


Anaren Integrated Radio

Anaren®

What'll we think of next?®

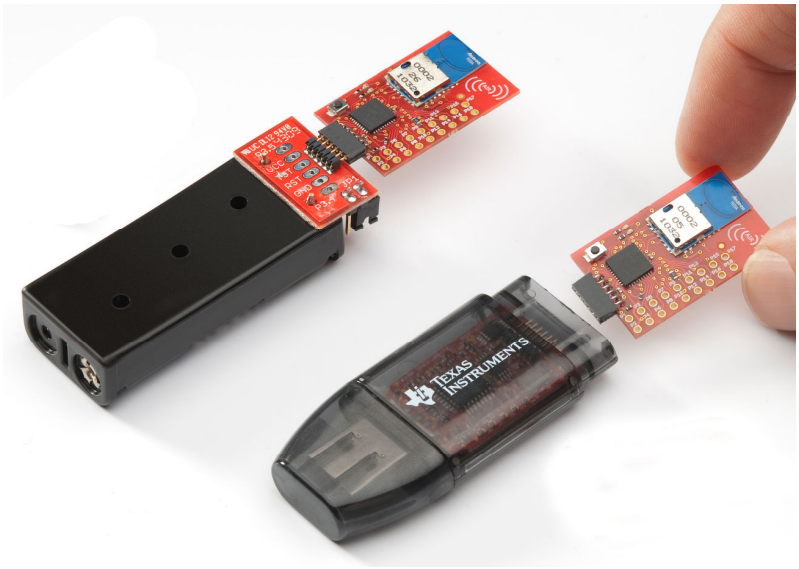
Paired Wireless Sensor / Gateway Demonstration using 2.4 GHz AIR Modules with TI SimpliciTI

Anaren Application Note: ANA-010

Geof Cohler – Regional Sales Manager

Mihir Dani – AIR Applications Engineer

August 7th, 2011




Sales Desk USA: Voice: (800) 544-2414 Fax: (315) 432-9121
Sales Desk Europe: Voice: (+44) 2392 232392 Fax: (+44) 2392 251369

Anaren Integrated Radio

In this demonstration, we use TI's EZ430-RF2500 demonstration kit -- together with Anaren's A2500R24A-EZ4x Target Boards and TI's SimpliciTI protocol -- to manage a powered-on receiver and battery-operated sleeping transmitter that are "paired" in the field. The Transmitter then simulates a sensor input by transmitting 12-bytes at a 16Hz rate to the receiver -- and sleeping otherwise. The Transmitter is optimized for low power consumption to maximize battery life. The Receiver is normally in a listening mode receiving, messages from the Transmitter, checking them for errors, and then moving the information out the UART for demonstration purposes.

At manufacturing time, all the Receivers are identically programmed, as are all the Transmitters. Each Receiver / Transmitter is programmed with a unique serial number that is later used in the pairing process. When originally programmed, the RX and TX units are unpaired -- and will go to "deep sleep" awaiting pairing.

A simultaneous button push on the RX and TX units will cause them to pair with each other, select a random channel to use for communication, and then to begin to transmit / receive. Status information can be retrieved from the Receiver's UART. Once paired and transmitting, the RX unit will toggle its green LED every time a packet is received; the blink rate is 1/2 the packet received rate -- i.e. 8Hz

Transmitter Operation:

The Transmitter basically has three major states:

1. Deep Sleep -- stays this way until pairing is completed (LPM4).

2. Transmitting -- continuously transmits 12-bit packets at 16Hz rate (sleeps LPM3 in the interim) at 0dBm Transmit power and 250Kb/s baud rate.

3. Pairing -- upon button push, looks for an RX unit also in Pairing mode, and attempts to complete a pairing. If the TX fails or times out, it will either revert to Deep Sleep or Transmitting mode (as it was before pairing).

The Transmitter is in deep sleep until paired. Pushing the button on the transmitter causes it to look for a RX unit to pair with; it will turn on the Green LED and blink the Red LED while looking. If it does not find an RX unit within approximately 5 seconds, it will go back to its prior status (either Deep Sleep or Transmitting). Once paired with an RX unit, the TX will transmit 12-byte packets at 16Hz continuously. It calls a routine called getData to retrieve the 12-byte message to send. In the demo, the data is an ASCII sequence.

Receiver Operation:

The Receiver also had three major states:

1. Deep Sleep -- stays this way until pairing is completed (LPM4).

2. Receiving -- continuously listens for packets, and receives, and toggles the green LED, and checks and prints status out the UART when received.

3. Pairing -- upon button push, looks for a TX unit also in Pairing mode, and attempts to complete a pairing. If the RX fails or times out, it will revert to either Deep Sleep or Receiving mode (as it was before pairing).

The Receiver is in deep sleep until paired. Pushing the button on the receiver causes it to look for a TX unit to pair with -- it will turn on the Green LED and blink the Red LED while looking. If it does not find a TX unit within approximately 5 seconds, it will go back to its prior state (either Deep Sleep or Receiving). Once paired, it will receive 12-byte packets and output occasional status information via the UART port. It calls a routine putData to move the 12-byte message and status information out the UART.

Paired Communications:

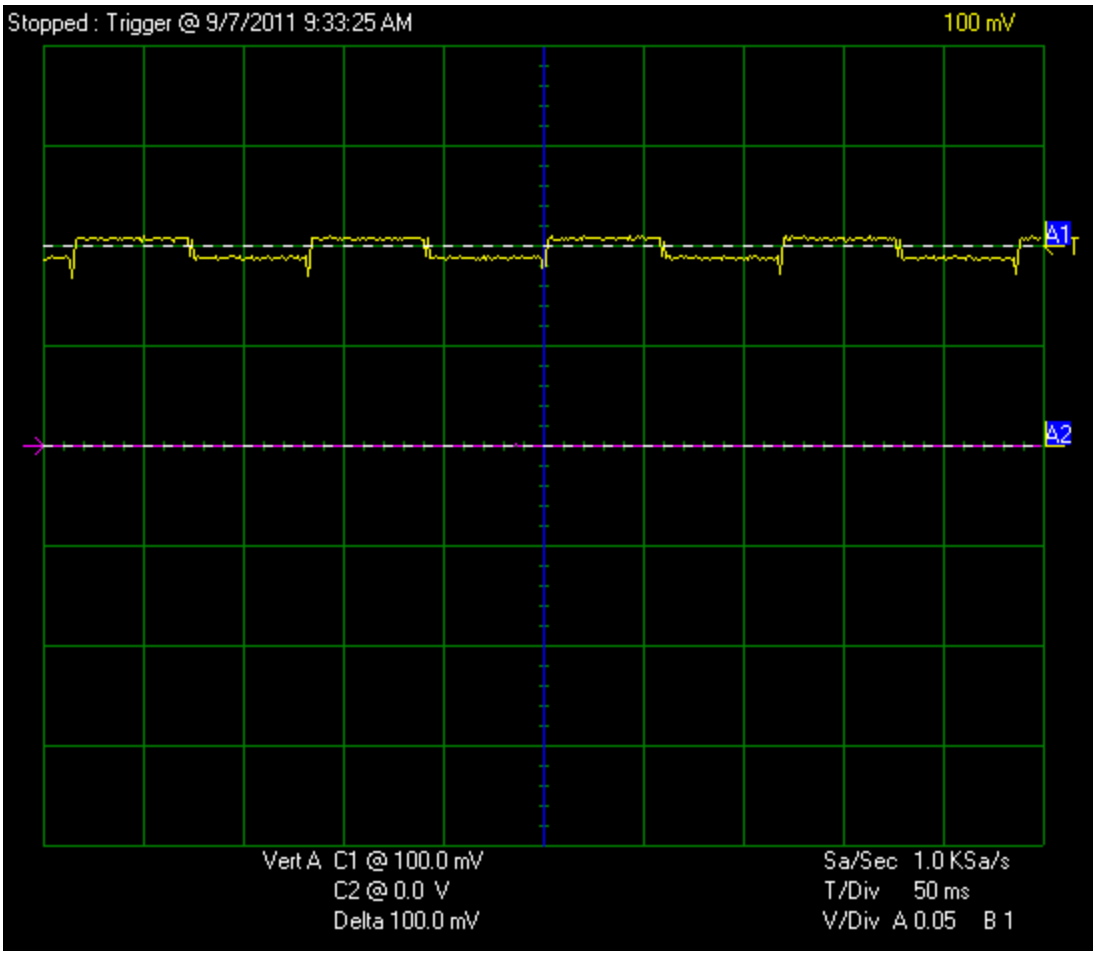
Two simple techniques are used to maximize the bandwidth available to an RX/TX pair, and to minimize interference from other pairs that are within radio range. The Receiver will only "listen" to messages received from its paired Transmitter. Thus if there are multiple pairs within radio range, one pair's TX will not affect another pair's RX. Plus, each RX/TX pair randomly selects a channel for operation from a list of available channels. So it is likely that two pairs will be on two different channels within the 2.4 GHz band. If two pairs are on the same channel, and within radio range, then there is some probability of packet collision. But it is minimized by the short transmission periods and long sleep periods.

Power Consumption / Timing:

When unpaired, the RX and TX units are in LPM4 with the Radio in Sleep mode, and current consumption is a few microamperes. Thus it is reasonable that Transmitters and Receiver units could be shipped with batteries installed, and still have a very long shelf life with only a few microamperes of drain.

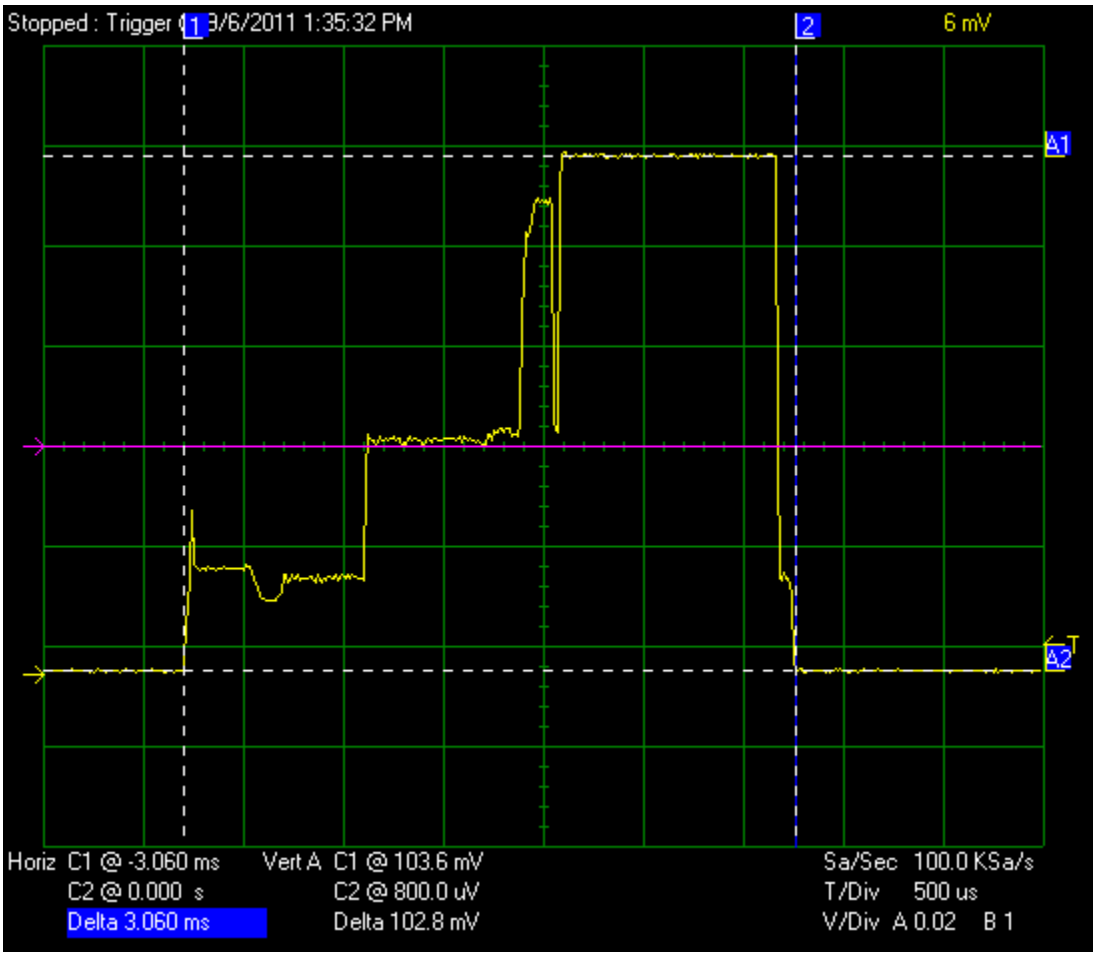
Receiver Power Cycle:

Once paired, the Receiver is in constant receive mode – drawing approximately 23mA fairly continuously – with some variation due to LED blinking. The scope trace below was taken using a 4.3ohm resistor. So the approximate 100mv reading corresponds to the 23mA average value.



Transmitter Power Cycle:

Once paired, the Transmitter has a more complex cycle focused on low-power operation. Based on the 62.5ms cycle time, the transmitter spends about 59.5ms in LPM3 power saving mode with the radio in SLEEP – drawing about 10uA. It then spends about 0.9ms powering up the radio – drawing 4.5mA, followed by 0.8ms calibrating the radio – drawing 10.7mA. Then, it will execute a "listen before talk" step for 0.2ms – drawing 21.4mA. Finally, it transmits its packet for 1.1ms at 24mA. So total power consumption for a 62.5ms cycle is approximately 44mA. See the diagram below for more detail.



Notes on potential power savings in the Transmitter:

1. The current setup does a calibration on every transmission – for maximum robustness. However these calibrations could be done every fifth transmission, eliminating about 7mA per cycle consumption.

2. The current setup will execute a “Listen before Talk” step on each transmission for maximum robustness. However, this could be eliminated – saving 4.3mA per cycle.

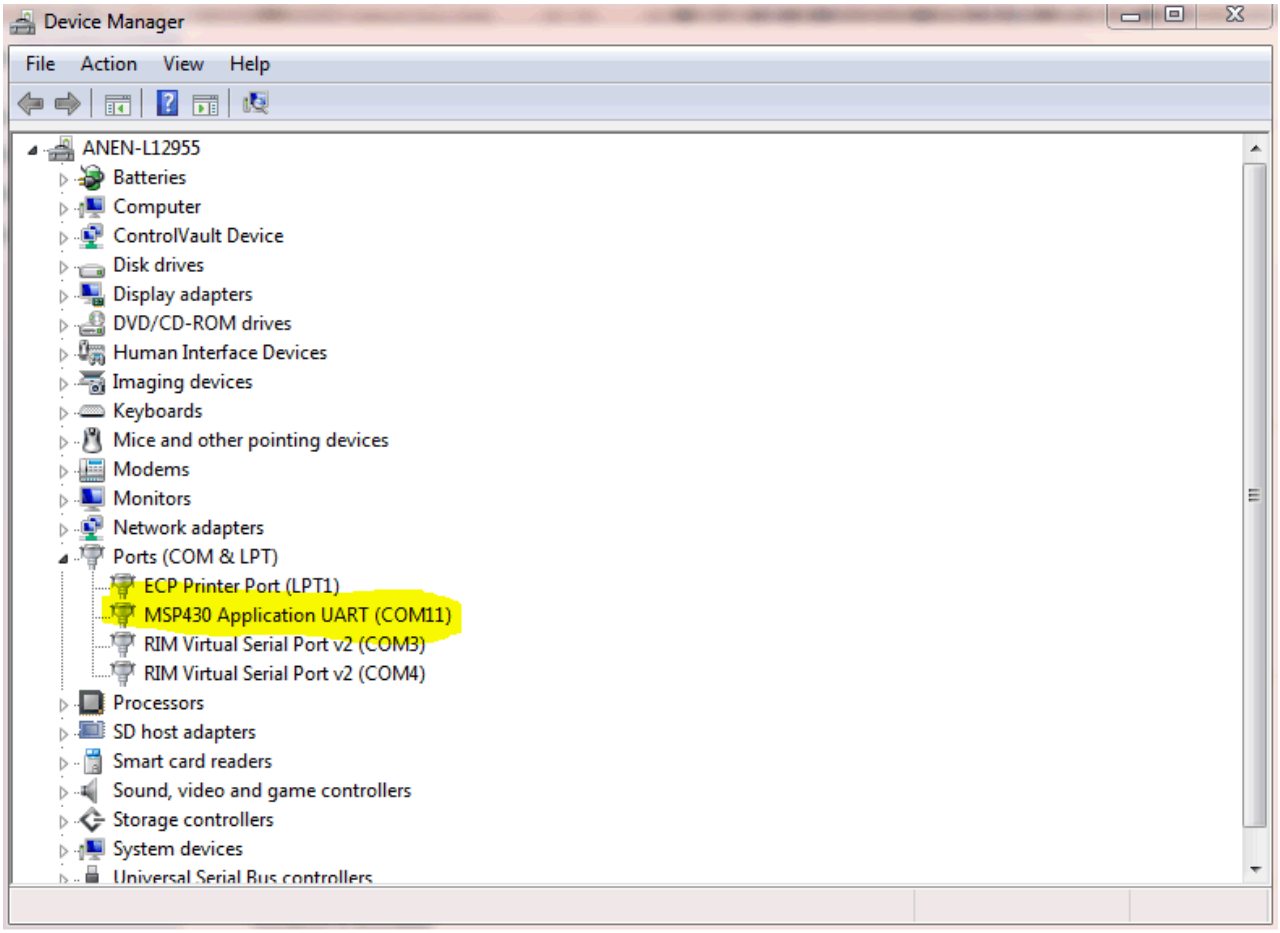
3. In short: The as-is total is 44mA per 62.5ms cycle, whereas the two reductions above will cut the current consumption by approximately 11mA per cycle (or approximately 25 percent).

Setting up the demonstration:

Download the Paired RX/TX Demo from the Anaren website. The demo comes with all source code and with pre-built code images.

On the two Anaren EZ4x Target Boards, label one A (for Access Point or RX) and one E (for End Point or TX); please note that the labeling can be done with a marker on the back of the Target Boards.

Plug the EZ430-RF2500 USB stick into the PC. Determine the COM port that the USB Stick is connected to (e.g. COM 11) -- using Device Manager.



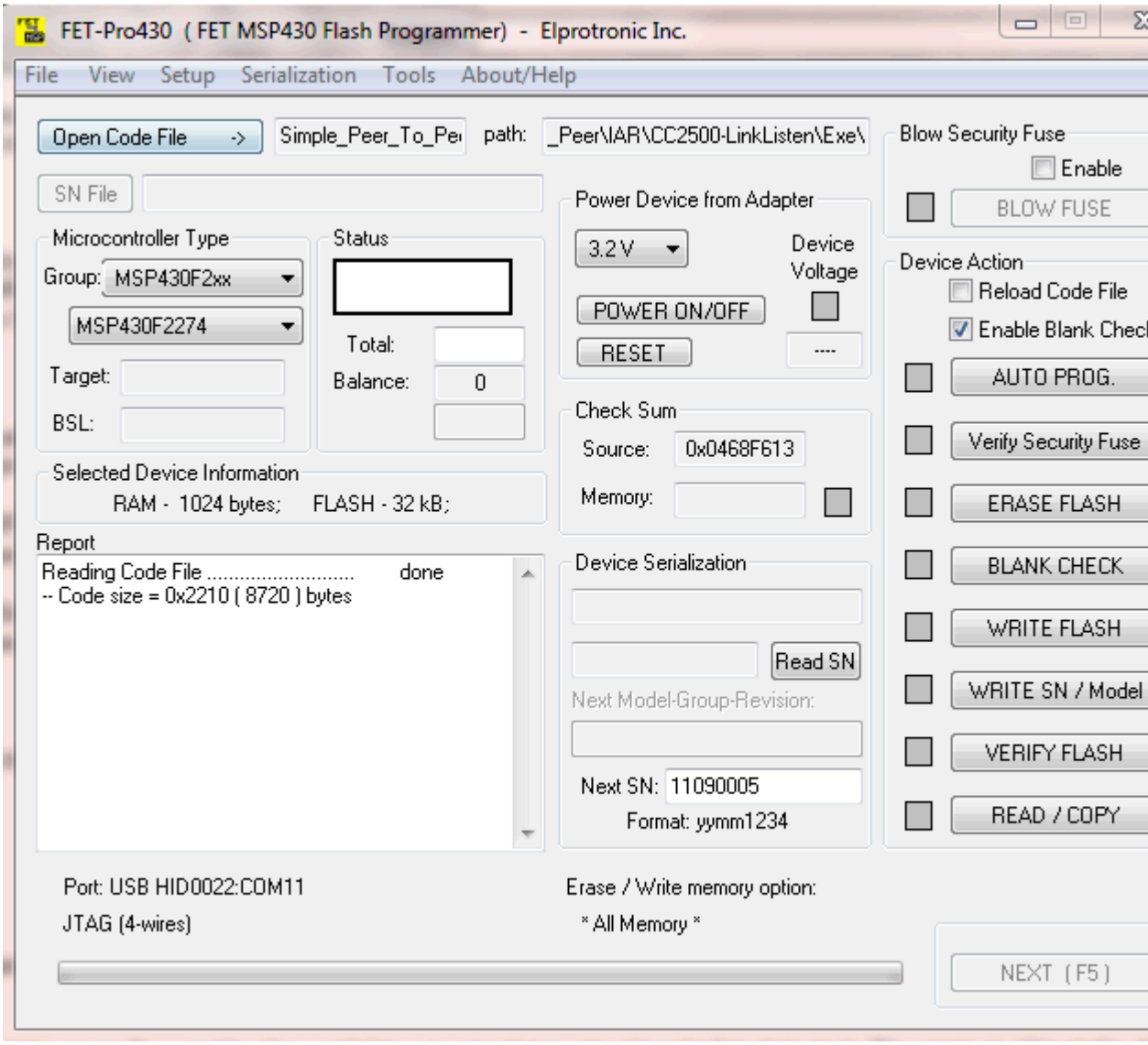
Using a programmer (IAR, Code Composer Studio, Elprotronic, etc), load the following images into the Target Boards by plugging the Target Board into the USB stick.

- Access Point -- RX unit -- use the file:
  ...\Projects\Examples\eZ430RF\Simple_Peer_to_Peer\IAR**CC2500-LinkTo**\Exe\Simple_Peer_To_Peer.d43
- Be sure to use a Serializing Loader, or to load a unique 4-byte serial number into address 0x10B0.
- The MSP-430 on the Anaren Target Boards is an MSP430F2274.
- So A - RX - LinkTo all go together.

- End Point -- TX unit -- use the file:
  ...\Projects\Examples\eZ430RF\Simple_Peer_to_Peer\IAR**CC2500-LinkListen**\Exe\Simple_Peer_To_Peer.d43
- Be sure to use a Serializing Loader, or to load a unique four-byte serial number into address 0x10B0.
- So E - TX - LinkListen all go together.

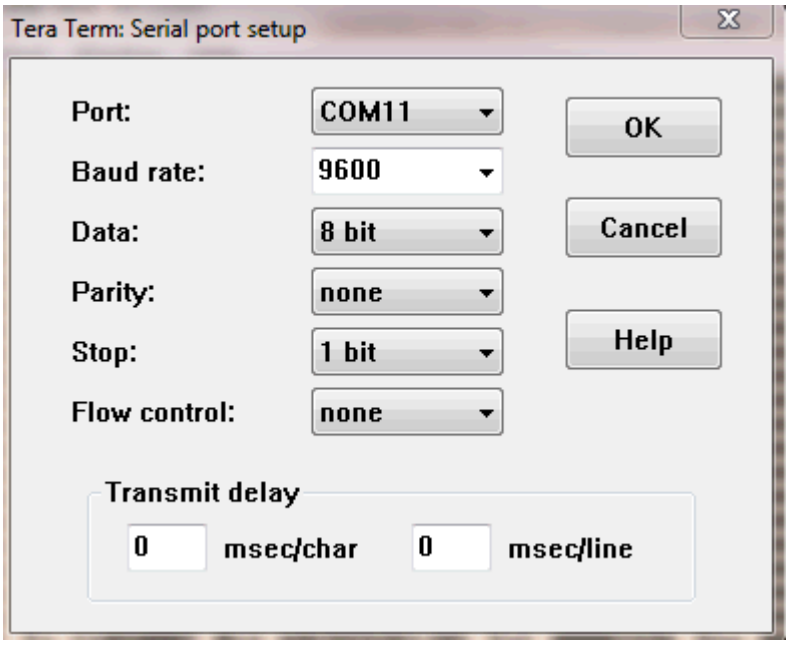
Example using Elprotronic:

Sales Desk USA: Voice: (800) 544-2414 Fax: (315) 432-9121
Sales Desk Europe: Voice: (+44) 2392 232392 Fax: (+44) 2392 251369



The boards are now loaded and, if powered on, will enter to the unpaired LPM4 status.

Run a Terminal Emulation program (like Hyperterm or TeraTerm) to connect to the COM port at 9600 baud, no parity, one stop bit, no flow control (9600, 8N1).



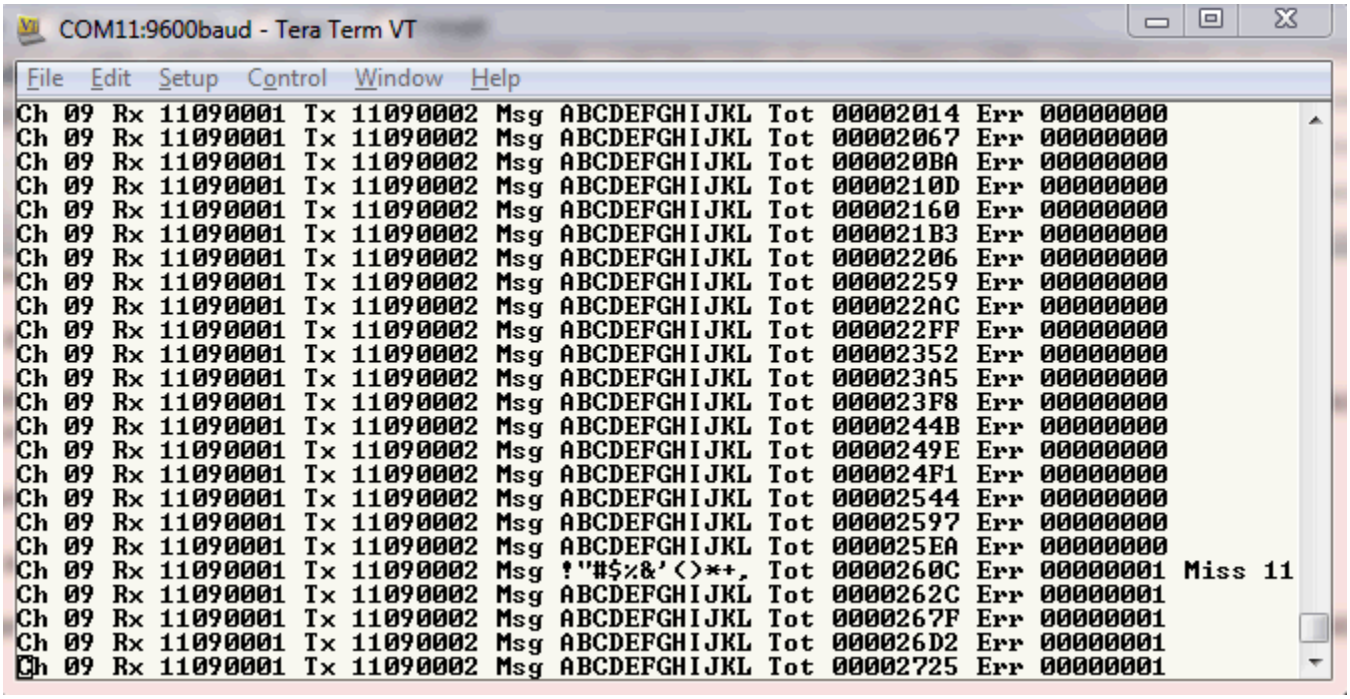
Once programmed, plug the End-Point or TX unit into the Battery Pack, and the Access Point or RX unit into the USB Stick.

Now the boards are ready to pair.

At the same time, push and release the button on both the RX and TX Target Boards. The green LED on both boards should light, and there will be some blinking of the Red LED. Then the TX LEDs should turn off, and the RX green LED should begin to blink at 8Hz. If this does not happen, retry this step. You may see occasional red blinks on the RX board -- indicating a packet loss. If there is excessive red blinking on the RX board, try re-pairing the TX/RX again to select a new channel.

Important: If, after pairing, you don't get a blinking 8Hz green LED on the RX board (or you get excessive red LED blinking on the RX board), try pairing again.

Once paired and transmitting, the COM port should get a printout about once every 6 seconds, plus each time an error (unreceived packet) is detected. The printout will look like this image below and you will see very few errors in general. (For demonstration's sake, we have forced an error so that you can see what it looks like in the printout.)



The information in the COM port printout is as follows:

Ch 09 -- Selected #09 of the random channels -- this is not the frequency, just a table index.
Rx 11090001 -- the Serial Number of the paired Receiver.
TX 11090002 -- the Serial Number of the paired Transmitter.
MSG -- the 12-byte message.
Tot -- the total number of packets received.
Err -- the total number of errors detected.
Miss -- the total number of consecutive packets that were lost (only on loss). This counts as one Err. So losing three consecutive packets will show Miss 03, but will add 1 to Err.

Additional Information:

The three main source files in the demonstration are:

1. Transmitter: …\Projects\Examples\Applications\main_LinkListen_wPairing.c
2. Receiver: …\Projects\Examples\Applications\main_LinkTo_wPairing.c
3. Support Functions: …\Projects\Examples\Applications\genericAppFunctions.c

The MSP-430 uses one port for input -- the push button, and 4 ports for output as follows:

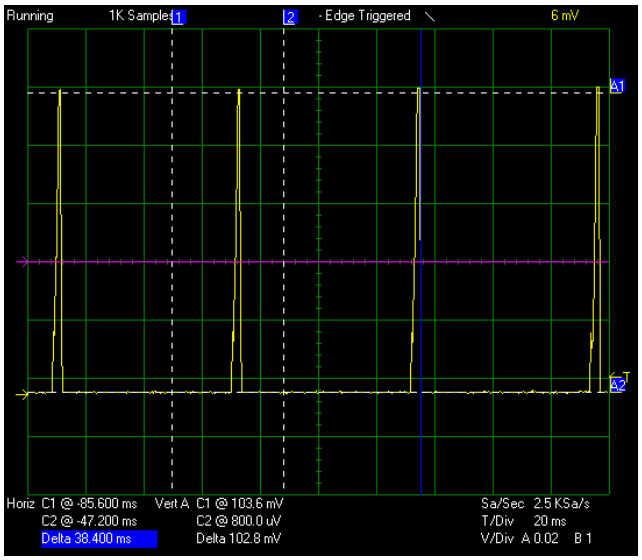
RX Unit:

- P1.0 -- Green LED -- On during pairing. Toggles during Receiving.
- P1.1 -- Red LED -- Toggles during pairing for debug. Turns on during receiving whenever one or more packets are lost consecutively.
- P1.2 – the Push Button.
- P2.0 -- Toggles at the 16Hz rate.
- P2.1 -- Mirrors the Red LED during Receiving.

TX Unit:

- P1.0 -- Green LED -- On during pairing; otherwise Off.
- P1.1 -- Red LED -- Toggles during pairing for debug; otherwise Off.
- P1.2 – the Push Button.
- P2.0 -- Toggles at the 16Hz rate.
- P2.1 -- Toggles before and after the SMPL_SEND call -- to show the transmit window in time.

By installing a 4ohm (or approximate) resistor across the power jumper on the battery pack, you can use a scope probe to monitor the current flow in the transmitter or receiver (divide voltage by the resistor value to get current). Here is an example trace showing the transmitter in low-power sleep mode most of the time with occasional packet transmissions; in this case, the resistor value was 4.3 ohms. So the peak current is about 24 mA.



Conclusion:

This demo can be used as a starting point for real applications. Care has been taken to minimize power consumption on the transmit side by assuming that the receiver is powered on and always listening. In addition, power consumption is minimized by doing un-acknowledged transmissions -- on the assumption that missing a single reading is not critical. The goal was to be generally useful, but not specific to any one application.

Areas that likely will need customization for specific applications:

- Transmitter:
  - We have chosen a 16Hz timer interrupt as the trigger for a transmission -- so that dataflow is readily apparent. However, a button push or other triggers can be used by modifying the MSP-430 code.
  - We have filled in 12-bytes of ASCII data in the message. This will need to be customized for application data.
- Receiver:
  - Output is sent to the UART. But, again, this can be modified for real applications.
  - There is no error correction implemented, though this function may be desired.
  - The error detection mechanism that is implemented is very specific to the test data being sent. As such, it will need to be changed for real data.

For questions or comments regarding this white paper, please do not hesitate to contact us at Anaren -- air@anaren.com

Sales Desk USA: Voice: (800) 544-2414 Fax: (315) 432-9121
Sales Desk Europe: Voice: (+44) 2392 232392 Fax: (+44) 2392 251369