Building Zigbee Radios Using Atmel AT86RF230B and Atmel Processors

This project consists of the circuit board design, parts list, and all software so you can build and use your own Zigbee radios for remote data gathering. Information is provided to guide you in building the radios using surface mount technology. The necessary software, following the Atmel model, is provided for two specific microcontrollers used as the basis for the project.

Warning: This is not a project for Beginners! You don't have to be an Electrical Engineer, but you do need to have intermediate level hardware and software skills. Although I try to explain a lot about the design and what's going on, this is a project, not a tutorial.

This project is complete in the sense that you can follow it to build your own Zigbee Coordinator and Zigbee Device and communicate between them. It is not complete in the sense that there's a lot more that could be done with these radios and the software, including making much larger and more complex networks, as well as exploring very low power modes. So there's room for many exciting sequels and future projects as time permits. But by making this all available at this stage I hope to inspire collaborators to contribute their projects as well.

Motivation

I had two goals in doing this project:

1. I'd like to be able to experiment with Zigbee radios for a lot less than about $25 each for commercial units. (Mine cost more like $8 each or less).
2. I wanted to see if it was possible to build working Zigbee radios at all without an RF Engineering degree. (It is.) My hope was for decent range. (These achieve about 100 feet including going from inside to outside.)

What's it do?

This project implements a Zigbee Controller and Device. Data collected by the device are transmitted regularly to the Controller which prints via its USB port to a terminal emulator running on a PC of some type (pick your favorite). As implemented, only one channel of the A/D is read, but more channels could easily be added.

Design Notes

Mostly, I followed Atmel's data sheet and App Note to design the radio. The wiggle antenna design was taken from the Cypress publication referenced below and adapted to 0.0625 inch thick circuit board. Since it has 50 Ohm impedance, I used Atmel's 50 Ohm matching circuit, also referenced below. Although this was a bit of a leap of faith since I'm not an RF guru, it all seems to work. Note that I did not include the filter that Atmel includes in their design. I'm OK with a little extra gain and since I'm not offering them commercially, I think it's OK. If neighbors complain of interference, I'll reconsider. So far, so good.

Wiggle antenna design (Cypress)
http://www.eetindia.co.in/STATIC/PDF/200812/EEIOL_2008DEC12_RFD_AN_01.pdf?SOURCES=DOWNLOAD

Atmel AVR2004: LC-Balun for AT86RF230 (matching network) http://www.atmel.com/dyn/resources/prod_documents/doc8113.pdf

Atmel AT86RF230 Data Sheet
http://www.atmel.com/dyn/resources/prod_documents/doc5131.pdf

Atmel Apps note and example circuit
http://www.atmel.com/dyn/resources/prod_documents/doc8092.pdf

Atmel Zigbee Software and Documentation. You can download it here, but you have to register.
http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4675

Circuit Board Design

The Eagle board and schematic files are included so you can have your own boards fab'ed. (Don't have Eagle? Get it here: http://www.cadsoftusa.com/freeware.htm The free version is perfect for this project.) One great place to get boards is through the Dorkbot group order that you can learn about here. http://dorkbotpdx.org/ . You don't have to be a member to take advantage of this – the boards will be mailed to you. The Gerbers for the appropriate design rules are available. If you decide to do your own boards and want to modify the design in any way, I suggest changing the 0402 footprints since I've had a bit of trouble with tombstoning on those parts in the matching network.

Parts List

Digikey Part Numbers

RF IC	AT86RF230-ZU-ND	IC TXRX ZIGBEE/802.15.4 32QFN
16MHz Crystal	887-1125-1-ND	CRYSTAL 16.000 MHZ 18PF SMD
C1,C2,C5	490-1283-1-ND	CAP CER 22PF 50V 5% C0G 0402
C3,C4	712-1251-1-ND	CAP CER 0.9PF 50V S 0402 UHI Q
C7,C8,C9,C10	490-1543-1-ND	CAP CER 1.0UF 10V 10% X5R 0603
C11, C12	490-1280-1-ND	CAP CER 15PF 50V 5% C0G 0402
L1,L2	587-1510-1-ND	INDUCTOR HIFREQ 4.3+/-0.3NH 0402
L3	445-3508-1-ND	INDUCTOR MULTILAYER .33UH 0402

Connectors (two of each): (Samtec):
MTMS-106-01-S-S-125
SLM-106-01-S-S
Note: Connectors are optional. You could solder directly to the board.

How do you build the boards? These two Instructables explain the process for building Extreme Surface Mount circuits in detail. Several members of DorkbotPDX have used this technique to successfully build tiny surface mount circuits of many types.
http://www.instructables.com/id/Extreme_Surface_Mount_Soldering/
http://www.instructables.com/id/Closing-the-Loop-on-Surface-Mount-Soldering/

This is the current best source for the solder paste stencils.
http://www.ohararp.com/Stencils.html

The Microcontrollers

Although many Atmel processors could be used (as long as they have over 32K flash), I will show specific examples using the AT90USB1286 and ATmega32U4 mounted on small pcbs and available as the Teensy2++ and Teensy2 respectively at this web site:

http://www.pjrc.com/teensy/?not_a_duplicate

Order one of each; more of the Teensy2 if you want more Devices.

The Teensys are connected to the radios as shown in the following table.

Radio	Teensy2	Teensy2++
J1, pin 1, /RST	Pin B5	Pin B5
J1, pin 2, SCLK	Pin B1	Pin B1
J1, pin 3, MISO	Pin B3	Pin B3
J1, pin 4, GND	Pin GND	Pin GND
J1, pin 5, MOSI	Pin B2	Pin B2
J1, pin 6, /SEL	Pin B0	Pin B0
J2, pin 1, GND	Pin GND	Pin GND
J2, pin 2, GND	Pin GND	Pin GND
J2, pin 3, SLP	Pin B4	Pin B4
J2, pin 4, IRQ	Pin D4	Pin D4
J2, pin 5, VCC	Pin +3.3V	Pin +3.3V
J2, pin 6, VCC	Pin +3.3V	Pin +3.3V

The Teensys need four leds (although only three are normally used; the fourth is for you, the user). Pick colors you like and put a 220 to 1000 ohm resistor in series. Here's where they connect.

Led	Teensy2	Teensy2++
0	Pin D7	Pin D7
1	Pin D6	Pin D5
2	Pin C7	Pin E7
3	Pin C6	Pin E6

Preliminaries

When you get your Teensy2 and Teensy2++, follow the directions on the website to download the software tools. You'll need these to build the software for this project. Follow the directions to test the Teensy downloader and bootloader. You should be able to load the FastBlink program to each Teensy and see it run properly.

The Atmel AT86RF230B must be operated at less than 3.4V. If you subject it to 5V, it will not survive! The Teensys run at 5V as delivered. Follow the instructions on the website to convert them to run at 3.3V. (Order the voltage regulators from Digikey when you order your other parts.) Once this modification is done, you may connect the AT86RF230B radios without worries. The USB still works.

How to build the software

Download the software package from Atmel and unzip it (see link above). You should end up with a large directory structure rooted at MAC_v_2_4_2. Read the Atmel documentation (AVR2025_User_Guide.pdf under MAC_v_2_4_2/Doc/User_Guide) to understand how to modify what they provide. Most of it needs no changes. There's a lot of information in the user guide, but you don't have to read it all at once. If your want to understand how to add a new device, pay particular attention to Section 10, Platform Porting.

The files that define the Teensy boards and the necessary make files are provided so you can download them as ZigBee_Project.zip. You must copy these files as follows:

Place the directory structure named AT86RF230B...T2PP under Coordinator into the corresponding area under App_1_Nobeacon/Coordinator. There will be many other similarly titled directories in the same area.
Do the same with the similarly named AT86RF230B...T2 under Device, placing it under App_1_Nobeacon/Device.
Replace the file App_1_Nobeacon/Coordinator/Src/main.c with the main.c provided under Coordinator.
Replace the file App_1_Nobeacon/Device/Src/main.c with the main.c provided under Device.

Now let's fix the PAL directories.
Under PAL/Inc, replace pal_types.h and pal.h with the versions provided.
Under PAL/AVR, copy the two directory structures provided for the AT90USB1286 and the ATMEGA32U4 into the AVR directory. Again, there will be several similar entries.
Under PAL/AVR/Generic/Inc, replace pal_internal.h and pal_uart.h with the versions I provide.
Under PAL/AVR/Generic/Src, replace pal.c with the version I provide.

Note that my code only supports the gcc tool chain. Since I don't have an IAR tool chain, I have no way to try it out.

Note that the usb_serial.c and usb_serial.h files I provide under AT90USB1286 are taken from the Teensy website. You should check that website and be sure you have the latest versions since these can change.

Actually building the code for the project requires that you build and load code for both the Controller (Teensy2++) and the Device (Teenys2) This project must be built from the command line. From the App_1_Nobeacon/Coordinator/AT86RF230B...T2PP directory do make from the command line. Do the same for the Device from App_1_Nobeacon/Device/AT86RF230B...T2. Everything should build properly. Use the Teensy downloader to download Coordinator.hex and Device.hex to your Teensys. Note that the downloader will automatically detect the exact type of Teensy that's connected to it. But it only deals with one at a time.

For those who are unsure about building the software, or who just need a sanity check, hex files are provided in the Hexes directory so you can program your Teensys with my code. This is perhaps a useful check, but only works if your hardware is identical. (If you copy this project, it will be.) You should still build the software yourself, or you won't be able to extend what I've done.

Operating the radios

Let's check out the Controller operation first. Use the Teensy Loader to load the Controller code onto the Teensy2++. Don't connect a radio just yet. Leave the Controller connected to a USB port and run a terminal emulator (I use Realterm; Hyperterm probably works also. Don't know about Linux or Mac.) to look at the output. With the Controller software running, hit Enter to start things off. You should see all the Leds blink rapidly and get a line or so of zeros in the terminal window. Unplug the USB cable. Now you can connect one of your radios (following the connection list above) and test it again. You should see Leds 0 and 1 turn on and stay on. When you get to this to work, you're ready to try the Device.

Load the Device software on the Teensy2. To do this, you will have to disconnect the Teensy2++ Controller unit. Test the Device while it is connected to the USB cable and with no radio plugged in to it. You should see all the Leds blink. The USB is used only for power and programming, there is no communication over it when the program is running. Unplug your USB cable. Now connect your other radio following the connection table above. You can also refer to the schematic for the radio. Connect the USB cable to power the radio. If all is well, Led 0 will turn on solid and Led1 will blink at a slow rate. (Note that the Led on Teensy also blinks; it's connected to D6 also. That's OK.)

OK! You are now ready to test your radios. Power the Device from a regulated 3.3V. Led 1 should blink while Led 0 is solid. Plug the Controller into the USB and start your terminal emulator. Give the Controller an Enter to start things off. Be patient for 30 seconds or so while the radios recognize each other. You'll see the leds on both Teensys go through a pattern. Leds 0 and 1 will go solid on both units while Led 2 on each of them should begin to blink. You should see regular output on the terminal each time they blink. The output is just a clock reading and the current value from Channel 0 (pin F0) of the A/D on the Device. You can hook up a pot as a voltage divider and see the values change.

Congratulations! You've got your Zigbee radios working and are ready to do more exciting projects.

The Next Challenges

Build a Mega328p version of the Device. Since no USB interface is needed for the Device, this should be easy to do. You can use a Teensy2 for the programmer, emulating the AVRISP-MKII.

On the software side, there's a ton to try. Besides making some more ZigBee Devices, I want to try a ZigBee Router (relays signals from a Device to the Controller, thus increasing the range). Reading the Atmel documentation suggests many experiements.

And of course there's the whole area of low power operation. Just what is the best way to get long battery life out of one of these radios? And how long can it be?

So much to try. So much to learn and share.