

## Project Final Report

### **Project Final Report Wireless Sensor Station over RF**

Daniel Rubin  
rubin@gwu.edu  
April 24, 2007

#### ***Project Abstract***

This project involved the development of a wireless sensor station communicating with a base station over a radio link. A pulse-width modulator temperature sensor fed readings into a smaller Z8 board, which was connected to a Radiotronix 433 MHz transmitter. On the main Z8 board, a Radiotronix receiver parsed a packet and extracted a floating point value. That value represents the temperature, and it was displayed on the LEDs.

I had to learn how to interface with the radio transmitter and receiver, implement an error correcting protocol similar to a UART – start and stop bits, a parity bit, and a packet checksum.

This was a very interesting and practical project. The final result is similar to wireless thermometers available to consumers, where a sensor is placed outside, and an LCD shows the reading.

#### ***Status***

The two Z8 boards communicate properly - the small Z8 with the temperature sensor constantly broadcasts a packet with the binary representation of a floating-point number, and the main Z8 constantly tried to receive a packet, displaying the contents on the LEDs when an inbound datagram is properly constructed.

I successfully got the temperature sensors working on the smaller Z8, which was not too difficult, but it was giving me problems for a little while.

Perhaps using a UART would have saved some problems, but it was very rewarding to go from bit banging on the radio to sending coherent data with reasonably high-level API. Developing the right technique for error correction was a challenge. Implementing it was challenging as well.

Fortunately I already had a good LED API, and I only had to worry about how the interrupts affected the radio interrupts. Other than that, the LEDs worked great.

One feature that I was unable to implement was a two-way protocol. It was so much work getting the uni-directional protocol working. Also, the protocol would not have added much value – why wait for a request to send the temperature when the sensor station can always send the temperature? I suppose for power consumption, which might be a good feature. A positive benefit to not having two-way communication is that the two Z8s can be set at a distance from each other, to further demonstrate the wireless nature of the communication. When both receivers were used, they both went into the breadboard, which meant they had to be adjacent to each other.

### *Specification*

Overall Design:

Temperature Sensor → Z86423 → RF Transmitter  
..... Radio Cloud.....  
RF Receiver → Z86401 → LED

The temperature sensor reused the code written for lab 3, with minor tweaks. The GPIO ports were different, and it needed to deal with interrupts to prevent the data from getting too corrupted. Also, the number of samples was adjusted. Too few samples, and the data can be wild, too many, and the operation is too slow and performance suffers. Finding the correct pins was a little bit trickier, since the Z86423 does not label the pins. With a pinout sheet from Zilog, it was straightforward enough.

Programming for the Z86423 was extremely easy, as it was virtually the same as the main Z8 used throughout the class. The project settings were different. The exact same serial on-chip-debugger connector was used, which worked out fine. Not having LEDs made it a little trickier to determine its status, but since it was similar, issues that needed live feedback were done on the main Z8.

The RF transmitter was simple enough when things started – antenna, ground, power, and data. But sending a single data bit is not especially helpful, so much abstraction was developed for the transmitter to be able to send a byte, poll to determine if the byte had been sent, then move on. The transmitting module will be described in the next section.

Receiving was a little easier, I just had to follow all of the rules established for transmitting, and know what to do when problems arose.

The Z86401 was thoroughly understood thanks to the rest of the course. The same goes for the LEDs, which were improved to be timer-based after the first lab.

Equipment used:

PWM 1-Wire MAX6676

MAXIM 6676 Low-Voltage, 1.8 kHz PWM Output Temperature Sensor

Zilog Z86423 Z8 Encore! (Small Z8)

Zilog Z86401 Z8 Encore! (Normal Z8 used throughout course)

Radiotronix 433 MHz Transmitter and Receiver

RCT-433-AS Transmitter

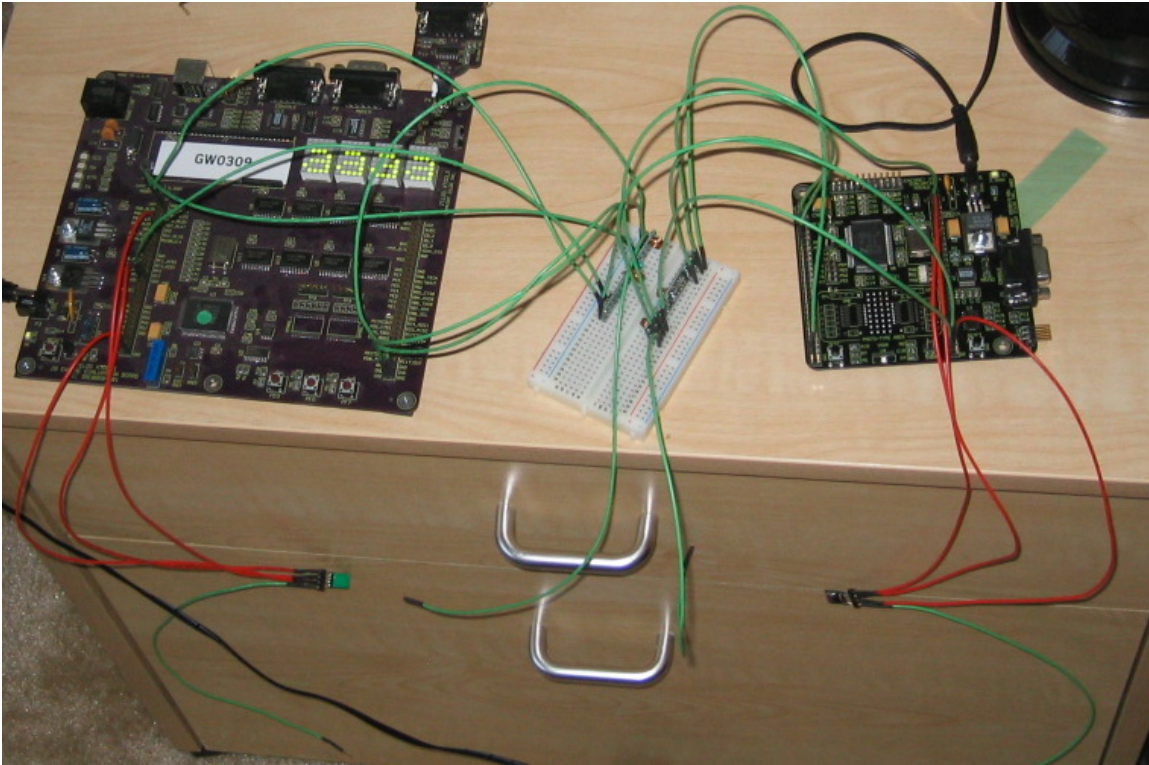
RCP-433-RP Receiver

The code written specifically for this project was the receiver and transmitter modules. These needed to compliment each other in order for the devices to properly communicate. Other modules were utilized:

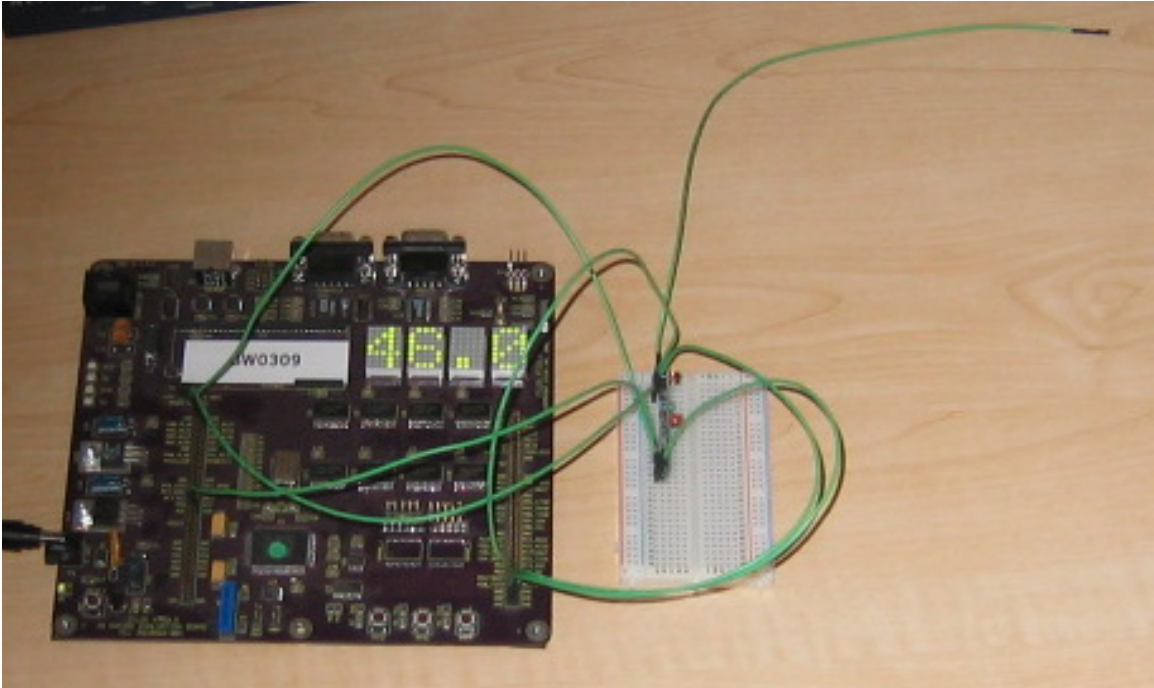
- Timers – to trigger an interrupt for when the radios should send/receive
- LED – for the main Z8 to display the results
- Temp sensor – the small Z8 needed to interface with the sensor to get the data

The main application for each Z8 needed to interface with all of these modules in a logical manner so that it could function properly.

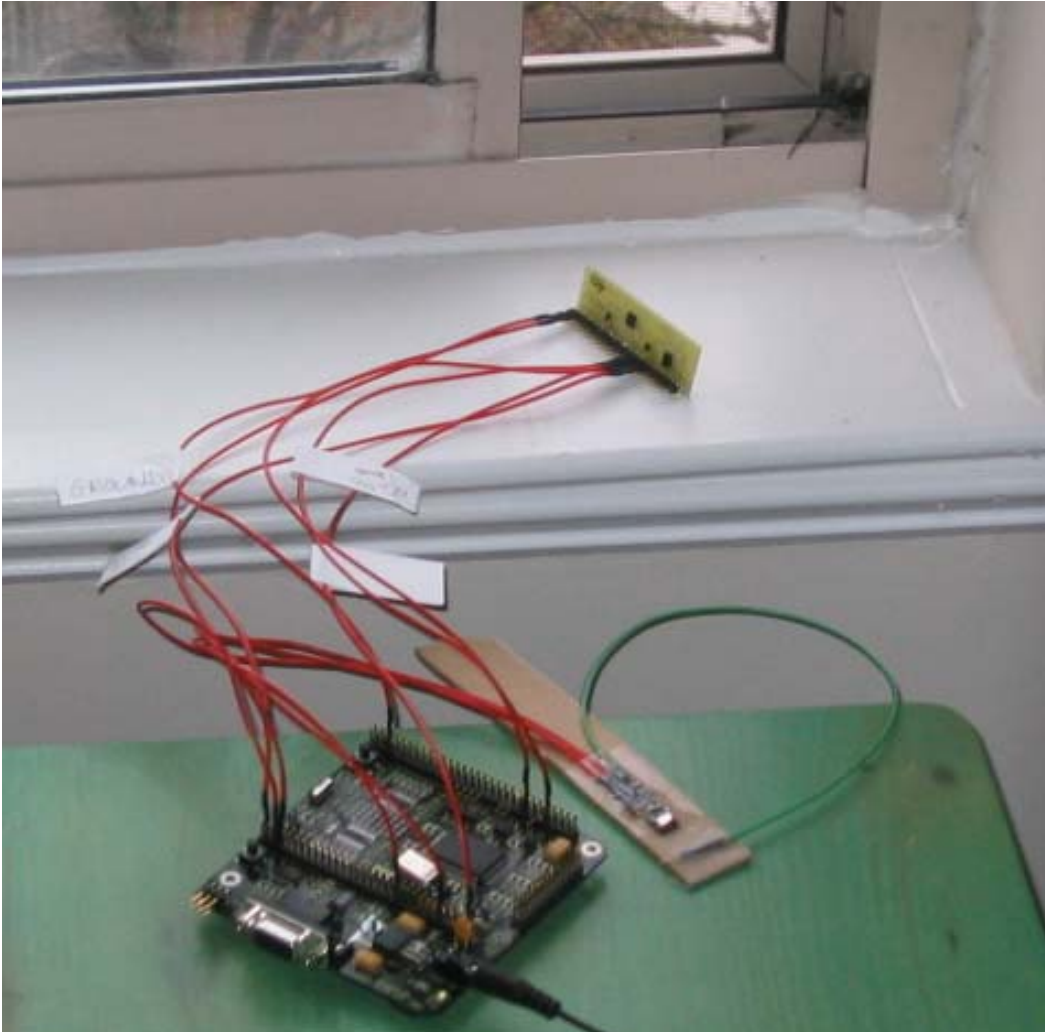
## Pictures



This is how the two boards were initially set up after the radio communications had been first implemented on a single board. Each board has a receiver in the breadboard, and they each have their own transmitter. The first thing that had to be done was to turn off the transmitter on the display board (left) and turn off the receiver on the transmit board (right) so that I knew I was sending data and receiving it correctly. The first few iterations were just sending bit patterns – 0x00, 0xff, and here 0xaa. It's possible that it was trying to send 0x55, but the framing was off and it came up 0xaa. Those were the primary problems before a robust error-correction system was implemented. Start and stop bits framed each byte, and each packet would have a known start and stop byte and a checksum byte.



Once the sensor station system was working, the unused equipment was removed. Here, the main Z8 displays the temperature it received from the smaller Z8. The top jumper cable is serving as the antenna.



Here is the smaller Z8, with the radio transmitter and temperature sensor attached. It was set up by the window to show that it could get the outside temperature, and the overall system would still work if the devices were several meters from each other. The transmitter pins are taped to a piece of non-conductive material (cardboard) because the jumper connectors did not snugly hold the transmitter pins. A source of great frustration would be to attempt to debug why the communications ceased (a programming bug?) only to find out a pin came loose on the transmitter. The breadboard held the receiver pins just fine.

### ***Implementation & Construction***

Radio Project Pin outs

Main Z8

Receiver  
RCR-433-RP

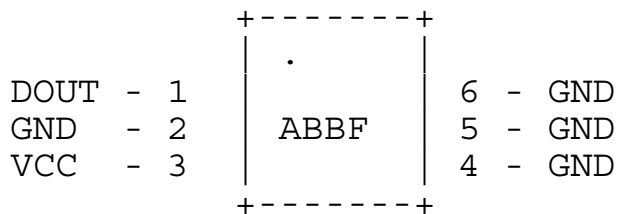
- 1 ANT - Antenna, 12" M-F Jumper
- 2 GND - GND
- 3 GND - GND
- 4 VCC - VDD
- 5 VCC - VDD
- 6 ANALOG - not connected
- 7 DATA - PB1
- 8 GND - GND

Small Z8

Transmitter  
Radiotronix  
RCT-433-AS

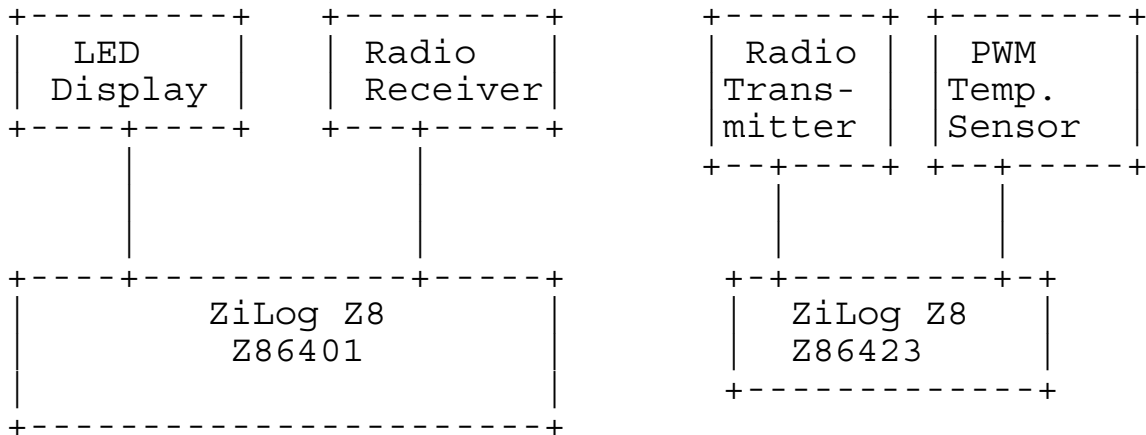
- 1 ANT - Antenna, 12" M-F Jumper
- 2 GND - JP2-56 GND
- 3 DATA - JP2-21 PC1
- 4 VCC - JP2-9 VCC 3.3V

PWM Temp sensor  
PWM 1-Wire MAX6676  
MAXIM 6676 Low-Voltage, 1.8 kHz PWM Output  
Temperature Sensor

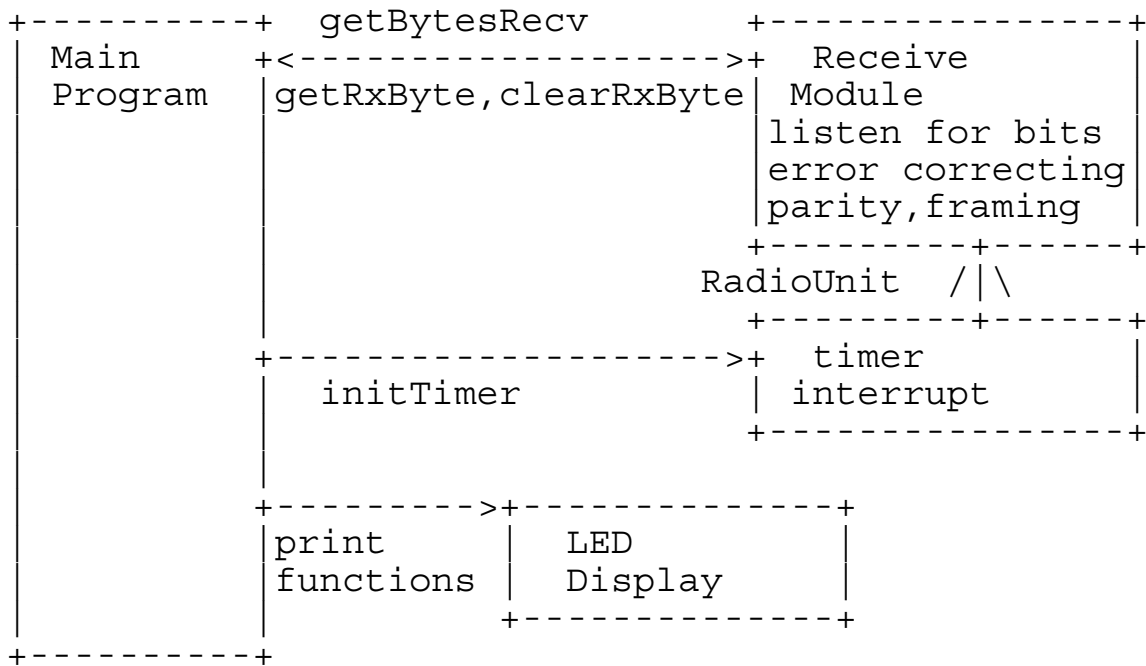


- V+ - JP2-59 VCC 3.3V
- GND - JP1-55 GND
- 6 - JP1-9 GND
- 5 - GND via GND Bus
- 4 - GND via GND Bus
- 3 - JP2-55 VCC 3.3V
- 2 - JP1-15 GND
- 1 - JP2-1 PE7

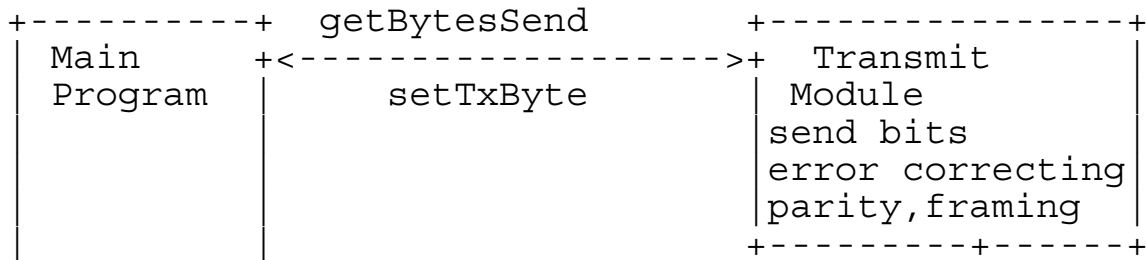
### Hardware Block Diagram

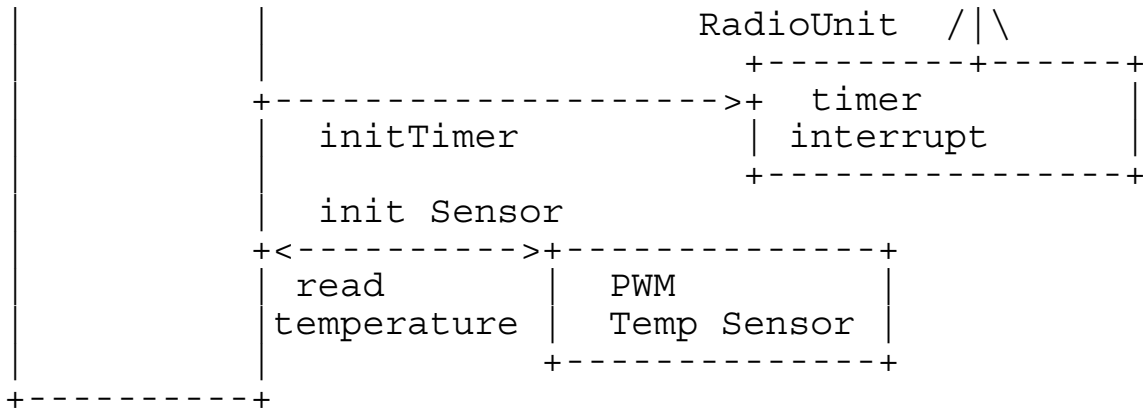


### Software Block Diagram Receiving Station



### Software Block Diagram Transmitting Station





The flow of the two programs, the sender and receiver, are both pretty simple. The challenge was in the implementation of radio communication itself.

They both initialize hardware into a known state – the LEDs, the timers, the temp sensor, and the radio transmitter and receiver.

The transmitter loops through sending a byte of the packet, and periodically getting a fresh temperature value.

The receiver constantly pulls in bytes. First it keys on the byte indicating the start of the packet – 0x39, then checks when it has received enough bytes to fill an entire packet. It checks that the last byte is also correct- 0x7f. Then the checksum is computed. If the checksum is correct- 0 – then the temperature data is extracted and displayed on the LEDs. If any of those steps fail, the receiver goes back to waiting for the first byte.

Extracting the temperature involves a little extra work since the four-byte value is encoded using 5 bytes. The high bit of each byte is a parity bit, so those bits are aggregated together in the 5<sup>th</sup> byte.

Note that a 4-byte float can be viewed as b[0] b[1] b[2] b[3], with b[0] as the most significant byte.

The example is for 70.5, or 0x42 0x8d 0x00 0x00

Packet diagram:

Byte	0	1	2	3	4	5	6	7
Holds	0x39	B[0] & 0x7f	B[1] & 0x7f	B[2] & 0x7f	B[3] & 0x7f	High bits hhhh000	Checksum	0x7f

				^ 0x55	^ 0x55			
Example	0x39	0x42	0x0d	0x55	0x55	0x20	0x09	0x7f

Parity bits aren't shown, even parity is used. That means that all bits xor'd together should be 0, or there should be an even number of 1's.

Tests were done on a single Z8 when possible, which was reasonable. I could send a bit to a transmitter and receiver on the same board, to make sure I could properly transmit a bit using the equipment. Another test was to have the transmitter broadcast a known bit pattern, and I would read it to determine what the problems were. That was how I realized (with the instructor's help) that I needed to frame my data with start/stop bits. I would send 0x55 and read 0xaa, something where my data was shifted by one bit.

Known good versions were saved, and when problems occurred, these would be referenced extensively. Incremental progress was key.

### *Retrospective*

Important design decisions:

Not using the UART, and implement the error correction myself

Perhaps I could have saved some grief and attempted to use the UART for framing the bytes, doing error correction, and offering status register bits to indicate when data had been sent. I chose to implement it myself after getting pretty far along when I was doing basic tests to confirm that the radios works. By then, it became an interesting problem to solve, and throwing away that work would not have been as educational. For a real production system, the UART should have been considered and implemented as an option, in order to compare the two for issues such as size of code, complexity, and operating speed.

One-way communications

One result of needing to take the time to implement the radio communications by hand was that I did not have time to create a two-way protocol. Another reason was that the application did not lend itself to two-communications in a natural way. What would the receiver send back? An acknowledgement? How would the sender do anything different with that knowledge? They would still keep broadcasting the temperature as it changed, rather than resending old data.

What did I learn

How do combine large functional modules of an embedded system, and how to apply the skills to go from data sheet to working implementation. The wireless temperature station combined many modules already created, along with a few new ones. That process had to be done in an organized, methodical manner to prevent disrupting the prior functionality. I took the time and effort to make that work.

Taking the radio transmitter and receiver data sheets, and understanding how that impacted my design was rewarding. I learned how to physically connect the parts, set up the Z8 GPIO, and then use the functionality of the hardware. This end-to-end process was challenging and interesting.

What would I do different

I would consider buying my own equipment so I could continue tinkering after the class was over. I'm overall satisfied with my approach, and how I managed to deal with the roadblocks along the way. Perhaps I would have bought fewer jumpers had I realized I didn't need so many, but I believe the company had a minimum order limit, so I might not have been able to save any money anyway.

### *Attachments*

Radio Data sheets (receiver and transmitter)

PWM Sensor data sheet

Source code for the Z86423 and Z86401

Pictures and one movie of the equipment in action

Brochure