School of Engineering Science
Simon Fraser University
Burnaby, BC V5A 1S6
eco-mt@sfu.ca

November 5, 2009

Dr. John Bird
School of Engineering Science
Simon Fraser University
Burnaby, British Columbia
V5A 1S6

Re: ENSC 440 Design Specifications for an Ecological Monitoring System

Dear Dr. Bird:

Attached is a document outlining the design specifications of our proposed ecological monitoring system, the ECOmonitor. Our project entails the monitoring of environmental conditions (CO_2 levels, temperature etc.) in remote areas using a wireless data 'hopping' technology.

The design specifications for the ECOmonitor have been outlined for the proof-of-concept model as well as algorithms that can be implemented in the final production. This document also discusses the system test plan that will be implemented to ensure proper functionality.

The ECOmonitoring Technologies Inc. team consists of five innovative and passionate engineers: Ryan Cimoszko, Amandeep Grewal, Brian Lee, Kianoush Nesvaderani and myself, Harvir Mann. If you have any questions or concerns about our functional specifications, please feel free to contact me by e-mail at eco-mt@sfu.ca.

Sincerely,

Harvir Mann
President and CEO
ECOmonitoring Technologies Inc.

Enclosure: *Design Specifications for an Ecological Monitoring System*

DESIGN SPECIFICATIONS FOR AN

ECOLOGICAL MONITORING SYSTEM

Project Team: Ryan Cimoszko
Amandeep Grewal
Brian Lee
Harvir Mann
Kianoush Nesvaderani

Submitted to: Dr. John Bird – ENSC 440
Steve Whitmore – ENSC 305
School of Engineering Science

Contact Person: Harvir Mann
hmann@sfu.ca

Issue Date: November 5, 2009
Revision: 1.1

EXECUTIVE SUMMARY

This document outlines the development and design pertaining to the proof-of-concept model of the ECOmonitor. We will discuss design considerations for the functional specifications marked as XX-I in our *Functional Specifications for an Ecological Monitoring System* document [1]. We will also provide justifications for the choice of components and algorithms that will be implemented in the final production.

The proof-of-concept model consists of a two monitoring stations, a base station, a database and webpage interface. The monitoring stations are made up of three sensors, a microcontroller and a wireless transceiver. For the proof-of-concept design, we chose to apply sensors that are suitable for fire detection: temperature, humidity and CO_2. The sensors and wireless transceiver are connected to the microcontroller. The microcontroller is used to communicate between the sensors and ADCs to obtain data, process data packets and send the data through the wireless transceiver. The base station will consist of a wireless transceiver and a microcontroller. When the data has been relayed back to the base station, the microcontroller will communicate with the host PC through USB to store the data. Using Python the data will be store into the MySQL database. The data will then be retrieved from the database and display graphically on a webpage. The user will have the option to select which data, which monitoring station and what time interval to view.

The primary technology regarding the ECOmonitor is the wireless data 'hopping' mechanism. We discuss difficulties arising from malfunctioning monitoring stations and provide a general algorithm for our design. Flow charts are presented to illustrate the process of each component. Finally a system test plan is presented to ensure proper functionality of the ECOmonitor. The proof-of-concept model is expected to conform to these design specification by the scheduled completion date of December 1, 2009.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ADC	Analog-to-Digital Converter
ANSI	American National Standards Institute
CSA	Canadian Standards Association
CSID	Current Station Identification
CSV	Comma Separated Values
DC	Direct Current
FCC	Federal Communication Commission
FIFO	First In First out
FRC	Frame Redundancy Check
FRC10	Flat Ribbon Cable Connector (10 pin)
GND	Ground
GPIO	General Purpose Input Output
HC	Hopping Control
HTML	Hyper Text Markup Language
LED	Light-Emitting Diode
OSID	Own Station Identification
PC	Personal Computer
PHP	PHP: Hypertext Preprocessor
RF	Radio Frequency
SRAM	Static Random Access Memory
TWI	Two wire interface
UART	Universal Asynchronous Receiver Transmitter
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VCC	Common-collector voltage
ID	Identification

1.0 INTRODUCTION

The ECOmonitor entails the construction of a sensor network that gathers data on certain environmental characteristics and communicates with a base station via a signal 'hopping' scheme. The data will be collected at each monitoring station and will be wirelessly relayed to the next monitoring station that is closest to the base station. Finally the data is sent back to the base station at which point the data will be analyzed and placed online. The design specifications for the ECOmonitor are described in this document.

1.1 SCOPE

This document outlines the design specification needed to satisfy the requirements in the *Functional Specification for an Ecological Monitoring System*. The requirements outlined in this document pertain to the proof-of-concept design as well as a portion of the final production (labeled as R[X-I] in the aforementioned document). Detailed explanation of each component of the ECOmonitor is presented in order for the reader to gain a full perspective of the project.

1.2 INTENDED AUDIENCE

The intended audience for the design specifications are the members of the ECOmonitoring Technologies Inc. team. This document will be used as a guideline to ensure that all the intended functional requirements have been met. Also, the team members will carefully follow the proposed test plan to ensure accurate and proper operation of the system.

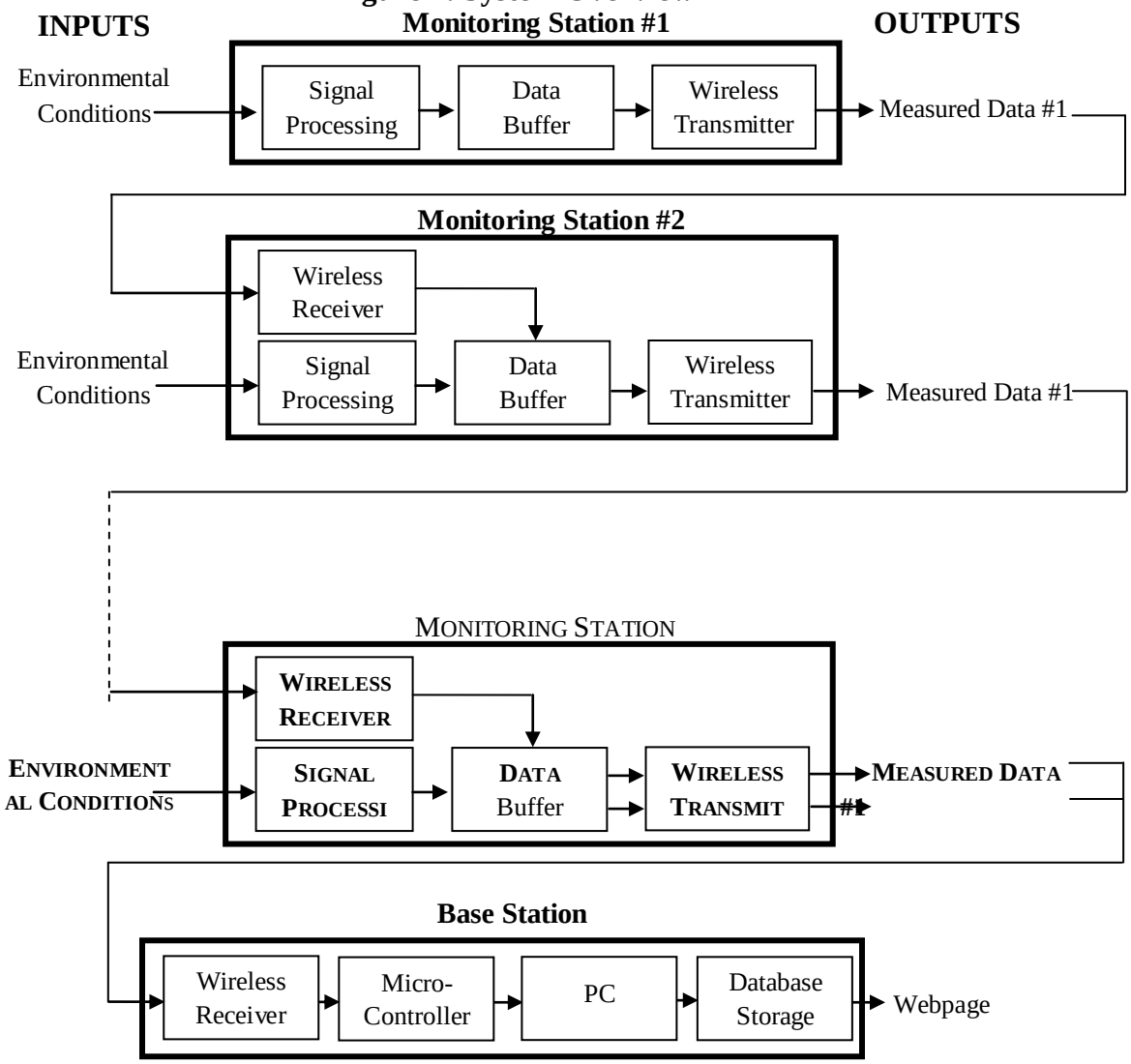
2.0 SYSTEM SPECIFICATIONS

The ECOmonitor will consist of multiple monitoring stations, a base station and a website component. Each monitoring station will be made up of a microcontroller, wireless transceiver and sensors. The sensors will periodically collect readings of environmental conditions which will be wirelessly propagated through each station ending at the base station. At this point, the data will be analyzed for errors and stored into the database. The user will then be able to select a monitoring station and time interval to view the data graphically on a webpage.

3.0 SYSTEM OVERVIEW

This section provides a high-level overview of the entire system design. Design details pertinent to the interaction of the various ECOmonitor components will be discussed, while design details specific to individual parts of the ECOmonitor will be found in their respective sections. The system overview is shown in **Figure 1.**

Figure 1: System Overview



Each monitoring station takes inputs from the environment through the use of a sensor array. The input signals produced from this sensor array are then processed within the micro-controller and placed in a data buffer internal to the microcontroller RAM memory. For the humidity and temperature sensors the data is retrieved via a Two-Wire interface, and for the CO_2 sensor an Analog-to-Digital convertor will be used to condition the signal before it is sent to the microcontroller. This data is stored in the microcontroller buffer until a flag is set giving the micro-controller permission to feed the contents of its buffer into the wireless transmitter. The specifics of how this permission is given to the micro-controller are shown in wireless 'hopping' algorithm figure and microcontroller figure. At the base station the data will be received via the wireless transceiver and relayed through the microcontroller which will then communicate with the host PC. The host PC will receive the data through a python interface and store it onto the database. The database will be tied to the webpage which will display all of the gathered data.

3.1 MONITORING STATIONS

Each monitoring station will include one wireless transceiver, a sensor array and a microcontroller. As shown in **Figure 2** and **Figure 3** the wireless transceiver is placed at the top of the enclosure, with the sensors being placed on the underside of the enclosure. The microcontroller will be placed between the wireless transceiver and sensor array to facilitate ease of wiring. The proof-of-concept model may not appear as shown in the figure.

Figure 2: Monitoring Station Top View

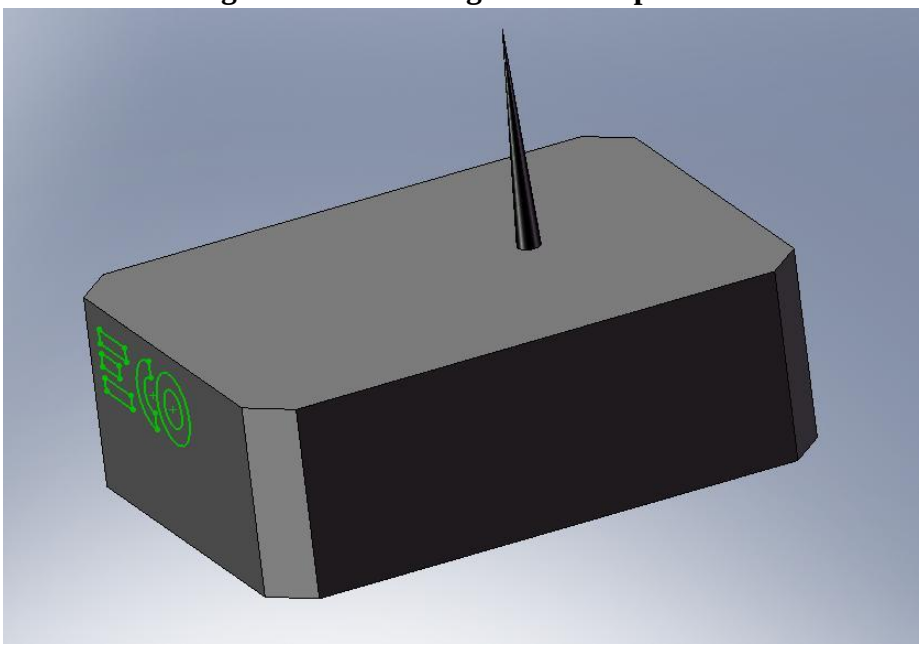
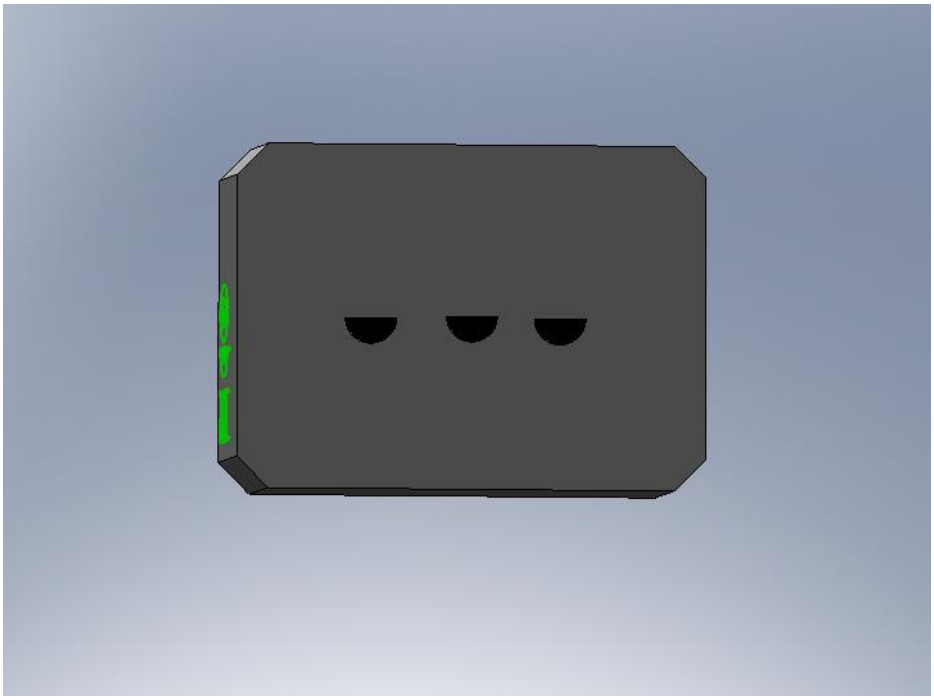


Figure 3: Monitoring Station Bottom View



The wireless transceiver antenna is not shielded from the elements when placing it at the top of the monitoring station. This improves vantage point for line of sight transmission to other monitoring stations. In order to protect the wireless transceiver from the elements sealant will be placed around the hole for the antenna.

The sensor array will contain a humidity, temperature and CO_2 sensor. Placing the sensors on the underside of the enclosure allows for the sensor station itself to partially shield the sensors from the elements while still allowing them to still freely receive inputs from the environment.

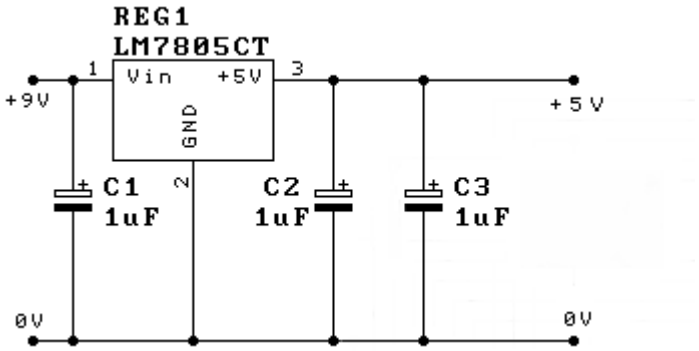
3.2 BASE STATION

The base station will consist of a wireless transceiver, microcontroller and a PC with internet connectivity as shown in **Figure 2**. There will be no sensor array underneath the device. The wireless transceiver will receive data from the monitoring stations which will then be processed by the microcontroller and sent to the PC to be posted online.

3.3 POWER SUPPLY

Power will be supplied to the sensor stations by a 9V battery connected to the microcontroller via a voltage regulator as shown in **Figure 4**. Any input power supply noise is filtered out through the capacitors. The 5V power source is then fed into the microcontroller, and the Microcontroller VCC ports are used to power the wireless transceiver and sensor array.

Figure 4: Voltage Regulator



4.0 MICROCONTROLLER

The primary functions of the micro-controller module for the ECOmonitor can be broken down into three functions:

- Communicate sensors and ADCs to obtain data
- Process data packets and communicate with XBee Wireless Module
- Communicate with the host PC for data collection

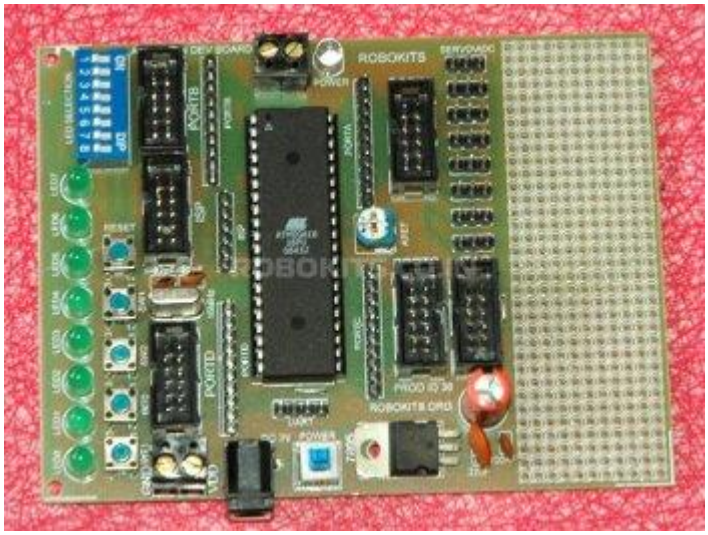
As the primary usage only requires just enough clock speed to process all data within a given amount of time and does not need a large amount of memory, the choice of the micro-controller module was scoped down to Atmel AVR products. We have managed to find a relatively small Robokits AVR 40 pin Development board which we mounted with Atmega32 micro-processor.

Atmega32 micro-processor features 2K Byte of Internal SRAM which provides more than the storage we need to temporarily store data packets being processed and also features 8MHz internal oscillator which gives more than enough speed. In addition, it has internal 8-channel ADCs allowing at most 8 Analog signals to be read without the need of extra ADCs unless we are looking for very high precision information. The 32 Programmable I/O lines provided are used for LED indicators to aid people installing the sensor stations and also for debugging.

4.1 COMPONENTS

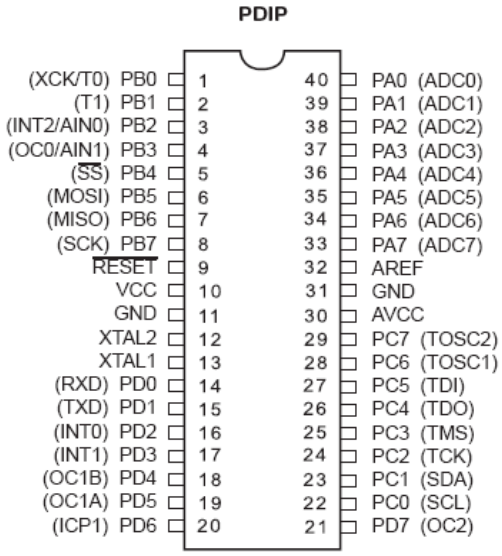
The Robokits AVR 40 pin Development board shown in **Figure 5** has FRC10 standard cables for connecting the GPIOs (PortA ~ PortD), ADCs, and TWI interface devices. There is an 11x34 PCB area where XBee Wireless Module is going to be installed.

Figure 5: Robokits AVR 40 pin Development Board



The pinout of the Atmega32 is shown in the **Figure 6** below:

Figure 6: Pinout of Atmega32 [2]



The Ports A to D can all act as GPIOs or AFIOs as noted in parenthesis. The Alternate Functions used are summarized in **Table 1**.

Table 1: Atmega32 Alternate Function Ports Summary

Ports	Function
PA0 to PA7 (ADC0~7)	Receiving analog voltage outputs from sensors
PB0 to PB7	LEDs for various indicating purposes
PC0 and PC1 (SDA and SCL)	TWI communication with various digital output sensors
PC2 to PC5 (TDI, TDO, TMS, TCK)	JTAG Debug mode
PD0 and PD1 (RXD and TXD)	USART/UART communication with XBEE Wireless Module and also the host PC

The Alternate Functions can be used by setting up proper over-riding signals (summarized in Table 21 in Atmega32 datasheet) [2]. Unconnected pins are going to be set to a defined level with internal pull-up resistors in order to avoid floating inputs leading to unneeded current consumptions.

An algorithm illustrating the microcontroller sensor read-in is shown in **Figure 13.** The first part of the algorithm shows initialization of variables. Then in the middle part, the averaging is done by taking several measurements and taking average defined by AvgCount. Then the micro controller checks if the HeaderFlag from the XBEE wireless module is set or not to decide if it should send out the newly read sensor data out to the next available station or not. The un-submitted data are stored in memory until later the HeaderFlag gets set.

4.2 MICROCONTROLLER – PYTHON COMMUNICATION

The communication between Atmega32 micro controller and the host PC is done through USART. For USART interface, pySerial is used with python. The built-in library functions enable us to connect to the micro-controller easily and the baud rate was set to 38400bps which gives 0.02% error (the error is filtered out during USART communication). The corresponding UBBR is 12 based on 8 MHz clock on Atmega32.

On the micro-controller firmware, it was found out after a simulation of 10000 data transfers that at least 1ms of delay is needed between each byte transmission on USART to avoid errors.

5.0 SENSORS

The sensors that we chose to implement for the proof-of-concept are directed towards the use of fire detection: temperature, humidity and CO_2.

5.1 HUMIDITY AND TEMPERATURE SENSOR

The major requirements we had for these sensors were that they were fairly accurate, had a good resolution, consumed less than 50 mW of power, and were able to detect temperatures up to 100 degree Celsius. Since these sensors are going to be used outdoors we also had to make sure that the sensors are able to operate properly in relatively cold or hot weather. We were able to find a large selection of sensors that met these requirements. We ultimately decided to go with the Sensirion SHT10 humidity sensor and the SA56004X temperature sensor.

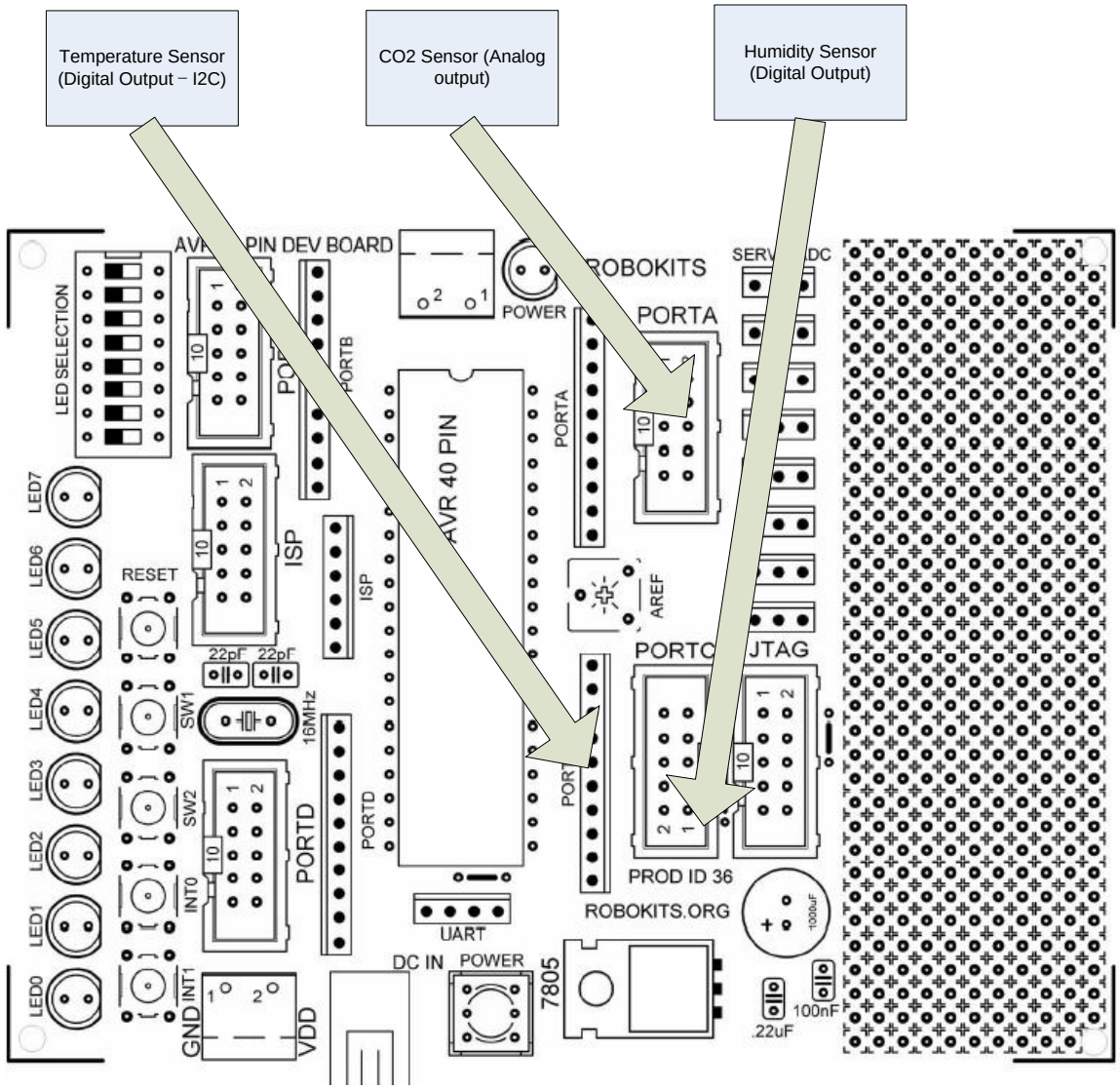
5.2 CO_2 SENSOR

For the detection of CO_2 levels we needed a sensor that was able to detect CO_2 levels over a fairly large range of values to reduce the number of false alarms that would be triggered by our system. These sensors give a CO_2 reading based on the PPM levels in a sample of air. We found that fires typically have a PPM value of over 5,000 PPM so we were looking for a sensor capable of sensing up to10,000 PPM. We also required that the sensor used less than 50 mW of power and would be able to function properly in temperatures between -40 and 80 degrees Celsius. We found four sensors that met these requirements. Three of them operated in the IR range and were priced between $ 120 and $ 400. Since we only had a budget of $ 950 and needed to create two units, we couldn't afford to purchase any of these sensors. The other sensor we found was the Futurlec MG811 CO2 Sensor. This sensor was significantly cheaper and still had a good enough resolution for our purposes.

5.3 SENSOR AND MICRO CONTROLLER MODULE COMMUNICATION

The CO_2, humidity, and temperature sensors are connected and communicated with the microcontroller as shown in the functional diagram in **Figure 7**.

Figure 7: Sensors' connections on Robokits AVR 40 pin Development Board [3]



The CO_2 sensors output analog voltages and they are connected to ADCs on Port A. The temperature sensors and humidity sensors output digital values through two-wire interface which can be communicated with TWI function on Port C of Atmega32 micro-controller.

5.4 TWI INTERFACE AND CONVERSION RATES

CO2 Sensor

The analog voltage levels will be taken by ATmega32's internal 10-bit ADCs and the conversion rate at our 8MHz clock is at max 200ms. The output voltages from CO2 will be scaled by programmable gain feature of internal ADCs to give us maximum range for the best reading resolution.

Temperature Sensor

SA56004X is a SMBus compatible, 11-bit remote/local digital temperature. Since this device is fully SMBus compatible, any SMBus specification could be used to develop the communication with the Atmega32 micro controller [4]. The address of our temperature sensor is defined by its ordering number and the address is defined as '1001 011'. The conversion rate is defaulted to 16 conversions/s which is roughly 62.5ms per conversion. For specific interface commands, refer to SA56004X_5 datasheet [5].

Humidity sensor

The interface is a common 2-wire interface with CRC checksum. **Table 2** summarizes the commands we would be using in our product (address is 000 by factory default). The maximum time of one 12-bit accuracy sensor reading is 80ms.

Table 2: Humidity Sensor Two-Wire Interface Commands

Command	Address + Command code
Soft reset (resets the interface, clears the status register to default values. Wait minimum 11 ms before next command)	00011110
Measure Relative Humidity	00000101
Read Status Register	00000111
Write Status Register	00000110

The default bit resolution for relative humidity measurements is set to 12 bits and thus the status registers do not have to be modified. For compensating non-linearity of the humidity sensor and for obtaining the full accuracy of the sensor, the formula below gives a both humidity and temperature compensated values of relative humidity.

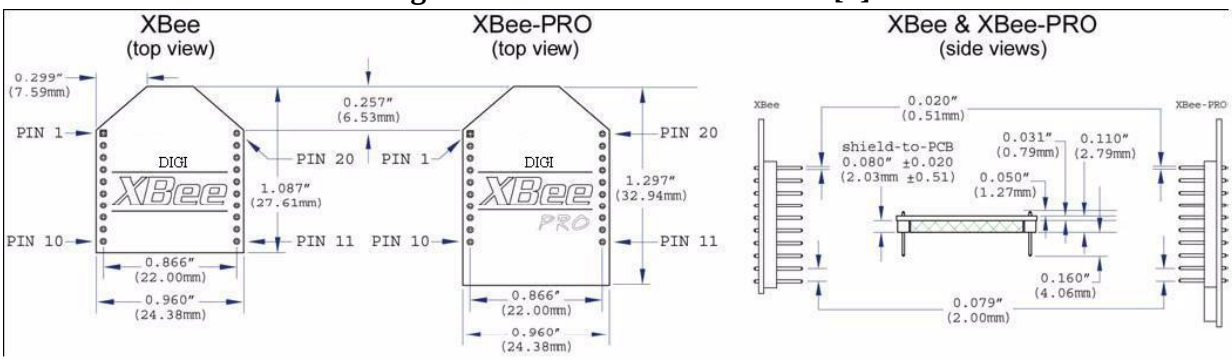
$$RH_{true} = (T_{\circ C} - 25)(0.01 + 0.00008 \cdot SO_{RH}) - 2.0468 + 0.0367 \cdot SO_{RH} - 0.0000015955\, SO_{RH}^{\ 2}$$

SO_{RH} represents the relative humidity sensor output and $T_{\circ C}$ is the temperature at where the humidity sensor is located. The temperature read out from the temperature sensor would replace the value of $T_{\circ C}$ every time the readout process is done.

6.0 WIRELESS TRANSCEIVER

To communicate between monitoring stations and the base station, a wireless transceiver was needed. From the requirements outlined in *Functional Specifications for an Ecological Monitoring System* we decided to use the XBee-Pro XSC RF Module [1]. This module was used in favor of other technologies due to its ease of use and superior transmission range when compared to other modules available at its price point.

Figure 8: XBee Wireless Module [6]



6.1 MECHANICAL DESIGN

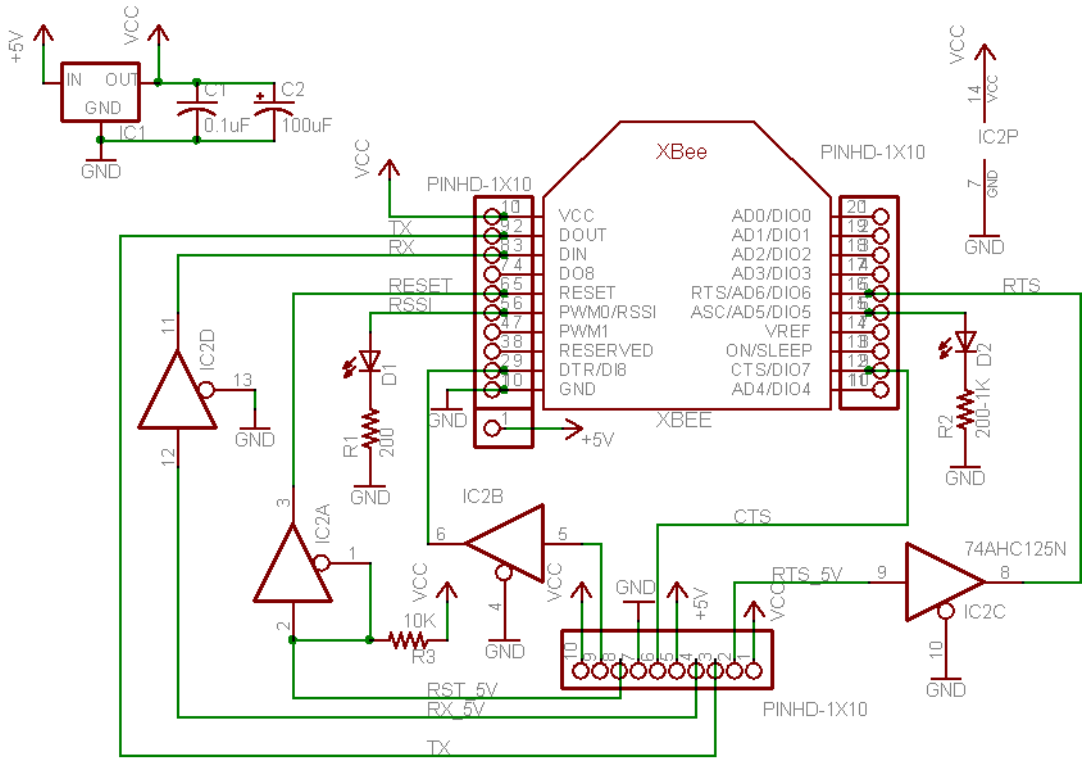
In order to minimize the effects of any interference created from the micro-controller and sensor circuitry the wireless module is to be placed near the top of the sensor station case with the wire antenna protruding through the case. This arrangement is also beneficial in ensuring maximum range as a higher vantage point is beneficial for the effective transmission range of the XBee module.

6.2 ELECTRONIC DESIGN

The XBee module is mounted and interfaced to the microcontroller through an adapter board. A protection circuit is used between the XBee module and the microcontroller interface. The protection circuitry, **Figure 9**, consists of a voltage regulator and an input buffer. The voltage regulator is a low dropout voltage regulator that can deliver up to 250 mA of current, and uses an input operating range between 2.3 to 6.0 V, within our microcontroller output voltage of 5 V. The nominal output voltage of the regulator is 3.3 V which is within the 3.0-3.6 V VCC voltage range of the XBee-Pro module. Due to the 5 V output voltage of the microcontroller it was

necessary to use a voltage buffer to interface with the wireless modules 3.3 V UART connections. For ease of debugging the RSSI (Activity) and ASSC (Power) ports of the XBee module were tied to LEDs.

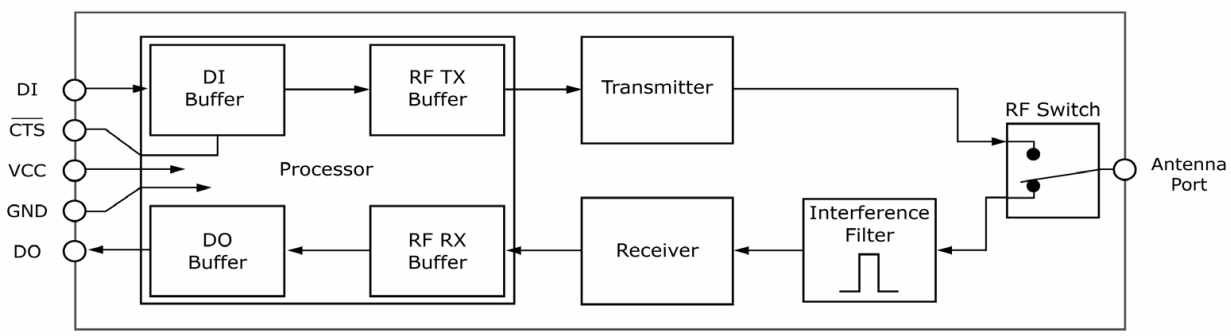
Figure 9: Protection Circuitry [7]



6.3 WIRELESS COMMUNICATION PROTOCOL DESIGN:

In order to create the 'hopping' protocol it was first necessary to understand the dataflow within the wireless module. As shown in **Figure 10** the XBee module contains both an input and output buffer. Both the input and output buffer's can hold 1000 bytes of data that is serially fed in 64 byte packets.

Figure 10: Internal Dataflow Diagram [6]



To initialize transmission the XBee module first sends an RF initializer packet that is used to capture the transmission channel for all transceivers on the network. After the channel is captured data is fed from the Data In buffer through the transmitter to the appropriate monitoring stations. The algorithm by which the transceiver stations send and receive these packets is outlined in Figure 14

Figure 14.

6.4 XBEE MODULE PACKET CREATION

In the construction of our data packets into the input buffer, we had to be aware of the fact that data is sent transmitted in packets of 64bytes. Since we are appending an identifier byte to each data packet it is crucial that we stay within this limit. The first byte of each data packet will contain the Current Station ID (CSID) and will be used to ensure data is not transmitted in the opposite direction of the base station. The second byte of the data packet will contain the Origin Station ID (OSID) that will be used by the base station to identify which location the data originated from. The data header composition is shown in **Figure 11**.

Figure 11: Data Header Composition

CSID(1 byte)	OSID(1 byte)	HC(1 byte)	Sensor Data (61 bytes)

6.5 WIRELESS COMMUNICATION FLOW CONTROL

The Hopping Control (HC) flag is used on a system level to denote whether the HeaderFlag has been set to 1 in any of the previous hops undergone by the data packet. The HC Flag was instituted primarily to safeguard against the adjacent sensor station failing. If the adjacent sensor station fails the system will now be capable of transmitting the data to the next sensor station and resume normal system operation. The HeaderFlag is essential in ensuring safe dataflow through the sensor network. If HeaderFlag =1 the station will be allowed to capture the transmission channel and transmit its data. A detailed diagram of the functionality of this is communication protocol is shown in algorithm **Figure 14**.

In order to ensure data collected from each station is not lost it was necessary to take into account the sensor capture time and capture intervals used by the sensors and ensure that wireless transmission of the last packet to the base station is completed before the input buffer of the new token holding station is full. Employing a sensor capture rate of 2 ms and a capture interval of 342.5 ms (refer to **Section 7.2**)it was found that it would take at least 3.15 s to fill the input buffer of the wireless module as shown in the following formula.

$$(capture\ time + capture\ interval)\ x\ \frac{1000byte(Data\ Input\ Buffer)}{\frac{64byte}{packet}} = total\ sensor\ capture\ time$$

We use a data transmission rate ≥ 38400 bps because with this data transmission rate we find using the formula below that our maximum transmission time will be 2.64 s. This transmission time will decrease as the wireless transmission rate is increased or number of sensor stations (# hops) is decreased. This maximum transmission time is determined using a packet length = 64 bytes, a wireless transmission rate = 38400 bps (4800 byte/s), and 99 hops. The resulting time is multiplied by two in order to compensate for the acknowledgement time and any other delays experienced during data processing by the microcontroller.

$$\max transmission time = \frac{packet length}{wireless transmission rate} \times \#hops \times 2$$

7.0 DATABASE

The database is used to store the data received from the monitoring stations. We chose the commonly used MySQL database software for its speed, ease of use and reliability.

7.1 BASE STATION AND DATABASE COMMUNICATION

We will be using the Python MySQL module to communicate with our database. Our database is located on a leased server space so we will need to connect and send data to it using this module. We initially proposed to create a CSV file at the base station that was written to every time a new packet was received. This CSV file would then be uploaded to the server, and eventually our database, using PHP scripting language. However, this method turned out to be inefficient since we would have to create a new file every time a packet was received. The Python MySQL module allows us to extract data from the packet at the base station and send it directly to our server, eliminating the any file creation at the base station. This reduces processing time, giving a more efficient solution.

7.2 COMMUNICATION PROTOCOL FOR BASE STATION AND DATABASE

Each of the monitoring stations will send data packets to the base station in a predetermined manner every minute via a transceiver hopping network. The base station will have a buffer capable of storing up to ninety-nine packets and will use the FIFO queue processing technique to grab packets from the buffer. This buffer size corresponds to the maximum number of units the ECOmonitor is capable of supporting. Every time a data packet is received at the base station it will be placed into this buffer. Packets will be retrieved from this buffer and processed using the following algorithm.

7.3 ALGORITHM

A flow chart of our algorithm can be seen in **Figure 15**. The number of stations in the ECOmonitor will be represented by the variable 'N' and may not exceed the system limit of ninety-nine. We will use a loop variable 'X' to represent the station which we expect to receive a packet from. If 'X' is the same as the OSID, then we know that we did receive a packet from station 'X' and thus it is functioning properly. On the other hand, if they aren't equal we know that at least one station didn't send a packet and thus the station(s) isn't functioning properly. To notify the system of this we will use a variable called 'Flag', which is active high. 'Flag' will be set to 0 if station X was successful in sending a packet.

Initially 'X' will be set to the number units in the system 'N' and 'Flag' will be set to 0 since we will be expecting to receive a packet from the Nth station first. The first condition we will check is if 'Flag' is equal to 1. Since we are assuming that this will be very first packet received, this condition will be false and thus we will grab a packet from the base station buffer. We will then extract the various data fields from the packet using the Python programming language. These fields will include the OSID value, temperature, CO_2, and humidity level. In the flow chart the variable DATA is used to represent readings from all three sensors.

Next, we will check if the received packet is actually from the expected by checking if OSID is equal to 'X'. If these two are equal we know that station X was successful sending its packet and thus we store the sensor readings into there respective variables (i.e. temperature, humidity, C02) and set 'Flag' to 0. If on the other hand they aren't equal, the packet from station X wasn't received indicating something has gone wrong at station X. If this happens we notify the system by setting 'Flag' to 1 and the respective data fields to 'undefined'.

Using Python we will store the readings for temperature, humidity, CO_2 and OSID in to a MySQL database. We will also time stamp the sensors readings to keep track of when the data was received. After storing the data into the database we will change the loop variable based on if 'X' is equal to 1. If 'X' is equal to one, this means that we have just finished processing data for station one and thus we will be expecting to receive a packet from the last station 'N'. Thus

we will set 'X' to N. If 'X' is not equal to one then we know that we haven't reach the first station so we will decrement 'X' by one.

After changing the loop variable we repeat the steps above to retrieve the sensors reading for sensor 'X' – 1 and so forth. This program will keep looping through the code, thus giving new data readings for each sensor at a fixed interval of time.

7.4 DATABASE STRUCTURE

The general format of the MySQL database is shown in **Table 3**. The first three columns contain the Date, Time and Station ID. The date will be in month/day format, the time will be in 24 hour clock and the Station IDs will be in sequential order. The next N columns will contain the sensor readings depending on the number of sensors implemented on the monitoring stations. The date and time will be automatically generated from the base station whenever data is received. This format is easily understandable and will provide simple communicate between the database and webpage.

Table 3: MySQL Database Format

Date	Time	Station ID	Sensor 1	Sensor 2	Sensor 3	…	Sensor N
Jan.1	00:00:00	1	$S1_{1,T0}$	$S2_{1,T0}$	$S3_{1,T0}$	…	$SN_{1,T0}$
Jan.1	00:00:00	2	$S1_{2,T0}$	$S2_{2,T0}$	$S3_{2,T0}$	…	$SN_{2,T0}$
Jan.1	00:00:00	3	$S1_{3,T0}$	$S2_{3,T0}$	$S3_{3,T0}$	…	$SN_{3,T0}$
Jan.1	00:00:00	⋮	⋮	⋮	⋮		⋮
Jan.1	00:00:00	n	$S1_{n,T0}$	$S2_{n,T0}$	$S3_{n,T0}$	…	$SN_{n,T0}$
Jan.1	⋮	⋮	⋮	⋮	⋮		⋮
Jan.1	24:00:00	1	$S1_{1,T24}$	$S2_{1,T24}$	$S3_{1,T24}$	…	$SN_{1,T24}$
Jan.1	24:00:00	2	$S1_{2,T24}$	$S2_{2,T24}$	$S3_{2,T24}$	…	$SN_{2,T24}$
Jan.1	24:00:00	3	$S1_{3,T24}$	$S2_{3,T24}$	$S3_{3,T24}$	…	$SN_{3,T24}$
Jan.1	24:00:00	⋮	⋮	⋮	⋮		⋮
Jan.1	24:00:00	n	$S1_{n,T24}$	$S2_{n,T24}$	$S3_{n,T24}$	…	$SN_{n,T24}$
⋮							
Dec. 31	00:00:00	1	$S1_{1,T0}$	$S2_{1,T0}$	$S3_{1,T0}$	…	$SN_{1,T0}$
Dec. 31	00:00:00	2	$S1_{2,T0}$	$S2_{2,T0}$	$S3_{2,T0}$	…	$SN_{2,T0}$
Dec. 31	00:00:00	3	$S1_{3,T0}$	$S2_{3,T0}$	$S3_{3,T0}$	…	$SN_{3,T0}$
Dec. 31	00:00:00	⋮	⋮	⋮	⋮		⋮
Dec. 31	00:00:00	n	$S1_{n,T0}$	$S2_{n,T0}$	$S3_{n,T0}$	…	$SN_{n,T0}$
Dec. 31	⋮	⋮	⋮	⋮	⋮		⋮
Dec. 31	24:00:00	1	$S1_{1,T24}$	$S2_{1,T24}$	$S3_{1,T24}$	…	$SN_{1,T24}$
Dec. 31	24:00:00	2	$S1_{2,T24}$	$S2_{2,T24}$	$S3_{2,T24}$	…	$SN_{2,T24}$
Dec. 31	24:00:00	3	$S1_{3,T24}$	$S2_{3,T24}$	$S3_{3,T24}$	…	$SN_{3,T24}$
Dec. 31	24:00:00	⋮	⋮	⋮	⋮		⋮
Dec. 31	24:00:00	n	$S1_{n,T24}$	$S2_{n,T24}$	$S3_{n,T24}$	…	$SN_{n,T24}$

8.0 WEBPAGE INTERFACE

The webpage interface is designed using the three following programming languages:

1. HTML
2. PHP
3. Javascript

HTML is the predominant markup language used to design the template of a webpage. PHP is the server-side scripting language that is able to connect and retrieve data from the MySQL database. Javascript is the client-side language that is used to produce the time-series plot. PHP and Javascript were chosen as the ECOmonitoring team was knowledgeable in programming these languages.

8.1 USABILITY DESIGN

The user interface consists of an online webpage with a graphical component providing the user the ability to view the data on any web browser. The user will be able to select the following parameters to graph:

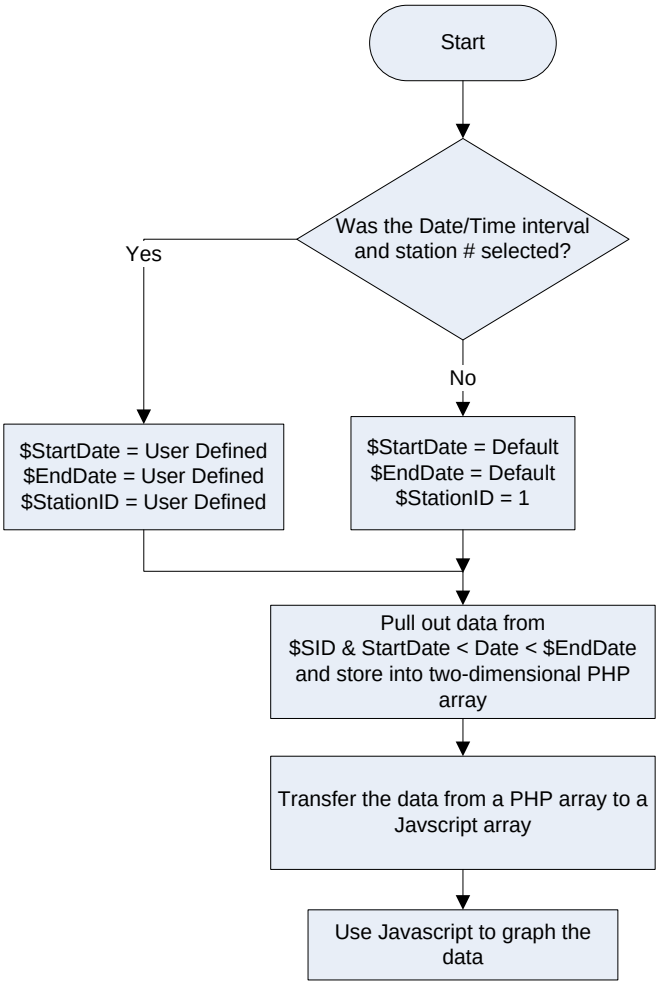
- Which data (ie. Temperature, Humidity, CO_2) to view
- Which monitoring station to view
- The time interval

One graph will be assigned to each data set, but multiple monitoring stations can be displayed on each graph. Once the user selects these parameters, a graph will display the specified data on the y-axis and the time on the x-axis. The user will be able to view the data in real-time as the page will refresh once every minute.

8.2 GRAPH DESIGN

The webpage will be able to graph the data hourly, daily, weekly and monthly for multiple monitoring stations. In order to retrieve data to minimize load times, the user can specify the exact monitoring stations to be viewed and for which time interval. If no time interval or monitoring station is specified, a default monitoring station (closest station) will be used and the graph will display the data hourly for the current day. **Figure 12** illustrates the flow chart of the graphing interface.

Figure 12: Graphing Interface Flow Chart



Whenever the page is loaded, the PHP script will check if the user has selected a time interval and monitoring station to view. If yes, then the three variables $StartDate, $EndDate and $StationID will be assigned the user's input. However, if the user has not selected a time interval then a default start date, end date and station ID will be assigned. The PHP script will pull the data according for $StationID between $StartDate and $EndDate. Each set of data (time, temperature, humidity, CO_2, etc.) that is pulled out from the database will be placed into their own two dimensional PHP array where the first dimension is the Station ID while the second dimension is the data reading. A function is used to convert that PHP array to a Javascript array. The Javascript array will finally be input into the graph.

9.0 ENVIRONMENTAL CONSIDERATION

While designing our system, environmental issues were taken into consideration. Our system creates minimal pollution since it is running off of a 5 volt battery for our current design, and will run off of a solar panel in the future. Since our system will be able to detect fire hot spots, we will be able to reduce the amount of pollution that fires contribute to plant. We will be using radio frequencies to communicate data between stations thus very minimal radiation will be transmitted. Our casing will be made from material ecologically inert, which will also reduce pollution.

10.0 SYSTEM TEST PLAN

10.1 TEST CASE #1

The purpose of this test scenario is to test the functionality of the individual components of our system. This test will be run under normal conditions (i.e. no fire) and with a 300 meter distance between stations and the host station. We will be testing the following functions:

1) Station 2 is able to successfully send a packet to station 1.
2) Station 1 is able to send the packet it received from station 1 to the base station.
3) Communication between base station and database work correctly.
4) The sensor readings are stored into the correct location in the database.
5) The plots for temperature, CO_2 levels, and humidity are updated every time a new packet has been inserted into the database.

We will also test if station 1 is able to successfully send its sensor readings to the host station and that the system is able to process the data as mentioned above. This will also test that our system is able to handle multiple stations sending packets at the same time.

10.2 TEST CASE # 2

This case will be used to verify that the hopping network works properly. Since our ECOmonitoring system will only consist of only 2 monitoring stations and a base station for our project, we will disconnect the monitor at station 1 to mimic a scenario where a monitor is down (i.e. failure). We will place the sensors 300 meters apart from each other and the host station 300 meters away from station 1 and 600 meters from station 2. To verify proper functionality we will be testing the following things:

1) Station 2 is able to transmit a data packet directly to the host station, thus bypassing station 1.
2) The host station is able to receive the packet sent from station 2.
3) The delivered packet contains correct sensor reading values.

4) Our communication protocol between the host station and server works correctly. It should be able to detect that no packets are being sent from station 1, indicating that it is down.
5) The temperature, CO2, and humidity values for station 1 are all set to undeclared in our database.
6) Our website stops creating an updated plot for station 1.

10.3 TEST CASE # 3

Here we will be testing how the ECOmonitoring system works when a fire has been detected. To mimic a wild fire, we will start a small controlled fire near one of the sensors. We will check that everything work correctly as in the cases above but we will also check that the webpage is able to conclude that a fire has been detected near the station.

10.4 TEST CASE # 4

This will be the last phase of our system test plan. This test will be used to check our systems ability to detect false alarms. To do this we will first release some CO_2 gas near station 1 and check if our system is able to detect the abnormal level. In this case our system should tell the user that an abnormal level has been detected for CO_2 gas but no fire is present. We will then keep the CO_2 levels constant and increase the temperature and humidity levels and check if it is able to detect this in a similar manner.

11.0 CONCLUSION

This document has discussed the proposed design specifications that meet the functional requirements of the ECOmonitor. Justifications for the chosen components have been discussed as well as a general algorithm for our 'hopping' technology design. Flow charts have been presented to illustrate the process of each component and finally a system test plan was presented to ensure proper functionality of the ECOmonitor. This document provides specific goals for the development and design of the ECOmonitor.

12.0 APPENDIX

Figure 13: Microcontroller sensor read-in algorithm

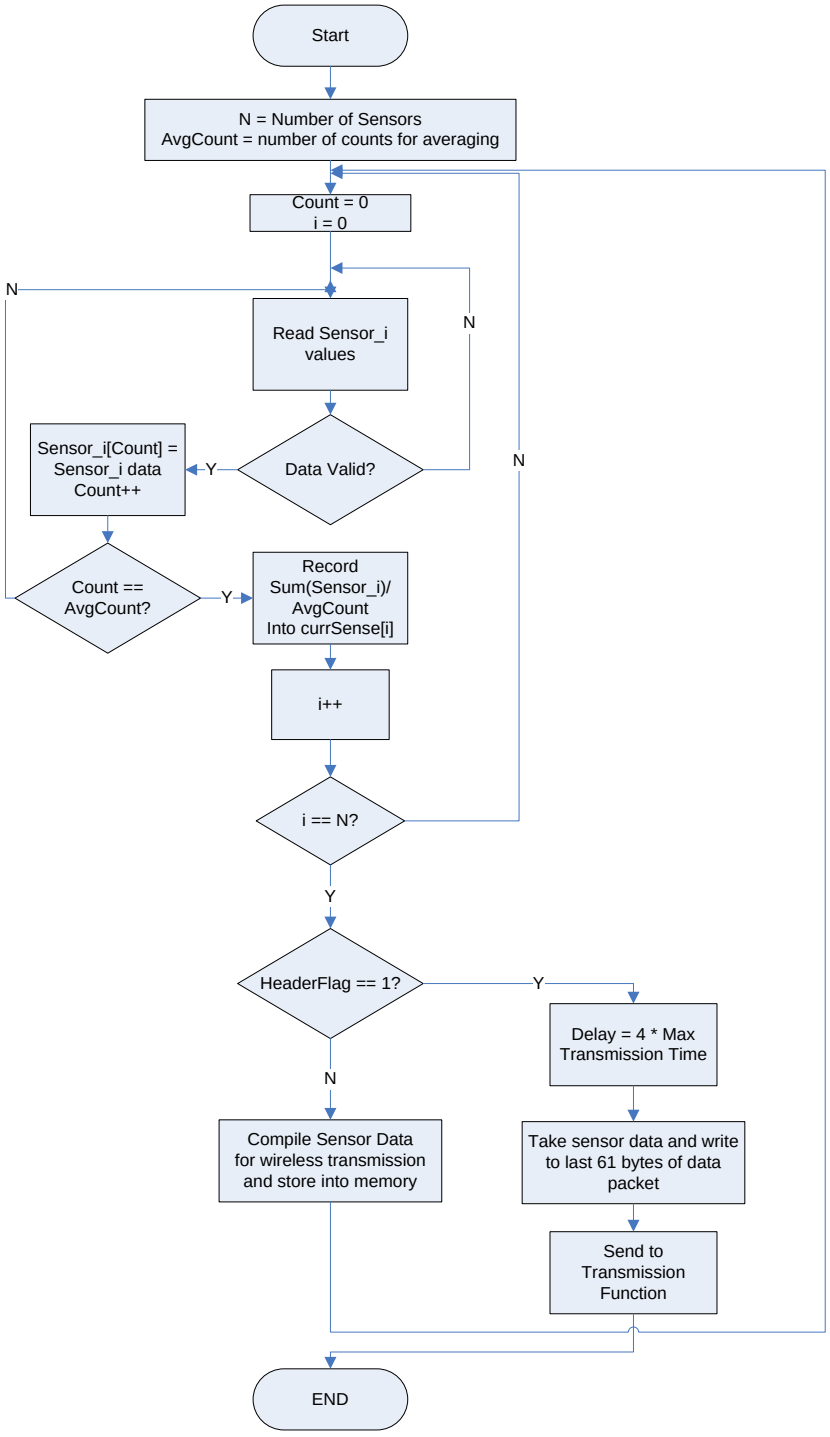


Figure 14: Algorithm for Hopping Schema for One Packet

Receiving Packet and Extracting Information

Checking Uni-Directional Transmission and Dead Stations

Recomposing the Packet and Transmitting

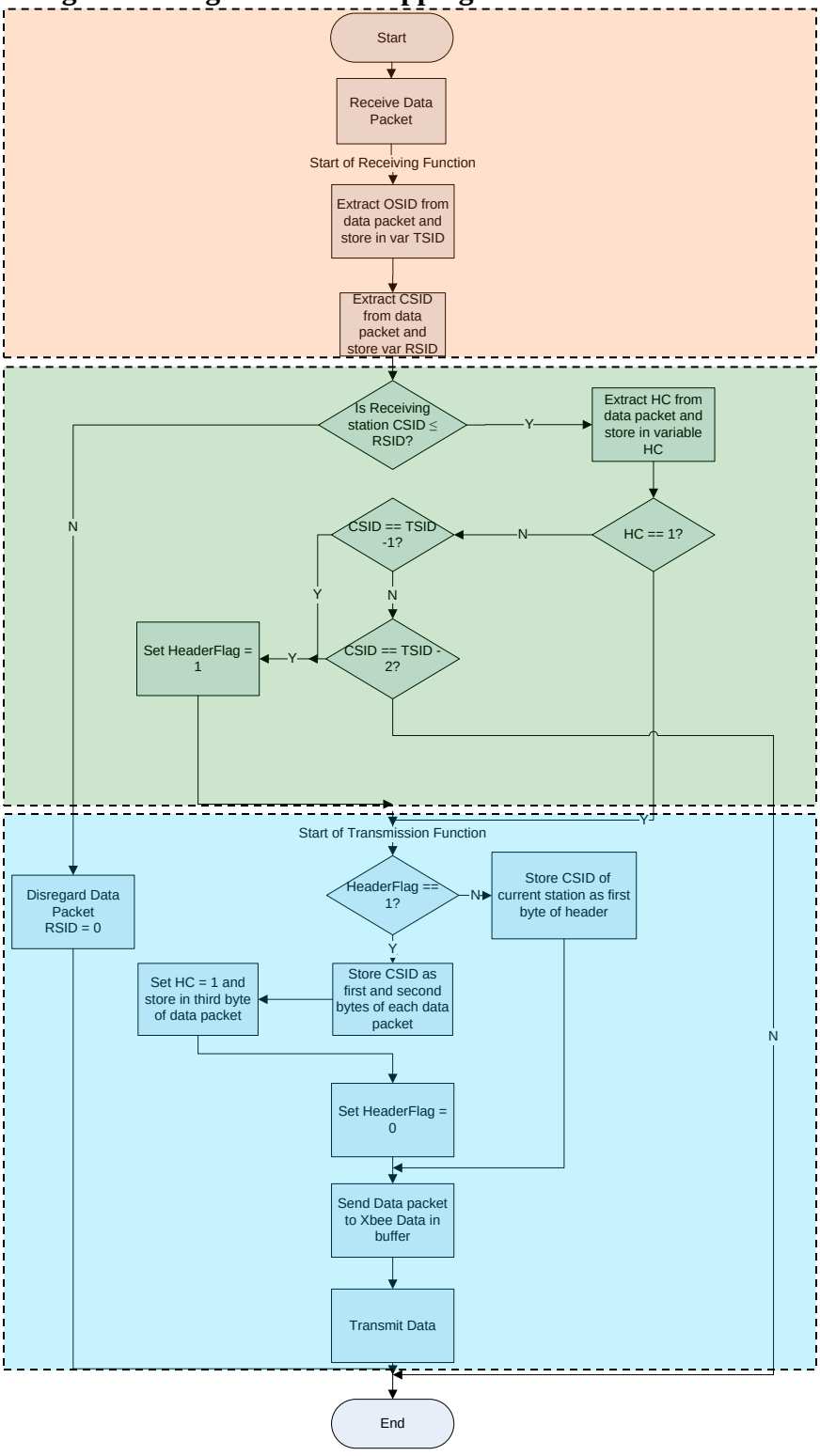
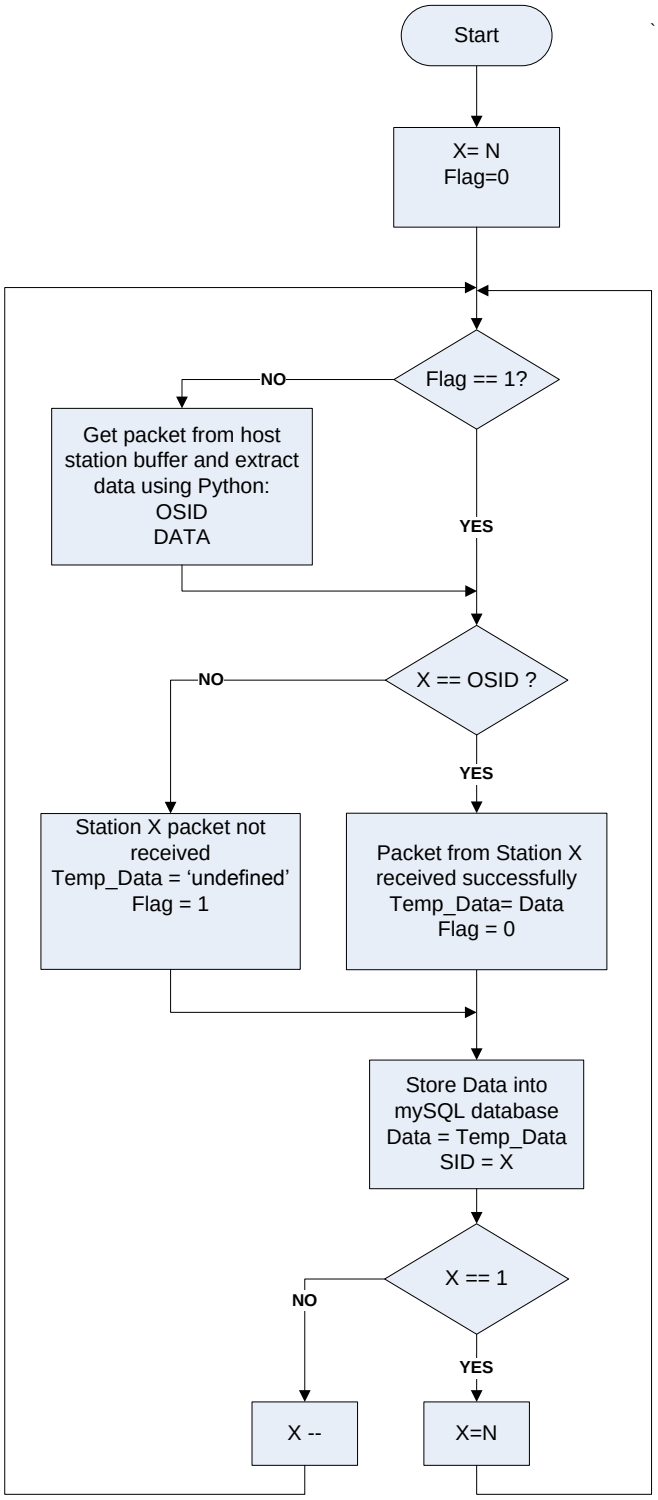


Figure 15: Base Station and Database Communication Algorithm



13.0 REFERENCES

[1] ECOmonitoring Technologies Inc., "Functional Specification for an Ecological Monitoring System", Simon Fraser University, Burnaby, BC, Canada, October 2009.

[2] ATMEL, "ATmega32 Datasheet" [Online]. Available: http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf. [Accessed: Oct. 24, 2005].

[3] Robokits, *AVR 40 Pin Development Board*, [Online]. Available: http://www.robokits.co.in/documentation/AVR%2040%20Pin%20Development%20Board.pdf. [Accessed: September 20, 2009].

[4] Sensirion, *Datasheet SHT1x*, [Online] Available: http://www.sensirion.com/en/pdf/product_information/Datasheet-humidity-sensor-SHT1x.pdf. [Accessed:September 26, 2009].

[5] NXP Semiconductors, *SA56004X*, [Online] Available: http://www.nxp.com/acrobat_download/datasheets/SA56004X_5.pdf. [Accessed: September 27, 2009].

[6] digi.com, product manual XBee Pro XSC, 2009 [Online]. Available: http://www.digi.com/register/index.jsp?mu=/pdf/ds_xbeemultipointmodules.pdf. [Accessed October 15, 2009]

[7] ladyada.net, Xbee adapter schematic, 2009 [Online]. Available: http://www.ladyada.net/images/xbee/xbee11sch.png. [Accessed: October 31, 2009]