envwi sense

ECE 403 Final Report

Timothy Albert & Ian Maines
May 2, 2014

The University of Maine

Abstract

The design, implementation, testing, and validation of a wireless temperature and humidity sensor network are described. Known as envwi sense, this sensor network is intended to measure the temperature and humidity of a remote location and wirelessly transmit the data to a central location to be logged and displayed. The wireless transmitters have a line-of-sight operable distance of approximately 450 feet. The project was calibrated and extensively tested for both sensor accuracy and for data transmission robustness using a climate controlled chamber. The project met and exceeded all specifications described in the project proposal. Temperature measurements were calibrated from 0^{o}C to 38^{o}C with an accuracy of $\pm 3^{o}$C. Humidity measurements were calibrated from 15% RH to 85% RH with an accuracy of ±8% RH. The data gathered by envwi sense is stored in CSV files, a database, and graphically displayed via a web page.

Table of Contents

List of Figures

List of Tables

1 Introduction

This report describes the design and implementation of envwi sense, a wireless temperature and humidity sensor network. Temperature and humidity are two important concerns when monitoring the climate of a building or other space. For example, computer data centers must maintain a specific temperature and humidity level to prevent damage to equipment. Homes and other buildings must maintain certain temperatures to prevent damage to plumbing and various other systems. Some devices similar to envwi sense are home weather stations which monitor exterior conditions and display them inside a home and a smart thermostat which monitors temperature within a home. With this in mind, envwi sense was designed to wirelessly monitor the temperature and humidity of remote locations and display that information to the appropriate personnel or systems.

envwi sense is a wireless sensor network used to display and log temperature and humidity data collected from various locations. The project has two main units. A "slave node" contains a temperature sensor and a humidity sensor. The slave node is battery powered and wirelessly transmits the temperature and humidity data, allowing it to be placed remotely. A "master node" collects all of the data transmitted from the slave nodes and stores the data in a database. The "master node" makes the values available in comma separated value (CSV) files which make it easy to open with Excel and other data analysis and statistical software. It also displays the humidity and temperature through a webpage. envwi sense can support many slave nodes. For the purpose of this senior project, only two slave nodes were built and used.

The project proposal for envwi sense may be found in Appendix A. The specifications for envwi sense are as follows:

1. Measure temperature from 0°C to 38°C with an accuracy of ±3°C
2. Measure humidity from 15% to 85% Relative Humidity (%RH) with an accuracy of ±8%RH.

3. Test ranges using the climate control chamber at the Advanced Manufacturing Center (AMC).
4. Design sensor circuits using unconditioned sensors to perform analog signal processing for microcontroller readings.
5. Design for sensor network communication with at least two slave nodes under 10 seconds.

The fourth specification means that the sensors must not be directly readable by the microcontroller, but must have some circuitry designed that would condition the output signal of the sensors for reading by the microcontroller.

Section 2 of this report discusses the functional operation of the project. Section 3 examines design details. Section 4 details the results of the project, and finally Section 5 is the conclusion.

2 The Breakdown

The breakdown section of this report describes the functionality of envwi sense and all of the major components that are used in the project. Figure 1 shows the block diagram of the project.

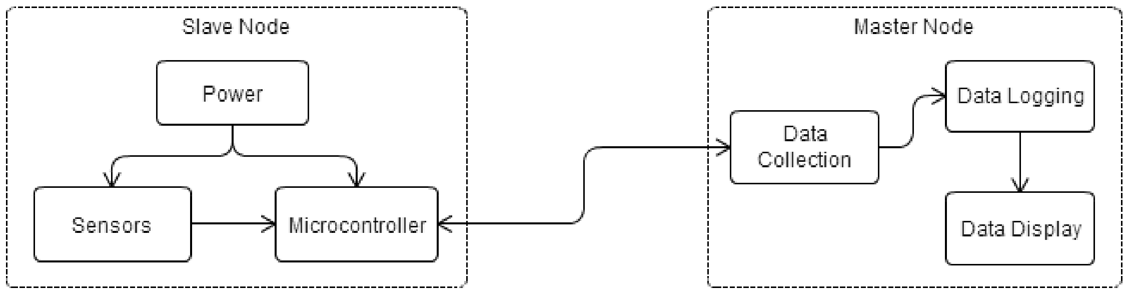


Figure 1: envwi sense Block Diagram

The hardware for envwi sense consists of two major circuits, the slave node(s) and the master node. The slave node power block consists of a battery with voltage regulation circuitry to provide power to the various components. The sensors block consists of sensors, sensor drivers, and measurement circuitry. The microcontroller is the brain of the operation, reading temperature and humidity and communicating this data to an XBee for wireless transmission to the master node. The master node consists or a Raspberry Pi computer running the Linux operating system, which is connected to a wireless transmitter used to communicate between each slave node and the master node. The wireless transmitter used is an XBee. The project was designed to be modular. Two main modules make up envwi sense, the slave node and the master node. The following sections are divided accordingly for ease of understanding and organization. Furthermore, the master node and slave node sections are broken down into hardware and software sections.

2.1 Slave Node

The slave node is where most of the electrical engineering theory went into practice. The slave node contains a temperature sensor, humidity sensor, current source, instrumentation amplifier, and a 555 timer circuit. It also contains a microcontroller that collects sensor measurements and sends the appropriate data to the XBee for data transmission. Each slave node has a unique address used to identify the slave node on the communication network. The slave node hardware is described in Section 2.1.1 and the software in Section 2.1.2.

2.1.1 Slave Node Hardware

The slave node contains physical sensors, measurement circuitry, a power supply, and data transmission circuitry. A printed circuit board (PCB) was designed for the slave node. The slave node consists of two PCBs, the sensor board and the communications board. The sensor board contains the sensors, sensor drivers, and power supply. The communications board contains the microcontroller, logic level shifter, and XBee. The PCB layouts are shown in Appendix B. Appendix C contains the project schematics and parts list. The functions of the different hardware on the slave node are explained in the following sections.

2.1.1.1 Power Supply

Hardware for the slave node begins with the power supply. The power supply consists of a single 9V battery, a +5V linear voltage regulator, and a +3.3V linear voltage regulator. The 5V linear voltage regulator powers the temperature and humidity sensor circuits. It also powers the microcontroller. The 3.3V linear voltage regulator provides power to the XBee. The PCB for the sensors was also designed to be driven by DC-DC converters as well as linear voltage regulators. The DC-DC converters are more efficient than the linear voltage regulators, therefore making the battery last longer and ultimately allowing for more temperature and humidity data to be collected.

2.1.1.2 Sensors

The next pieces of hardware on the slave node are the sensors. Two types of sensors exist on the slave node, one for temperature and the other for humidity. The temperature sensor is a resistance temperature detector (RTD). The RTD has a positive temperature coefficient, meaning the resistance increases as the temperature increases. The humidity sensor is a capacitive humidity sensor, meaning the capacitance of the sensor changes as the humidity changes. More specifically, as the relative humidity increases, the capacitance also increases. The two sensor circuits are completely separate from each other because they require very different biasing networks.

2.1.1.3 Sensor Drivers and Measurements

In order for each sensor to be measured, it must be excited with a specific signal by a sensor driver. The sensor driver for the RTD consists of a current source. The current source pushes a constant 100μA through the RTD. Since the resistance of the RTD changes with respect to temperature and the current stays constant, the voltage across the RTD will change with respect to temperature. The temperature is measured by amplifying the voltage across the RTD using an instrumentation amplifier. The instrumentation amplifier measures a differential voltage across the RTD and amplifies the difference between the two inputs. The amplified signal is measured by an analog-to-digital converter (ADC) pin on the microcontroller.

The sensor driver for the capacitive humidity sensor is a 555 timer. The humidity sensor is used to set the output frequency of the 555 timer. Since the capacitance changes with respect to humidity, the output frequency of the 555 timer will also change with humidity. The humidity sensor is measured using a frequency counter on the microcontroller. The frequency counter is further explained in Section 2.1.2.3 of this report.

2.1.1.4 Microcontroller and XBee

The final pieces of hardware are the microcontroller and the XBee. The microcontroller is used to measure the sensors and send the calibrated data to the XBee for transmission. The gathering and transmission of data is done in software, as discussed in Section 2.1.2. The XBee is used to wirelessly communicate with the master node. It operates at 2.4GHz. XBees have three operational modes. The “network coordinator” mode creates a network for other XBees to connect to. Each XBee can relay data between two XBees which are in range to itself, but out of range from each other. Each network of XBees must have one network coordinator. The next mode is “router”. Routers can also relay data between XBees that are out of range. The final mode is “endpoint”. XBees configured as an endpoint cannot relay data and can be put to sleep to conserve power. The master node operates as the network coordinator, and the slave nodes operate as routers. In this way, each node is capable of relaying data to another node.

2.1.2 Slave Node Software

The software on each slave node has a simplified set of tasks, leaving most of the computation to the master node, where more processing power is available. The slave node software is responsible for reading raw data from the sensors, and communicating that data on-demand to the master node. This code has been broken down into the blocks shown in Figure 2.

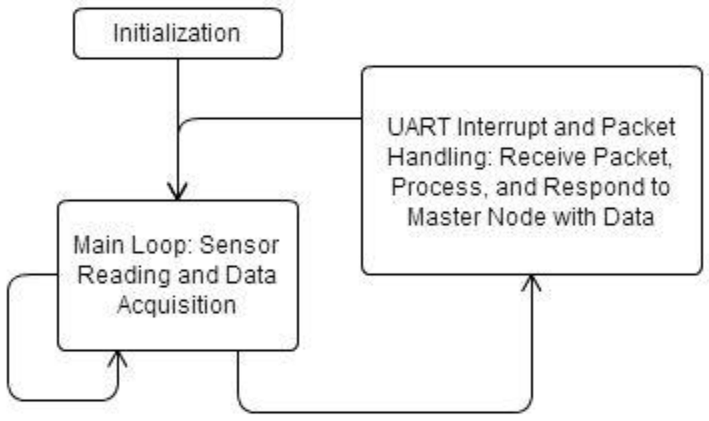

Initialization
UART Interrupt and Packet Handling: Receive Packet, Process, and Respond to Master Node with Data
Main Loop: Sensor Reading and Data Acquisition

Figure 2: Slave Node Software Block Diagram

Each block in Figure 2 is discussed in the following sections, beginning with initialization. Descriptions of how each sensor is read may be found in section 2.1.2.3. A description of the packets used to communicate data may be found in Section 2.1.2.4. Descriptions for the packet handling, the UART interrupt, and data transmission may be found in section 2.1.2.5.

2.1.2.1 Initialization

The slave node software runs on a PIC16F1783 microcontroller. The software starts by initializing the internal clock, the serial communication module that communicates with the XBee, the Analog to Digital Converter (ADC) that reads from the temperature sensor, the timer module that reads from the humidity sensor, and the interrupt that receives data from the master node through the XBee link. The packet receive buffer, which is used to hold a partial packet until all eight bytes of the packet have been received, is also initialized to empty.

2.1.2.2 Main Loop Execution

The main loop is designed primarily to hold the microcontroller in a communication idle while it waits for the master node to request data. The communication idle is a state of the slave node during which it is neither receiving nor transmitting data. During the communication idle, the microcontroller continuously reads from each sensor while it allows the serial interrupt to fill the receive buffer. When the master node requests data, no delay is required for the slave node to read each sensor. The lack of a delay exists because the data values for temperature and humidity are constantly updated. Once the buffer is full with eight bytes, the software breaks out of the main loop into the code to process the packet. Once the packet has been processed and a reply sent, the code reenters the main loop and again continuously reads from the sensors, as described next.

2.1.2.3 Sensor Reading and Data Acquisition

Because the output of the temperature sensor is a temperature dependent voltage, the temperature sensor is read using the ADC module. The ADC has 12-bits of resolution, and the raw data can range from 0 to 4095, although for the temperature sensor designed the range is much less than that. The time taken to read from the ADC module is approximately 5µs according to the PIC16F1783 datasheet shown in Appendix D.

The humidity sensor is read using a timer. The output of the humidity sensor is a humidity-dependent square wave, the frequency of which changes with humidity. The frequency measurement of the 50% duty cycle square wave produced by the humidity sensor is performed using a timer. The count of the timer is set to increment whenever the timer detects a rising edge from the sensor. For the microcontroller to read from the humidity sensor, the value of the timer is set to zero. The timer is then enabled and the controller waits for one second. At the end of the one second, the controller stops the timer from counting. The number of rising edges detected by the timer in one second is equal to the frequency of the input signal. The frequency data is stored as a 2-byte number. Both temperature and humidity are stored and transmitted as 16-bit integers.

2.1.2.4 Packet Description

Two types of packets were utilized in envwi sense, a data request packet and a data return packet. A data request packet is transmitted from the master node to a specific slave node and contains the address of the slave with which the master node is communicating and the type of data it is requesting. A data return packet is transmitted from a slave node to the master node and contains the data requested by the master node. Figure 3 shows the packet structure.

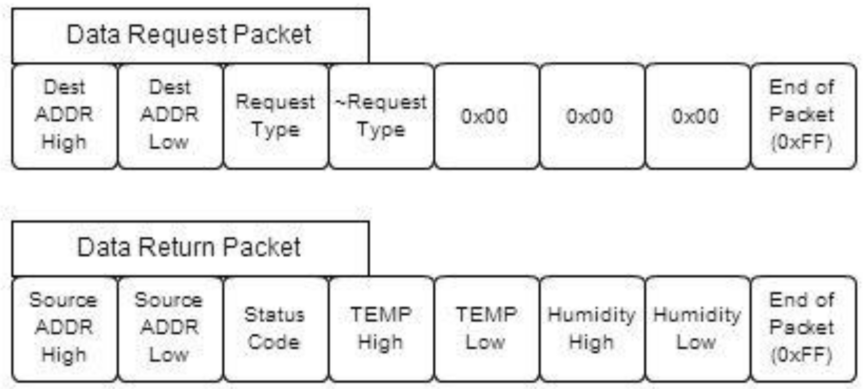


Figure 3: Data Packet Structure.

As shown, the packets include features such as the "end of packet" byte to ensure data integrity. The end of packet byte was used to correct a specific data transmission issue. The issue was that during transmission in a high-noise environment such as the climate-controlled chamber used for calibration and testing, it would sometimes happen that some of the bytes of a packet would not be received. The receive buffer would fill the bytes received from the first packet, and then the rest of the buffer would be filled by some of the bytes from the next packet. Before the End-of-Packet byte was used, there were no methods to detect this sort of transmission error, and when data was extracted from the packet, it would be erroneous. The packets do not use a Checksum or CRC, partly because the XBee uses a CRC on each byte transmitted by it and automatically retransmits any byte that was corrupted in transmission.

2.1.2.5 UART Interrupt and Packet Handling

The receive buffer is filled by the serial interrupt using a global counter variable to track how many bytes have been received. Once the buffer is filled by a packet, the packet is checked to ensure that it is a valid packet. As discussed in Section 2.1.2.4, each packet contains the address of the slave node to which or from which the packet was sent. The address of the packet is checked against the address of the slave node receiving it. If the packet was intended for the slave node, several other bytes in the packet are checked for conformance with expected data. The packet is either discarded if these checks fail and any new data in the receive buffer is flushed, or the packet is accepted and a reply is constructed. To construct the reply packet, the slave node fills the packet with the most recent data from each sensor. The slave node then transmits this packet to the master, and resumes continuously reading from the sensors.

2.2 Master Node

The master node is the brain of envwi sense. The job of the master node is to communicate with each slave node and make use of data collected from each node. The master node performs the majority of the data processing; whereas, the slave nodes have the simpler job of collecting and transmitting data upon the request of the master node.

2.2.1 Master Node Hardware

The master node has much less hardware compared to the slave node. The hardware consists of a Raspberry Pi and an XBee. The Raspberry Pi is a small low-power Linux computer used as the web server and data logger for the project. It collects data from the slave nodes and stores the data in a database. The XBee is the wireless transmitter used for communication between master and slave nodes. The Raspberry Pi communicates with the XBee using the on-board 3.3V serial port.

2.2.2 Master Node Software

The master node runs three programs that work together to form the functionality of envwi sense. The responsibility of each program is shown in Figure 4.

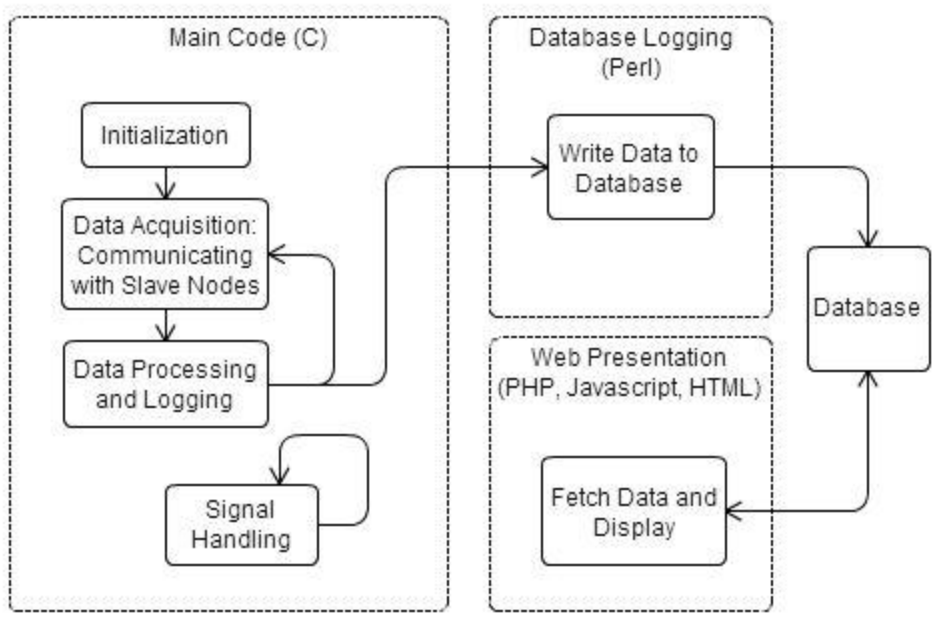


Figure 4: Master Node Software Block Diagram

Each of the programs used on the master node are discussed in the following sections, beginning with the main code written in the C language, followed by the Perl script for adding data to the database, and finally the PHP code used for the website.

2.2.2.1 Main Code

The main code is responsible for communicating with the slave node(s) via the XBee. It handles all data communications and processing, and logs the data collected for viewing and use. The data is stored in both raw and converted formats. The raw format is the value read by the slave microcontroller. The converted format is the values in ºC and %RH. The data is stored in comma separated value (CSV) files. Each time the program runs, two CSV files are created. One file is created for the raw data, and the other file for the converted data. Debug information is logged into a text file each time the program is run. The collected data is also stored in a database in its converted form for use by the website, or potentially other database applications. The main code consists of one primary C code file and several other C header and code library files. Using Makefiles, the libraries are compiled into a statically linked library, which is then linked with the primary C code file by another make file.

2.2.2.1.1 Initialization

The main software operates first by allocating memory for packets, data, and strings followed by configuring the Raspberry Pi's on-board serial port to communicate with the XBee. It then opens the files needed to log data and debug information. Once these operations have been completed, the master node may begin communicating with slave nodes. Communication with the slave nodes is done using 8-byte packets. A description of these packets may be found in Section 2.1.2.4. Before the main program can be used with the Raspberry Pi's on-board serial port, some configurations on the Rasbian operating system must be changed. See Appendix E for information on these configuration changes.

2.2.2.1.2 Data Acquisition and Communication with Slave Nodes

The master node collects data from each slave node. It begins by requesting data from each slave node. To accomplish this, it sends a data request packet to the slave node and then waits for a response. The software will wait for a timeout period of two seconds for a response before moving on to the next slave node. The master node does this until it has collected data or timed out from each slave node in its list. During this process, data is logged in the raw data log as well as the debug log. The debug log provides useful information if a problem with the system ever occurs. Data processing begins once all of the slave nodes have been read from since the slave nodes deliver data in the raw format as read by the microcontroller. See Section 2.1.2.3 for more information about these formats.

2.2.2.1.3 Data Processing and Logging

Data conversion is performed by the master node since the data is delivered from the slave nodes as read by the microcontroller. Data conversion takes place using calibration equations assigned to each slave node. Once the data has been converted from raw format into degrees Celsius and % Relative Humidity, it is written to the CSV file. After the converted data has been stored in the appropriate CSV file, the main program then calls a Perl script and passes the data values to the Perl script as arguments. See Section 2.2.2.2 for information on the Perl Script. Once the data has been written into the database by the script, the main code returns to the beginning of the list of slave nodes and begins collecting data again.

2.2.2.1.4 Signal Handling for Program Termination

The Main Code can be terminated using several POSIX signals. These signals are handled by the program in order for the program to terminate safely. The signals handled are SIGINT, SIGQUIT, and SIGTERM. These are signals from the Linux operating system which indicate that the program should quit. The primary signal used for this is the SIGINT signal, which is generated when “Control+C” is entered into the terminal from which the main program was run.

2.2.2.2 Database Logging

The main code uses a script written in Perl to write data into the database. The database contains what is known as a ‘table’, which is a division of a database generally used to contain related data. envwi sense uses a single table to store the data in the database. Along with the stored data is a time stamp from the time that the data was collected. The Perl script creates the table in the specified database if it does not exist and then inserts the data into that table. The script then exits, returning an error code on failure.

2.2.2.3 Web Presentation

The data gathered by envwi sense is presented in graphical format on a web page. The web page is served by a web server running on the Raspberry Pi, and a website written to display data from the database. The web server used is Lighttpd with PHP5 and SQLITE3 for databasing. The web page consists of one PHP file which displays data from the database on a webpage. The PHP code makes use of Javascript in order to dynamically display elements on the page, including the clock, program and slave node status indicators, and the temperature and humidity data for the past hour. The graphs use the graphing library Highcharts, and the data is inserted into the graphs using PHP to load the data from the database.

3 Details

This section of the report describes the engineering design decisions made for envwi sense and the reasons behind those decisions. Consistent with previous sections, this section is divided into slave node and master node, and further into hardware and software.

3.1 Slave Node

As mentioned previously, the slave node is the portion of the project that is placed remotely to measure temperature and humidity and transmit it to a central location. The following section describes the decisions made during the design process of the slave node.

3.1.1 Slave Node Hardware

Most of the hardware design occurred in the slave node. The following sections detail the design choices associated with hardware on the slave node.

3.1.1.1 Power Supply

The power supply consists of a 9V battery, which was used to power all of the components for the slave node. The slave node needs 5V and 3.3V to power the various components. Linear voltage regulators were chosen to provide the necessary voltages on the sensor board and communication board. Two linear voltage regulators were used to regulate the battery voltage from 9V to the required +5V and +3.3V. The LM7805 was chosen because it provides a constant +5V output and does not require any biasing resistors. The LD1117 was chosen because it outputs a constant +3.3V and also does not require any biasing resistors. The 5V linear voltage regulator provides power to the MCP6004 operational amplifier, the LM555 timer, and the microcontroller. The 3.3V linear voltage regulator provides power to the XBee and the logic level shifter. As previously mentioned in Section 2.1.1.1, the PCB was designed to use three-pin DC-DC voltage regulators instead of the linear voltage regulators. Using the DC-DC regulators allowed envwi sense to capture an additional three hours of temperature and humidity data.

3.1.1.2 Sensors

To measure temperature, an RTD was chosen because of its sensitivity over a small range of temperatures. Typically, thermocouples are used when the temperature will vary by 1,000 degrees Celsius. However, the proposed temperature range was only from 0ºC to 38ºC. Over the specified range, an RTD is more precise and accurate. The chosen RTD also has a positive temperature coefficient.

To measure relative humidity, a capacitive humidity sensor was chosen because of its response over a wide range of humidity. The proposed humidity range was from 15% RH to 85% RH. This range cannot be covered by a resistive humidity sensor, therefore a capacitive humidity sensor was chosen. The sensor works by exposing some of the dielectric material of the capacitor to the surrounding air. The dielectric material absorbs and releases moisture to maintain equilibrium with the surrounding air. As the water content in the dielectric material changes, the dielectric constant of the material changes and linearly affects the capacitance of the sensor.

3.1.1.3 Sensor Drivers and Measurements

The RTD is biased using a 100µA current source and the voltage across the RTD is measured. Since the resistance changes relative to the temperature and the current stays the same, then the voltage across the RTD also changes relative to temperature. A current source of 100µA was used for power consumption and measurability reasons. A smaller current would use less power but would also make the change in voltage smaller as temperature changes, making the measurements less precise. A larger current would result in a larger voltage swing as temperature varies, but the circuit would consume more power. The extra power consumption would also generate heat which could cause measurement errors with the circuit. The battery would drain faster and the slave node would not transmit data for as much time.

The chosen RTD also played a factor in the design process. The RTD has a resistance of 100Ω at 0ºC and with a known 100µA, a power consumption by the RTD of 1µW occurs at 0ºC. The RTD has a temperature rating from -50 to 500ºC and has a sensitivity of

3,850 ppm/^{o}C. The current source was designed so that at 19^{o}C, the center of the specified temperature range, the output of the instrumentation amplifier would be 2.5V, half way between the voltage references of the ADC. The schematic can be seen in Figure 10 in Appendix C. Placing the output of the amplifier in the center of the voltage range allows for maximum voltage swing in both directions which ultimately results in maximum temperature swing.

An instrumentation amplifier was designed to amplify the small voltage across the RTD. The gain of the amplifier was determined using (1).

$$A_d = \frac{R_9}{R_7}(1 + \frac{R_4}{R_5}) \tag{1}$$

Where R_4 = 20kΩ, R_5 = 220kΩ, R_7 = 10kΩ, and R_9 = 100kΩ. The instrumentation amplifier can be seen in Figure 10 of Appendix C. The instrumentation amplifier has a calculated gain of 233V/V.

The capacitive humidity sensor was used as part of a 555 timer circuit. The 555 timer output frequency is set by a charging capacitor. By placing the humidity sensor as this capacitor, the output frequency of the 555 timer will change with respect to relative humidity.

$$f = \frac{1.44}{(R_{11} + R_{12}*2)\, C_H} \tag{2}$$

The resistors R_{11} = 10kΩ and R_{12} = 120kΩ were calculated so that the output of the 555 timer would be 20kHz at a base capacitance of C_H = 330pF. The microcontroller measures the output frequency, and using a conversion equation, the humidity is determined from the measured frequency.

3.1.1.4 Microcontroller and XBee.

The XBee was chosen from among other wireless transceivers due to its built-in software redundancy, security, and other similar software features that make it reliable and useful as compared to its competitors. The specific model of XBee was chosen due to its range and cost. A low range XBee was selected as this project is not intended to bridge long distances, which would reduce battery life. The cost of the XBee was also a factor since each node in the project must have one, and they are the second most expensive part after the Raspberry Pi, consisting of more than triple the cost of an entire slave node's parts alone.

3.1.2 Slave Node Software

The slave node software is used only to read from the sensors and communicate with the master node. Limiting the amount of processing done by the slower microprocessor on the slave node by delegating it to the faster master node increased the performance of the system. Figure 2 in Section 2.1.2 shows the block diagram of the slave node software.

3.1.2.1 Data Transmission as Packets

The slave node uses 8-byte packets to transmit and receive data. Section 2.1.2.4 discusses the configuration of the two types of packets used. The packets were designed with some redundancy, including some per-byte redundancy as well as the end of packet field, designed to indicate a "framing error". A framing error would occur if partial packets were received by either the master or the slave. This byte successfully allows the receiving entity to determine if a problem occurred during transmission, and discard the packet if it is not valid. During the testing of the project, it was noted that if framing errors were encountered, without a way to detect and recover from these, the project would have no way to determine if the data received was valid, and would log data from a garbage packet. The addition of redundancy into the packets allows the project to recover from data transmission errors.

3.1.2.2 Interrupts

Interrupts can be configured on a processor to allow the processor to process data or respond to a change immediately by interrupting the current operation and handling whatever caused the interrupt. Interrupts were selected for receiving serial data so that the microcontroller would not have to wait to receive data from the master node, or risk losing a byte of incoming data if multiple bytes of data were received during some microcontroller operation. Interrupts allow the microcontroller to handle the asynchronous data communication from the master node. The use of interrupts allows the microcontroller to read from the sensors while it is not communicating with the master node so that when a request for data arrives from the master node, the slave node has recent data to send it. This use of interrupts removesany latency needed to read from each sensor before transmission.

3.1.2.3 Data Representation

Temperature and humidity data on the slave node are represented as 2-byte integers. This representation standardizes data representation on the slave node among all sensor types, and allows for more abstraction for data handling and transmission. The data on the slave node consists exclusively of the raw values read from the sensors. The data is not converted until after it is transmitted to the master node. Not converting the data from raw format frees processing time on the slave node by delegating the conversion to the more powerful master node, and also allows for both temperature and humidity data to be transmitted in one packet.

Packets are constructed using a C struct. The C struct allows for type and bounds checking rather than using a C array. The struct contains a single C array that is eight bytes in length. The struct can be type-checked by the compiler and each instance of the packet struct is guaranteed to contain an eight char array, thus removing the need for bounds checking as the array is always guaranteed to be eight bytes long.

3.1.2.4 Microcontroller

The microcontroller unit (MCU) chosen for the project is the Microchip PIC16F1783. This MCU was chosen for several reasons. First, the microcontroller is available in many package types, including PDIP. The PDIP package type facilitates prototyping on a breadboard without the need for breakout boards or waiting to test anything until a PCB could be ordered. The variety of package types meant that transitioning to a surface mount package once a PCB was available would also be an option. The 16F1783 has all of the peripherals needed for the project and many more which facilitate expansion of the project, and also has a 32MHz internal clock. The 32MHz clock is more than fast enough to perform all of the necessary operations without compromising performance. Among the MCUs meeting the above qualifications, the 16F1783 also stood out due to its low power consumption and low cost operation.

3.2 Master Node

The master node is the controller of the sensor network and provides data acquisition and logging to the project. The following describes in detail the design decisions involved in the making of the master node.

3.2.1 Master Node Hardware

The master node consists of a Raspberry Pi and XBee. The Raspberry Pi was chosen due to its small size, low power consumption, portability, low cost, and the built-in serial port. As a Linux computer, the Raspberry Pi is able to easily store large amounts of data, run a web server, and process more data than a standard embedded microcontroller, as well as easily display data to a screen. Thus, the Raspberry Pi was chosen over a desktop or laptop computer or a microcontroller. The Raspberry Pi is not the only small, inexpensive Linux computer. However, among its competitors, it is generally less expensive, is more widely in use, and has much documentation and support. Refer to section 3.1.1.4 for information on the decision regarding the XBee.

3.2.2 Master Node Software

As discussed in Section 2, the master node software is written primarily in three languages: C, PHP, and Perl. The program written in C performs the majority of the work of the project as its function is to facilitate communication with the slave nodes and log data. The C code uses a Perl script to write the data into a SQLite3 database. A Perl script was chosen because it provides an interface between the C code and the database, an intuitive SQLite3 library. The PHP webpage, composed of PHP, Javascript, and HTML, reads from the database to display the data.

Temperature and humidity are the most important data used by the master node software. The C code represents temperature and humidity as 2-byte integers when they are received from the slave nodes. Once the data is converted to degrees C or % RH, they are stored as floating point values. The floating point values natively allow for storage and display the converted data as decimal numbers. Packets are represented as C structs containing an eight byte array as discussed in Section 3.1.2.3. Temperature and humidity data are represented in the SQLite3 database as “FLOAT”. The timestamp stored in the database is of the type “DATETIME”, and contains the local date and time when the data was measured.

The master node program that communicates with the slave nodes uses 8-byte packets to transmit and receive data. These packets are described in Section 3.1.2. The details of the packet design are discussed in Section 3.1.2.1.

4 Results

After extensive calibration and testing, the results of the envwi sense project are described below. The specifications were met and well exceeded by the project as shown in Table 1.

Table 1: Projects Specifications and Results

Specification	**Project/Ability**
0ºC - 38 ºC ±3ºC	Met
15%RH - 85%RH ±8%RH	Met
Tested in Sub-Zero Chamber	Yes
Sensor Conditioning Circuitry	Designed and Used
10 Second Communication	5 Seconds

Section 4.1 describes how envwi sense was calibrated and tested, Section 4.2 shows the results from the temperature validation data, Section 4.3 shows the results from the humidity validation data, and Section 4.4 shows how the data is displayed.

4.1 Calibration and Testing

The sensors were tested and calibrated to a Cincinnati Sub-Zero Temperature Chamber. The Sub-Zero Chamber is a large metal enclosure, with metal walls on the inside and outside and insulation in between. The chamber is automated to vary temperature and humidity. The sensors were tested over the entire specified range. During the testing, some difficulties were encountered in establishing a reliable communications link between the master node and the slave nodes. The master node was placed outside of the Sub-Zero Chamber with the XBee's antenna positioned through a hole in the wall, and the slave nodes were placed inside. The metal walls of the chamber presented some difficulty for the XBees to communicate, as they were all contained within a somewhat small metal box. Another problem is that the Sub-Zero Chamber uses multiple large compressor motors. It seemed that during the times that these motors were in operation, the XBees would experience more difficulty communicating.

During the initial testing, many problems were encountered and much data was lost. This data loss was due to the fact that the system had no way of telling if a packet had been received correctly. The largest problem was a framing error, where only some of the bytes in the packet would be received. A framing error means that the packet that was read by the master or slave node would contain parts of multiple packets, or an incomplete single packet. Initially the system had no method for recovering from this error, and would log the faulty data until the program was restarted. Building error checking into the packets allowed the system to recover from a framing error. Once the error checking was added, some data was lost, but the system was able to recover from this error condition.

To verify that the project met its temperature and humidity specifications, the Sub-Zero Chamber was set to run two tests, one for temperature and one for relative humidity. The chamber was set to test the limits of our specified range for both temperature and relative humidity.

4.2 Temperature Results

The first test decreased the temperature in the chamber to 0ºC, held it there, and then increased the temperature to 38ºC and maintained that temperature. The results are shown in Figure 5.

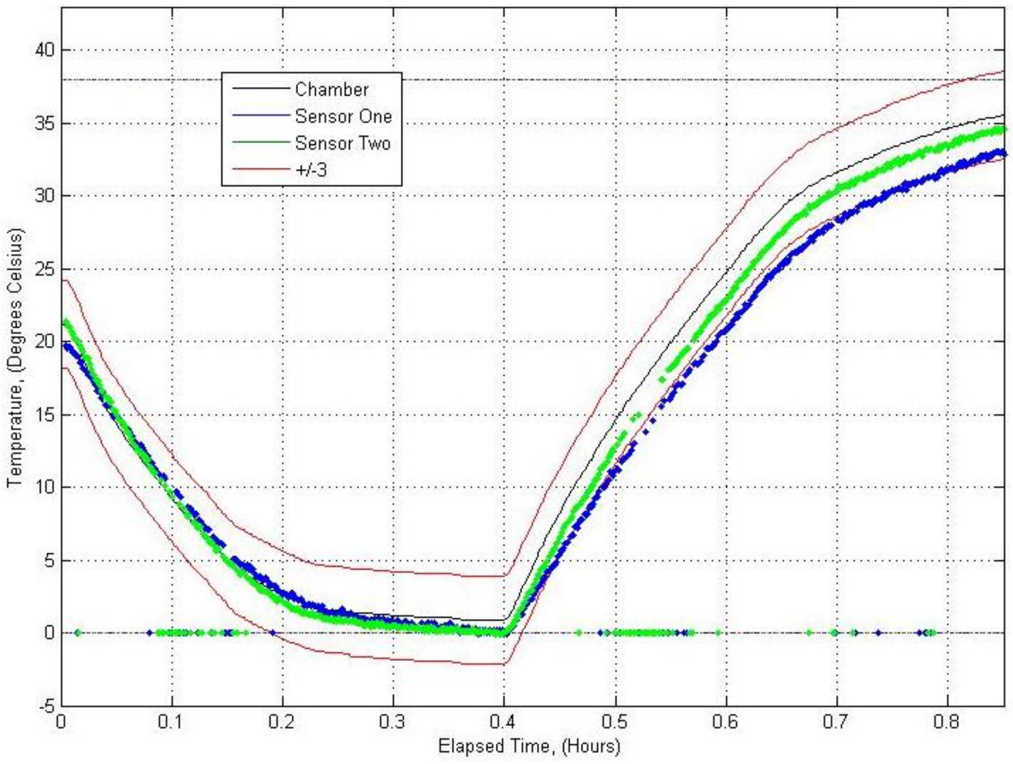


Figure 5: Temperature Validation

As shown, when in steady-state, the sensors stayed within the ±3^{o}C specification. Lost data was chosen to be represented as zeroes to visually explain large gaps in the graph. The data shown in Figure 5 was converted from the raw values using the following calibration equations.

$$Temperature\ ℃ = 0.121 \times (ADC\ value) - 214 \tag{3}$$

$$Temperature\ ℃ = 0.128 \times (ADC\ value) - 266.5 \tag{4}$$

Equation (3) was the conversion equation for slave node 1, and (4) for slave node 2.

4.2 Relative Humidity Results

The second test held the humidity at 15% RH, and then brought it up to 85% RH. Throughout the entire test, the temperature was held at 20^{o}C. The results from the humidity sensor validation are shown in Figure 6.

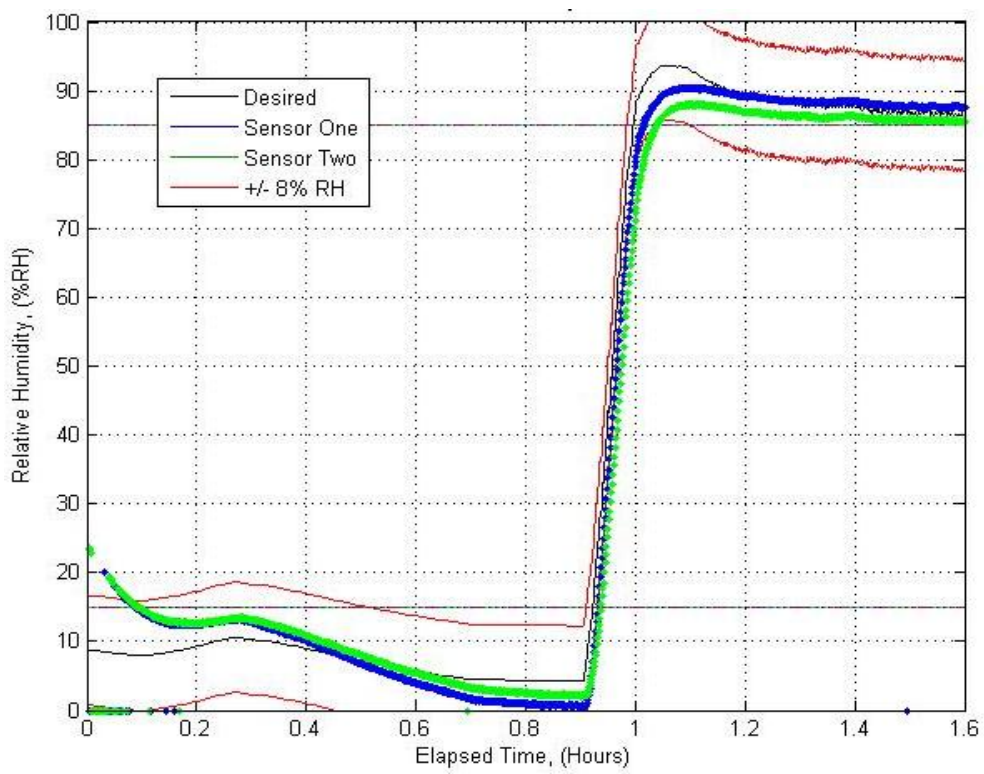


Figure 6: Humidity Validation

The relative humidity sensors also met the proposed specification. To get these results, the following calibration equations were used:

$$Humidity\ \%RH = 770{,}000 \times (555\ Period) - 384 \qquad (5)$$

$$Humidity\ \%RH = 770{,}000 \times (555\ Period) - 381 \qquad (6)$$

Equation (5) is the conversion equation for slave node 1, and (6) for slave node 2.

4.4 Data Display

The collected data may be viewed on the project website. Figure 7 shows the web page as it displays collected data.

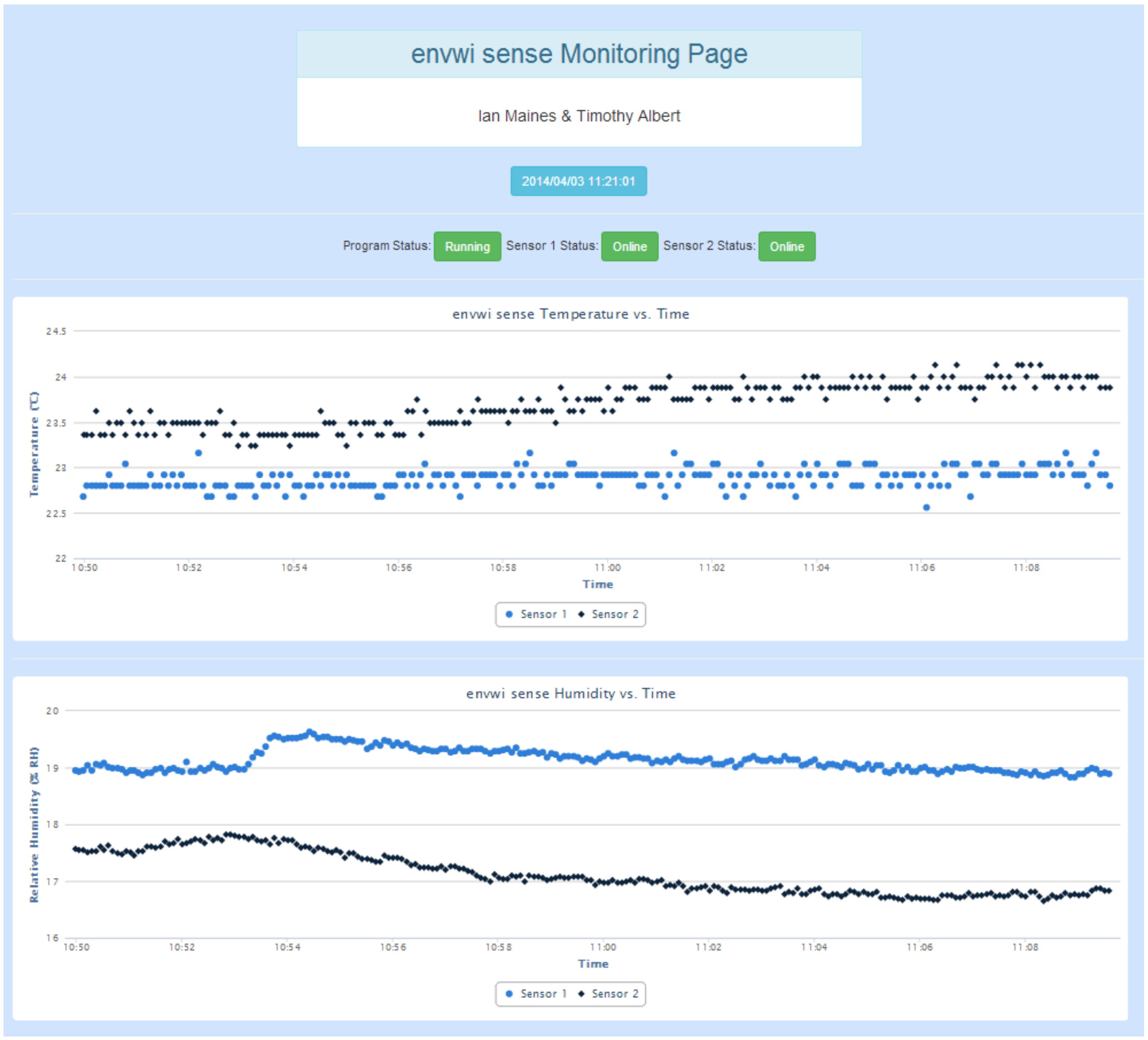


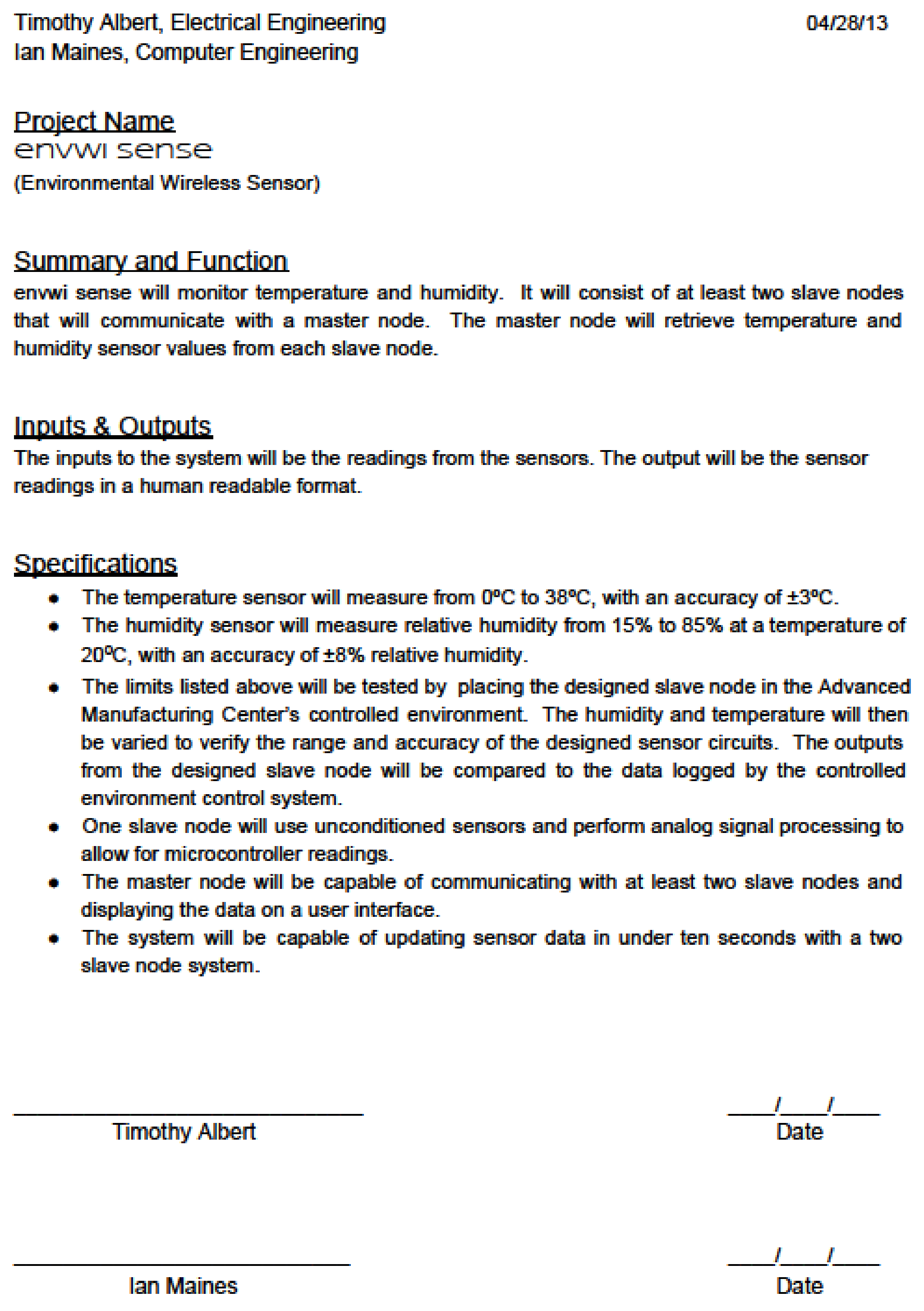
Figure 7: Webpage Display.

As shown in Figure 7, the web page displays data collected and recorded from the sensors. The website only displays data from the past sixty minutes. The website also has displays indicating whether the project is running and the sensors are connected. If the master node is intermittently unable to connect to a slave node due to the master node not receiving a response from the slave node, the slave node status indicators will change color and displays a warning that indicates that packets are being lost This is another feature of the website that is shown in Figure 7. The Raspberry Pi has a limited amount of RAM, and as databases are somewhat RAM intensive, accessing a large database is slow. The web page displays a warning banner if the database has more than 8,000 entries so that the database may be trimmed if desired. The database will still operate until the SD card on the Raspberry Pi is full. However, once the database contains more than 12,000 entries, the PHP script which gets the data from the database will time out, and the web page will not display any data.

5 Conclusions

This report described the envwi sense capstone project. envwi sense consists of a master node that wirelessly receives temperature and humidity data from a slave node. The slave node uses discrete components to measure the temperature and relative humidity and makes the sensor output measurable by a microcontroller. A RTD is the type of sensor used to measure temperature, along with a capacitive humidity sensor used to measure humidity. XBees are used to transmit and receive the data wirelessly and the Raspberry Pi is used to collect the data and display it to a web page. The envwi sense project was tested to measure temperature from 0^{o}C to 38^{o}C with an accuracy of $\pm 3^{o}$C, and was also able to measure relative humidity from 15% RH to 85% RH with an accuracy of ±8% RH. Extensive testing was performed to verify that the sensor network met the specifications. The testing was completed in a Sub-Zero climate controlled chamber. This chamber presented some difficulty for the XBees to communicate, but through the integration of error checking into the packets, the system has been able to recover from all communication difficulties presented by the chamber, with only minimal data loss. This loss is not expected to impact the usefulness of the project, as the project is not intended to work within confined metal walls or high noise areas. Overall, the project works well and accomplishes its goals while displaying the data using a clean interface, and meeting all specifications.

Appendix A: envwi sense Project Proposal



Timothy Albert, Electrical Engineering — 04/28/13

Ian Maines, Computer Engineering

Project Name

envwi sense

(Environmental Wireless Sensor)

Summary and Function

envwi sense will monitor temperature and humidity. It will consist of at least two slave nodes that will communicate with a master node. The master node will retrieve temperature and humidity sensor values from each slave node.

Inputs & Outputs

The inputs to the system will be the readings from the sensors. The output will be the sensor readings in a human readable format.

Specifications

- The temperature sensor will measure from 0ºC to 38ºC, with an accuracy of ±3ºC.
- The humidity sensor will measure relative humidity from 15% to 85% at a temperature of 20ºC, with an accuracy of ±8% relative humidity.
- The limits listed above will be tested by placing the designed slave node in the Advanced Manufacturing Center's controlled environment. The humidity and temperature will then be varied to verify the range and accuracy of the designed sensor circuits. The outputs from the designed slave node will be compared to the data logged by the controlled environment control system.
- One slave node will use unconditioned sensors and perform analog signal processing to allow for microcontroller readings.
- The master node will be capable of communicating with at least two slave nodes and displaying the data on a user interface.
- The system will be capable of updating sensor data in under ten seconds with a two slave node system.

______________________________ ____/____/____

Timothy Albert — Date

______________________________ ____/____/____

Ian Maines — Date

Appendix B: PCB Layouts

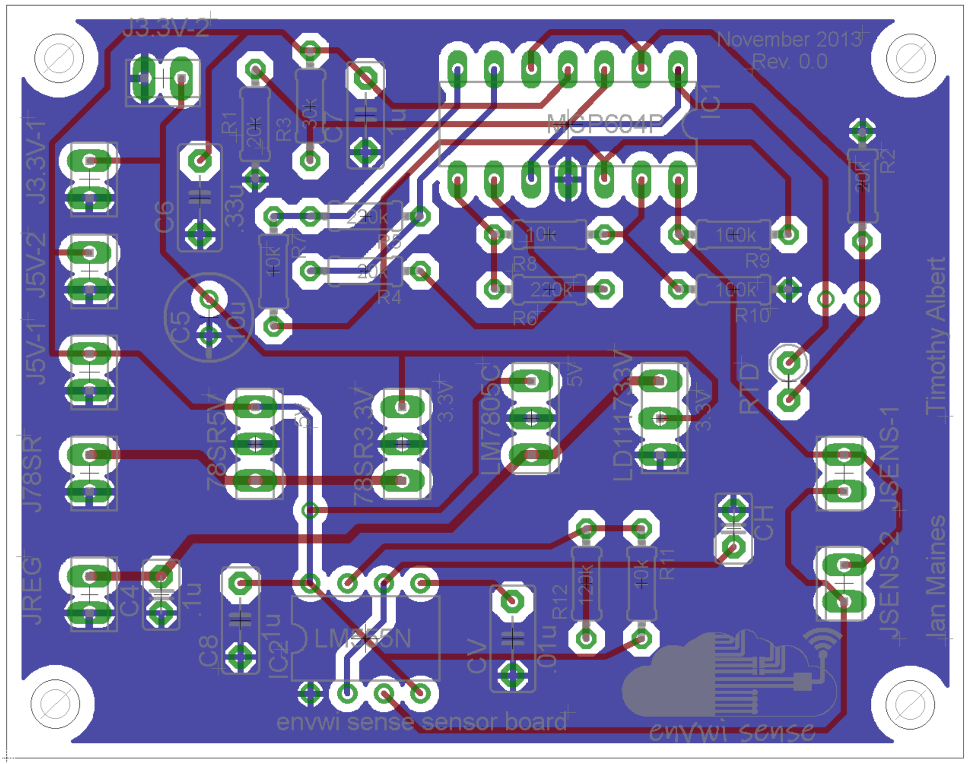


Figure 8: Sensor Board PCB Layout

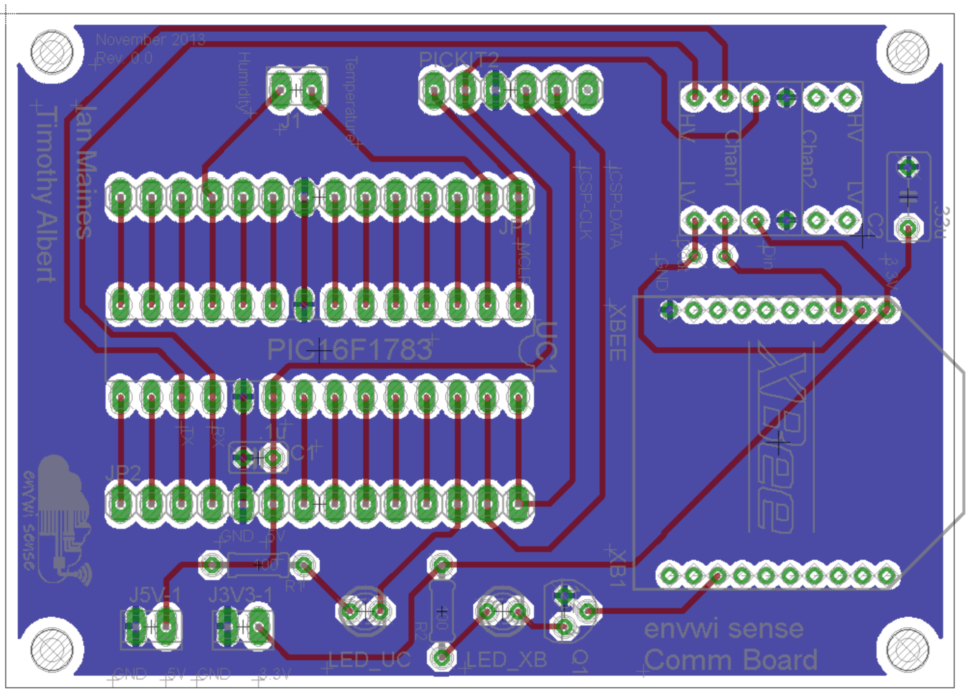


Figure 9: Communications Board PCB Layout

Appendix C: Project Schematics and Parts List

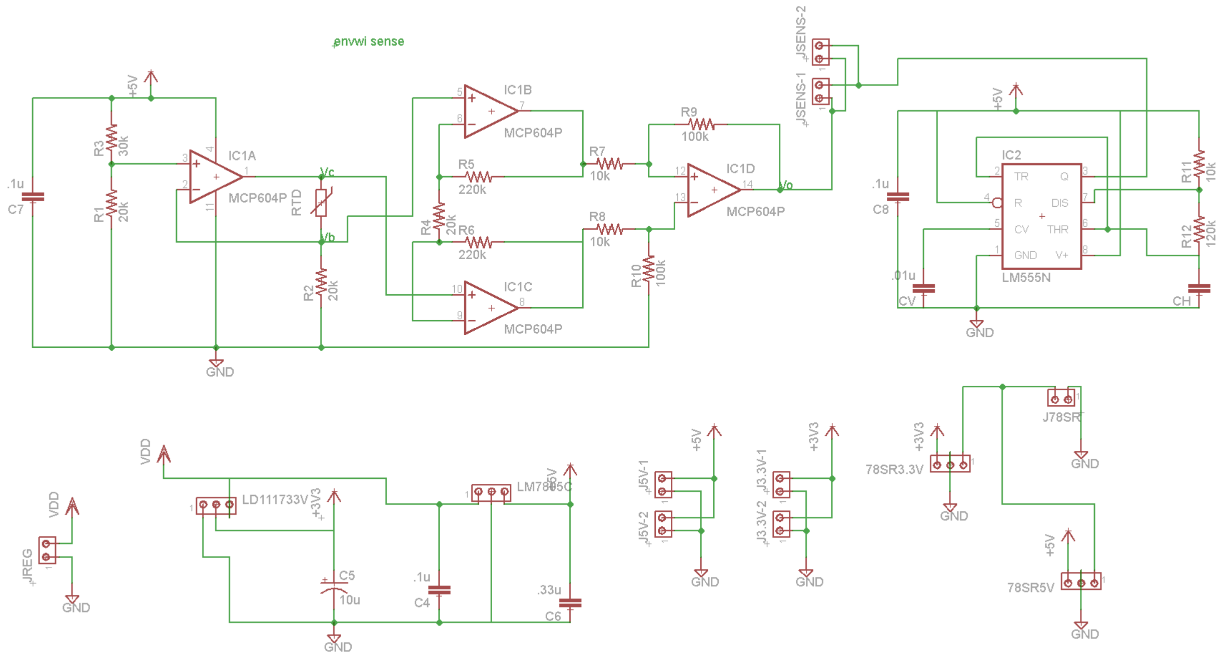


Figure 10: Schematic of Sensor Board Hardware

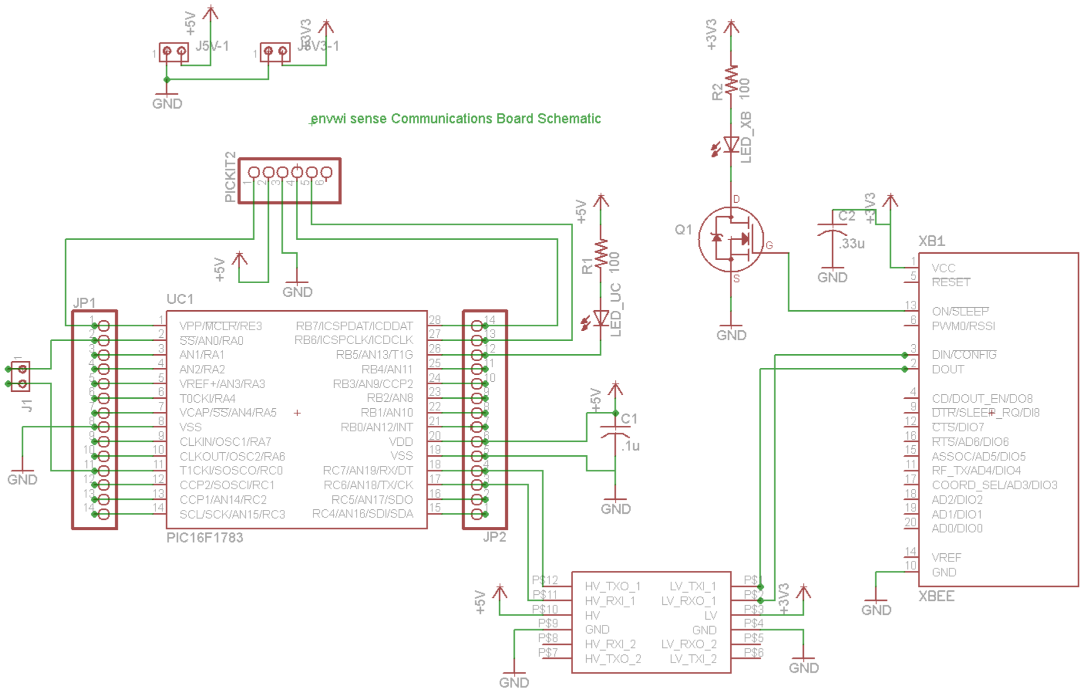
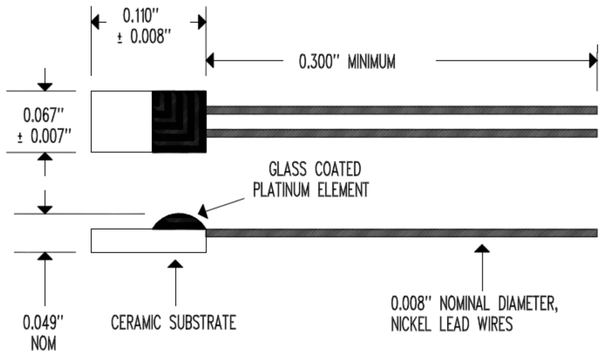


Figure 11: Schematic of Communications Board Hardware

Table 2: Project Parts List

Reference	**PART NUMBER**	**DESCRIPTION**	**QUANTITY**
UC1	PIC16F1783	Microcontroller	1
XB1	XB24-Z7SIT-004	Wireless transmitter	2
IC1	MCP6004	Quad Op-Amp	1
JSENS-(1,2), J5V-(1,1,2), J3.3V-(1,2), J78SR, J1, J3V3-1, JREG	MTA Connector	MTA connector	11
CH	HCH1000	Capacitive Humidity Sensor	1
C4, C7, C8, CV	0.1u	0.1uF Capacitor	4
C2, C6	0.33u	0.33uF Capacitor	2
R1, R2(sensor), R4	20k	20k Resistor .25W	3
R3	30k	30k Resistor .25W	1
R2(comm.)	100	100 ohm Resistor .25W	1
R5, R6	220k	220k Resistor .25W	2
R7, R8	10k	10k Resistor .25W	2
R9, R10, R11	100k	100k Resistor .25W	3
R12	120k	120k Resistor .25W	1
RTD	PPG101B1	100 ohm RTD	1
IC2	LM555N	555 Timer	1
LD111733V	LD111733V	3.3V Linear Regulator	1
LM7805C	LM7805C	5V Linear Regulator	1
78SR3.3V	78SR3.3V	3.3V DC-DC Converter	1
78SR5V	78SR5V	5V DC-DC Converter	1
Q1	2N7000	MOSFET	1
-	LLC	Logic Level Converter	1
LED_XB	LED_XB	Communications LED	1
LED_UC	LED_UC	Microcontroller LED	1
PCB_SENSOR	PCB_SENSOR	Sensor and Power PCB	1
PCB_COM	PCB_COMM	Communications PCB	1

Appendix D: Datasheets


0.110" ± 0.008"
0.300" MINIMUM
0.067" ± 0.007"
GLASS COATED PLATINUM ELEMENT
0.049" NOM
CERAMIC SUBSTRATE
0.008" NOMINAL DIAMETER, NICKEL LEAD WIRES

RTD ELEMENT RESISTANCE @ 0°C = 100 Ω ± 0.12%
RTD ELEMENT ACCURACY @ 0°C = ± 0.30°C
RTD ELEMENT DIN 43760 ACCURACY CLASS = "B"
TCR = 3,850 ppm/°C
TEMPERATURE RATING = −50 TO +500°C
THERMAL TIME CONSTANT = 10 SECONDS MAXIMUM (1m/SEC MOVING AIR)
DISSIPATION CONSTANT = 2.5 mW/°C NOMINAL (1m/SEC MOVING AIR)
MAXIMUM APPLIED CURRENT = 1 mA

SEE MANUFACTURING SPECIFICATION (LAYER 1)

"D"	DIMENSION "L" FOR 100 Ω RTDS WAS 0.095" ± 0.008"	10/11/08	DD
"C"	LEAD WIRE WAS 32 AWG PLATINUM PLATED NICKEL	08/25/05	DD
---	ISO RELEASE	04/11/03	DD
"B"	UPDATED LEAD WIRE AWG & LENGTH, ADDED NOTE	08/14/02	RD
"A"	UPDATED LEAD WIRE AWG & TYPE	07/15/02	RD
REV	REVISION RECORD	DATE	APP

SCALE NONE	U.S. SENSOR CORP. (C) COPYRIGHT 714-639-1000 www.ussensor.com
DRAWN BY DAN DANKERT	
DATE 07/15/02	PLATINUM RTD SENSOR
REV. "D"	P/N PPG101B1
LAYER 0 OF 2	

HCH-1000 Series

Table 1. Specifications (T_A= 25 °C [77 °F], Input Voltage = 1 V_{RMS}, Frequency = 20 kHz)

Characteristic	Condition	Min.	Typ.	Max.	Unit
Normal capacitance	at 55 %RH	310	330	350	pF
Sensitivity	10 %RH to 95 %RH	0.55	0.60	0.65	pF/%RH
Humidity hysteresis	–	–	±2	–	%RH
Linearity	–	–	±2	–	%RH
Response time	30 %RH to 90 %RH	–	15	–	sec
Temperature coefficient	5 °C to 70 °C [41 °F to 158 °F]	0.15	0.16	0.17	pF/°C
Long-term stability (drift)	–	–	0.2	–	%RH/year
Operating temperature range	–	-40 [-40]	–	120 [248]	°C [°F]
Operating humidity range	–	0%	–	100%	RH
Operating frequency range	–	1	–	100	kHz

Figure 1: Frequency Characteristics

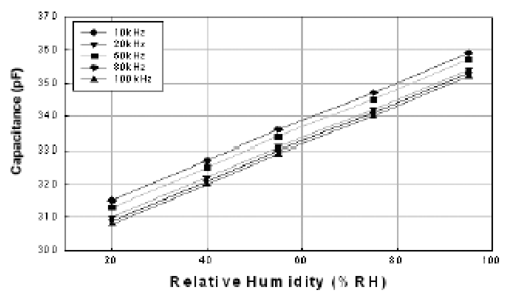


Figure 2: Typical Humidity Response (Sensitivity = 0.6 pF/%RH.)

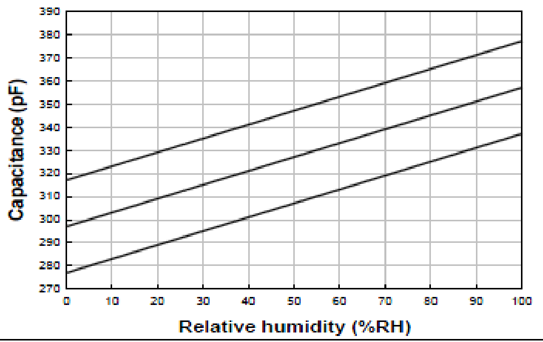
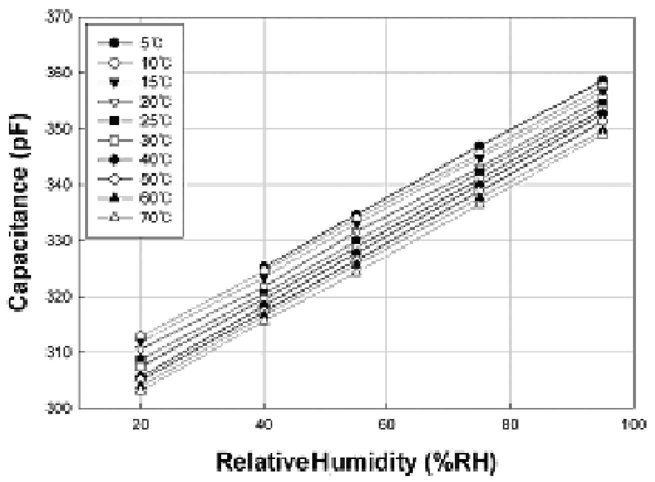


Figure 3: Temperature Characteristics (At 1 V_{RMS} and 20 kHz.)



17.4 ADC Acquisition Requirements

For the ADC to meet its specified accuracy, the charge holding capacitor (CHOLD) must be allowed to fully charge to the input channel voltage level. The Analog Input model is shown in Figure 17-4. The source impedance (RS) and the internal sampling switch (RSS) impedance directly affect the time required to charge the capacitor CHOLD. The sampling switch (RSS) impedance varies over the device voltage (VDD), refer to Figure 17-4. **The maximum recommended impedance for analog sources is 10 kΩ.** As the source impedance is decreased, the acquisition time may be decreased. After the analog input channel is selected (or changed), an ADC acquisition must be done before the conversion can be started. To calculate the minimum acquisition time, Equation 17-1 may be used. This equation assumes that 1/2 LSb error is used (4,096 steps for the ADC). The 1/2 LSb error is the maximum error allowed for the ADC to meet its specified resolution.

EQUATION 17-1: ACQUISITION TIME EXAMPLE

Assumptions: $Temperature = 50°C$ *and external impedance of* $10k\Omega$ $5.0V$ V_{DD}

$$\begin{aligned} T_{ACQ} &= \text{Amplifier Settling Time} + \text{Hold Capacitor Charging Time} + \text{Temperature Coefficient} \\ &= T_{AMP} + T_C + T_{COFF} \\ &= 2\mu s + T_C + [(Temperature - 25°C)(0.05\mu s/°C)] \end{aligned}$$

The value for T_C *can be approximated with the following equations:*

$$V_{APPLIED}\left(1 - \frac{1}{(2^{n+1}) - 1}\right) = V_{CHOLD} \qquad ;[1]\ V_{CHOLD} \text{ charged to within 1/2 lsb}$$

$$V_{APPLIED}\left(1 - e^{\frac{-Tc}{RC}}\right) = V_{CHOLD} \qquad ;[2]\ V_{CHOLD} \text{ charge response to } V_{APPLIED}$$

$$V_{APPLIED}\left(1 - e^{\frac{-Tc}{RC}}\right) = V_{APPLIED}\left(1 - \frac{1}{(2^{n+1}) - 1}\right) \qquad ;\text{combining [1] and [2]}$$

Note: Where n = number of bits of the ADC.

Solving for T_C:

$$\begin{aligned} T_C &= -C_{HOLD}(R_{IC} + R_{SS} + R_S)\ \ln(1/8191) \\ &= -10pF(1k\Omega + 7k\Omega + 10k\Omega)\ \ln(0.000122) \\ &= 1.62\mu s \end{aligned}$$

Therefore:

$$\begin{aligned} T_{ACQ} &= 2\mu s + 1.62\mu s + [(50°C - 25°C)(0.05\mu s/°C)] \\ &= 4.87\mu s \end{aligned}$$

Note 1: The reference voltage (VREF) has no effect on the equation, since it cancels itself out.

2: The charge holding capacitor (CHOLD) is not discharged after each conversion.

3: Maximum source impedance feeding the input pin should be considered so that the pin leakage does not cause a voltage divider, thereby limiting the absolute accuracy.

Appendix E: RPISERIALCONFIG

This appendix describes the steps necessary to configure the Raspberry Pi for use with envwi sense. Although any serial port may be used, the on-board 3.3V serial port called “ttyAMA0” was chosen. By default, this serial port is used as a console by the Rasbian operating system. This appendix describes how to change that configuration so that the on-board serial port may be used by envwi sense.

Two files must be modified in order to use the serial port. These are `/boot/cmdline.txt` and `/etc/inittab`. Before modifying these files, it would be wise to create a backup of them in case there is a problem with other system components caused by their modification. The line in `/boot/cmdline.txt` that is in the current version of Rasbian reads:

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4
elevator=deadline rootwait
```

Note that the above code is a single line in the file. It must be modified to remove mentions of ttyAMA0. The final result should read as follows

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2
rootfstype=ext4 elevator=deadline rootwait
```

Note that the above code is a single line in the file. Next, any line mentioning ttyAMA0 in `/etc/inittab` must be commented out. This can be done using the ‘#’ symbol.

In the current version of Rasbian, the end result will be as follows

```
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

The Raspberry Pi will then be configured for envwi sense to use the serial port.