WISENET

STUDENTS: JOSEPH DUNNE

DAVID PATNODE

DEPARTMENT: ELECTRICAL ENGINEERING

FACULTY: DR. MALINOWSKI

DR. SCHERTZ

SENIOR DESIGN PROJECT

2002-2003

ABSTRACT

WISENET is a wireless sensor network that monitors the environmental conditions (such as light, temperature, and humidity) of labs and offices in Jobst Hall. This network is comprised of nodes called “motes” that form an ad-hoc network to transmit this data to a computer that functions as a server. The server stores the data in a database where it can later be retrieved and analyzed via a web-based interface. The project works successfully with an implementation of one sensor mote.

Table of Contents

Illustration Index

Background

The technological drive for smaller devices using less power with greater functionality has created new potential applications in the sensor and data acquisition sectors. Low-power microcontrollers with RF transceivers and various digital and analog sensors allow a wireless, battery-operated network of sensor modules ("motes") to acquire a wide range of data. The TinyOS project at University of California, Berkeley (http://today.cs.berkeley.edu/tos/) has created a real-time operating system to address the priorities of such a sensor network (low power, hard real-time constraints, robust communications). MIT even recognized wireless sensor networks and TinyOS as one of the ten emerging technologies that will change the future (Technology Review, 1 February 2003). "Wireless Sensor Networks for Habitat Monitoring" (see the References section) describes an in-depth study of implementing wireless sensor networks for real-world habitat monitoring using TinyOS and the then-current Mica motes.

The first goal of WISENET was to create a new hardware platform to take advantage of newer microcontrollers with greater functionality and more features. This involved selecting the hardware, designing the motes, and porting TinyOS. Once the platform was completed and TinyOS was ported to it, the next stage was to use this platform to create a small-scale system of wireless networked sensors. The purpose of this system was to monitor environmental conditions in the labs and offices in the ECE department at Bradley University.

System Description

The overall system block diagram is shown in figure 2. There were two primary subsystems (Data Analysis and Data Acquisition) comprised of three major components (Client, Server, Sensor Mote Network).

Primary Subsystems

There were two top-level subsystems – Data Analysis and Data Acquisition.

Data Analysis: This subsystem was software-only (relative to WISENET). It relied on existing Internet and web (HTTP) infrastructure to provide communications between the Client and Server components. The focus of this subsystem was to selectively present the collected environmental data to the end user in a graphical manner.

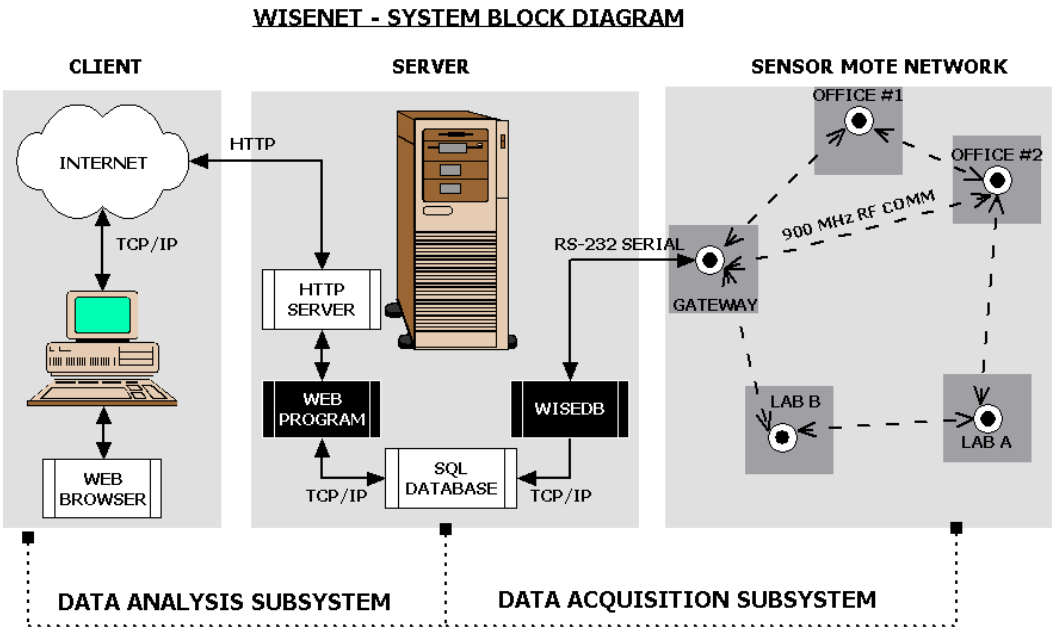

Figure 1: WISENET System Block Diagram

Data Acquisition: The purpose of this subsystem was to collect and store environmental data for later processing by the Data Analysis subsystem. This was a mix of both PC & embedded system software, as well as embedded system hardware. It was composed of both the Server and Sensor Mote Network components.

System Components

There were three primary system components: Client, Server, and Sensor Mote Network. Each is described below, complete with an input/output block diagram.

Client (Figure 2): The Client component was necessary but external to the development of WISENET. That is, any computer with a web browser and Internet access could be a Client. It served only as a user interface to the Data Analysis subsystem.

Server (Figures 3 and 4): The Server was a critical component as the link between the Data Acquisition and Data Analysis subsystems. On the Data Analysis side was an web (HTTP) server hosting a web application. When a page request came in, the web server executed the web application, which retrieved data from the database, processed it, and returned a web-page which the web server transmitted to the Client. For the Data Acquisition system there was a daemon (WiseDB) running to facilitate communication with the Sensor Mote Network. This daemon was responsible for collecting raw data packets from the Sensor Mote Network. These packets were then processed to

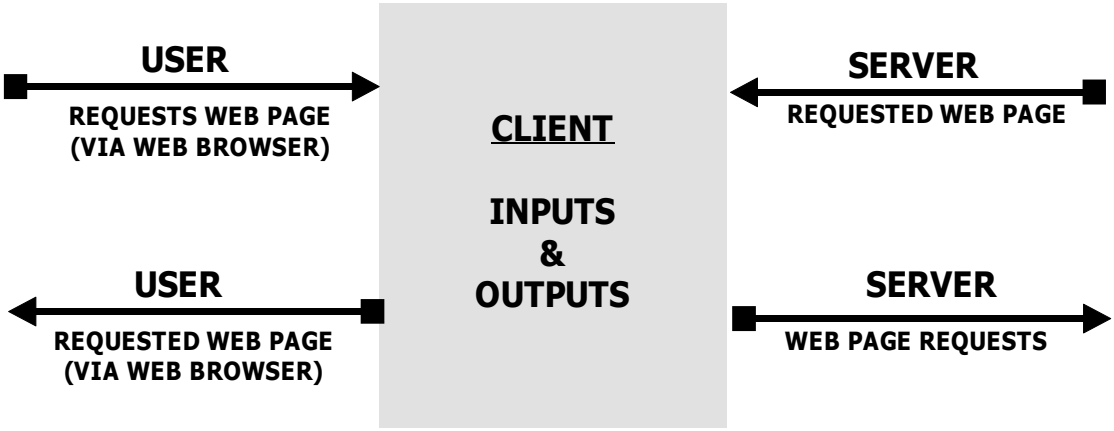

Figure 2: Client Component Inputs / Outputs

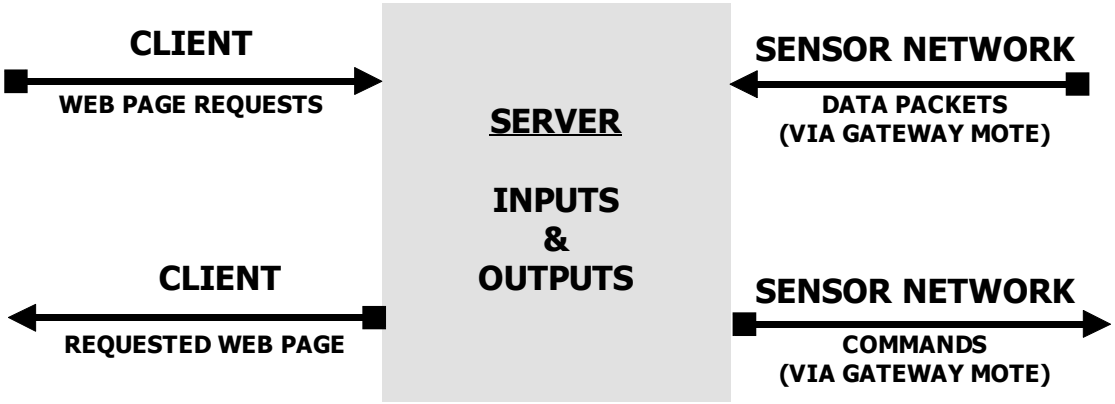

Figure 3: Server Component Inputs / Outputs

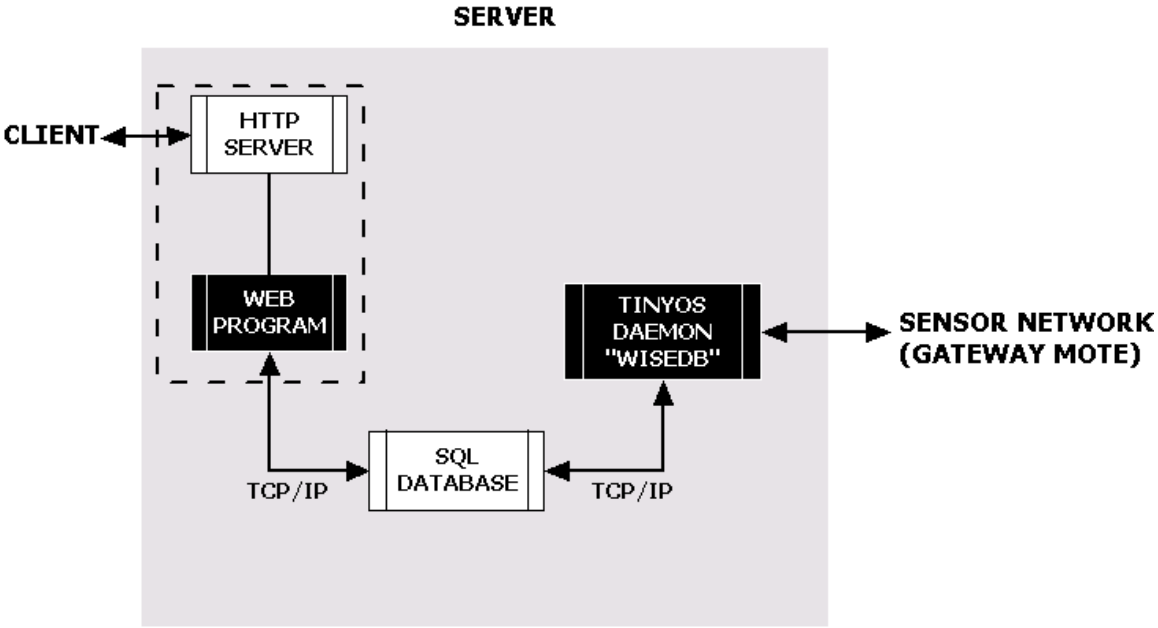

Figure 4: Server Component Block Diagram

convert the raw data into meaningful environmental data. This processed data was then inserted into the database. Thus the database was the link between the Data Analysis and Data Acquisition subsystems. The Server also had the potential to send commands to the Sensor Mote Network (via the gateway mote), although this functionality was not explored in WISENET.

It should be noted that since the SQL database connections can be made via TCP/IP, only the web server and web-program (see figure 4) needed to be located on the same physical machine. The web server, the database, and WiseDB could all be on different physical machines connected via a LAN or the Internet. This allowed a flexible Server component implementation that was useful during WISENET development.

Sensor Motes (figures 5 and 6): The primary focus of WISENET was the development of the Sensor Mote Network component. It was the component responsible for collecting and transmitting raw environmental data to the Server. There was also the potential for the motes to receive commands from the Server, although that functionality was not implemented in WISENET. Uses for this feature would include server-based synchronization and wireless network reprogramming.

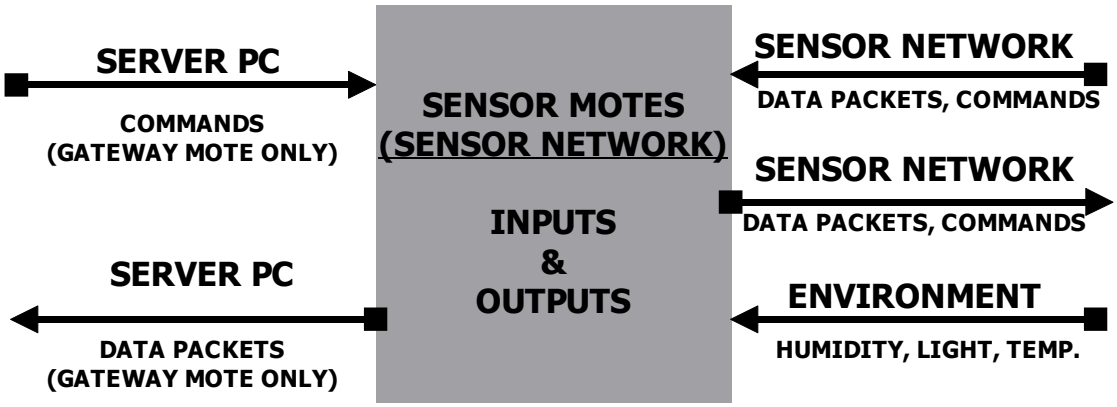

Figure 5: Sensor Mote Component Inputs / Outputs

This component consisted of two parts. The first was the sensor mote. The primary purpose of the sensor mote was to collect and transmit raw environmental data. When not doing this, it went into a low-power idle mode to conserve energy. Another aspect of the sensor motes involved ad-hoc networking and multi-hop routing; however, due to limited resources WISENET was not able to test these functions. See Fgure 6 for the block diagram of a standard sensor mote. Specific hardware details will be discussed in the Hardware Design section.

The gateway mote was the second part of the Sensor Mote Network. Its purpose was to serve as the liaison between the Server and the Sensor Mote Network and deliver all the data packets to WiseDB. In theory both standard and gateway motes could be implemented on the same hardware PCB and

with the same software. For WISENET, however, resource and time constraints necessitated the use of slightly different hardware and software configurations for gateway versus standard motes, as is described below.

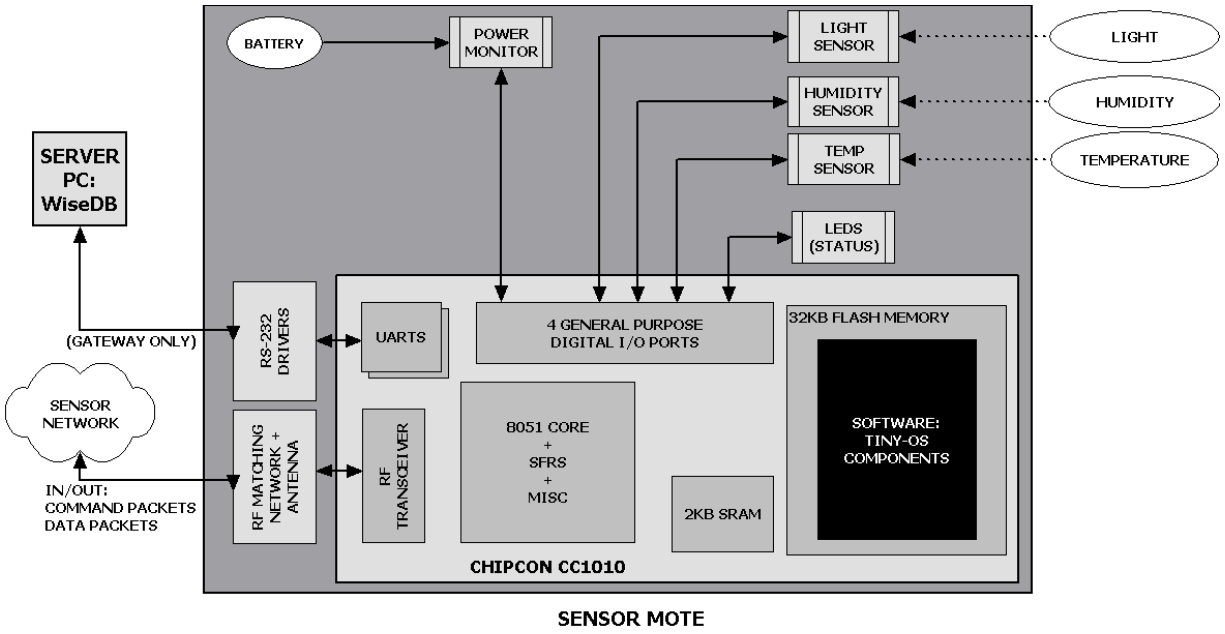

Figure 6 : Sensor Mote Block Diagram

Hardware Design

The selection of components for the sensor motes was a critical process in the early development of WISENET. Great functionality and low power were two of the highest priorities in evaluating the fitness of both the microcontroller and the sensor candidates. Thanks to Honeywell, Inc. WISENET was introduced to the new state-of-the-art Chipcon CC1010 microcontroller with integrated RF transceiver. After a little research it was decided the CC1010 would make the perfect microcontroller. It had the following feature list:

1. Optimized 8051-core
2. Active (14.8 mA), Idle (29 μA) and sleep (0.2 μA) power modes
3. 32 kB flash memory
4. 2 kB +128 bytes SRAM
5. Three channel 10-bit ADC
6. Four timers / Two PWMs
7. Hardware DES encryption/decryption
8. Hardware random bit-generator
9. Fully integrated UHF RF transceiver (433 MHz / 868 MHz nominal)

- Programmable output power (-20 to 10 dBm)
- Low current consumption (11.9 mA for RX, 17.0 mA for TX at 0dBm)
- RSSI output that can be sampled by the on-chip ADC

Honeywell generously provided a development kit for WISENET which included a socketed evaluation board (CC1010EB) and two evaluation modules (CC1010EM). The evaluation board provided access to all of the analog and digital pins on the CC1010, as well as two serial ports, a parallel programming port, RF network analysis ports, and other peripherals. Each evaluation module featured the CC1010, RF network hardware, an antenna port, and an analog temperature sensor. The modules connected to the evaluation board via two TFM-D sockets. These sockets also allowed the possibility of designing a custom expansion board.

Choosing the sensors involved additional research. WISENET was designed to measure light, temperature, and humidity. There were many digital temperature sensors available, but there was a much smaller selection of digital humidity and light sensors. A larger selection of analog sensors was available; however, analog sensors tended to require more power and be less precise than their digital counterparts, in addition to requiring more complex circuitry. For these reasons, digital sensors were given higher priority. Two new sensors provided the required functionality. First, Sensirion released the SHT11, a digital temperature and humidity sensor with ultra low power consumption (550 μA while measuring, 1 μA when in sleep mode), a 14 bit analog to digital converter, and the desired accuracy (±5% relative humidity, ±3ºC). It also featured a simple serial interface. The light sensor chosen was the Texas Advanced Optoelectonic Solutions (TAOS) TSL2550 ambient light sensor with SMBus interface. This sensor also featured ultra-low power (600 μA active, 10 μA power down), a 12-bit analog to digital converter, and dual photo diodes. The TSL2550 uses both photo diodes to compensate for infrared light and to produce a measurement that approximates the human eye response. Interestingly, both of these sensors were also used on a Mica mote sensor board design by the TinyOS development team.

The final stage of hardware design involved creating the add-on module. The final WISENET Add-On Module is shown with attached CC1010EM in figure 7. The schematic, shown in figure 15 in Appendix II, was created using Orcad Capture CIS. The PCB layout was done using Orcad Layout Plus, a program that is part of the standard Orcad package. Figures 16 and 17 in Appendix II show the top and bottom PCB silkscreens, respectively. The netlist was imported into Layout Plus and then the traces were routed using that software. A PCB layout tutorial using Orcad is available online: http://www.seas.gwu.edu/~ecelabs/appnotes/PDF/PCB.pdf

The Add-On Module PCB is 2 layers and has a 50 pin TFM-D connector which plugs into the CC1010 evaluation module. The WISENET Add-On Module has the two digital sensors described above. The Sensirion SHT-11 humidity and temperature sensor has a 2-wire proprietary serial interface. The TAOS TSL2550 digital light sensor uses an SMBus serial interface. SMBus is a standardized 2-wire serial interface. The layout was carefully designed such that the light, temperature and humidity sensors were not underneath the evaluation module when it was plugged into the board, which would make them useless. The board included a DC-DC converter and battery monitor from Maxim-IC called the Max1676. An RS-232 to CMOS level converter chip, the Max3221, was added to enable a sensor mote to function as a gateway mote. A jumper was included so that the Max3221 could be enabled or disabled to save power if a particular mote was not configured to use the serial port. Three status LED's were placed on the board for troubleshooting purposes. Due to resource and time constraints, through-hole components were used for their availability and ease of wiring.

Figure 7: CC1010 Evaluation Module with WISENET Add-On Module

Software Components – Commercial Off The Shelf Products

The server used for WISENET had four commercial off the shelf applications installed on it that worked together to create the Data Analysis portion of the Server component. Apache, MySQL, and PHP are open-source products freely available on the Internet. In addition, Chart-Director the trial version of the commercial application Chart-Director was used.

- Apache is a standard web-server which makes a web document available on the Internet. The Apache web-server is available at: http://www.apache.org/
- PHP is a web programming language which allows dynamic web-pages. It is also designed to be used with a database and included many built-in functions for interfacing with MySQL. The PHP hypertext processor is available at: http://www.php.net/
- MySQL is a database that can contain any type of data and is accessed by a TCP/IP (Internet) call. The MySQL database server is available at: http://www.mysql.com/

- Chart-Director is a program that generates a graph from raw data. It is available in many languages such as PHP, ASP, C++, and others. Chart-Director is available for download from: http://www.advsofteng.com/index.html

Instructions on how to set up a web-server with Apache, PHP, and MySQL were available at: http://Internetmaster.com/installtutorial/index.htm The server for WISENET was set up very similarly to the setup described in the tutorial described above. In addition, it used the Chart-Director software. The documentation on how to install Chart-Director is available under the documentation folder inside the zip file available at this location (choose the PHP version for Windows): http://www.advsofteng.com/download.html

Software Components - Custom

WISENET was also composed of three custom software components: the web program, WiseDB, and a port of TinyOS. Each of these will be discussed in more detail.

WISENET's web program was written in PHP and utilized the Chart-Director charting software. The web application queried MySQL database for the data in the requested date range, then used Chart-Director to generate a graph of that data. A screen-shot of the form interface is shown in Figure 8. The interface was self-explanatory. The 'Moteid' field allowed the data from a single mote to be displayed, or checking the box allowed data from all the motes to be displayed on the graph. Pushing the 'Generate Graphs' button showed three graphs on the page (or a text message if there was no data

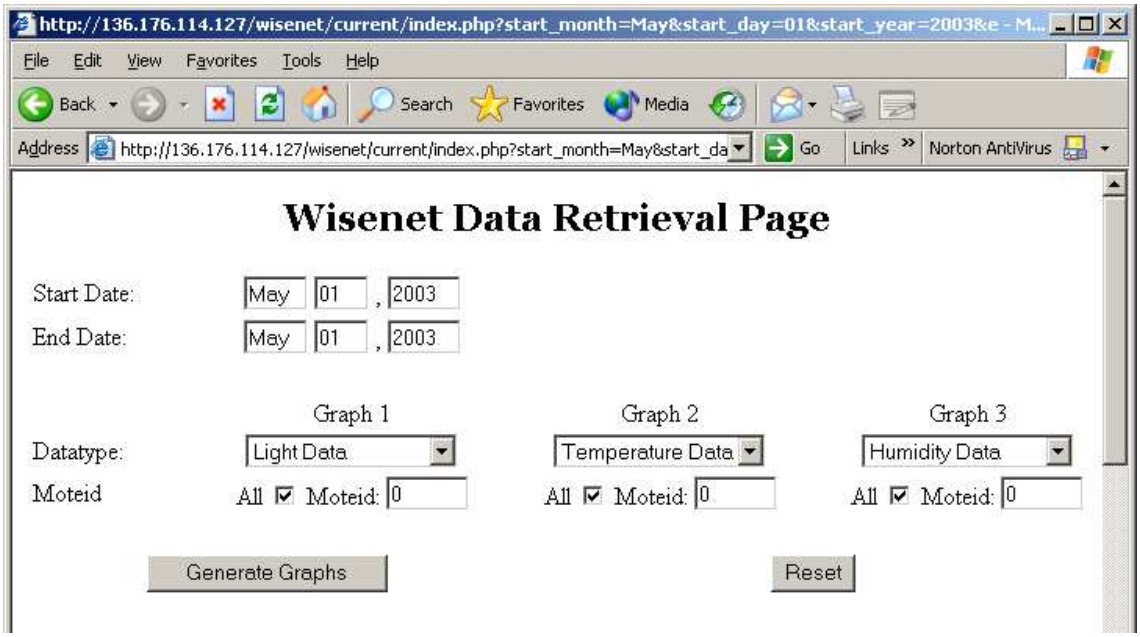

Figure 8: WISENET's Web-Based Data Retrieval Page

for that date range). Figure 9 shows a screen-shot of a sample graph of temperature data obtained on May 11, 2003. The data was retrieved using built-in functions of PHP. PHP was developed to have tight integration with MySQL, and thus interfacing the two is very simple. An example line of code from PHP is as follows:

```
$datatypedesc = mysql_query( "SELECT `description` FROM data_types
WHERE `data_type` = $data_type ");
```

The above statement gets the text field contained in the **description** field that is linked to the **data_type** field with the quantity equal to a local PHP variable called **$data_type**. With the exception of the PHP variables (which makes it easier to use), everything contained inside the **mysql_query (“”)** is SQL standard syntax, and given that the SQL syntax is known, interfacing with the database is very simple.

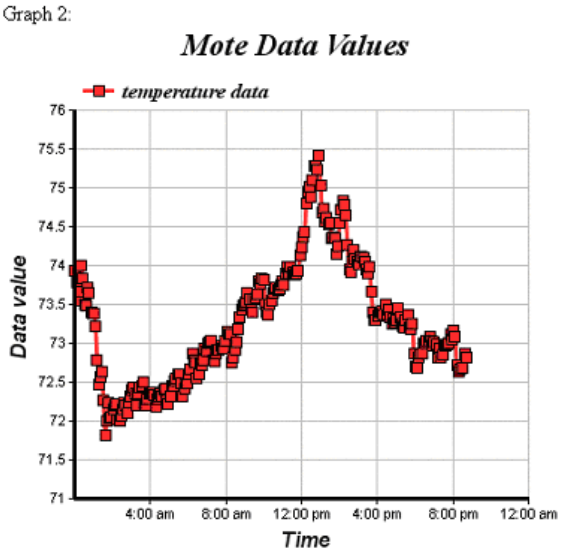

Figure 9: Temperature Graph Generated by WISENET's Web Interface

WiseDB is the custom software component that interfaced with the Sensor Mote Network via a serial link to the gateway mote and with the MySQL database via a TCP/IP link to the MySQL server application. See Figure 3 for more information about how WiseDB interacted with the rest of the system. WiseDB was written in C++ and utilized two open-source API’s (application programming interface). The first API used was MySQL++ for MSVC++. The source for this API was available from MySQL's website. See MYSQL++ 1.71 for Microsoft Visual C++ 6.0 located at: http://www.mysql.com/downloads/api-mysql++.html

The MySQL++ API allowed for a connection to the MySQL database with reasonably simple code. C++ was not as easy to use as PHP, so the code to perform a query is a little less elegant, but still not too complex:

```
query << "select data_type from data_types where description = " <<
      description;
Result res=query.store();
Result::iterator i;
i=res.begin();
```

```
Row row = *i;
data_type = row["data_type"];
```

The query shown here gets the integer field contained in the **data_type** field that is linked to the **description** field with the text matching a local string variable called description. Everything contained inside the **mysql_query ("")** is SQL standard syntax, and given that the SQL syntax is known, interfacing with the database is somewhat simple. As shown, once the query is performed by the line: `Result res=query.store();` an iterator must be used to parse the data stored in the result data block.

The second API used in WiseDB was a C++ serial API written by Ramon Klein. This API allowed a very simple interface to the serial port. An example of code used in WiseDB is shown:

```
serial.Read(&header.packettype, sizeof(unsigned char));
```

The **packettype** variable is passed by reference from the Read function. The above line of code is used to read a character from the serial port. (The `sizeof(unsigned char)` is used to represent the length of a byte.) The write operation was just as simple to use. Clearly this API made it easy to use serial communication in a C++ program. The main flow of WiseDB was to wait until a packet was received and then calculate light, temperature, and humidity values from the raw data. These values were then inserted into the MySQL database.

The final custom software component involved porting TinyOS to the CC1010-based hardware platform described in the Hardware Design section. As previously mentioned, TinyOS was a real-time operating system designed for use in sensor network applications where low-power, limited resources and hard real-time constraints are critical parameters. See J. Hill's Master Thesis (listed in the References section) for more information about the theory and design of TinyOS. The newest version of TinyOS (1.x) has been re-written using nesC, which is an extension to the C language that is designed based on the structure and core concepts behind TinyOS. For more information about nesC see **The *nesC* Language: A Holistic Approach to Network Embedded Systems** in the References section.

The first step to porting TinyOS to the CC1010-based platform was to determine how an application is created and compiled. nesC and the TinyOS build process were based on a Linux development environment, which was emulated under Microsoft Windows using Cygwin (see http://www.cygwin.com/). The nesC compiler preprocesses the TinyOS source code files into ANSI C, which is then fed to an actual compiler for cross-compiling and linking for the AVR microcontroller. WISENET development needed to be done under Cygwin running on Microsoft

Windows 2000/XP, with final compiling and linking performed by commercial Keil Software tools (http://www.keil.com/) augmented with the Chipcon CC1010IDE (Integrated Development Environment, see http://www.chipcon.com/index.cfm?kat_id=2&subkat_id=12&dok_id=55). There were some fundamental incompatibilities between these two processes, which meant some of the TinyOS and nesC tools had to be modified, and other new tools created to bridge the gap. A flowchart of the build process is shown in Figure 10. In brief, the C language file was produced by **nesc1.exe.** The script **nesc-compile** was modified to pass source code to the custom post-processor script **keil_ppp**. This script resolved any incompatibilities between the source code and the Keil C51 compiler syntax and saved the modified code, ready for inclusion into a new Keil project file. An improvement to this process would be the automatic creation of the project file, based upon a user-supplied template.

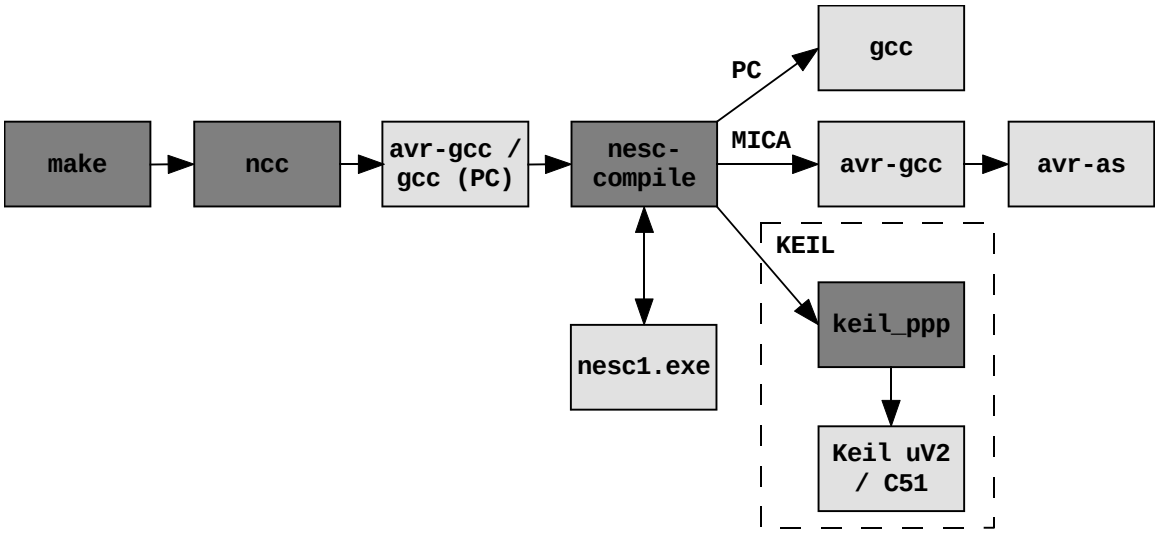

Figure 10: TinyOS Build Process Flowchart

After modifying the build process, porting of TinyOS commenced. A new subdirectory off the **tinyos-1.x/contrib/** path was created to contain the development files. Thanks to the highly modular architecture of TinyOS, all of the hardware functionality was abstracted in components. Components were then 'wired' together to form more complex components and complete applications. See Figure 11 for a diagram of this hierarchical approach. One important goal of WISENET was to completely replace the lower-layer functionality (the layers enclosed in the dashed-line box in Figure 11) to permit existing higher-level components and applications to be immediately implemented on the new hardware platform without modification. Existing low-level code for the Mica mote was used as a template for the CC1010 code where applicable.

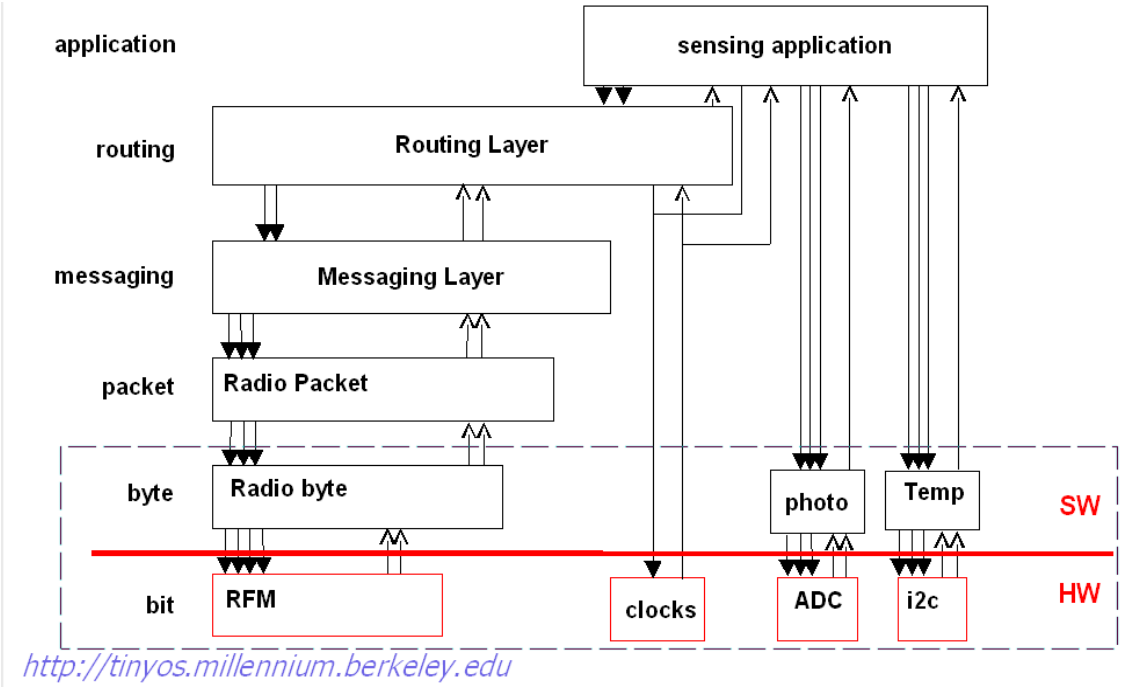

Figure 11: TinyOS Layers

One significant component not implemented on the CC1010 was the default radio communications (RFM) stack. Instead a newer and more flexible radio communications stack was used. The S-MAC, a medium access control (MAC) protocol designed for sensor networks, featured reduced energy consumption as well as improved scalability and collision avoidance. See **An Energy-Efficient MAC Protocol for Wireless Sensor Networks** (listed in References) for more information. The S-MAC code also featured a compatibility component that allowed applications based on the S-MAC protocol to use the Active Messaging components in TinyOS, ensuring core compatibility with all of the communications components.

The final application that was developed for WISENET was called "SensorSleepAppSMAC." Its flowchart is shown in Figure 12. It incorporated nearly all of the major components, such as sensor reading, radio transmission (via S-MAC protocol), and power management. An expansive diagram of

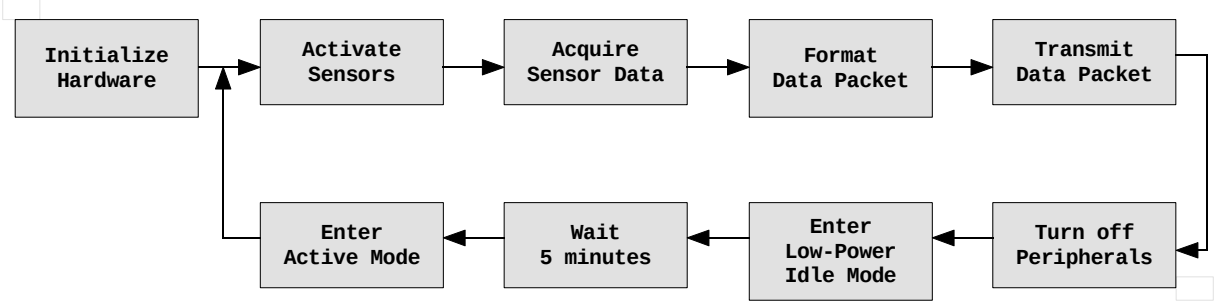

Figure 12: SensorSleepAppSMAC Flowchart

every component used in the application is shown in Appendix III. This diagram also indicates approximately to which TinyOS layer (see Figure 11) each component corresponds. Due to the lack of motes (only two were available, and one of those was used as the gateway mote), advanced network features such as multi-hop packet routing and ad-hoc network detection were not implemented. These functions are already available in TinyOS in the higher layers, so future implementation should not be exceptionally difficult.

Simulations

Simulation of the entire WISENET system was not possible; however simulations were used on certain components to ease development. The serial component was one part of the system that was simulated to allow development and testing before integration with the rest of the system. It was tested using a loop-back serial connection on the server. GWsim was a program written to simulate a gateway mote receiving a data packet from the network and forwarding it to the serial port. WiseDB was tested and debugged using this loop-back connection. This simulation helped immensely in the final hours before WISENET was operational for the first time. Only minimal additional debugging was necessary when the Sensor Mote Network finally became active.

Results

WISENET was a completely operational system. It measured light, humidity, and temperature data from a single mote with the WISENET Add-on Module. A gateway mote was set up using the CC1010 evaluation module and evaluation board. WiseDB received the data packets from the gateway mote and converted the raw sensor data to lux (for light), percentage (for relative humidity), and degrees-Fahrenheit (for temperature). The converted values were inserted into the MySQL database with a time-stamp and moteid to identify from which mote the data came (which, in this particular implementation, was trivial since there only was one mote). The web interface allowed the user to look at all three sets of data (light, temperature, and humidity) within a specified date range. A graph was then generated for each data type based on the data within the given range.

As previously mentioned, the TinyOS application written for WISENET (SensorAppSleepSMAC) incorporated every major component (except advanced network-layer functionality). See Appendix III for a component diagram of the application. The application worked as expected; however, there are many code optimizations that could be made to increase performance, reduce code size, and reduce energy consumption. In addition, more aggressive power-saving could be implemented in the application.

There are some aspects of WISENET that were not operational. The battery-power regulator on the Add-On Module did not produce the required 3.3VDC. An external 3.3VDC power supply was used to work around this problem. The problem appeared to be an inductor that was used in the DC-DC converter circuit; it did not meet the specification that the MAX1676 reference circuit required. The inductor used had a maximum current rating that was $1/10^{th}$ of the required switching current specification of 1 amp. Further testing is required.

Another part of the WISENET Add-on Module that was not functional was the Max3221 RS-232 to CMOS converter IC. The IC received the correct inputs but did not provide the appropriate outputs. It was believed that this chip was damaged by the high current that was generated by the Max1676 with the improper inductor. More testing must be done on the circuit to confirm this.

Future Work

There are a number of avenues for future extension of this project.

1. Expand the sensor mote network by adding more motes. This would allow the development and testing of advanced network-layer functions, such as multi-hop routing.
2. Create a new PCB design that integrates the CC1010EM design with the sensors and power hardware on a single-board. Another interesting feature would be to develop or adopt a standard expandable plug-in sensor interface in both hardware and software
3. Research alternative energy sources to extend mote battery life. Possibilities include solar cells and rechargeable batteries.
4. Continue the development of TinyOS in general and 8051-based systems in particular. This could include improving the tools or optimizing component code (especially Chipcon-supplied library code).
5. Improve web interface – this could take many facets, from simply increasing the flexibility of the charting options to making it more graphical (e.g. a map of motes and physical locations) to performing complete data analysis (highs/lows, day/week/month/yearly averages, etc.)

Conclusions

Wireless sensor networks are getting smaller and faster, increasing their potential applications in commercial, industrial, and residential environments. WISENET, as implemented, represents one commercial application (office environment monitoring). However, the limit of applications depends only upon the sensors used and the interpretation of the data obtained. As the technology improves

and new low-power digital sensors become more readily available, motes will increase functionality without increasing power consumption and will expand the wireless sensing market.

There are many strengths upon WISENET draws – first and foremost, the open-source development community. Nearly every component of the project used open-source software and tools (exceptions include Keil µVision2, the Orcad package, and Microsoft Visual Studio). The open nature of the software encourages developer participation and communication. This was indispensable during work on TinyOS and nesC – the developers were able to provide additional information and ideas for circumventing obstacles that halted WISENET's development

WISENET also draws on the strength of TinyOS's design methodology. The modular structure of TinyOS allows rapid application development once the low-layer components are developed. This component-based approach is perfect for developing custom applications quickly and easily. In addition, the abstracted nature of the components allows new features, sensors or platforms to be developed and implemented with relative ease.

There are strengths on the server side, as well. The TCP/IP-based connections between server components allow a distributed server model. This allows an outdated computer to be recommissioned as the WiseDB server, while an existing web and database server can be used for the web program and database, respectively. Or a single dedicated server can be used. Flexible implementation is an important consideration for security and scalability purposes.

WISENET and its success demonstrate some of the power of wireless sensor network technology. This is a field that will see tremendous growth in the near future as microcontrollers and sensors improve their performance while lowering their energy consumption. TinyOS is an operating system designed to meet the needs of wireless sensing applications; its modular design methodology and open-source development are keys to its success. Finally, the emphasis on web-based interfaces allows complex graphical user interfaces to be rapidly designed and tested on a variety of platforms, simplifying the end-user experience.

Appendix I: Data Sheet

After careful consideration, the following specifications were chosen:

- Environmental Conditions
  - -20 – 80º C
  - 0 – 100% Relative Humidity (RH)
- Sensor Accuracy
  - Light Sensor: ±10% Lux
  - Humidity: ±5% RH
  - Temperature: ±3º C
- Power
  - 3.3V Operating Voltage
  - Battery-powered with 2xAA batteries
  - Goal: 6 Month Battery Life with a 5 minute update interval
- RF
  - Operating Range: >50 feet indoors
  - Nominal Frequency: 868MHz
  - Manchester Encoding
- Web-based graphical user interface
- Compatibility with TinyOS applications

Appendix II: WISENET Add-On Module

Wisenet Addon Module

General Description

This Wisenet Addon Module is an expansion to the CC1010 Evaluation Module from Chipcon. It is designed for environmental monitoring applications. The addon module was developed to be used with the Wisenet software.

Board connections

The Wisenet Addon Module connects to the CC1010 Evaluation module through the P2 connector on the board. The pin descriptions of this connector are listed in Fig 1

Features

The Expansion board features 4 major components. One component is a voltage regulator with low battery detection provided by a chip from Maixm, the Max1676. Another is a CMOS to RS-232 converter chip with autoshutdown. There are also 2 sensor modules. One monitors temperature and humidity, the Sensirion SHT11. And the other, the TAOS TSL 2550, monitors light.

Pin#	CC1010 Pin	Pin Description	Input / Output	Addon Board description
1	RSSI_AD2	RSSI / ADC CH2	INPUT	NC
3	AD1	ADC CH1	INPUT	NC
5	AD0	ADC CH0	INPUT	TEST_POINT
7	TEST_N	SET THIS TO VDD	INPUT	VDD
9	RESET_N	RESET THE CC1010	INPUT	NC
11	PROG_N	FLASH PROGRAM	INPUT	NC
13	P2.7	General IO	I/O	INVALID' of Max3221 (RS-232)
15	P2.6	General IO	I/O	LBO' of Max1676 (low batt)
17	P1.7	General IO	I/O	S_CK of SHT11 (serial clock)
19	P1.6	General IO	I/O	S_DATA of SHT11 (serial data)
21	P1.5	General IO	I/O	SMB_DATA of TSL2550
23	P0.3	General IO	I/O	SMB_CLOCK of TSL2550
25	P0.2	MISO	I/O	NC
27	P3.0	RXD0	I/O	RX -> Max3221 (RS-232)
29	P3.1	TXD0	I/O	TX -> Max3221 (RS-232)
31	P3.2	INT0 – level/edge trig.	I/O	Green LED
33	P2.5	General IO	I/O	Yellow LED
35	P2.4	General IO	I/O	Red LED
37	P2.3	General IO	INPUT	Enable Serial (EN' of Max3221) active low
39	3.3V_RF	VDD	Power	VDD (regulated from Max1676)

Fig 1 – Pin descriptions of TFM-D connector.

Figure 13: WISENET Add-On Module Datasheet, Pg 1

Jumpers

There is a jumper on the board, JP1. JP1 has two purposes, one is to indicate to the microcontroller that the serial converter chip, the Max 3221, is enabled though pin P2.3. This pin is an input and is used only to monitor the status of the jumper.

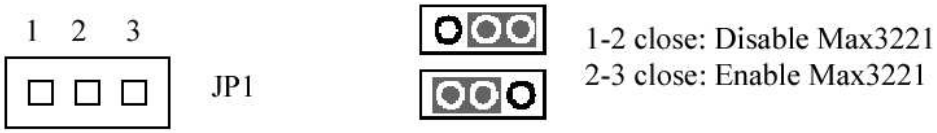

Fig 2 – Jumper JP1 description

The enable pin of the Max3221 is active low, so position 2-3, connects pin 2 to ground and grounds P2.3 of the microcontroller thus indicating that the Max3221 is disabled.

Test pins

Several test pins are available on the addon module. They are listed in the Fig 3.

Test pin #	Name	Purpose
TP1	Vin	Voltage in connection Min = 1.1V, Max = 5.5V
TP2	Gnd	Ground connection
TP3	TX	RS-232 TX pin for serial communication
TP4	RX	RS-232 RX pin for serial communication
TP5	ADC0	Analog to digital converter input

Fig 3 – Test pins available

Figure 14: WISENET Add-On Module Datasheet, Pg 2

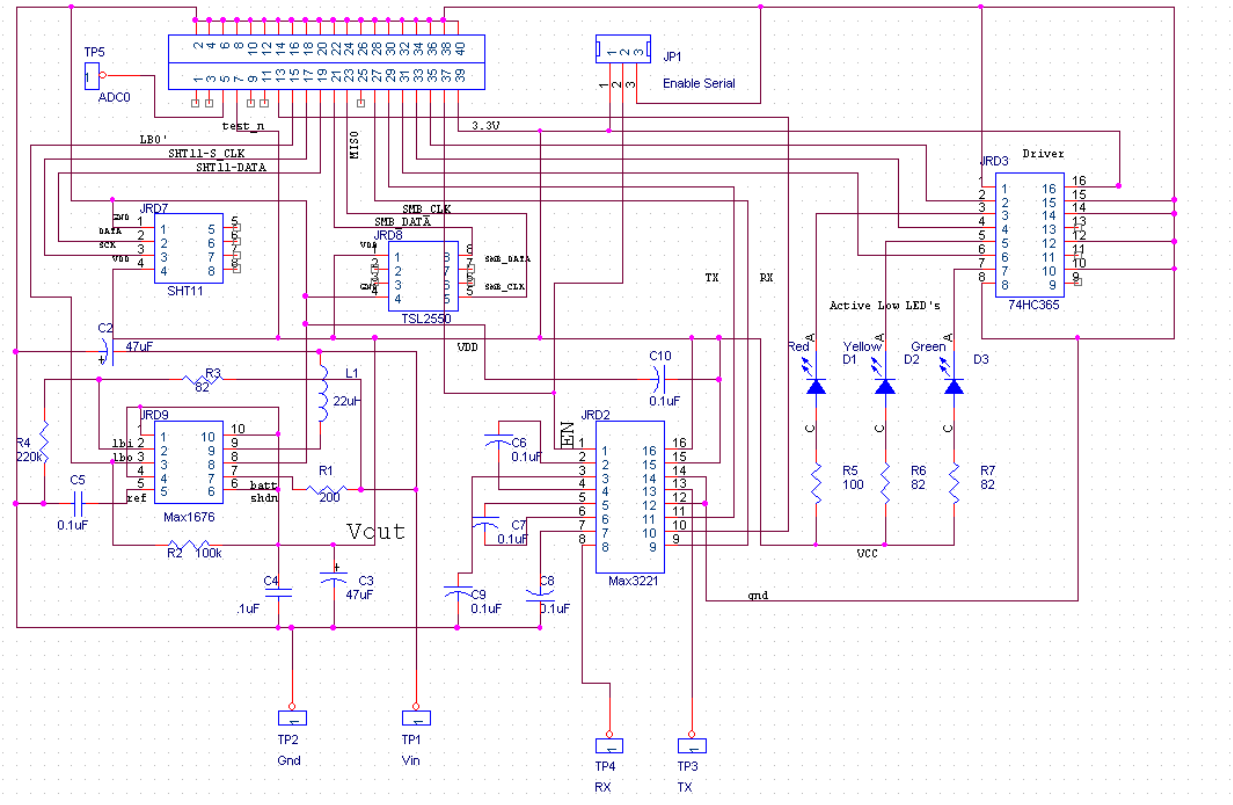

Figure 15: WISENET Add-On Module Schematic

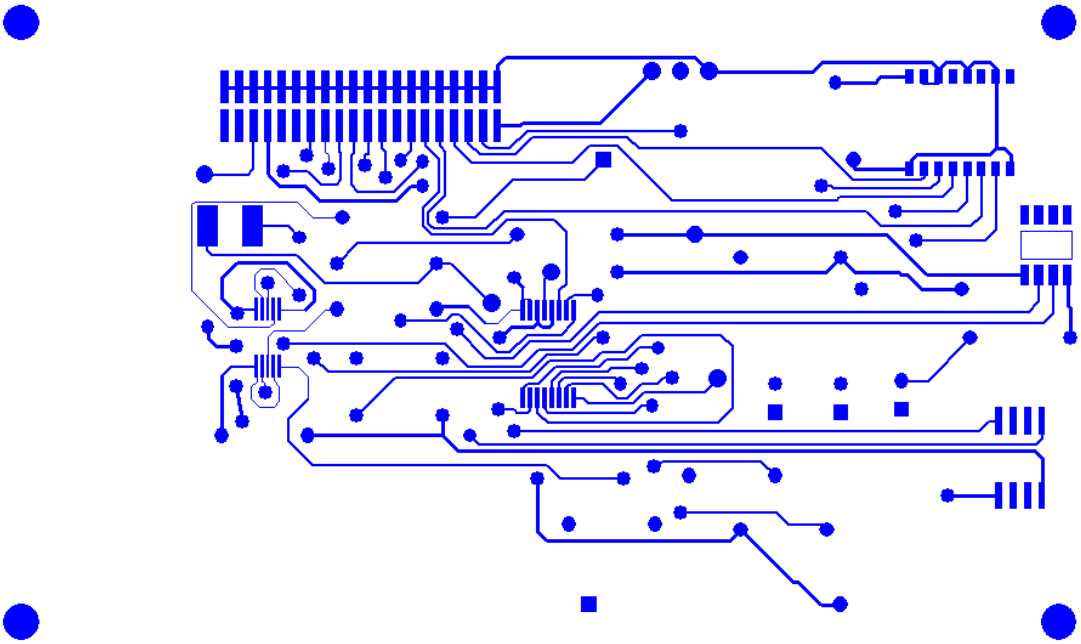

Figure 16: WISENET Add-On Module PCB Layout – Top Layer

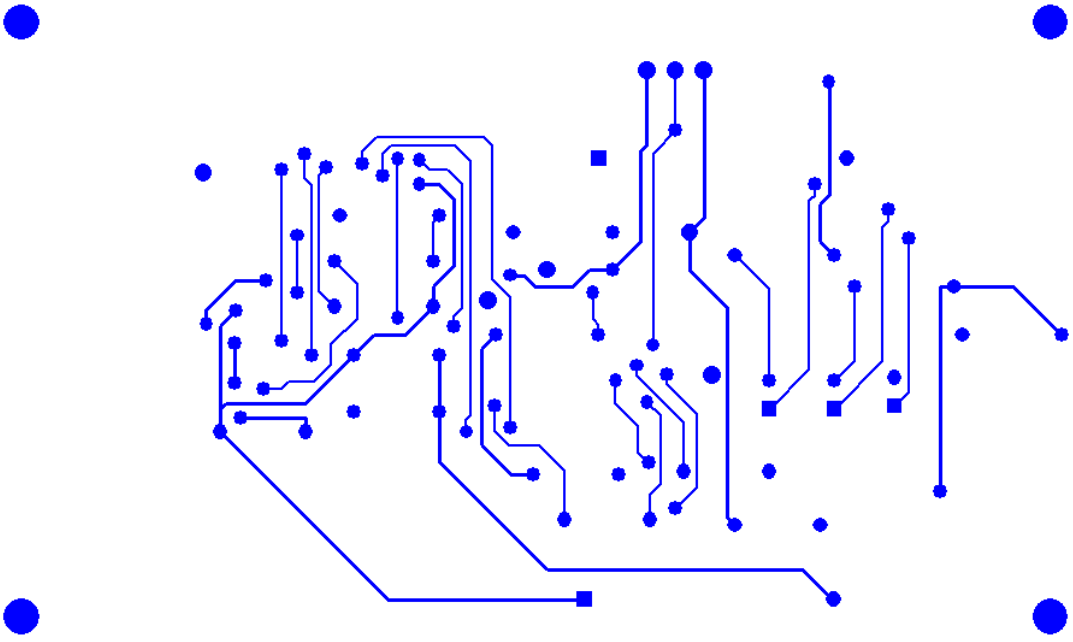

Figure 17: WISENET Add-On Module PCB Layout – Bottom Layer

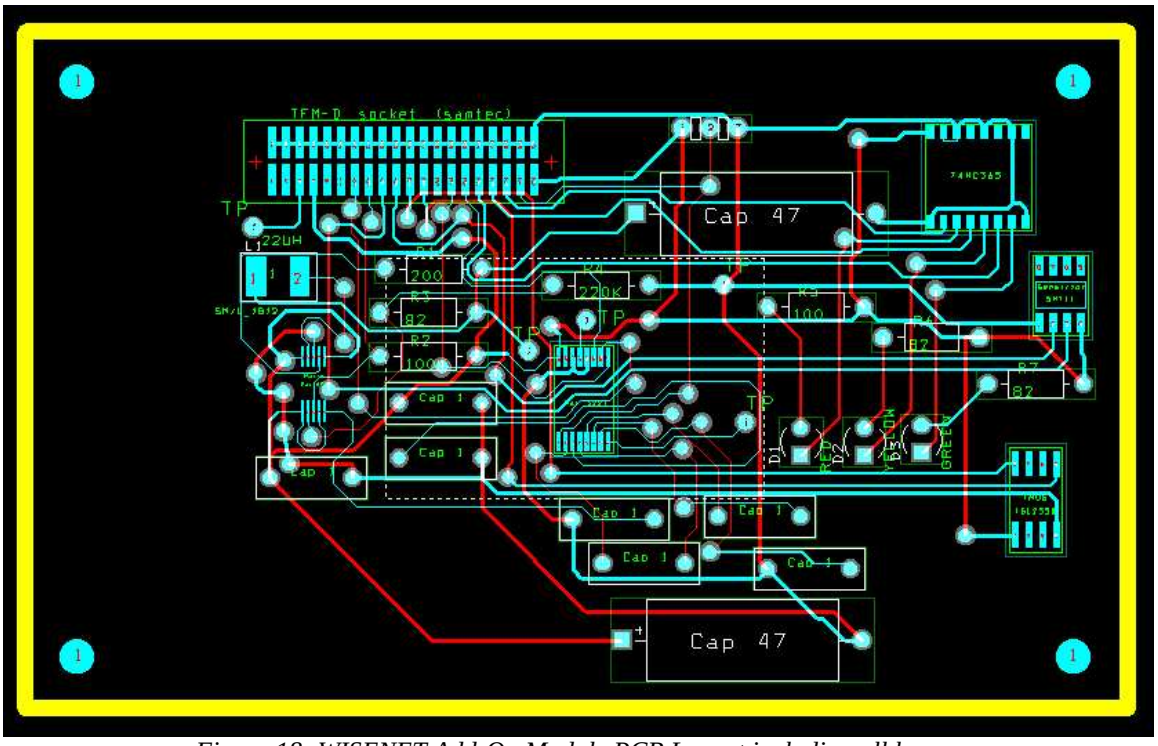

Figure 18: WISENET Add-On Module PCB Layout including all layers

Appendix III: SensorSleepAppSMAC

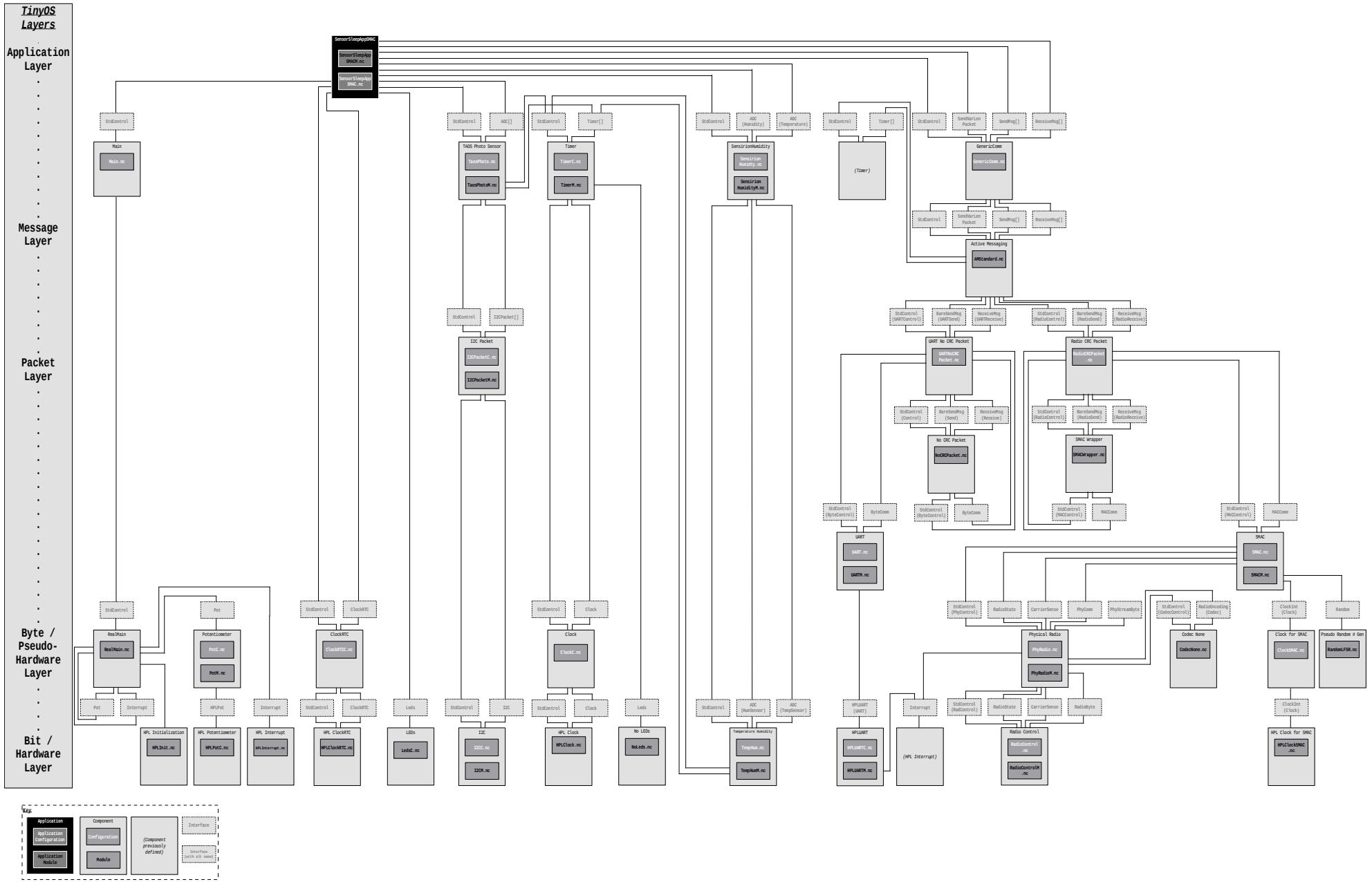

Figure 19: Full Component Diagram - SensorSleepAppSMAC Application (See attached full-size diagram)

Appendix IV: Applicable Patents

There are a number of patents that are relevant to this project involving wireless sensor networks. Some of them are:

- "Reprogrammable remote sensor monitoring system (5,959,529)
- "Wireless integrated sensor network using multiple relayed communications (6,208,247)
- "Early warning detection and notification network for environmental condition (6,023,223)
- "Modular architecture sensing and computing platform (6,402,031)
- "Distributed topology learning method and apparatus for wireless networks (6,414,955)

Appendix V: Applicable Standards

WISENET takes advantage of a number of different standards, such as:

- TCP/IP Protocol
- Hypertext Transfer Protocol (HTTP)
- Structured Query Language (SQL)
- Serial Link (RS-232)
- Micro-Controller Serial Link (SMBus)
- FCC regulations (unlicensed ISM Bands)

References

The following sources were used in the research and design of this project.

1. Atkinson. **MySQL++: A C++ API for MySQL, vers 1.7.9**, *online posting*. 12 Dec 2002. <http://www.mysql.com/downloads/api-mysql++.html>.
2. Gay, Levis, et al. **The *nesC* Language: A Holistic Approach to Network Embedded Systems**, *online posting.* To appear in Proceedings of Programming Language Design and Implementation (PLDI) 2003, June 2003. 13 May 2003. <http://today.cs.berkeley.edu/tos/papers/nesc.pdf>.
3. Hill. **A Software Architecture Supporting Networked Sensors,** *online posting*. Master Thesis, Dec. 2000. 12 Dec 2002. <http://today.cs.berkeley.edu/tos/papers/TinyOS_Masters.pdf>.
4. Hill, Szewczyk, et al. **System architecture directions for network sensors,** *online posting.* Jason ASPLOS 2000. 12 Dec 2002. <http://today.cs.berkeley.edu/tos/papers/tos.pdf>.
5. Klein, Ramon. **Serial Communication Library for C++**, *online posting.* 01 Dec 2002. 12 Dec 2002. <http://home.ict.nl/~ramklein/Projects/Serial.html>.
6. Mainwaring, Polastre, et al. **Wireless Sensor Networks for Habitat Monitoring**, *online posting.* 2002 ACM International Workshop on Wireless Sensor Networks and Applications September 28, 2002. 13 May 2003. <http://www.cs.berkeley.edu/~polastre/papers/wsna02.pdf>.
7. Torvmark, K. H.. **Application Note AN017: Low Power Systems Using the CC1010**, *online posting*. 13 May 2003. <http://www.chipcon.com/files/AN_017_Low_Power_Systems_Using_The_CC1010_1_1.pdf>.
8. Ye, Heidemann, et al. **An Energy-Efficient MAC Protocol for Wireless Sensor Networks**, *online posting*. In Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002). 13 May 2002. <http://www.isi.edu/%7Eweiye/pub/smac_infocom.pdf>