Remote Weather Sensors Over ZigBee

Ian Will

April 26, 2011

1 Project Abstract

The project a remote weather station which reports weather conditions periodically from a remote sensing platform to a central controller. The remote sensor is implemented using an XBee series 1 radio and a few analog sensors. The central controller is a Zilog Z16 in the ZNEO Contest Kit. Each sensor periodically measures temperature, humidity, and light levels, sending them to the controller over 802.15.4 (ZigBee). Upon receive a measurement, the central control unit adds the current pressure and logs all received records. Logged data from two remote sensors and from the control station are displayed on the LED panel and controlled through four buttons. More detailed data analysis is supported through a shell that operates over a serial connection.

2 Status

The most significant deviation from the original proposal was to reduce the number of microcontrollers used. The original proposal had one Zilog at each weather sensor. Instead, it was more efficient to use the built in Analog-to-Digital converters on the XBee to send samples withour requiring an additional microcontroller at each sensor node. This required that all sensors have analog output, meaning the I2C pressure sensor had to be moved to the central control station. It also required a swap of humidity sensors (replacing the varying capacitance Parallax HS1101 for the analog voltage output from the Honeywell HIH-4000). An unexpected difficulty came in logging the data. Compared to the 4K of RAM available on the ZNEO, the 22 byte data records were quite large. After the size required by the program, there was room for less than 100 records. To support longer periods of data logs, I added a logging library which stores records in the upper section of EROM. This ended up taking a considerable amount of time debugging. The proposal mentioned two possible exensions which I did not have time to implement due in part to the unexpectedly long debug time for the flash logger. An LCD display attached to the central control unit was procured from Adafruit, and C libraries were provided for the AVR. I spent some time porting them to Zilog, but ultimately had to abandon the effort in favor of having a working data logger. A camera sensor was not purchased,

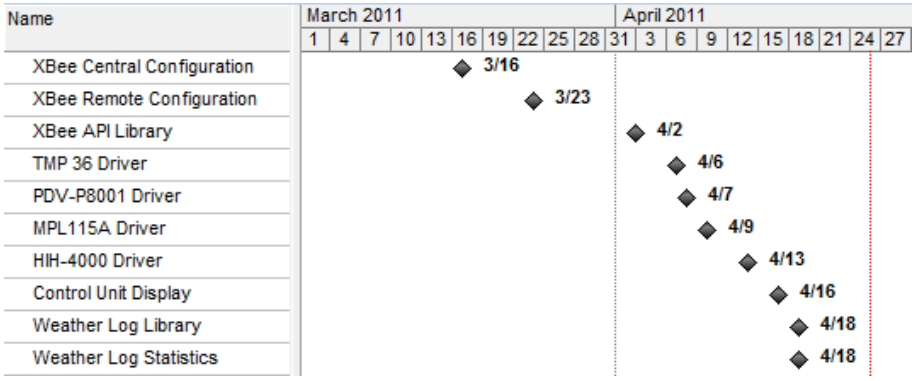

Figure 1: Project Milestones

Item	Cost	Source
ZNEO 16F Series Contest Kit	$49.99	www.mouser.com
XBee Adapter Kit	$10.00 (x 3)	www.adafruit.com
Digit XBee Series 1 (XB24-AWI-001)	$23.00 (x 3)	www.adafruit.com
Honeywell Humidity Sensor (HIH-4000)	$24.41 (x 2)	www.digikey.com
API. Photocell (PDV-P8001)	$1.00 (x 3)	www.adafruit.com
Freescale I^2C Barometer Eval. Board (KITMPL115A2I2C)	$15.00	www.newark.com
Analog Devices Temp Sensor (TMP 36)	$2.00 (x 2)	www.adafruit.com
5V Regulator (LM7805)	$1.25	www.sparkfun.com

Table 1: Materials, Costs, and Sources

and would have added significant complexity to the project, for which there was not time.

3 Specification

The system was composed of two primary hardware platforms: The ZNEO Microcontroller and the XBee 802.15.4 radio. A handful of analog sensors supplied the additional functionality. The ZNEO was used within the ZNEO Contest Kit, which includes the microcontroller and a number of peripherals and pins designed to make prototyping convenient. An itemized list of major parts and materials is provided in table 1. The XBee radios were connected to breadboards through adapter boards. The two remote sensors connected using adapter kits from adafruit.com, which include 5V to 3.3V conversion logic. The remote units rely on the XBee adapter board for 5V to 3.3V conversion necessitated by the 5V HIH-4000 Humidity Sensor and the 3.3V XBee and TMP 36 sensors. The 9V to 5V conversion used by remote units was built following the sparkfun Beginning Emedded Electronics Tutorial 1 (www.sparkfun.com/tutorials/57).

3.1 System Description

The project is composed of two types of physical units: a remote sensor, and a central control station. Remote sensors are implemented with an XBee radio and analog sensors. Remote sensors require no other microcontroller besides what is included with XBee radios. These radios can be configured to read analog voltages from certain pins, convert them to a digital value, and transmit that value in a data packet. Each remote sensor is configured to read three analog sensors, measuring temperature, humidity, and light levels, and to send that data back to the central unit over 802.15.4 communications periodically. The details of XBee radio configuration are discussed in more detail in the Implementation section.

A weather collection system can be composed of many XBee-based remote sensors. The number is not restricted beyond the data logging capability of the central control unit. For purposes of demonstration, I implemented two remote sensor units.

In addition to the remote sensor units, there is a single central control unit to which all remote sensor units report measurements. The central control unit logic is run on a Zilog Z16 microcontroller which communicates with a XBee radio over one of its two UARTS. The second UART is used for serial communication with a computer, over which a shell is provided with commands to query data logs.

The central controller runs in a continuous loop, checking for data frames received from its XBee radio and occasionally polling the sensors connected to the central control unit. When measurements are received, they are converted from analog voltage measurements into temperature, humidity, and resistance values in appropriate units. Because remote sensors do not feature a native pressure sensor, the central control unit reads from its I2C pressure sensor and adds that pressure reading to the data record. The measurement is then inserted into a logging library which stores a limited number of measurements in RAM, dumping them to EROM as the RAM buffer fills.

In addition to receiving measurements from remote sensors, the central control unit also has built in sensors for measuring conditions at the controller. This provides contrasting information on conditions indoors versus outdoors, or in other rooms. The central unit uses one of the Z16 built in timers as both a button poll timer and an internal clock counter. Every second the internal clock is incremented, and the central unit reads from its local sensors and submits its measurments to the data logger.

Temperature is stored in Fahrenheit, relative humidity in percent, pressure in inches of water, and light as resistance in KΩ. Light is stored as a resistance because no formula to convert measured resistance to Lux was provided on the data sheet. Eventually the user interface should correlate ranges of resistance with certain light values or conditions and display, for example, sunny, cloudy, direct sun, shadow, indoor, full moon, and completely dark conditions. Storing measured resistance in the database instead of qualitative descriptors provides more flexibility for changing light descriptor thresholds at a latter time.

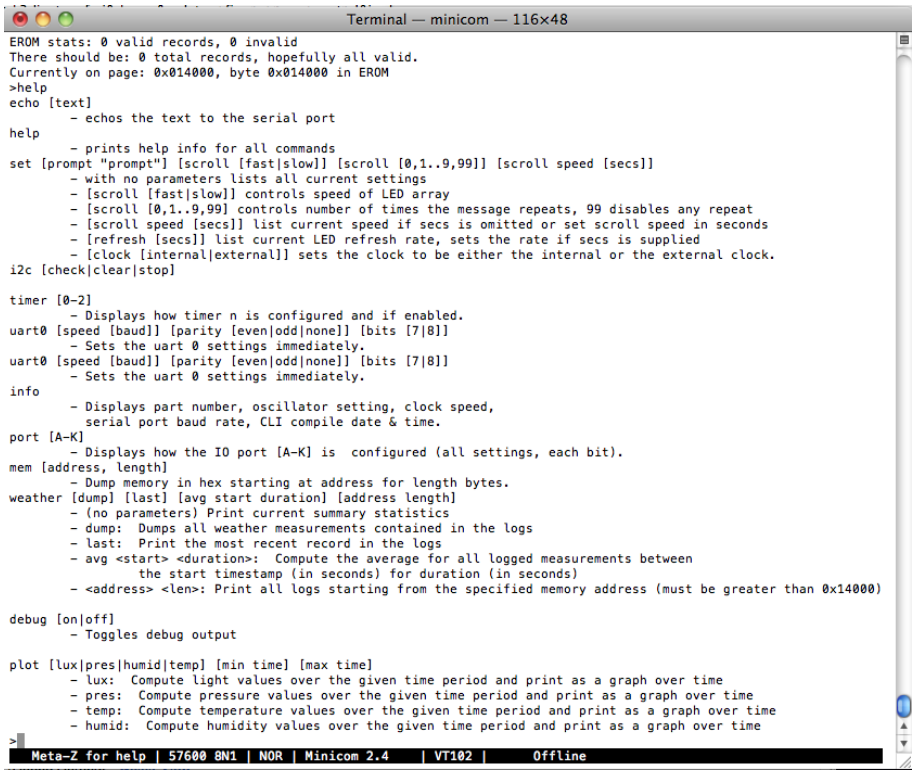

Figure 2: Shell Help

The central control unit provides a simple interface for viewing current conditions at each station using the LED panels built into the ZNEO contest kit. Each of the three stations (central unit, remote unit 1, remote unit 2) can be selected using the three switches on the contest kit board. The currently selected station is displayed using the auxilary modem LEDs. A fourth switch is wired to PF5, and changes which value is displayed on the LED panel. It cycles through temperature (in F), pressure (in inches of water displaced), relative humidity, light (the resistance in KΩ is displayed), timestamp of last measurement, and last received signal strength (in dBm). More detailed information on weather logs is available through shell commands over serial. Supported commands are described in the help output in figure 2.

The weather command prints records according to some filtering criteria, either dumping all records, printing the last received record, or printing all records at a range of memory addresses. The plot command creates bar graphs of measurements over a specified time interval.

Implementation and Construction

XBee Configuration

Configuration of XBees is done prior to installation to define the rate of data polling, the interval at which updates are sent to the central controller, which central controller data should be reported to, and how the remote unit is identified to the central control unit. The configuration details are listed below.

AT Command	Value	Description
Remote Unit		
MY	1111 or 2222	16-bit address
ID	3317	Personal Area Network (PAN) ID
IR	0x9C4	Pin Sample Rate of 2.5 seconds
IT	1	Transmit after every 1 sample
DL	1234	Destination XBee is central unit
D0	2	Digital IO Pin 20 in Analog-Digital Conversion (ADC) mode
D1	2	Digital IO Pin 19 in ADC mode
D2	2	Digital IO Pin 18 in ADC mode
CH	C	802.15.4 Channel C
VR	10e8	Firmware version 10e8
Central Unit		
MY	1234	16-bit network address
ID	3317	PAN ID
IR	0	Disable DIO/ADC lines
IU	1	Enable UART output
IA	FFFF	I/O Input Address, accept all packets
CH	C	802.15.4 Channel C

Table 2: XBee Configuration AT Commands

The frame processing logic requires that the two sensors IDs be 0x1111 and 0x2222 in order for the LED panel display logic to work properly. The remote units' DL setting must match the central unit's MY setting. The remote units' IR setting is too frequent for persistent use, but provides quick feedback for demos. When configured for a persistent installation, the data rate should be reduced.

The remote sensor stations are driven by XBee series 1 radios. Attached to each XBee sensor were a Honeywel HIH-4000 humidity sensor, an Advanced Photonix Inc (API) PDV-P8001 CdS Photoconductive Photocell, and an Analog Devices TMP 36 temperature sensor. The HIH-4000 and TMP-36 are analog sensors which vary voltage out as a function of the humidty and temperature measurements respectively. The PDV-8001 varies resistance as a function of the amount of light exposed. This was inserted in a voltage divider circuit and connected to an XBee analog pin. Pins are read at some interval (configurable

for each XBee via AT commands). The in-class demo used 5 ms poll intervals. After every poll, the measurements are sent to an XBee radio connected to the ZNEO central control station. Measurements are retrieved through a UART as XBee data frames. The frames are parsed and records are inserted into the logging database.

XBee radios are configured in two ways, depending on whether it will be used as the central controller or as a remote sensor. The central controller is configured to have the ID 1234 (hex) using the MY AT command. It is also configured to accept analog pin readings from all sensors using IU FFFF.

Remote sensor XBee radios must be configured to send communications to the central controller by issuing the DL AT command. They must also be configured to read from pins 18-20 as analog measurements using the D0, D1, and D2 commands with argument '2.' Finally, the sample interval must be set using the IR command and the number of samples to accumulate locally before broadcasting is set using the IT command. For the in-class demo, remote sensors were configured with IR xxx and IT 1, so that they would quickly communicate changing conditions to the central controller. For longer-term data logging, and IR of about 5 minutes may e more appropriate, and the IT could hold a few samples, perhaps up to and hour or so, befor transmitting them to the central unit. This would reduce power consumption and would help forestall exceeding EROM capacity, which requires overwriting data.

Software

The software is broken into libraries which group related functionality into seperate components, as shown in figure 3. The Central Control Application Logic is contained in main.c. It initializes all necessary libraries and contains the central control loop. The shell library (shell.h / shell.c) accepts input from the serial port over UART0 and interprets commands to configure the central control unit and to query the weather record database. The central logic registers three additional shell commands: weather, debug, and plot. The settings library allows dynamic configuration of clock speed, UART baud rate, and LED scroll and refresh rates. The LED library displays messages on the central unit's LED panels. Messages are scrolled column-by-column continually in response to switch events which are interpreted in the button_isr and update_led methods in main.c. The button library (buttons.h / buttons.c) does button debouncing. The clock library controls setting and querying the clock speed.

The XBee API library interfaces with an XBee operating in "API" mode (Digi's term). An XBee in API mode sends data in a nested frame format (as is usual for data exchanged over a network). Interpreting these "API" messages requires parsing the frame structure to extract the necessary information. The XBee API library (xbee_api.h / xbee_api.c) defines structs and methods for interpreting that data. Parsing is handled by defining a struct with variables of byte lengths and order that exactly match a section of the frame. An arriving frame is buffered in xbee_api.c in a buffer, and when it has been completely received, it is interpreted by casting the pointer to the buffer as a struct pointer

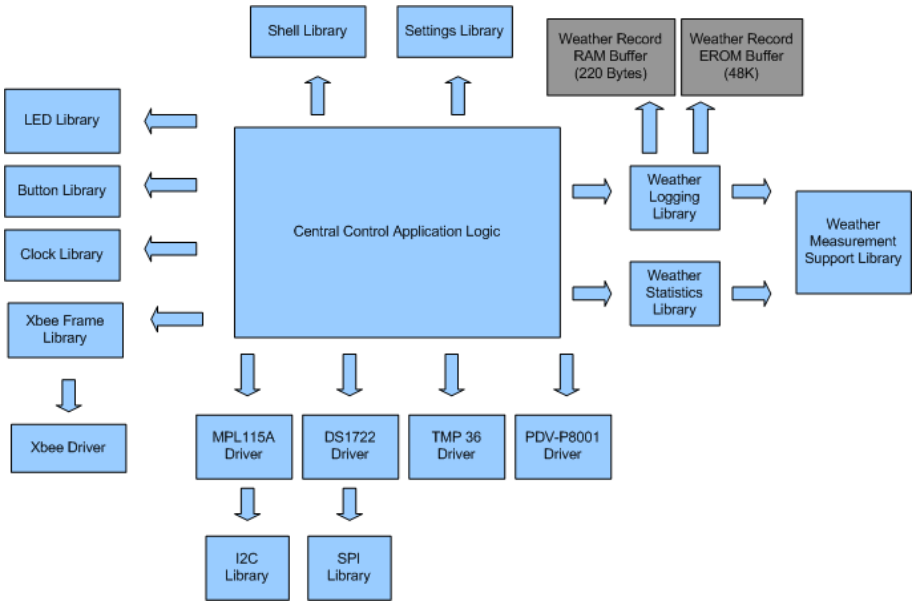

Figure 3: Software Block Diagram

(xbee_frame *). Fields can then be individually accessed through the struct without requiring any other parsing logic. This is possible because network frame fields and lengths are well defined and do not vary. The XBee API library supports the common frame header, and the I/O Data contents (for API Command ID 0x83). More details on XBee API mode can be found in [1], and [2].

The XBee API library relies on an underlying XBee driver (xbee.h / xbee.c) to read characters from UART1. This is a ZNEO specific library, as it interacts directly with status registers for UART1. Characters are read by polling, checking the UISTAT0 register for new data, and storing characters in a buffer. Each time XBee API is polled via the xbee_check_frame method (once per main loop cycle), it checks for a new character. If that character represents the frame start character (0x7E), the frame buffer pointer is reset to index 0. As more characters are received, the pointer is incremented until the frame length is reached. At that time, xbee_check_frame will return 1, indicating there is a new frame available. Further interact with the XBee frame can be done using xbee_get_source, xbee_get_rssi, xbee_get_samples, xbee_print_frame, and xbee_print_raw_frame functions.

The MPL115A library (MPL115A.h / MPL115A.c) interfaces with the Freescale MPL115A2 I^2C Digital Barometer [3]. It relies on an underlying I^2C library (i2c.h, i2c.c) to for support methods, but does interface directly with ZNEO I^2C control registers. The library's responsibilities include reading coefficients from the chip initially, caching the coefficients locally, reading pressure values

from the chip, and using the coefficients to adjust the read value. This library took a significant portion of time to write and debug. It relies heavily on detailed instructions, equations, and sample code found in [4], although there were some errors. The most notable error is that the start conversion command (0x12) must be followed by another byte (0x01) before sending a stop bit. This was only documented in the "I^2C Simplified for Communication" section on page 12, and was not included in the (otherwise) more detailed I^2C Write Mode / Read Mode sections. Extracting the floating point coefficients from the response bytes was also not straightforward or well explained in the documentation. I eventually determined the correct interpretation of the bytes after reading suggestions on the Freescale Internet forum. I write the parsing code myself after consulting with posted example code. The results are in the coeff_to_float method in MPL115A.c.

The drivers for other sensors involved only minimal code. The TMP36 library (TMP36.h/c) consists only of the equation to convert an ADC voltage measurement (relative to 3.3V) to a temperature in Celsius, and logic to convert Celsius to Fahrenheit. The PDV-P8001 library (PDV-P8001.h/c) consists of a similar conversion formula, and a method that interfaces with the ZNEO ADC to read from PB0/ALG0. The latter method is used by the central unit when constructing local measurements. The DS1722 library (ds1722.h/c) relies on an SPI library (spi.h/c), both of which were supplied by Dan Eisenreich as example code. The DS1722 temperature sensor is used to measure the temperature at the central control unit.

The weather logging and statistics libraries are the largest and most complex of the software required for this project. The weather logging library (weather_log.h/c) defines the weath_meas struct which defines the contents of a measurement record and the logic required to store and retrieve as many of those records as possible. Each record consists of four floats storing humidity, lux (actually stores resistance), pressure, and humidity; an four-byte integer timestamp, and a two-byte short station identifier (the XBee node source's 16-bit network address is used). This results in a 22 byte data record.

The major functions are put_measurement, which inserts a record into the database, and the iterate methods (iterate_all_logs, iterate_all_erom_logs), which take a pointer to a callback function which is called once for every log in the database. The logging library manages two separate memory spaces which buffer records. The first is a small buffer in RAM, which is restricted to 10 records (220 Bytes) due to the limitations of 4KB of available RAM. The second buffer consists of 24 pages of EROM Flash at the top of the address space (0x14000 to 0x1FFFF). Each page of flash memory is 2KB, meaning the EROM buffer is 48KB and can store about 2,200 records before overwriting data. As records are inserted via the put_measurement function, they are added to the RAM buffer. When the RAM buffer is full, the contents are written to the next available location in EROM and the RAM buffer is cleared. The sequence by which the EROM buffer is updated is depicted in figure 3.1.

In addition to managing the two data buffers, the logging library also contains logic to iterate through those buffers from any point for a given number of

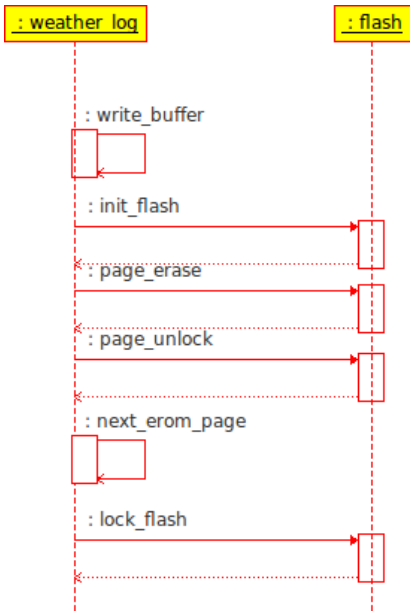

Figure 4: Log Insertion Sequence

records. The three iterate methods (iterate_all_logs, iterate_all_erom_logs, and iterate_ram_logs) take function pointers as arguments. The function is called for each record that is iterated over. This allows application logic to exist apart from detailed knowledge of the buffer structure or memory locations, and also facilitates reuse of buffer traversal logic without copy-paste duplication. The iteration logic was surprisingly difficult to debug, so this level of modularity was critical to isolating bug fixes to a single method and elliminate the copy-paste-errors development anti-pattern.

The weather logging library responsibilities are managing the two data buffers to keep it to a managably size. Application logic is implemented in the weather statistics library (weather_stats.h/c). It currently defines three types of methods: w_histo_temp, w_print_stats, and w_get_average. The histo functions (there is one for each of the four measured weather parameters) use the logging library's iteration logic and maintain min, max, and average for all records traversed. The variation of a particular value over time is then displayed over the serial port as a horizontal bar chart. This was initially intended to be a histogram, but binning as a function of time was more informative than as a function of value. In this manner, the system can show changes to values over time. The sequence of calls that provide a graph of conditions over time is shown in figure 5.

With all of the aforementioned libraries in place, the main loop's responsibility is to alternately poll the serial input for shell commands, and the XBee for incoming frames, adding weather records to the weather log as they arrive.

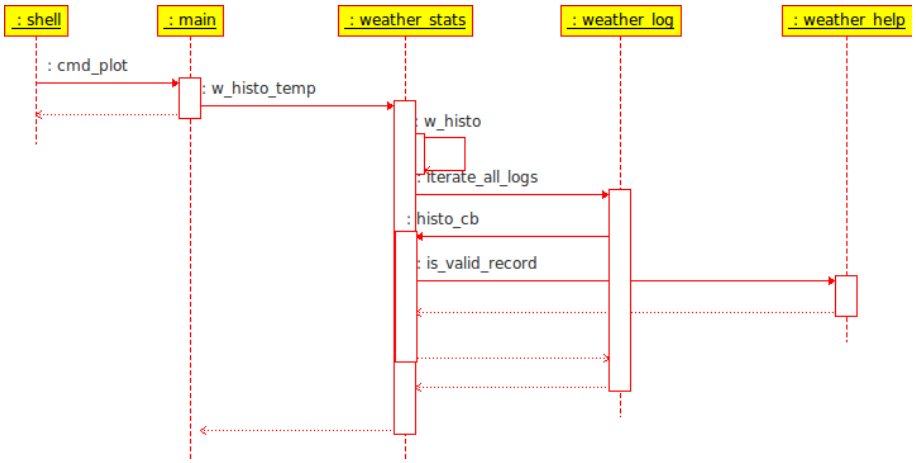

Figure 5: Plot Command Sequence

Pushbuttons are polled periodically, and local conditions are measured occasionally (the interval was set to 1 second for the demo, but should be must less for practical use). As conditions are updated, or in response to pushbutton controls, the message on the LED panel is change to reflect current conditions at a selected sensor unit. Display for any of three sensor units are selected by pressing one of the switches on the ZNEO Contest Kit. A fourth pushbutton cycles through the weather parameters and other informational messages (timestamp and latest received signal strenght).

Hardware

The hardware configuration for the project is very straightforward. Some soldering is necessary to assemble the adafruit XBee adapter kits, but there are excellent instructions on their website. Wires connect three I/O pins on the XBee to each of the three sensors on the remote units. The most challenging aspect of hardware design was accomodating both 3.3V and 5V sensors, and driving them from a battery power source. The XBee and TMP 36 sensors require 3.3V power, but the HIH-4000 requires 5V power. Conveniently, the XBee adapter board includes a 5V to 3.3V regulator which was able to provide sufficient current to drive the TMP 36, the XBee and the photoresistor hardware. The HIH-4000 was driven from 5V, which was downconverted from a 9V battery using an LM7805 5V regulator. The beginning electronics tutorial on www.sparkfun.com provides very detailed instructions on using the LM7805. With voltage regulation in place, the remaining step was to down-convert the output of the HIH-4000 to be below 3.3V to avoid saturating the XBee's ADC (which uses 3.3V as its VREF). To do this, I used a voltage divider composed of a 10K Ω and 22K Ω and resistor. A circuit diagram of the remote units is provided in figure 6.

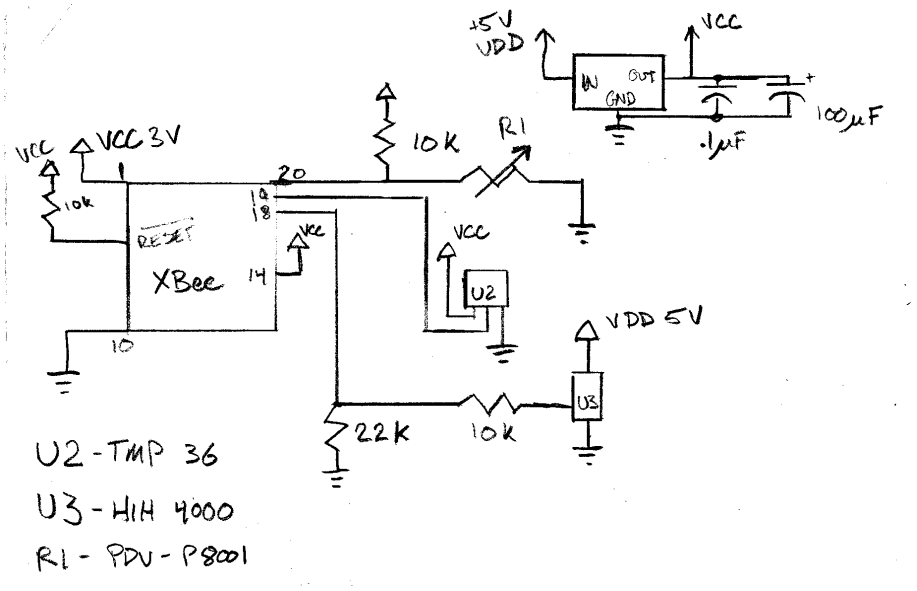

Figure 6: Remote Unit Circuit Diagram

The central control unit hardware could be operated entirely from the ZNEO Contest Kit's 3.3V VDD pins, as the HIH-4000 can be replaced with a much less expensive Parralax HS 1101. The Z16 can then interface with a simple RC circuit formed by the varying-capacitance of the HS 1101 and known quantity resistors. The frequency of the RC circuit can be measured and the capacitance can be calculated to determine current humidity. Unfortunately I didn't have time to implement this, as I didn't have the correct value resistors to follow the directions in the HS 1101 specification. The circuit diagram for the central control unit is displayed in figure 7. The XBee only needs one connection wire (DOUT/XBee pin 2 to RX / Z16 PD 4), since the central control unit doesn't transmit to remote units. The photoresistor is configured in an identical voltage divider as that used by remote sensors, and is connected to PB0/ALG0. A fourth pushbutton is connected to PF5 through a 10k pullup, just as the four pushbuttons provided with the contest kit. Finally, the MPL115A2 I^2C chip is connected to PA6 and PA7 for Z16 SCL and SDA alternate functions.

Photos of the completed project follow. In the photos, one remote sensor is powered off a 9V battery, the other is powered from a variable-voltage wall-wart. The central unit is on the largest (red) breadboard.

Retrospective

I approached the project with the primary goal of learning about ZigBee and XBee radios. Secondary goal was gaining more experience working with analog

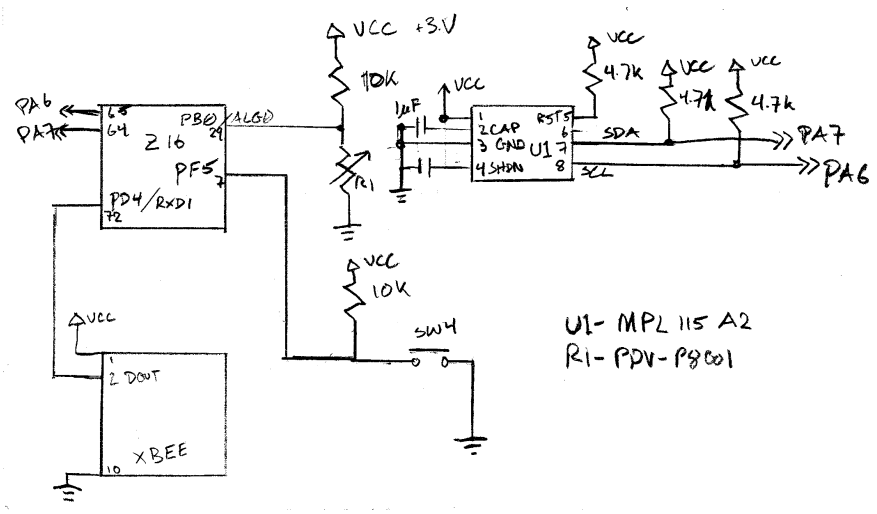

Figure 7: Central Unit Circuit Diagram

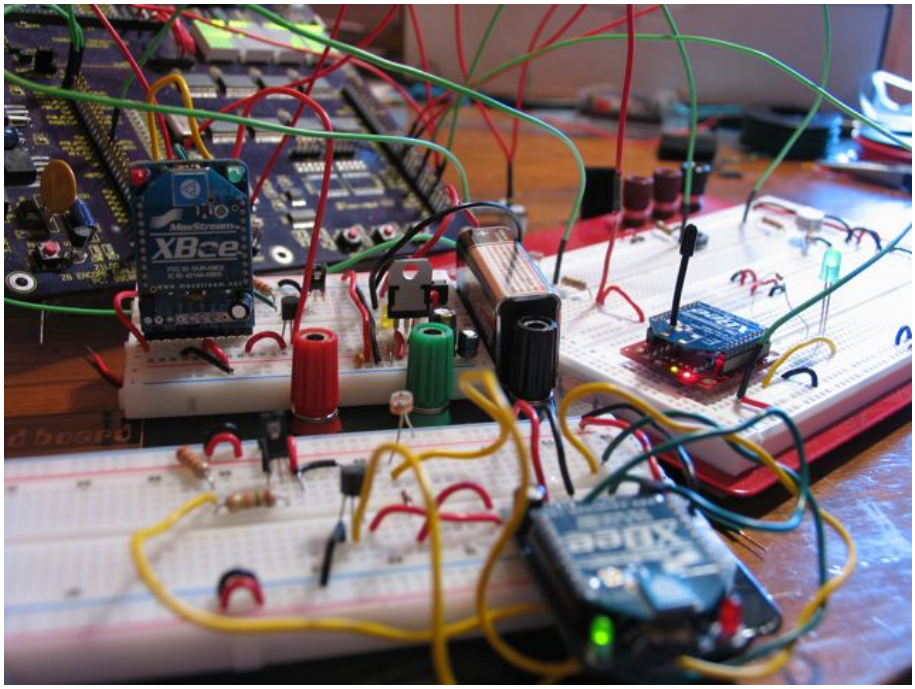

Figure 8: Weather Measurement System

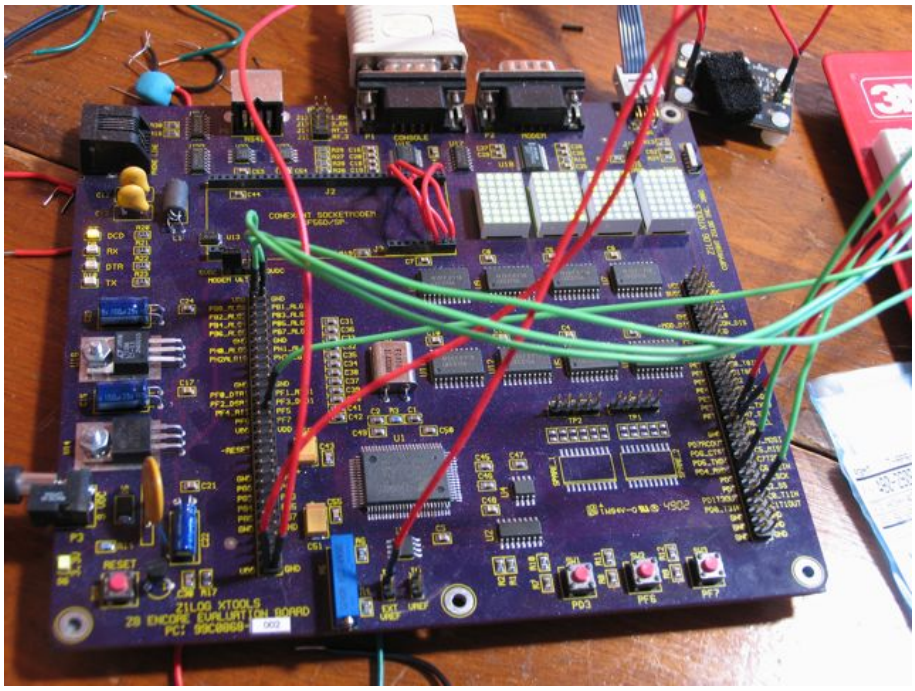

Figure 9: Z16 Flash Microcontroller Contest Kit

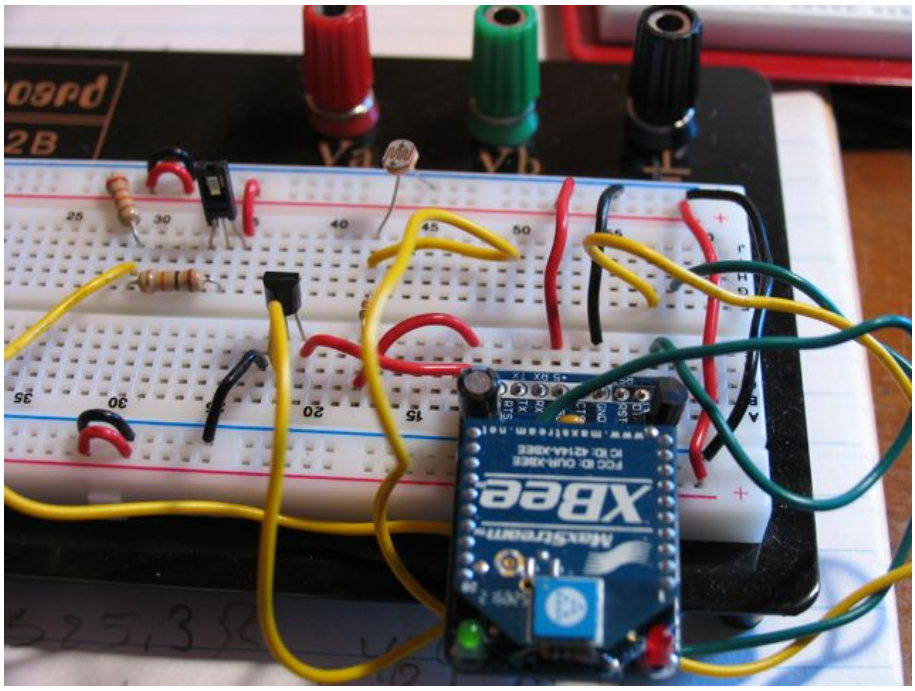

Figure 10: Remote Unit 1111

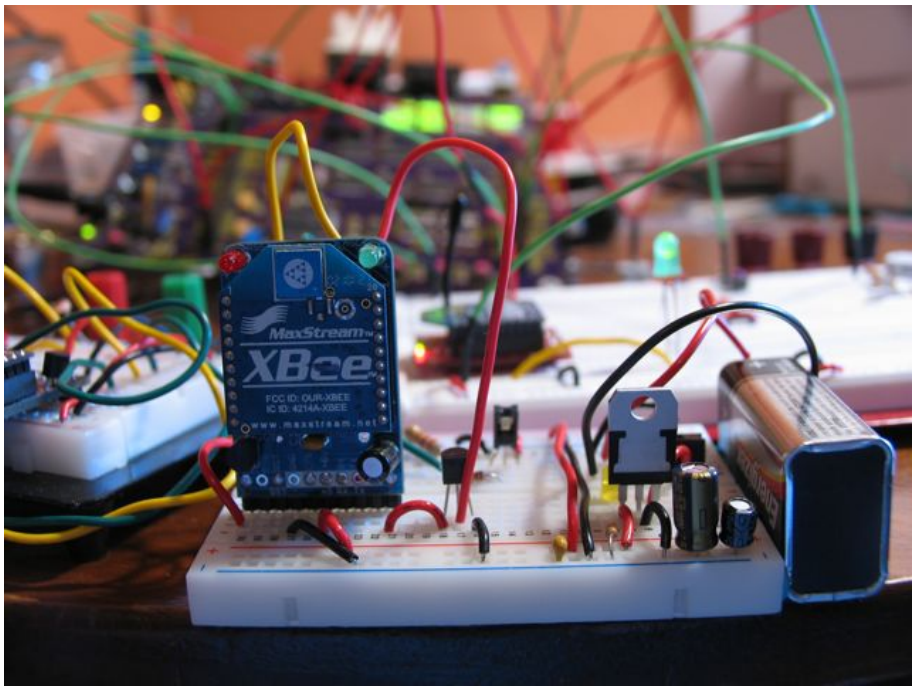

Figure 11: Remote Unit 2222

sensors, and having a system which could monitor conditions at home, giving better data with which to understand energy usage and to inform gardening decisions (like which window is best to put seedlings in, and when its safe to keep plants outdoors overnight). The primary goal influenced a major design decision: using XBee radios. I choose series 1 radios because I had them available from a previously planned project (Adafruit's Tweet-a-Watt), and because reading suggested they were easier to configure. Series 2 (or 2.5) XBee's would be necessary to leverage higher level ZigBee features, which would be required to interface with commercial ZigBee devices. I found series 1 configuration very easy, and think it was the correct choice of technology for the project.

The second most important design decision was to limit remote units to include only the control logic available on the XBee itself; no additional microcontroller would be used at remote units. This decision avoided some complexity and significant cost, but added certain limitations as to which sensors could be used. This was a mid-course decision, which I made after learning of the built in Analog-Digital converters in XBees. It required a switch from the less expensive Parallax HS 1101 Humidity Sensor (which varies capacitance as a function of humidity) for the more expensive Honeywell HIH-4000 humidity sensor. The HIH-4000 that I purchased was about $20, and included per-part calibration information, which was overkill for the project. More exhaustive searching may have turned up other analog humidity sensors with less detailed calibration for less cost. The late switch to HIH-4000 added an unexpected hiccup when they arrived, and further reading of the datasheet revealed a requirement for 5V power supply. All other sensors used with the remote unit required or were configured for 3V power. Fortunately the XBee adapter boards I used (from Adafruit) had built in 5V to 3V converters. I was able to power the HIH-4000 and the XBee adapter boards from 5V on breadboard rails, and run 3V from the XBee adapter chip to the TMP-36 and PDV-P8001 sensors. Because the XBee VREF is limited to VCC, the HIH-4000 output voltage had to be divided to bring it down to 3.3V. This oversight when selecting the HIH-4000 part could have been very inconvenient. It was fortunate to have the 5V to 3.3V convesion already featured on one of the included components.

The last major design decision was a system for data storage. The most significant aspect was choosing what type of data to store, and where to store it long term. Initially I had thought RAM would hold a sufficient number of records to avoid needing to address the problem until the system needed weeks of uptime. However, consulting the map file produced by the ZDS II compiler indicated there was very little RAM available in the 4K featured on the Z16. My first attempt was to use the 16K I2C external memory chip, for which drivers were already available. However, there were conflicts between that chip and the I2C pressure sensor. After a few days of working the problem, I eventually had everything working except the memory chip wouldn't reliably release the SDA line, causing the Z16 to always report busy I2C bus status after interacting with the memory chip. Eventually I gave up on the external memory when I realized the Z16 has 128K of ROM.

After consulting the map file again, I decided to use 24 pages at the top of

EROM for data logging. This gave a comfortable buffer between the current size of the central control unit program and the beginning of data logging, in case program modifications were necessary in the future. Each page of ROM is 2K, providing 48K of ROM for persisting weather measurements long term.

Weather measurement records consist of four floats (4 bytes each), one four byte integer timestamp, and a two byte short storing the station identifier from which the measurement originated. Each record is 22 bytes long. For simplicity, records will not wrap between pages, so each page can store 93 records. This allows for 2,232 measurements in ROM. After all ROM pages are full, the buffer will wrap around, erase the first data page, and overwrite records starting from the oldest first. An improvement on this algoithm could avoid completely overwriting old data by reducing the first 23 pages into the final page, then beginning to overwrite again at the first page. This would allow data to be persisted indefinitely, though with progressively less and less time resolution as the 24 page buffer is cycled through.

Implementing this buffer in EROM proved more difficult than anticipated. The complication of needing special erom pointers with the complexity of maintaining a circular buffer and requiring debugging on the Z16 made this a more complex task than expected. In hindsight, implementing a circular buffer doesn't seem like it should have been as difficult as it proved to be.

Attachments

- ZNEO Specification
- Contest Kit Specification
- XBee Specification
- HIH-4000 Specification
- TMP 36 Specification
- MPL115A2 Specification
- DS1722 Specification
- HS1101 Specification
- How to Implement the Freescal MPL115A Digital Barometer

References

[1] MaxStream, "XBee / XBee-PRO OEM RF modules product manual v1.xAx," 2007.

[2] Digi, "XBee / XBee-PRO OEM RF modules product manual v1.xEx," 2009.

[3] Freescale, "Miniature i2c digital barometer product specification," 2010.

[4] J. Young, "How to implement the freescale MPL115A digital barometer," 2009.