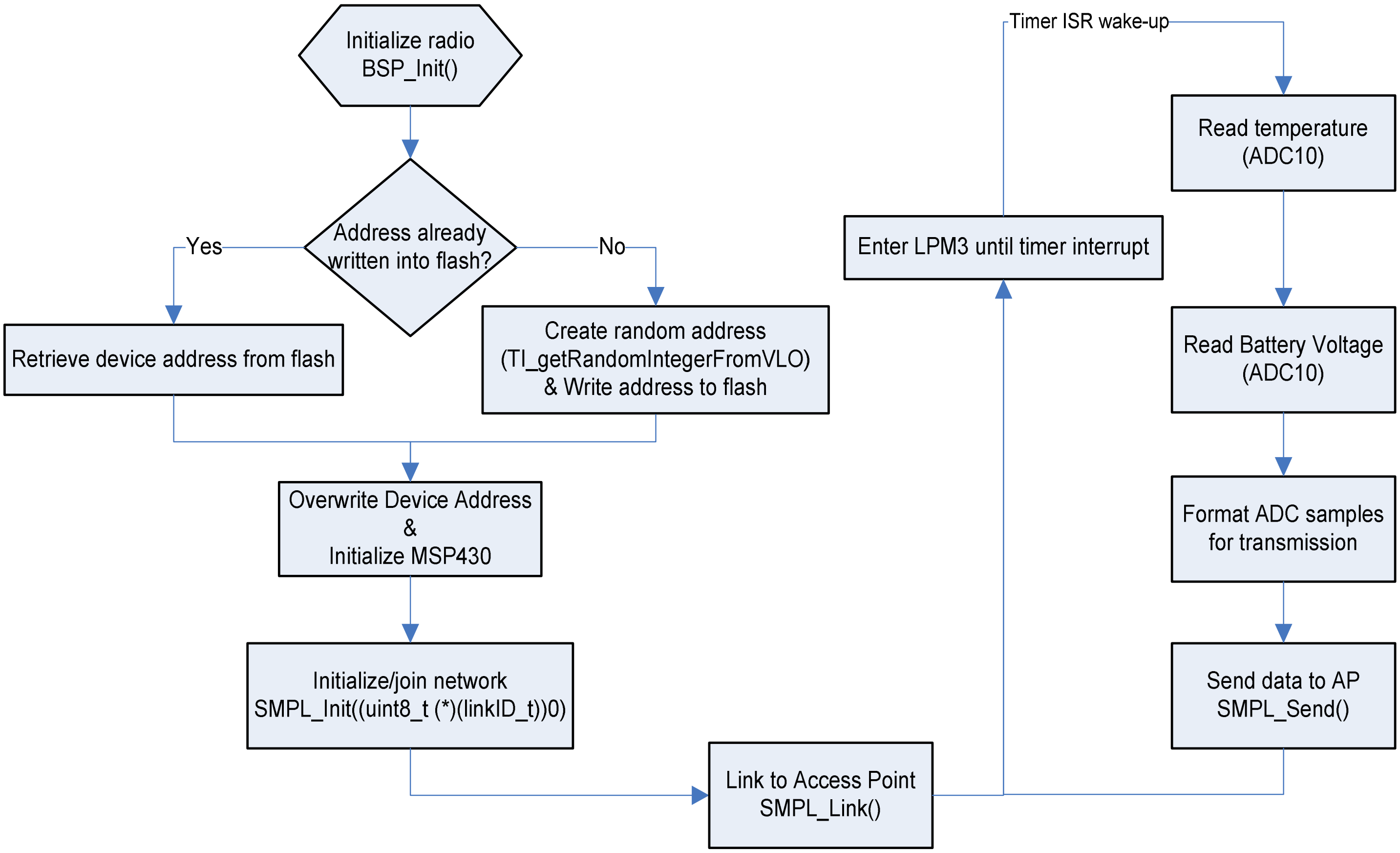

Initialize radio
BSP_Init()
Address already written into flash?
Yes
No
Retrieve device address from flash
Create random address (TI_getRandomIntegerFromVLO) & Write address to flash
Overwrite Device Address & Initialize MSP430
Initialize/join network SMPL_Init((uint8_t (*)(linkID_t))0)
Link to Access Point SMPL_Link()
Enter LPM3 until timer interrupt
Timer ISR wake-up
Read temperature (ADC10)
Read Battery Voltage (ADC10)
Format ADC samples for transmission
Send data to AP SMPL_Send()

Figure 7. Flowchart for demo_ED.c

This application has no need for a callback function that indicates the receipt of messages from other nodes on the network, because an ED's responsibilities are only to transmit its collected data. If an ED were to be capable of receiving messages, it could do so in two ways:

1. In the case that an ED sleeps and wakes up to receive messages from the AP, it would call SMPL_Receive() on wake-up to sample the AP output buffer for any stored messages.
2. In the case that an ED is always on and always listening for incoming messages, it would implement a callback function similar to the sCB function in the demo_AP.c firmware.

3 Performance Overview

The memory requirements of the built code are shown in Table 1. The build was executed using IAR Embedded Workbench for MSP430 KickStart v5.10 with optimization settings set to *Balanced → Low*. Changing the optimization settings for the project may result in unexpected behavior and is not recommended.

Table 1. Memory Requirements for Wireless Sensor Monitor v1.02

Memory Requirements	ROM	RAM
Access Point	9,922 Bytes	724 Bytes
End Device	6,616 Bytes	395 Bytes

To analyze the current profile of the application, the following hardware setup was used:

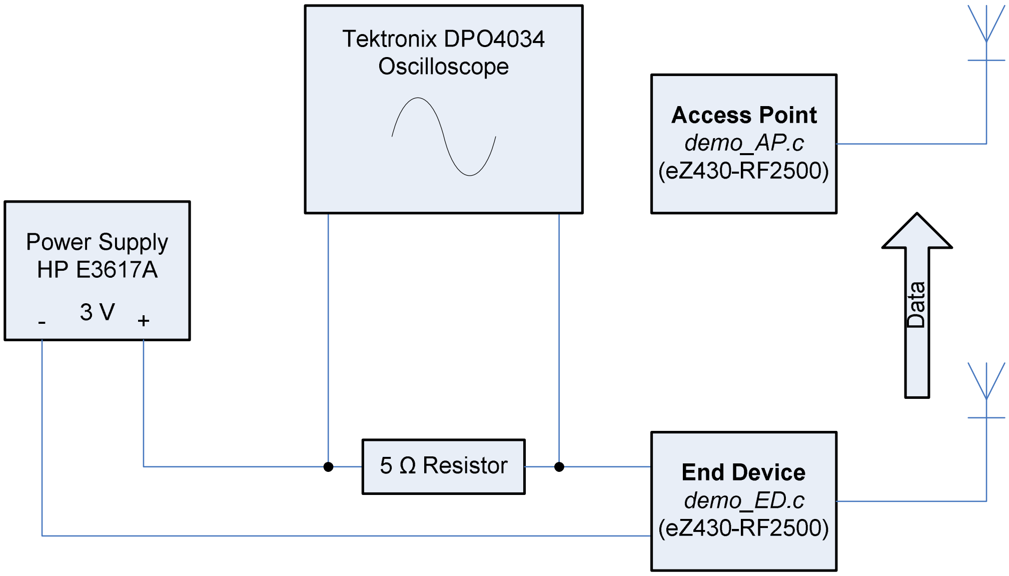


Figure 8. End Device Test Platform

The oscilloscope shot in Figure 10 shows the current profile of an ED over 4 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.

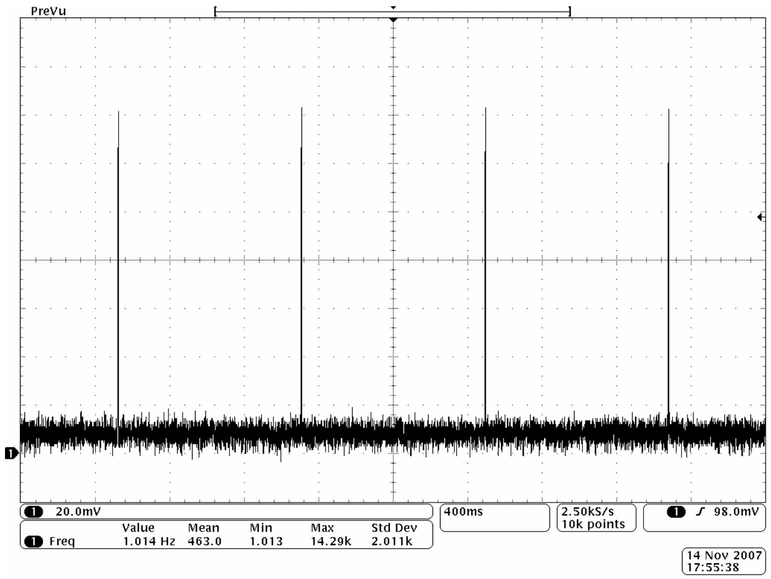


Figure 9. End Device 4-Second Current Profile

Figure 11 shows one of the spikes after decreasing the time step and averaging over 64 samples.

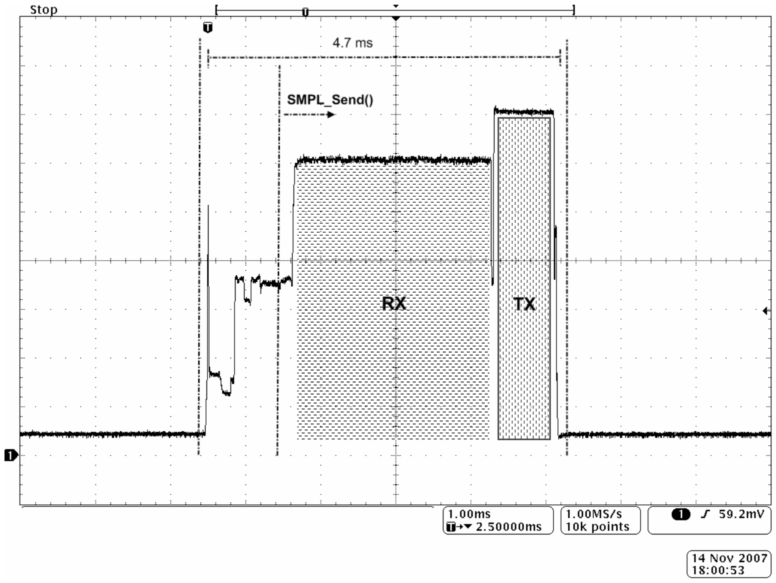


Figure 10. End Device Transmission Current Profile

At this time scale, the waveform can be more closely analyzed for specific events in hardware or software and their respective power contributions. An important consideration in this waveform is the presence of a voltage offset due to measurement error of about 6.8 mV, or when divided by the 5 Ohm resistor, a current measurement error of about 1.36 mA. This is significant as the expected sleeping current of the MSP430 plus the CC2500 radio amounts to only 1.3 µA (900 nA [MSP430] plus 400 nA [CC2500])[*], a magnitude of ×1000 less than the displayed sleep current.

[*] Verified as within range using a multimeter in series with the positive lead of the power supply and the V_{CC} line of the eZ430-RF2500 target board during operation [5]

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 discussion.

Table 2. Significant Current and Timing Contributions

Hardware[†]	Current		Software[‡]	Time for Execution	
	Value	Unit		Value	Unit
Sleep Modes			**Oscillator ('A' in Figure 12)**		
MSP430 Low-Power Mode 0 (LPM0)	1.1	mA	XOSC Startup time	300	µs
MSP430 Low-Power Mode 3 (LPM3)	900	nA	**Ripple Counter ('B' in Figure 12)**		
CC2500 Sleep State	400	nA	Timeout before CHP_RDY Hi→ Lo	150	µs
MSP430 Active Mode			**PLL ('C' in Figure 12)**		
8 MHz = DCO = SMCLK, 3 V	2.7	mA	Calibrate Frequency Synthesizer	809	µs
MSP430 ADC10			**Temp Sample ('1' and '2' in Figure 12)**		
$f_{ADC10CLK}$ = 5.0 MHz, ADC10ON = 1, REFON = 1, REFOUT = 0, ADC10DIV = 0x4 (ADC10CLK / 5)	850	µA	ADC10 REFON + delay for stabilization	130	µs
			LPM0 + Sync + temp sample (30 µs) + conversion (13 x ADC10CLK cycles)	44	µs
CC2500 Modes			**Volt Sample ('3' and '4' in Figure 12)**		
Idle	1.5	mA	ADC10 2.5V REFON + delay for stabilization	130	µs
Receive (RX) (weak input signal, DEM_DCFILT_FILT_OFF = 0, 250 kbps)	18.8	mA	LPM0 + Sync + temp sample (8 × ADC10CLKs) + conversion (13 × ADC10CLK cycles)	22	µs
Transmit (TX) (250 kbps, 0-dB output power)	21.3	mA	**FIFO ('5' and '6' in Figure 12)**		
			Prepare message for transmission	140	µs
			Write message to TX FIFO	110	µs
			RX/TX Modes ('D' and beyond in Figure 12)		
			RX mode protocol overhead	2.56	ms
			Transmit the message	800	µs

The MSP430 contributions to program flow and power consumption are somewhat straightforward, as the events can be traced through an analysis of the application 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. A brief introduction to the radio is necessary to fully understand why the MSP430 is able to execute – better yet, should execute – its application in parallel to radio events, saving both time and current consumption.

Figure 12 shows all of the events from Table 2 partitioned in the waveform, excluding most of the RX and none of the TX modes of operation, shown in Figure 11.

[†] Numbers attained using data sheets from the MSP430F2274 and CC2500 radio. [5] [6]

[‡] Numbers attained using data sheets from the MSP430F2274 and CC2500 radio and oscilloscope measurements. [5][6]

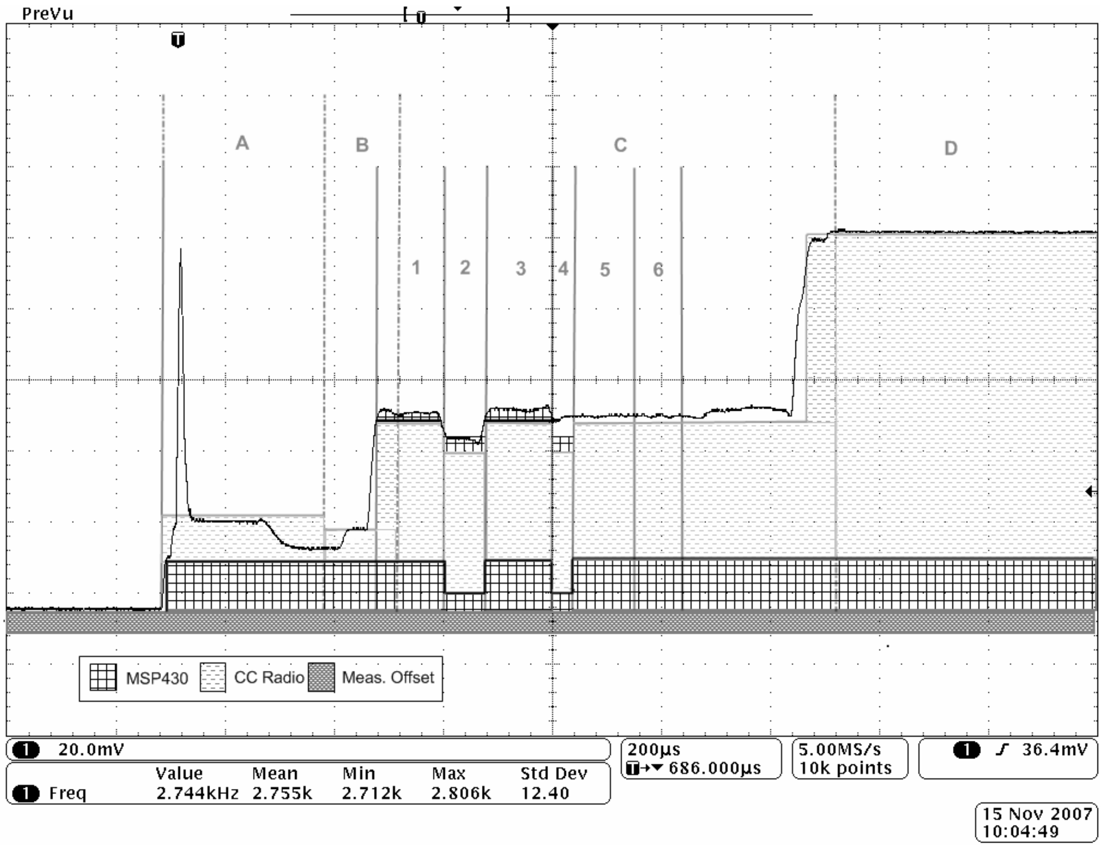


Figure 11. End Device Transmission (Zoomed)

The radio events are denoted using the dotted lines in the chart, whereas MSP430 events use the solid gray lines. The four radio events (A, B, C, and D) occur every time that the radio is woken from sleep by a SMPL_Ioctl() call and are a requirement for a successful reception or transmission of information. They are:

1. XOSC startup – ‘A’

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

2. Ripple Timer Timeout – ‘B’

   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 – ‘C’

   After XOSC startup, the radio defaults to its IDLE mode, where it awaits the ripple timer’s timeout. However, the SimpliciTI wakeup function forces the radio directly into RX mode after initialization to IDLE. 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.

NOTE: If the SMPL_Transmit( ) or SMPL_Receive( ) functions are called during the time that the PLL is being calibrated, program execution becomes undefined due to the fact that the radio is in between states.

4. RX Mode – 'D'

   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). When the CCA check is complete, the radio can then transmit its intended application payload.[§]

Due to these four radio requirements, it is important to understand that the largest contributor to current consumption is not the transmission of the bytes itself (TX mode), but rather the CC radio and MSP430 initializing that transmission. To calculate the average current consumption for the application, this report then uses two methods:

1. Calculation by hand
2. Use of the oscilloscope to take the integral of the voltage curve

Calculating the average by hand is done by first separating the radio and MSP430's current consumption components.

Table 3. Expected Current Consumption

Radio Event	Current Consumed	Time Executed	Amps * Seconds Consumed
XOSC startup	2.7 mA	300 µs	810 nA*s
Ripple counter timeout	1.75 mA	150 µs	262 nA*s
PLL calibration	7.5 mA	809 µs	6,068 nA*s
RX mode	18.8 mA	2.56 ms	48,128 nA*s
TX mode	21.3 mA	800 µs	17,040 nA*s
		Total	**72,308 nA*s**
MSP430 Event	**Current Consumed**	**Time Executed**	**Amps * Seconds Consumed**
MSP430 Active Current	2.7 mA	4.634 ms	12,512 nA*s
MSP430 LPM0 Current	1.1 mA	66 µs	73 nA*s
ADC10	850 uA	326 µs	277 nA*s
		Total	**12,862 nA*s**
		Transmission Total	**85,170 nA*s**

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 = 1.3 uA * (period_of_transmission – application_execution_time)* (1)

= 1.3 [uA] * (1 [s] – 4.7 [ms])

= **1.29 uA*s**

[§] For more information on CC2500 settings, see the *CC2500 Single-Chip Low Power RF-Transceiver, Rev. A*. [6]

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)

$\text{average\_current\_consumption}_{\text{EXPECTED}}$ = (1,294 [nA*s] + 85,170 [nA*s]) / 1 [s]

= **86.46 uA**

For comparison, the second method used to calculate the average current consumption – integrating the curve using an oscilloscope – resulted in an area under the curve of 501.4 µV*s. This can be used directly, as the integral includes the entire period of the transmission waveform:

$\textit{average\_current\_consumption}_{\textit{MEASURED}}$ *= (measured_voltage_reading / 5 Ohms) / period_of_transmission* (3)

= (501.4 [µV*s] / 5 [Ohms]) / 1 [s]

= **100.28 µA**

Given the offset voltage of the oscilloscope calculation, the slight simplification of the current analysis, and general measurement error, a 14% difference between the two numbers is acceptable. To calculate the life expectancy of an ED on the network, and assuming that two AAA batteries will maintain a 1000 mA*hr rating under the hypothetical condition in which the batteries hold their voltage ideally and until their capacity is exhausted:

hours_of_operation = current_rating / average_current (4)

Calculated life expectancy:

= 1000 [mA*hrs] / .08646 [mA] = 11566 [hrs] / 24 [hrs/day] = 481.9 [days] / 365 [days/yr] = 1.32 years

= **1 year, 3 months, and 25 days**

Measured life expectancy:

= 1000 [mA*hrs] / 0.10028 [mA] = 9972 [hrs] / 24 [hrs/day] = 415.5 [days] / 365 [days/yr] = 1.14 years

= **1 year, 1 month, and 19 days**

Figure 13 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 and, in turn,
2) Fit as many bytes into the transmission packet as is feasible for the application.

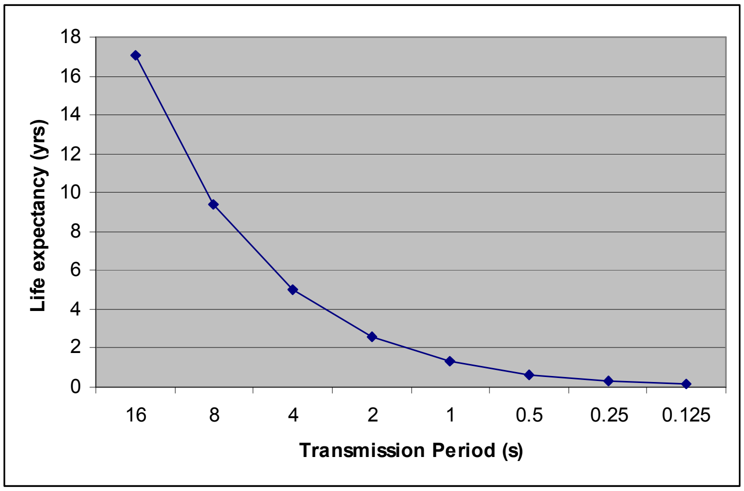


Figure 12. Years of Operation vs Transmission Interval

Using this document, the supporting documentation, and the Wireless Sensor Monitor v1.02 firmware the reader now has all the tools he needs 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 .

References

1. *eZ430-RF2500 User's Guide* (SLAU227)
2. *SimpliciTI Developer's Notes*
3. *Measuring Power Consumption with CC2430 and Z-Stack* (SWRA144)
4. *TI Delivers SimpliciTI™ Network Protocol for Simple, Low-Power RF* Networks (SC-07149)
5. *MSP430F22x2, MSP430F22x4 Mixed Signal Microcontroller, Rev. B* (SLAS504)
6. *CC2500 Single-Chip Low-Cost Low-Power RF Transceiver, Rev. A* (SWRS040)
7. *Random Number Generation Using the MSP430* (SLAA338)

Appendix A. Wireless Sensor Monitor v1.02 FAQ's

5.1 Which version of IAR is required to launch the eZ430-RF2500 Wireless Sensor Monitor v1.02 project?

IAR KickStart R5.10 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.

5.2 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.

5.3 The Access Point always measures a hotter temperature than the End Device

The Access Point has a constant supply of power (USB), therefore it can remain in active mode at all times in order to listen for End Device packets and join requests. A drawback of having the CPU on constantly is that the A/D will measure a slightly higher temperature than a device that remains in LPM3.

5.4 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.

5.5 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.

5.6 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.

5.7 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 will require 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 v1.02 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 v1.02 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. The following is a screenshot of the GUI using the two devices that come with the eZ430-RF2500 kit.

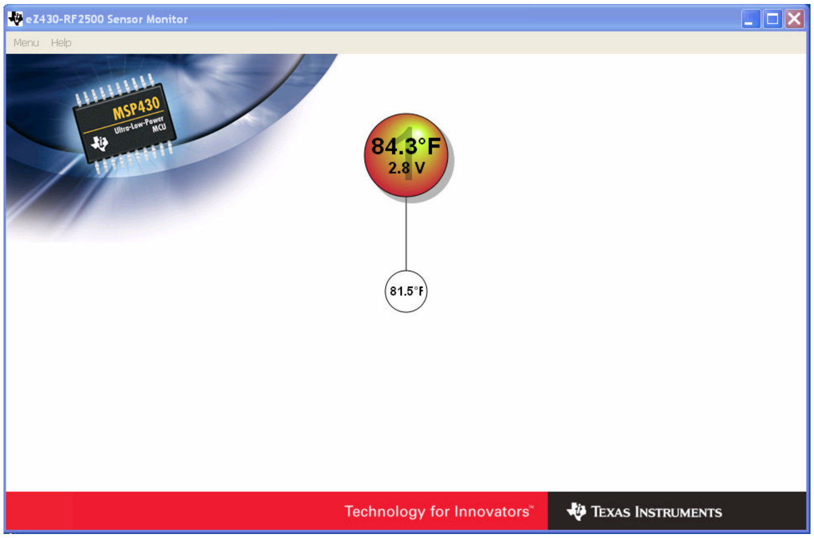
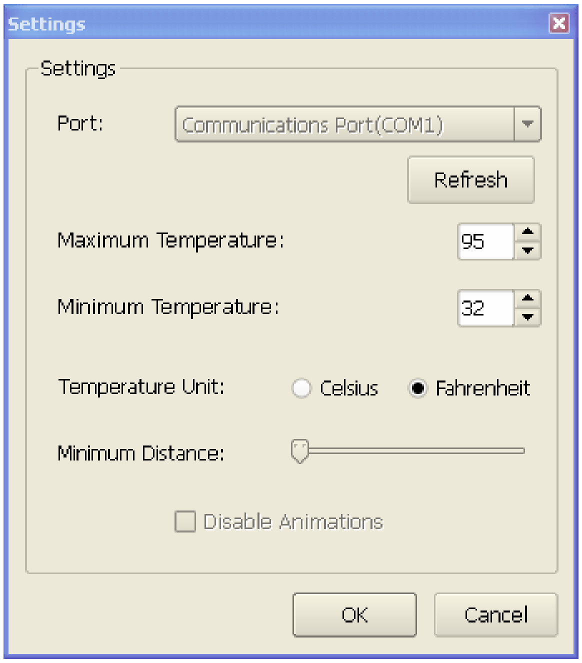


Figure B-1. Network Visualizer Screen

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 B-2. 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.



Settings

Port: Communications Port(COM1)

Maximum Temperature: 95

Minimum Temperature: 32

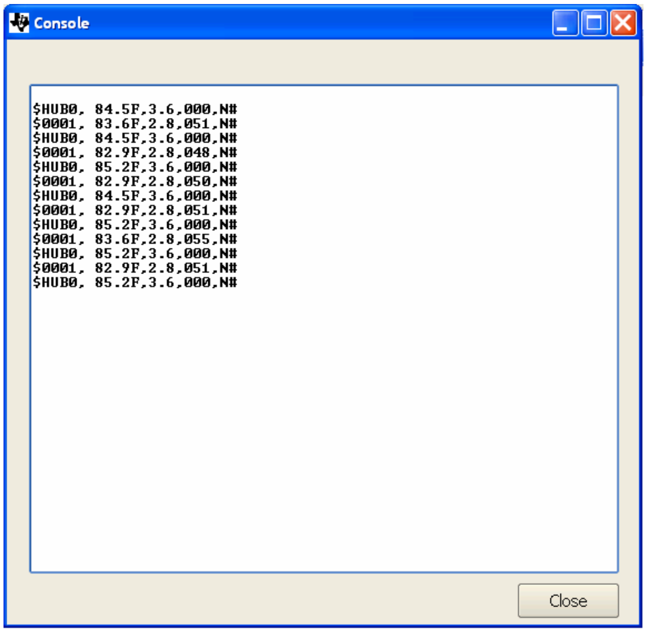
Temperature Unit: ○ Celsius ● Fahrenheit

Minimum Distance:

□ Disable Animations

Figure B-2. 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 B-3. For further questions on the Network Visualizer GUI, please reference the Help menu.



```
$HUB0, 84.5F,3.6,000,N#
$0001, 83.6F,2.8,051,N#
$HUB0, 84.5F,3.6,000,N#
$0001, 82.9F,2.8,048,N#
$HUB0, 85.2F,3.6,000,N#
$0001, 82.9F,2.8,050,N#
$HUB0, 84.5F,3.6,000,N#
$0001, 82.9F,2.8,051,N#
$HUB0, 85.2F,3.6,000,N#
$0001, 83.6F,2.8,055,N#
$HUB0, 85.2F,3.6,000,N#
$0001, 82.9F,2.8,051,N#
$HUB0, 85.2F,3.6,000,N#
```

Figure B-3. 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**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power	www.ti.com/lpw	Video & Imaging	www.ti.com/video
Wireless			
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265