TinyBrainz
CSM3 WSN Project

Client Report

Brandon Ardiente, Steven Arth, Claire DuPont

Field Session

June 21, 2007

Table of Contents

1 - Abstract

There is a gross under-representation of minorities and females in the Information Technology and science, technology, engineering and math (STEM) fields. One approach suggested to remedy this situation is to develop hands-on activities where students can perform their own experiments to aid in the learning process. Information Technology Experiences for Students and Teachers (ITEST)[1] proposes to meet this need by designing and implementing a year-round informal education program through the use of wireless sensor networks (WSNs).

The goals of the ITEST program are as follows:

1. Develop and implement an informal year-round intervention based on WSN hardware, software and communications that is team-based, engaging to students, and related to students' daily environments.

2. Improve student attitudes (e.g., perceptions of, value of, self-confidence in, etc.) regarding learning, school, IT, and STEM.

3. Improve student attitudes towards pursuing a future in IT and/or STEM.

4. Improve student performance in IT and STEM related areas.

Our Role:

We are using a collection of wireless, portable sensor *nodes* that measure environmental and physical parameters, such as temperature and barometric pressure, to create a wireless sensor network. This network is used to design activities that can be performed easily by upper elementary and middle school students during allotted class times and support information being taught in a normal classroom setting.

2 - Introduction

The goal of this project is to create a curriculum with math and/or science related activities for elementary and middle school students with the use of wireless sensor networks (WSNs). These activities should increase the interest of students in math and science and meet Colorado State Science and/or Math Standards. Relevant client application development, wireless sensor network (WSN) hardware, and appropriate documentation should be included. The curriculum should be implemented by a teacher with relative ease with help from a Colorado School of Mines representative familiar with setup and operation of the WSN. It is desired that the education and complexity level of the activities be adjustable for students in grades 3 through 8. We are assuming that cost is not a major issue since schools will not be purchasing the sensor nodes. Rather, grant money will likely fund the project, and Colorado School of Mines will be responsible for deploying the equipment as needed.

3 - Design

We have been given the opportunity to use four MPR400CB radio boards (see Figure 1), three MTS310CA sensor boards (see Figure 1), and one MIB520CB interface board to work with for this project. Since we had four radio boards, one was used as part of the base station and the other three were nodes. Although some of the equipment is slightly older than the motes (the radio board and sensor board are called a mote when put together) we had originally researched, the MTS310CAs have the sensing capabilities that we plan to use for the middle school activities and are capable of using current software. This mote contains a thermistor, 2-axis accelerometer, 2-axis magnetometer, light sensor, buzzer, three LEDs, and microphone. The activities we have chosen will use the thermistor, microphone, 2-axis accelerometer, light sensor, LEDs and buzzer.

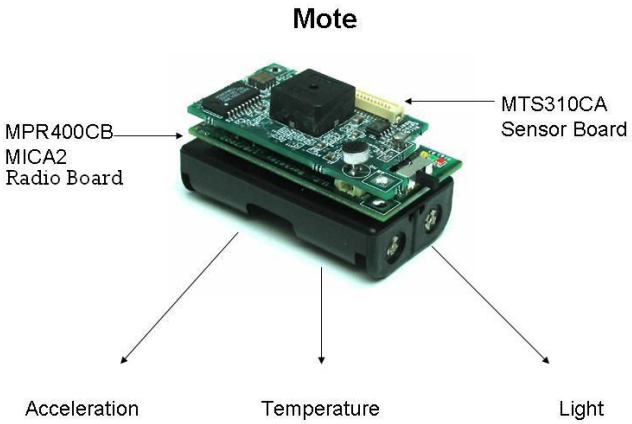


Figure 1 – This is an image of the motes we used in our project. They each consist of a power supply, the MICA2 radio board, and the MTS310CA Sensor Board.

3.1 - Requirements

3.1.1 Functional Specifications

1) Design a system of nodes that are able to relay information between each other. This system needs to be able to transmit data to the base computer. The base will use this information to display data for students to use/interpret for their activity (see Figure 2).

2) Create classroom activities that meet Colorado Math and/or Science Standards.

3) There is the possibility of allowing students to view and slightly modify code to gain a better understanding of how the sensors work.

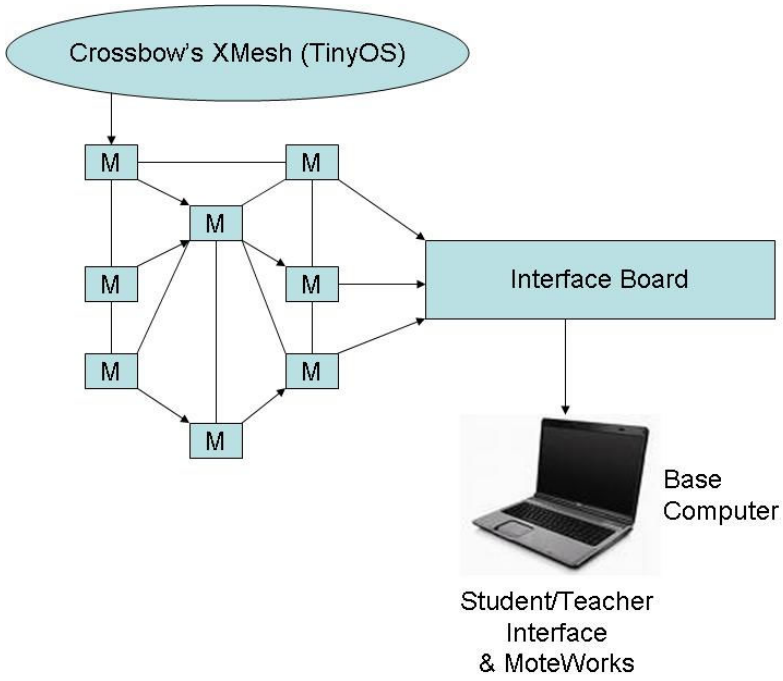


Figure 2 – This is the overall layout of the WSN. Since the motes were manufactured by Crossbow, they come with a modified version of TinyOS called MoteWorks. When in place, the motes relay information off of each other until the data is finally sent to the interface board. The interface board then processes the data and transmits the data to the base computer where we can utilize Crossbow's MoteView software to present the information in a form useful for students and teachers.

3.1.2 Nonfunctional Specifications

1) Since wireless sensors have a lot of versatility, both indoor and outdoor activities can be created for students. These activities can be done both on and off of school premises.

2) The activities will be specifically designed with upper elementary and middle school students in mind. In light of this, the need for programming skills will not be emphasized. However, setup of a WSN will not necessarily be possible for the average teacher. A CSM student/representative will be needed on-site to configure everything and explain the basic operation of the equipment.

3) No specific requirements have been set as far as quality assurance, but in order for this project to be feasible it will need to be reliable and have little to no maintenance. Since it is unlikely a school district will be able to afford a WSN setup, the burden will be placed elsewhere (most likely funded by grants). Once the initial setup has been purchased it may be difficult to secure funds for replacement equipment.

4) Documentation is crucial in this project. Teachers who are considering implementing these projects as part of their curriculum need to fully understand what is involved and how it works. Make all documentation available on the Toilers website for easy reference[2] and in a series of web pages specifically designed for this project.

5) The WSN must be as simple and flexible as possible. Make the completed system capable of adding new nodes of various types that will be compatible with the original hardware. This will reduce the future costs that will be incurred with the system. Should the project commit to one brand of WSN technology, a consequence of this requirement will be following that brand's upgrade and compatibility paths.

6) Since the motes run on nesC (an extension of the C programming language), we will need to be able to program efficiently in this language.

3.2 - Activities

We have designed five activities for students. The first allows students to take a mote home with them and collect temperature data outside of their homes to be analyzed the following day when they return to school (see Appendix A). Another activity measures acceleration of toy cars on a ramp taking into account different factors, such as the height of the ramp, the material covering the ramp, and the mass of the car (see Appendix B). The third activity will require students to bury motes in environmentally-diverse locations and draw conclusions to a temperature-related problem based on the data collected (see Appendix C). The fourth activity will use the acceleration sensor to create a step counter (see Appendix D). Lastly, the sound sensor will be used to demonstrate the changes in decibel levels from various sounds in a classroom or other environments (see Appendix E).

Each of these activities has been specifically designed to utilize the capabilities of the sensing boards while meeting several math and science standards as specified by the state.

3.3 - HTML Pages

We have created a set of basic HTML pages designed for teachers to gain a better understanding of the activities. These pages include questions the teachers will introduce to students, the purpose of the activities, and step-by-step procedures for implementation. The HTML pages do not address the programming aspects of the activities.

3.4 - Wireless Sensor Network

The wireless sensor network formed by our Crossbow MICA2 motes runs on Crossbow's MoteWorks, a modification of the open-source TinyOS. TinyOS is an embedded operating system written in the nesC programming language as a set of cooperating tasks and processes.

MoteWorks itself is a software suite consisting of firmware, programming tools and maintenance utilities that allow us to easily develop applications for and deploy the motes.

The MoteWorks mote firmware uses the XMesh networking library, an extension of the TinyOS MintRoute library. Crossbow claims XMesh achieves 95-99.9% message delivery reliability versus MintRoute's 65-70%[3]. Furthermore, the firmware forms an ad hoc wireless network within seconds, can repair the network when nodes go offline or signal quality decays, and can actively run necessary network operations from each mote running exclusively on battery power. An ad hoc wireless network is a computer network in which the communication links are wireless because each node is willing to forward data for other nodes. XMesh's functionality is entirely out of the box; we will not have to modify any code for the purposes of our project.

The MoteWorks suite comes with a Windows-based development environment for viewing, editing, compiling and downloading code. We will use this environment to extend the functionality of our motes. Currently, our motes require the use of a base station (a PC) in order to obtain the various sensor readings. We have extended the mote firmware to provide the user with audiovisual notifications so that the user may use the mote and draw important information from the mote without having to use the base station software.

The base station software consists of Crossbow's MoteView software. MoteView provides a visual representation of network topology, sensor readings over time, and network link quality. Our activities will use MoteView's existing functionality, and teachers will have exclusive access to the software.

4 - Implementation

4.1 - Hardware Recommendation

We researched several current off the shelf wireless sensor node devices for our project. After careful consideration of ease of development and ease of deployment, we determined that the Moteiv Tmote Invent wireless sensor nodes would be the best hardware to use with our project. Full documentation of our hardware recommendation may be found in Appendix F. However, our clients requested that we use hardware that they already owned, so we changed our design approach to accommodate Crossbow MICA2 motes instead of the Tmote Invent motes.

4.2 - Mote Usability and Firmware

For our activities, our clients suggested different mote usage scenarios. One such scenario involved students being able to obtain readings from their motes without the use of a base station, a PC and client-side software. This scenario required the mote to relay relevant sensor data to a user directly, without the use of the wireless sensor network. As a result, we had to devise a method by which our MICA2 motes, with only 3 LEDs and a buzzer, could accurately relay sensor data such as temperature, acceleration, light intensity, etc.

Another usage scenario involved students being able to directly control when a mote would take a given reading. Our original hardware recommendation, the Moteiv Tmote Invent, came equipped with a programmable user-input button which we had originally intended on programming to allow the user to control when the mote took a sensor reading. However, due to our hardware change to the MICA2 mote, we were not able to utilize any sort of user input in regards to taking a sensor reading. As such, we had to program the motes to take readings at specified intervals, and to devise a method by which the mote users would be able to identify when a reading has been taken and when that reading is being displayed.

Lastly, we had to implement our display design into the existing XMesh networking functionality so that we may still monitor our motes using Crossbow's MoteView software.

4.3 - Sensor Data Display Method

Our original design involved the use of the 3 LEDs as place value markers and the buzzer as a counter. The yellow LED served as the hundreds place, the green LED served as the tens place, and the red LED served as the ones place; for a given number, the place value would remain lit while the buzzer sounded off for each unit in that place value. For example, to relay the number 123, the yellow LED would light up and the buzzer would buzz once, then the green LED would light up and the buzzer would buzz twice, and finally the red LED would light up and the buzzer would buzz three times. (See Appendix G for example)

After implementing our original design, we discovered that the buzzer equipped on the MICA2 mote was obnoxiously loud and high-pitched, and that a group of MICA2s sounding the buzzer sounded almost like a fire alarm. To save the sanity of the teachers who will be using our motes (and to save our own), we made modifications to our sensor data display method. The new display method uses the same LED and place value relationships as the previous method, but the new method blinks each place value rather than buzz the value. For example, to relay the number 123, our motes will now *blink* the yellow LED once, the green LED twice, and the red LED three times. Once we completed and tested this second design, we decided that it would become the standard by which our motes will display sensor data to the user without using a PC.

Due to the given limitations of our mote hardware, we decided that the mote firmware should only be able to display one type of sensor data reading in a given program. We wrote our code so that, while every available sensor is read during a reading, only one type of reading is actually sent to the LEDs as flashes. For example, if a student is currently using the mote to take temperature readings via the LED display and wishes to switch to taking acceleration readings, the mote must be reprogrammed to the acceleration-specific code.

In our temperature display implementation, we found that, in order for our temperature display to be useful, we had to convert the sensor's default analog to digital conversion (ADC) values to degrees Celsius values *before* displaying the result on the mote's LEDs. The MTS310 User's Manual provided the following equation for the conversion[4]:

$$1/T(K) = a + b \times \ln(R_{thr}) + c \times [\ln(R_{thr})]^3$$

where:

$R_{thr} = R1(ADC\_FS-ADC)/ADC$

a = 0.001010024

b = 0.000242127

c = 0.000000146

R1 = 10 kΩ

ADC_FS = 1023, and

ADC = output value from Mote's ADC measurement.

T(C) = T(K) – 273

We had difficulty implementing this code into our motes because we could not figure out how to perform power and logarithm functions from within nesC. We thought that we had to somehow include the C-standard Math.h file so that we could use power and logarithm functions. However, we later learned that the Math.h file is already included in our TinyOS distribution by default; thus, we simply had to properly call the power function and the logarithm function in order to perform our conversion. (Refer to our code attachment in Appendix H)

4.4 - Identifying Sensor Reads and Sensor Displays

The MICA2 mote's lack of a user input button forced us to write firmware that took sensor readings at specified time intervals while displaying the reading results after each interval. In order for our display to be useful to students, we needed a way for students to know exactly when the mote is taking a reading and when the mote is displaying the value via the LEDs.

TinyOS, being an event-driven operating system, uses a timer interface that will fire a timer event once every specified time interval. We decided to abstract this interface to "ticks". A "tick" is the length of time between two successive timer events; with the aid of the yellow LED, we further defined a tick as the yellow LED flashing on (after one timer event) and turning off (after the second timer event, hence the two successive timer events). An unsigned integer at the beginning of our code defines the number of ticks that the firmware must cycle through before it attempts to take a sensor reading. We wrote code to sound off the buzzer during the last three ticks before a reading is taken to help users anticipate the start of the sensor reading. The result is a "3…2…1" countdown effect using the buzzer. Once the firmware cycles through the specified number of ticks, all three LEDs light up to signify that the mote is currently taking a sensor reading. The mote packages the readings and sends the readings across the XMesh network, and upon completing this task, the mote beeps and all of the LEDs turn off. As soon as the LEDs turn off, the mote displays one of the sensor data readings, based on the current code (see Sensor Data Display Method on previous page).

4.5 - Preserving Existing XMesh Functionality

XMesh is Crossbow's customized implementation of TinyOS' MintRoute networking library. In order to utilize Crossbow's powerful MoteView client-side application, our motes must implement XMesh for wireless network communications. An XMesh library exists such that we may import and use it in our firmware. However, in the interest of time, we decided to take existing XMesh-enabled firmware and add our custom sensor display method to the existing code.

Crossbow's distribution of MoteWorks comes with source code for the default, XMesh-enabled applications that interface well with MoteView. In order to merge our sensor data display functionality with the existing XMesh code, we had to utilize the existing code's timer. We modified the timer event handler code to account for a "tick" countdown (a tick is equal to one second), our sensor readings and our LED display. By using Crossbow's existing code, we saved ourselves the trouble of learning how to interface with Crossbow's networking solution and instead were able to focus on interfacing with sensor data and with LEDs.

5 - Results

Our firmware implementation relies heavily on Crossbow's existing code and libraries (particularly XMesh) since we wrote our display code on top of the existing code. Furthermore, our code specifically interfaces with the MTS310CA sensor board. As a result, our code is very platform-specific and is almost exclusively geared toward Crossbow MICA2 motes equipped with an MTS310CA sensor board.

Our number display mechanism is fairly generic and should be able to be ported to any TinyOS-enabled mote with a 3 LED display. However, because of our heavy reliance on existing XMesh code (and its GUI, MoteView), changing platforms from the Crossbow MICA2 to a different platform would require a complete rewrite of the networking code, as well as an understanding of a completely different client-side application, since MoteView is exclusive to Crossbow.

Maintaining use of our existing firmware locks the project's future hardware to Crossbow's MICA2 upgrade paths. Rewriting the networking code and client-side application interface in order to accommodate a different platform would have magnitude enough to warrant a project of its own. On the other side, our activities are generic enough that they may be used with any wireless sensor node device that utilizes the sensors that our activities call for; the only additions necessary would be programming the on-mote display and programming the wireless data transmission.

Our clients requested that we demonstrate one of our activities with a group of students. We chose "A Home for Kermy" as our demonstration activity; therefore, we fully implemented the display and networking firmware to work with temperature readings. Our firmware successfully read temperatures, sent the data back to the base station where the values could be read via MoteView, and displayed the data to the user (in Celsius units) via the LED display.

Since this project was completed after the normal school year was over, we tested the activity on twelve accelerated students studying 3rd and 4th grade material (note that "A Home for Kermy" is designed for 5th to 6th grade students, but material was omitted or altered to accommodate this grade level). Before we started the activity we tested the students on what they already knew about reading graphs, finding averages, and doing fraction multiplication with calculators (we were informed prior to the activity that the students had not learned fraction multiplication yet). All of the students could do the fraction multiplication with a calculator, and half of the students could also interpret readings on a graph. The students seemed really excited about the topic and the background story; therefore, trying to include a game or fun character was a good addition to the activity. We felt we could have given a better introduction to the activity before we started gathering data. One thing we specifically needed to talk about was how to calculate the average of three numbers. This information could be added to the Teacher's Packet.

The students had difficulty focusing at times and were rowdier than we expected. Performing the activities in groups smaller than four could have helped solve this problem, but we only had three motes to use. Thus, groups had to be larger than the two or three students the activity initially called for. We also realized that the motes needed to be programmed to take readings more frequently than once per minute. This would have saved some time and kept the students more focused. Students at the younger elementary age level have a short attention span, so making them wait for a minute between readings made the activity more difficult.

We had a few firmware issues in which the LED display would display an incorrect value, but these issues were temporarily remedied by turning off the mote and then turning it back on. We could not determine the cause of these errors; the motes would take 3 or so proper temperatures then return false values. We had a few hardware issues as well. Since our activity required students to bury our motes and since we did not have adequate housing for our motes, our motes suffered broken wires and disconnected boards that resulted in adverse results in our activity.

However, despite our software and hardware issues, our demonstration received positive acclaim from our test group when we surveyed them at the end of the activity. We also found that the majority of the students did not find the activity to be too difficult and the instructions were very clear. Most said the motes were necessary and very easy to use, and that the data from the motes was not difficult to interpret. All of the students found the activity fun and interesting, which positively addressed one of our biggest concerns.

6 - Conclusions and Future Directions

In the future we hope that other people will be able to focus on creating more activities for students since only minor modifications will be needed to adapt the software we create for new activities. There is the possibility of receiving funding for this project so it can be more widely used and upgraded as well. One of the limitations to future contributions to this project deals with hardware compatibility with other mote brands; if motes other than Crossbow are purchased the programming may need modifications. However, without having another brand of motes to investigate this issue it is unclear how extensive these modifications may be.

7 - Glossary

ADC – analog to digital converter
Ad hoc wireless network – a computer network in which the communication links are wireless because each node is willing to forward data for other nodes
Client-side – refers to operations that are performed by the client in a client-server relationship (a client is typically a computer application, such as a web browser, that runs on a user's local computer or workstation and connects to a server as necessary)
Event-driven programming – the flow of the program is determined by user actions (mouse clicks, key presses) or messages from other programs
Firmware – software that is embedded in a hardware device
Interface board – the main component of the base station, used to program and communicate with the motes
IT – Information Technology
ITEST – Information Technology Experiences for Students and Teachers
LED – light emitting diode
MICA2 – a third generation mote module used for enabling low-power, wireless sensor networks
MintRoute – the standard routing protocol of TinyOS
Mote – electronic communication devices that make up a wireless sensor network
MoteView – an application that interfaces between a user and a deployed network of wireless sensors
MoteWorks – Crossbow's open, integrated, standards-based platform for the development of wireless sensor network OEM devices and systems
nesC – a programming language built as an extension to the C programming language with components "wired" together to run applications on TinyOS
Node – a device connected to a network (see "Mote")
Radio Board – Sometimes called the "mote" because it is the brains of the mote component. The radio board is responsible for collecting data from the sensor board, transmitting data, and storing and running the code that is to be executed.
STEM – science, technology, engineering, and math
TinyOS – an embedded operating system written in the nesC programming language as a set of cooperating tasks and processes
WSN – Wireless Sensor Network
XMesh – (Crossbow's routing protocol) – creates a network consisting of nodes (Motes) that wirelessly communicate to each other and are capable of hopping radio messages to a base station where they are passed to a PC or other client

References:

1) http://www2.edc.org/itestlrc/ - ITEST Homepage

2) http://toilers.mines.edu/Toilers/MiddleSchoolWSNProject - Middle School WSN Project Homepage

3) http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MoteWorks_OEM_Edition.pdf - MoteWorks OEM Edition Software Platform Description

4) http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf - Crossbow MPR-MIB Users Manual

Appendix A

Temperature Activity

Grades: 3rd and 4th

Science Standards:

STANDARD 4:
Earth and Space Science: Students know and understand the processes and interactions of Earth's systems and the structure and dynamics of Earth and other objects in space. (Focus: Geology, Meteorology, Astronomy, Oceanography)

4.2 Students know and understand the general characteristics of the atmosphere and fundamental processes of weather.

5) recognizing that the Sun is a principal source of Earth's heat and light
6) describing existing weather conditions by collecting and recording weather data *(for example, temperature, precipitation, amount of cloud cover)*

Math Standards:

STANDARD 6:
Students link concepts and procedures as they develop and use computational techniques, including estimation, mental arithmetic, paper-and-pencil, calculators, and computers, in problem-solving situations and communicate the reasoning used in solving these problems.

- demonstrate conceptual meanings for the four basic arithmetic operations of addition, subtraction, multiplication, and division
- add and subtract commonly-used fractions and decimals using physical models (for example, 1/3, 3/4, 0.5, 0.75)
- demonstrate fluency with basic addition, subtraction, multiplication, and division facts without the use of a calculator

Key Question: How much does the sun affect temperature?

Engage: Below are a few questions to ask students prior to starting the activities:

- How many miles away from the earth is the sun?
  Minimum distance = 146 million km (91 million miles)
  Maximum distance = 152 million km (94.5 million miles)
- What is the temperature like on the sun?
  Temperature at core = 14 million degrees C (22.5 million degrees F)
  Temperature at surface = 5,500 degrees C (9,932 degrees F)
  Temperature of sunspots = 4,000 degrees C (7,232 degrees F)
- Why is the temperature different at the equator than at the poles?

Hypothesize: What consecutive temperature readings will give the largest change in temperature (See Readings 1-4 below)?

Part 1: Students each take a node home to measure the temperature around their homes. The nodes will be programmed to take outdoor temperatures at approximately the following times:

Reading 1. when students get home from school
Reading 2. when the sun goes down (right after sunset)
Reading 3. before students go to bed (after the sun has been down for several hours)
Reading 4. in the morning before students return to school

If time and weather permit, take the temperature at the same times during a cloudy day when the sun is not out at all. In class the following day students will calculate the difference in the temperature data they gathered for each day.

Calculate:	**Sunny Day**	**Cloudy Day (if applicable)**
Reading 1 – Reading 2 =	_____	_____
Reading 2 – Reading 3 =	_____	_____
Reading 3 – Reading 4 =	_____	_____
Reading 1 – Reading 4 =	_____	_____
Reading 1 – Reading 3 =	_____	_____
Reading 2 – Reading 4 =	_____	_____

Questions:

- Analyzing only the readings from the sunny day, which set of consecutive readings (Readings 1-2, Readings 2-3, or Readings 3-4), if any, had the greatest change in temperature? How did this result compare to your hypothesis? Give reasons why you think this result occurred.
- If measurements were able to be taken on a sunny and cloudy day, how does the difference in the temperature readings on the cloudy day compare to the difference in temperature readings on the sunny day?
- Which day had smaller temperature differences? Explain why you think this happened.

Part 2: Students can work in small groups for this activity. Give each group two sensors. Students place a node in the window of the room in the sun and another node on the other side of the room in the shade. Take several temperature readings from both nodes.

Questions: Did one side of the room give warmer temperature readings? If so, tell which side and why you think this higher temperature occurred.

Analyze: What do these two activities say about how the sun affects temperature?

References:

http://www.windows.ucar.edu/tour/link=/earth/statistics.html - Windows to the Universe "Earth Statistics"
http://www.windows.ucar.edu/tour/link=/sun/statistics.html - Windows to the Universe "Sun Reference Data"

Appendix B

Cars

Age: Grades 6-8

Note: Break off into teams

Materials: toy car, ramp, stopwatch, meter stick, sandpaper and towels to cover the ramps, wax paper for ramp, masking tape

Learning Goal & Instructional Objectives
Students conduct experiments to collect data (time and distance) for calculating the average speed of a car as a function of its mass, the height of the ramp, and surface of the ramp.

Instructional Objectives:
1. Calculate the speed of an object using the appropriate formula with data gathered in the laboratory and printed information.
2. Calculate the acceleration of an object using the appropriate formula with data given in the problem.

$$s = \frac{d}{t}$$

$$a = \frac{v_f - v_i}{\Delta t}$$

Key Questions: What factors affect the average speed of a small toy car?
How does each factor affect the motion of the car?

State science standards met:

1. Students understand the processes of scientific investigation and design, conduct, communicate about, and evaluate such investigations.
5. Students know and understand interrelationships among science, technology, and human activity and how they can affect the world.
6. Students understand that science involves a particular way of knowing and understand common connections among scientific disciplines.

To the Teacher: Emphasize that acceleration is defined as the change in velocity divided by the change in time.

Multiple Intelligences:
Logical-Mathematical Intelligence—
Consists of the ability to detect patterns, reason deductively, and think logically. This intelligence is most often associated with scientific and mathematical thinking.
Linguistic Intelligence—
Involves having a mastery of language. This intelligence includes the ability to effectively manipulate language to express oneself rhetorically or poetically. It also allows one to use language as a means to remember information.
Spatial Intelligence—
Gives one the ability to manipulate and create mental images in order to solve problems. This intelligence is not limited to visual domains. Gardner notes that spatial intelligence is also formed in blind children.

Engage: You could introduce the ideas of speed and acceleration by discussing major car races. The following questions may be useful for gaining interest from the class.

1. How fast do you think those cars are going?
2. How can a driver figure out the car's speed if the car's speedometer is not working?
3. How does a race car driver change the speed of the car?
4. What is the difference between acceleration and speed?

Each car will need to have a mote attached to its top to measure the acceleration of the car.

Part 1: Height of ramp
1. With the ramp flat on a table or floor, place the back of the car's wheels at one end of the ramp and measure the distance from the front of the car to the end of the ramp. Record this distance on the data sheet.
2. Raise the ramp up on the blocks. Measure the height in meters and record on the data sheet.
3. Place the back of the car's wheels at the top end of the ramp.
4. Release the car as you start the stopwatch.
5. Stop timing when the front of the car gets to the bottom of the ramp. Record this time on the data sheet.
6. Repeat steps 3-5 two more times then calculate the average time and record on the data sheet.
7. Calculate the average speed of your car by using the formula:
   speed = distance/time or $s = d/t$.
8. Raise one end of the ramp on two blocks and repeat steps 2-7.
9. Raise one end of the ramp on three blocks and repeat steps 2-7.

Part Two: Mass of Car (Height stays the same, mass changes)
1. Raise one end of the ramp on one block. Measure the height in meters and record on the data sheet.
2. Place the back wheels of the car at the top end of the ramp.

3. Time how long it takes to reach the bottom of the ramp. Record on the data sheet.
4. Repeat steps 2 and 3 two more times then calculate the average time and record on the data sheet.
5. Add a known mass to the car then repeat steps 2 - 4 recording all measurements on the data sheet.
6. Add a second known mass to the car then repeat steps 2- 4 recording all measurements on the data sheet.

Part Three: Surface of the ramp (Height of ramp changes, mass stays the same)
1. Raise one end of the ramp on 2 blocks. Measure the height in meters and record on the data sheet.
2. Place the back wheels of the car at the top end of the ramp.
3. Time how long it takes to reach the bottom of the ramp. Record on the data sheet.
4. Repeat steps 2 and 3 two more times, then calculate the average time and record on the data sheet.
5. Cover the surface of the ramp with a higher friction material such as a towel or sandpaper.
6. Repeat steps 2 - 4 two more times then calculate the average time and record on the data sheet.
NOTE: If the car stops before it reaches the bottom of the ramp, measure the distance from the top of the ramp to the back wheels and record this distance on the data sheet.
7. Cover the ramp with waxed paper.
8. Repeat steps 2 - 4 two more times then calculate the average time and record on the data sheet.

Teachers: Have students record their calculations in a lab journal and communicate their findings in a class discussion.

Part Four: Calculation, Graphs and Analysis
1. Calculate the average speed for each line in the chart using the formula:
   speed = distance/time or s = *d/t*.
2. Collect the acceleration values from the base computer and record them in your data sheet.
3. Calculate the average acceleration for each line in the chart using the formula:
   Acceleration = (velocity final – velocity initial)/change in time or $a = (v_f - v_i)/\Delta t$
   (where velocity initial is 0 and velocity final is the speed calculated in part one)
4. Compare your calculated acceleration to the value given by the sensor. How accurate are your results? If your results aren't accurate, why might this be?
5. Graph the data from Parts One, Two, and Three on separate graphs.
6. Identify the manipulated variable and responding variable for Parts One, Two, and Three.
7. Write about your findings in your journal and attach your graphs.

Data Sheets*

Part 1: Does the height of the ramp affect the average speed of the car?

Distance Traveled (m)	Height of Ramp (m)	Time (s)	Average Time (s)	Average Speed (m/s)	Average Acc. (m/s²)	Acc. From Mote (m/s²)
	[redacted]	Average time (s)				
	[redacted]	Average time (s)				
	[redacted]	Average time (s)				

Part 2: Does the mass of the car affect the average speed of the car?

Distance Traveled (m)	Mass on Car (g)	Time (s)	Average Time (s)	Average Speed (m/s)	Average Acc. (m/s²)	Acc. From Mote (m/s²)
	[redacted]	Average time (s)				
	[redacted]	Average time (s)				
	[redacted]	Average time (s)				

Part 3: Does the surface of the ramp affect the average speed of the car?

Distance Traveled (m)	Surface of Ramp (m)	Time (s)	Average Time (s)	Average Speed (m/s)	Average Acc. (m/s²)	Acc. From Mote (m/s²)
	[redacted]	Average time (s)				
	[redacted]	Average time (s)				
	[redacted]	Average time (s)				

Elaborate: Have students respond to the following questions in their science journals and then discuss their responses as a team.

Analysis:

1. Why did you do each timing three times, and then average them?

For more accuracy in experiments, measurements should be done more than once.

2. How did the height of the ramp affect the average speed of the car?

The average speed increased because gravity was able to act on the car over a longer vertical distance.

3. How did the mass of the car affect its average speed?
When the car goes down the ramp, gravity pulls at it with the same acceleration no matter what its mass is. (Teacher note: Students have problems with this concept because many of them believe that heavier objects will hit the ground first.) The final velocity of each car no matter what its mass should be the same velocity. There are some factors that may influence the results. These include air resistance, wheel traction, and the pressure exerted on the car's axles due to the increased mass. (Teacher note: A discussion of Galileo before the exploration might help the students understand this concept.)

4. How did the surface of the ramp affect the average speed of the car?
The more the friction between the ramp and the car wheels, the slower the car will travel. The wax should make the car go faster. However, if the friction is too little, the wheels will not be able to have traction with the surface and might go slower. Students must look at their data to see if that situation happened.

5. What other factors do you believe would affect the average speed of the car? How do you think that each factor would affect the speed?
This is open to student opinion. They might want to test wheel size (circumference) or change the front of the car so it has less air resistance. Let them visualize their cars and what they could do to change the speed.

6. What sources of error were involved in this lab? How do you think each source of error affected your results?
Timing correctly was probably the area of most error. If the time was wrong, it would definitely affect the average speed accuracy. To try to overcome this error possibility, three time measurements were done for each trial.

7. What was the initial velocity of the car at the point that you let the car go?
It was 0 meters per second because the car was not moving until it was released.

8. If a device measured the final velocity of a car as 2 m/s south on the ramp, calculate the acceleration of the car if it took 4 seconds to reach the bottom of the ramp. Use the

$$\text{acceleration} = \frac{\text{final velocity - initial velocity}}{\text{change in time}}$$

formula: $a = \frac{2\ \text{m/s} - 0\ \text{m/s}}{4\ \text{s}} = 0.5\ \text{m/s/s or } 0.5\ \text{m/s}^2\ \text{[South]}$

Teachers: Emphasize the scientific method throughout this project

Reference: http://www.coe.uh.edu/texasipc/units/motion/cars.pdf - Integrated Physics and Chemistry

Appendix C

A Home for Kermy

A Wireless Sensor Node Activity for 5th and 6th Grade Students

Objectives

1. Students will be able to use wireless sensor nodes to take temperature readings.
2. Students will be able to create and compare bar graphs according to the temperatures.
3. Students will be able to calculate the average of three numbers.
4. Students will be able to apply scientific reasoning and think critically to solve a problem.

Colorado Model Content Standards Met - Science

1. Students understand the processes of scientific investigation and design, conduct, communicate about, and evaluate such investigations (CMCS Science 1)
    a. using appropriate tools, technologies, and measurement units to gather and organize data;
    b. interpreting and evaluating data in order to formulate conclusions;
    c. communicating results of their investigations in appropriate ways *(for example, written reports, graphic displays, oral presentations)*;
    d. using metric units in measuring, calculating, and reporting results;
    e. giving examples of how collaboration can be useful in solving scientific problems and sharing findings.

2. Students know and understand common properties, forms, and changes in matter and energy (CMCS Science 2)

    a. Students know that matter has characteristic properties, which are related to its composition and structure (2.1)

        i. examining, describing, comparing, measuring, and classifying objects based on common physical and chemical properties *(for example, states of matter, mass, volume, electrical charge, temperature, density, boiling points, pH, magnetism, solubility)*;

    b. Students know that energy appears in different forms, and can move (be transferred) and change (be transformed) (2.2)

        i. measuring quantities associated with energy forms *(for example, temperature, mass, speed, distance, electrical charge, current, voltage)*;

    c. Students understand that interactions can produce changes in a system, although the total quantities of matter and energy remain unchanged (2.3)

        i. identifying and classifying factors causing change within a system *(for example, force, light heat);*

Colorado Model Content Standards Met - Mathematics

1. Students use algebraic methods to explore, model, and describe patterns and functions involving numbers, shapes, data, and graphs in problem-solving situations and communicate the reasoning used in solving these problems (CMCS Mathematics 2)

    a. represent, describe, and analyze patterns and relationships using tables, graphs, verbal rules, and standard algebraic notation (2.1)

2. Students use data collection and analysis, statistics, and probability in problem-solving situations and communicate the reasoning used in solving these problems (CMCS Mathematics 3)

    a. read and construct displays of data using appropriate techniques (for example, line graphs, circle graphs, scatter plots, box plots, stem-and-leaf plots) and appropriate technology (3.1)

    b. display and use measures of central tendency, such as mean, median, and mode, and measures of variability, such as range and quartiles (3.2)

    c. formulate hypotheses, draw conclusions, and make convincing arguments based on data analysis (3.4)

Activity Introduction

Kermy is a Kermludgeon from the planet Kermotron. One day, while Kermy was off exploring the vast expanse of the universe, Kermy's ship, the K-Pod, was struck by an asteroid. The asteroid knocked the K-Pod toward a big blue planet, and before he knew it, Kermy was caught in the planet's gravitational field and was plummeting towards its surface.

Kermy crashed landed in a sandbox in what appeared to be a elementary school playground. His K-Pod completely destroyed, Kermy must now find a place to live until he can repair his ship.

Problem

Kermludgeons are very sensitive creatures that can only live in soil (much like a plant) that has the right temperature. Students must investigate the different kinds of soil in the school playground and to determine where Kermy would be able to live.

Materials

Wireless Sensor Nodes

Clear Sandwich Bags (for protecting the nodes during burial)

Colored Pencils (for drawing graphs)

1 Copy of the Student Packet per Student

Setup

Select 5 or more playground locations that students will take temperature readings from. Try to pick locations that would have large differences in temperature, such a spot that is always shady or a spot that is always under the sun. Consider taking the students to a nearby park if the school playground does not offer such temperature-diverse locations. Note that each location should be relatively dry, as the wireless sensor nodes are not waterproof and must be buried in order to take proper temperature readings.

Select one or two locations that will serve as Kermy's desired home. Make sure these locations are very similar to each other in temperature so that students will be able to correctly choose the location given a hint (Kermy loves the cold) or a range of temperatures (Kermy likes temperatures of 50-60F). Consider taking the temperature of each location before performing the activity to get an idea of the range of temperatures the students will encounter during the activity.

Develop one or more hints to help students determine which location Kermy might live in. Consider using a formula that will help the students figure out what Kermy likes (for

example, a simple conversion from Fahrenheit to Celsius). Of course, hints will be based on which location(s) serve as Kermy's desired home.

Note that this lab focuses on temperature at a high level; it does not extensively investigate the quality of different kinds of soil, but simply requests that students perform qualitative analysis of those soil types. Teachers may introduce soil types to their students to fit this activity; however, such an introduction is not presented here.

Engage

Begin the activity with a brief overview of Kermy's problem. Then engage the students using questions that will hook their interest in temperature and wireless sensor nodes. Examples of such questions are below.

1. What can affect an object's temperature?
    a. External Sources (the sun)?
    b. Physical Properties of the Object (darker color absorbs more heat)?
    c. External Objects (heat transfer)?
2. What tools are available for us to take temperature readings of objects?
3. How might technology help us to take temperature readings of objects?

Explore

1. Form teams of 3-4 students (may be driven by how many motes you have)
2. Have teams assign roles to each student in the team:
    a. Investigator - The person who uses the sensor
    b. Recorder - The person who writes down the data the sensor gives
    c. Calculator - The person who calculates values from the data recorded
    d. Students will rotate through roles for each type of soil tested.
3. Take each team out to the playground and identify the locations they will test.
4. Give the students a demonstration of the following steps:
    a. Using the Sensor – Place the sensor inside of a clear sandwich bag and partially bury the sensor in the desired location. Turn on the sensor. The sensor will beep twice immediately before taking a temperature reading. The sensor will then report the temperature in a series of beeps and LED flashes. Refer to Appendix 1 for information on how to interpret the sensor's report.

b. Recording the Temperature – Have the students fill out the included data sheet with the name of the location being tested, three temperatures taken at that location, and any observations about the location (the ground is under shade most of the time, there are no plants and the sun shines constantly, etc.)

c. Performing Calculations – Have the students calculate the average of the three temperatures for each location. Have the students create a bar graph of each location with a bar for each temperature. Have them draw a dotted line across the three bars representing the average of the three temperatures.

d. Have each team spend no more than 5 minutes at each location.

5. Once each group has finished collecting data from each location, have each group present a location and its results (try to do one unique location per group). After the presentation, have the group post the graph for that location on the blackboard.

6. Compare and contrast the different locations and temperatures with the class. Ask them questions about why some locations may have such high temperature readings, while others have low ones. Tie each group's recorded observations (bare ground, excessive sunlight, etc.) to the temperatures and ask for the students to make the proper connection.

7. Offer each team a clue about where Kermy might like to live. This can either be direct (he hates hot places) or indirect (use this equation to find out what Kermy's desired average temperature is). Give each team 5 minutes to determine which location(s) Kermy might like. Consider offering some sort of reward for the teams that successfully determine a place for Kermy.

Variations

1. Have the students take temperatures at each location during different parts of the day (morning, lunch time, afternoon). Have the students create a line graph by plotting the average temperature of each location against the time of day that the reading was taken. Then have the students determine Kermy's desired home based on a hint involving time (for example, Kermy does not like to live in places that change temperature by more than 10 degrees throughout the day).

References

1. Sun Facts and General Sun Information (Regents of the University of California) - http://cse.ssl.berkeley.edu/segwayed/lessons/startemp/l1.htm (Sun Cartoon Image)

2. Worm Cartoon Image - http://onefoggy.tripod.com/images/worm.jpg

3. Crossbow MIH400 Demonstration - http://www.xbow.com/Products/Product_images/Wireless_images/comicbook(sound).swf (Sensor Node Image)

Appendix D

Step Counter Activity

Age: 6th Grade

Note: It may be helpful to pair students off to save time and help students with data interpretation.

Objectives:
1. Calculate the number of steps taken based on acceleration data presented on a graph.
2. Calculate velocity based on distance and time.

Questions to consider as you work:

Does the location of the sensor on the body affect the readings it gives? If so, in what way does it change the readings?
Which location of the sensor on the body gave the best readings and why do you think this is?

Part 1: Choosing a data collection method
1. Have the students attach a mote to their waists.
2. Instruct the students to take several steps, while trying to be consistent in stride length and speed.
3. Then have them run for about 15 seconds.
4. Now have students attach a mote to their arms.
5. Repeat steps 2 and 3.
6. Have the students attach a mote to their ankles.
7. Repeat steps 2 and 3.

Note: To get an accurate approximation for the number of steps taken with the mote attached to the ankle, you must multiply the number of steps counted in the graphs by 2. This is because the mote is only reading half the steps when it is attached to one leg.

Part 2: Let the race begin (make sure you know and record race distance for later calculations)

1. Line the students up and have them race with motes attached to the body part earlier determined to be optimal.
2. Keep track of the first 4 people to complete the race.
3. Repeat the first 2 steps a few times, possibly with new racers.

Optional Calculation: Use the following equation to calculate the velocity of the first 4 runners to finish in each race.

velocity = distance/time

Explain why this is an average velocity.

Analysis:

1. How do the step counts compare with each runner?

2. Was there any consistency between the number of steps taken and the overall speed of the runner?

3. Was your hypothesis correct?

Appendix E

Noise Pollution Activity

For 5th Grade

Science Standards:

STANDARD 1:
Students understand the processes of scientific investigation and design, conduct, communicate about, and evaluate such investigations.

1.0 identifying and evaluating alternative explanations and procedures

2.0 asking questions and stating hypotheses that lead to different types of scientific investigations (for example, experimentation, collecting specimens, constructing models, researching scientific literature)

3.0 creating a written plan for an investigation

4.0 using appropriate tools, technologies, and measurement units to gather and organize data

5.0 interpreting and evaluating data in order to formulate conclusions
communicating results of their investigations in appropriate ways (for example, written reports, graphic displays, oral presentations)

Objective: Students will measure the intensity of various sound sources at different times of the day and in different locations. Students will observe the changing levels.

Background and Safety Information:

Noise pollution can be defined as sounds or noises that are loud, annoying and harmful to the ear. Loud sounds can damage hearing, not only in humans, but also in animals. For example, noises in the ocean, such as the sounds of ships and other industrial activity, can interfere with a marine mammal's use of sound for navigating, hunting and communicating.

Sounds that are very loud and short in duration can damage a person's hearing quickly. Loud, continuous sounds can cause long-term effects on hearing. In general, sounds can be characterized by their frequency (or pitch) and intensity (or loudness). The vibrations that produce a sound are cyclical and are measured in hertz (Hz). One hertz (Hz) equals the number of cycles that occur per second. An adult with good hearing can generally hear frequencies in the range 20 to 15,000 Hz. Children can usually hear frequencies above 20,000 Hz.

The sound pressure level is measured in decibels (dB). For example, a whisper can occur in the range of 20-30 dB, while normal conversation is about 60 dB. A person that is shouting in your face could easily exceed 80 dB! The correspondence between decibel levels and perceived

loudness is fairly simple. A difference of 3 dB in noise level is barely noticeable, yet it represents a doubling of the acoustic energy involved. For a noise to sound twice (or half) as loud, a difference of about 10 dB is required. For example, a lawn mower measured at 80 dB will sound about twice as loud as a hair dryer at 70 dB.

If conducting experiments near automobile traffic, reinforce safety guidelines to your students. If experiments are conducted near areas of loud noise (i.e. jackhammer at a school construction site), use proper ear protection.

Hypothesize: Will the Sound Levels in a classroom increase or decrease during a class period? Why?

Procedure:

1) Place the wireless sensor node (WSN) in an open area of the classroom.
2) Collect data throughout an entire class period.
3) Print out copies of the graph of the data for students to examine.

Questions:

- At what point was the sound level reading the highest? The lowest?
- Were there any spikes on the graph that had large changes in sound? What could these changes have been caused by (i.e. laughter, school bell, textbook falling on the floor, etc.)?
- How did the sound levels compare to your hypothesis and expectations? Explain why any difference in sound level occurred.
- How might a reading from one class period differ from another? Why?

Extensions: It may be a good idea to place the WSN in another classroom to analyze sound levels after collecting data from the classroom of the students doing the experiment. This will ensure that the presence of the sensors don't influence students to create or refrain from creating noises. However, this does make it more difficult to interpret what caused certain changes in the sound level. Some interesting classes or other places to collect data may include the following:

1) band or choir class
2) gym class
3) lunch period
4) a school sporting event (outdoor or indoor)
5) traffic in the school parking lot

Compare the sound levels of these areas with the data from the students' classroom. Why did certain places have higher or lower readings than the students' class?

Discuss other sound-related issues, such as how human-generated noise can disturb an environmental habitat. This will hopefully encourage some interest in how individuals working

in scientific and technological fields are becoming increasingly sensitive to this issue. Ask students for examples.

References:

http://www.pasco.com/experiments/middle_school/february_2003/home.html#purpose - PASCO Scientific

Appendix F

Narrative on WSN Hardware Selection for the Middle School WSN Project

1. Abstract

We found the Moteiv Tmote Invent to be the best solution for the Middle School WSN Project's hardware requirements. We based our decision on the Tmote Invent's ease of development, ease of deployment, and feature set.

2. Background

The goal of the Middle School WSN Project is to create a working science and/or math curriculum for Colorado students grades 3 through 8 that utilizes wireless sensor node technology (Reqs 3.1.1, 3.2.6). The curriculum must include any hardware, documentation and training materials necessary for successful, guided implementation in an elementary or middle school classroom (Reqs 3.2.2, 3.2.4).

Our WSN hardware selection is based on various factors, including ease of development, ease of deployment, and feature set (Req. 3.1.1). In this document, we present a recommendation for the WSN hardware and detail how our chosen hardware meets our selection factors.

3. Recommendation

We researched current off the shelf wireless sensor nodes (hereafter referred to as "motes") during week one of the project. We found four models of motes from three separate vendors that would be reasonable for use in our project. After careful consideration of each mote's ease of use and feature set, we found that the **Moteiv Tmote Invent**[1] is currently the best solution for our WSN hardware needs. The remainder of this document explains our "moteiv"ation for choosing this particular mote.

3.1 Ease of Development

Since the project's classroom activities vary in their use of the WSN hardware, we will have to develop software for use with the motes that tailors to each activity. Developing software for each activity involves developing the software that runs on a given mote, as well as the software that runs on a PC functioning as a wireless sensor network base station.

The Moteiv Tmote Invent, when purchased through Moteiv's Tmote Invent Kit[2], comes with Moteiv's Tmote Tools, a software suite designed to aid in the creation and maintenance of a wireless sensor network[3]. The suite offers a graphical view of network topology, link status and basic sensor readings (the temperature currently recorded at each mote, for example). The Tmote Tools software will thus be beneficial for both the development and deployment stages of our project.

The Invent Kit also comes with a Moteiv distribution of TinyOS. TinyOS is a widely-used operating system designed for motes; thus, there are many support channels available on the internet. Moteiv distributes their TinyOS version with Application Programming Interface (API) documentation and sample code for using the various features of the mote. All mote development must be written in NesC (an extension of the C programming language) in order to properly interface with TinyOS. Since our development skills in C are limited, having the ample documentation and sample code is crucial in effective mote development.

3.2 Ease of Deployment

Although our project requires the presence of a Colorado School of Mines student or representative familiar with the WSN technology to deploy the motes (Req. 3.2.2), the motes themselves should still be relatively easy to set up and use (Req. 3.2.5).

Our classroom activities require that students from grades 3 through 8 personally handle each mote (Req. 3.2.2). Thus, our motes must be sturdy enough for young children to use. The Tmote Invent comes in a hard plastic housing that does not expose any of the internal hardware, thus the mote will be safe for use by young children.

The Tmote Invent supports automatic on-demand ad-hoc networking; thus, the motes themselves will reconfigure themselves (and the network) automatically if any motes are added to or removed from the network.

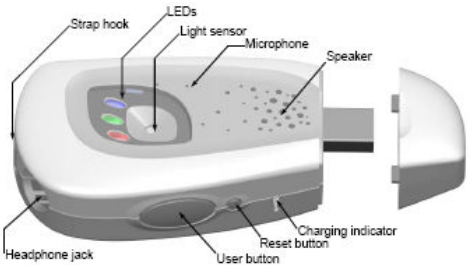
The Tmote Invent Kit comes in a box with molded encasings. This allows safe, easy transportation of up to eight motes.

Finally, the Tmote Invent contains a built-in lithium ion rechargeable battery. The mote enters charge mode when connected to a PC via USB, and an LED indicator on the mote itself gives the charge status of the mote. With a built-in battery, the hassle of purchasing and installing batteries for motes that run on removable batteries is eliminated.

3.3 Feature Set

The Tmote Invent has built-in light, temperature, acceleration and sound sensors. For our project's intents and purposes (considering scope) these four sensors are more than enough to demonstrate a working curriculum.

Figure 1



The mote also has LEDs, a built-in speaker, a microphone, a user button and a headphone jack (see Figure 1). The LEDs will allow the mote to provide instant feedback to the students. The speaker is capable of reproducing voice-quality audio and serves as another way to provide user feedback. The microphone may be used to record audio that can be sent to all other nodes for playback. The user button allows further interactivity with the mote; we can program the button to force the mote to check its sensors, for instance. Lastly, the headphone jack allows for an even greater range of audio output than the built-in speaker.

The mote has 10 KB of RAM for applications and 48 KB of flash memory for storing sensor readings. Each mote operates on a 2.4GHz, 250kbps wireless connection with an indoor range of 50 meters and an outdoor range of 125 meters.

4. References

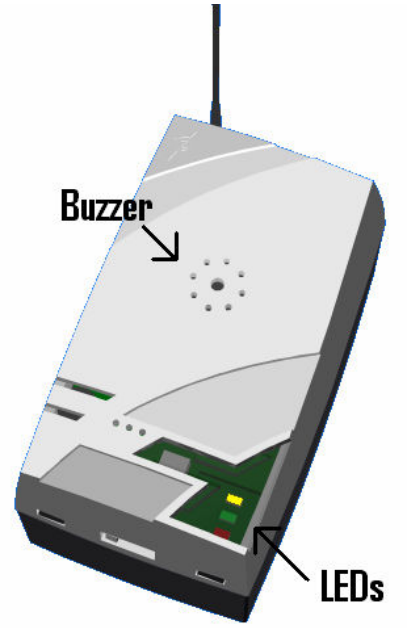
1. Moteiv Tmote Invent Product Page - http://www.moteiv.com/products/tmoteinvent.php

2. Moteiv Tmote Invent Kit - http://www.moteiv.com/xcart/product.php?productid=5

3. Moteiv Tmote Invent User Guide - http://www.moteiv.com/products/docs/tmote-invent-user-guide.pdf

Appendix G

Interpreting the Sensor Node's Report



This appendix explains how to interpret the Crossbow MICA2's audiovisual cues to determine a reading.

Note that these audiovisual cues are only available on TinyBrainz-programmed sensor nodes; they are not available on MICA2s purchased directly from Crossbow. We will refer to a TinyBrainz-programmed sensor node as a "TinyBrainz mote" for the remainder of this document.

Anticipating the Report

The TinyBrainz mote blinks its yellow LED as a tick when it is not taking a reading or sending a radio message. By default, the TinyBrainz mote will take a reading every 30 ticks. The mote will sound a long beep for each of the last 3 ticks to signify that it is about to take a reading. After these ticks, all 3 LEDs will light up. After the mote takes its reading, the mote will beep quickly and the LEDs will turn off. The mote will then display the value of the reading.

Obtaining Numerical Values

The TinyBrainz mote reports numbers in a consistent manner, regardless of the kind of reading it takes. The three LED colors each correspond to a base-10 place value. The yellow LED corresponds to the hundreds place, the green LED corresponds to the tens place, and the red LED corresponds to the ones place. Starting with the yellow LED, the mote will blink the current place value, one blink per unit. The mote will continue in this fashion until the number is reported. The mote will sound a long beep to signify that the number has been displayed.

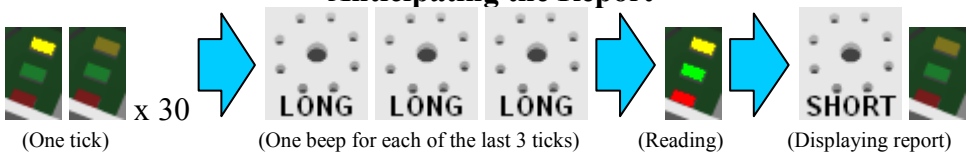
For example, to report the number 27, the mote will blink its green LED twice and its red LED seven times. For the number 134, the mote will blink its yellow LED once, its green LED three times and its red LED four times. Refer to the Quick Reference guide below.

Quick Reference

LED Key

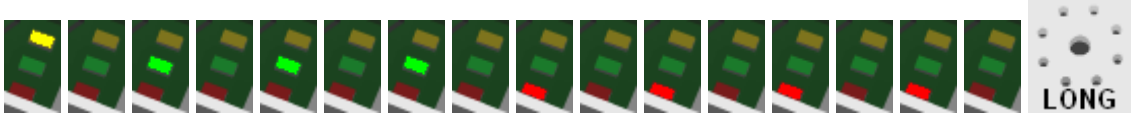
Yellow	Hundreds
Green	Tens
Red	Ones

Anticipating the Report



(One tick) (One beep for each of the last 3 ticks) (Reading) (Displaying report)

Report Example: 134



Appendix H

```
/**
 * Temperature-enabled Sensor Application with LED Display
 * Part of the TinyBrainz Project
 *
 * This application allows a MICA2 mote equipped with an
 * MTS310CA sensor board to read in temperatures, then
 * convert those temperatures to Celsius and display the
 * temperatures using the standard TinyBrainz sensor display
 * method.  Refer to the Teacher Packet for more information
 * on the display method.
 *
 * For use with "A Home for Kermy" and other projects needing
 * real-time temperature readings.
 *
 * Original code by Crossbow Technology, Inc.
 *
 * @author Brandon Ardiente, Steven Arth, Claire DuPont
 *
 * June 14th, 2007
 *
 */
/*
 * Copyright (c) 2004-2007 Crossbow Technology, Inc.
 * All rights reserved.
 * See license.txt file included with the distribution.
 *
 * $Id: XMTS310M.nc,v 1.5.4.6 2007/04/26 20:18:11 njain Exp $
 */

/**
 * XSensor multi-hop application for MTS310 sensorboard.
 *
 * @author Martin Turon, Alan Broad, Hu Siquan, Pi Peng
 */

/*****************************************************************************
 *
 *****************************************************************************/
#include "appFeatures.h"
includes sensorboard;

module XMTS310M {
   provides {
         interface StdControl;
```

```
  }
  uses {
// RF Mesh Networking
        interface MhopSend as Send;
        interface RouteControl;

        interface XCommand;
        interface XEEControl;

// Battery
  interface ADC as ADCBATT;
  interface StdControl as BattControl;

//Temp
        interface StdControl as TempControl;
        interface ADC as Temperature;

//Light
        interface StdControl as PhotoControl;
        interface ADC as Light;

// Mic
        interface StdControl as MicControl;
        interface Mic;
        interface ADC as MicADC;

// Sounder
        interface StdControl as Sounder;

// Accel
        interface StdControl as AccelControl;
        interface ADC as AccelX;
        interface ADC as AccelY;

// Mag
        interface StdControl as MagControl;
        interface ADC as MagX;
        interface ADC as MagY;

        //interface ADCControl;
        interface Timer;
        interface Leds;

#if FEATURE_UART_SEND
        interface SendMsg as SendUART;
#endif
```

```
        command result_t PowerMgrEnable();
        command result_t PowerMgrDisable();
        command void health_packet(bool enable, uint16_t intv);
        command HealthMsg* HealthMsgGet();
    }
}

implementation {

    enum { START, BUSY, SOUND_DONE};

#define MSG_LEN  29

    TOS_Msg    gMsgBuffer;
    TOS_Msg    msg_buf_radio;
    TOS_MsgPtr msg_radio;
    HealthMsg *h_msg;

        // Begin TinyBrainz code addition
        norace uint16_t displayValue;
        norace uint8_t ticks = 0;
        norace uint8_t ticksUntilRead = 60;

        norace uint8_t hundreds = 0;
        norace uint8_t tens = 0;
        norace uint8_t ones = 0;

        float fTemp;
        float temperature, a, b, c, Rthr, tTemp;

        norace bool isSounding, isDisplaying = FALSE;
        // End TinyBrainz code addition

    bool sleeping;              // application command state

    norace XDataMsg readings;

    char main_state;
    norace bool sound_state, sending_packet,sensinginsession;
    norace uint8_t        miccnt;
    norace uint32_t  val;

    // Zero out the accelerometer, chrl@20061206
    norace uint16_t accel_ave_x, accel_ave_y;
    norace uint8_t accel_ave_points;
```

```
/*****************************************************************************
 * Task to xmit radio message
 *
 *    msg_radio->addr = TOS_BCAST_ADDR;
 *    msg_radio->type = 0x31;
 *    msg_radio->length = MSG_LEN;
 *    msg_radio->group = TOS_AM_GROUP;
 *****************************************************************************/
 /**
  * send_radio_msg
  *
  * This function is called when the mote is ready to transmit sensor
  * information across the XMESH wireless network.
  *
  * We have modified this code to signal to the mote that once
  * the mote is done transmitting wirelessly, it should run
  * displayNumber() to display the value of displayValue on the LEDs.
  * This is actually called from the "Timer.fired" function.
  *
  */
   task void send_radio_msg() {
          uint8_t   i;
          uint16_t  len;
          XDataMsg *data;

          call Sounder.start(); //Leds.yellowOn();
          // Fill the given data buffer.
          data = (XDataMsg*)call Send.getBuffer(msg_radio, &len);

          for (i = 0; i <= sizeof(XDataMsg)-1; i++)
             ((uint8_t*)data)[i] = ((uint8_t*)&readings)[i];

          data->board_id  = SENSOR_BOARD_ID;
          data->packet_id = 1;
          //data->node_id   = TOS_LOCAL_ADDRESS;
          data->parent    = call RouteControl.getParent();
          data->packet_id = data->packet_id | 0x80;
#if FEATURE_UART_SEND
          if (TOS_LOCAL_ADDRESS != 0) {
             call PowerMgrDisable();
             TOSH_uwait(1000);
             if (call SendUART.send(TOS_UART_ADDR, sizeof(XDataMsg),
                                    msg_radio) != SUCCESS)
             {
                   atomic sending_packet = FALSE;
                   call Sounder.stop(); //Leds.yellowOff();
```

```
                call PowerMgrEnable();
            }
        }
        else
#endif
        {
            // Send the RF packet!
            if (call Send.send(BASE_STATION_ADDRESS,MODE_UPSTREAM,msg_radio,
sizeof(XDataMsg)) != SUCCESS) {
                atomic sending_packet = FALSE;
                call Sounder.stop(); //Leds.yellowOff();
            }
        }

        // Signal that the mote should now display the value of
        // displayValue on the LEDs
        isDisplaying = TRUE;

        return;
}

        /**
         * initialize
         *
         * This function is called when the mote is first started.
         *
         * We have made modifications to secure the timer_rate at 1000ms
         * rather than variable rate.  The timer_rate may still be changed
         * via the command interface on the MoteView software package.
         *
         */

  static void initialize() {
    atomic {
        sleeping = FALSE;
        main_state = START;
        sound_state = TRUE;
        isSounding = FALSE;
        sending_packet = FALSE;

        // This line will set the timer to fire every 1000 ms, or each second
        timer_rate = 1000;

        // TinyBrainz removal
        /*
            #ifdef APP_RATE
```

```
                                timer_rate = XSENSOR_SAMPLE_RATE;
                #else
                #ifdef USE_LOW_POWER
                                timer_rate = XSENSOR_SAMPLE_RATE  +
((TOS_LOCAL_ADDRESS%255) << 7);
                #else
                                timer_rate = XSENSOR_SAMPLE_RATE +
((TOS_LOCAL_ADDRESS%255) << 2);
                #endif
                #endif
            */
            miccnt=0;
            val=0;
            sensinginsession=FALSE;
        }
    }

    static void start()
     {
      call BattControl.start();
      call Mic.gainAdjust(64);  // Set the gain of the microphone. (refer to Mic)
      call MicControl.start();
      call TempControl.start();
      call PhotoControl.start();
#ifdef MTS310
      call AccelControl.start();
      call MagControl.start();
#endif
     }

          /**
           * displayNumber
           *
           * This function is the TinyBrainz-standard for displaying values on
           * a 3-LED mote.  This particular implementation uses code
           * specific to temperature conversion from raw ADC to Celsius.
           *
           */

    static void displayNumber() {
     if (hundreds == 0 && tens == 0 && ones == 0 && isDisplaying == TRUE) {

            // Begin temperature-specific conversion code
            tTemp = (float) displayValue;

            if(tTemp == 0){
```

```
                temperature = 0;
        }
        else{
                a = 0.001307050F;
                b = 0.000214381F;
                c = 0.000000093F;
                Rthr = 10000 * (1023 - tTemp) / tTemp;

                temperature = 1 / (float)(a + b * log(Rthr) + c * pow(log(Rthr),3));
                temperature -= 273.15F; // Convert from Kelvin to Celcius
}
        fTemp = temperature;
        // End temperature-sepcific conversion code

        // The following code is standard for all TinyBrainz applications,
        // the difference here being that 'displayValue' has been replaced by
        // 'fTemp.'
hundreds = fTemp / 100;
        fTemp = fTemp - (hundreds * 100);
        tens = fTemp / 10;
        fTemp = fTemp - (tens * 10);
        ones = fTemp;
        isSounding = FALSE;
        call Leds.yellowOff();
        call Leds.greenOff();
        call Leds.redOff();
    }

    // For each value in the hundreds place, flash the yellow LED
    if (hundreds > 0) {
      if (isSounding) {
        call Leds.yellowOff();
                hundreds = hundreds - 1;
                isSounding = FALSE;
      }
      else {
                        call Leds.yellowOn();
                        isSounding = TRUE;
              }
    }

    // For each value in the tens place, flash the green LED
    else if (tens > 0) {
      if (isSounding) {
        call Leds.greenOff();
                tens = tens - 1;
```

```
                isSounding = FALSE;
            }
            else {
                        call Leds.greenOn();
                        isSounding = TRUE;
                }
        }

        // For each value in the ones place, flash the red LED
        else if (ones > 0) {
            if (isSounding) {
                call Leds.redOff();
                ones = ones - 1;
                isSounding = FALSE;
                if (ones == 0){
                    isDisplaying = FALSE;
                    ticks = 0;
                        call Sounder.start();
                }
            }
            else {
                        call Leds.redOn();
                        isSounding = TRUE;
                }
        }
}

  task void battstop()
  {
        call BattControl.stop();
  }
  task void tempstop()
  {
        call TempControl.stop();
  }
  task void photostop()
  {
        call PhotoControl.stop();
  }
  task void Micstop()
  {
        call MicControl.stop();
  }


  task void Accelstop()
```

```
  {
      call AccelControl.stop();
  }


/****************************************************************************
 * Initialize the component. Initialize ADCControl, Leds
 *
 ****************************************************************************/
  command result_t StdControl.init() {

      atomic msg_radio = &msg_buf_radio;

  //  MAKE_BAT_MONITOR_OUTPUT();  // enable voltage ref power pin as output
  //  MAKE_ADC_INPUT();          // enable ADC7 as input
  call BattControl.init();
      // usart1 is also connected to external serial flash
      // set usart1 lines to correct state
      //  TOSH_MAKE_FLASH_SELECT_OUTPUT();
    TOSH_MAKE_FLASH_OUT_OUTPUT();          //tx output
    TOSH_MAKE_FLASH_CLK_OUTPUT();          //usart clk
      //  TOSH_SET_FLASH_SELECT_PIN();

    call Leds.init();
    call TempControl.init();
    call PhotoControl.init();
    call MicControl.init();
    call Mic.muxSel(1);  // Set the mux so that raw microhpone output is selected
    call Mic.gainAdjust(64);  // Set the gain of the microphone. (refer to Mic)
#if FEATURE_SOUNDER
    call Sounder.init();
#endif

#ifdef MTS310
    // Zero out the accelerometer, chrl@20061206
    accel_ave_x = 0;
    accel_ave_y = 0;
    accel_ave_points = ACCEL_AVERAGE_POINTS;

    call AccelControl.init();
    call MagControl.init();
#endif

if (TOS_LOCAL_ADDRESS==0)
    call PowerMgrDisable();
```

```
    initialize();
    return SUCCESS;

  }
 /****************************************************************************
 * Start the component. Start the clock.
 *
 ****************************************************************************/
  command result_t StdControl.start()
  {
        call StdControl.stop();
        h_msg = call HealthMsgGet();
        h_msg->rsvd_app_type = SENSOR_BOARD_ID;
        call health_packet(TRUE,TOS_HEALTH_UPDATE);
        call Timer.start(TIMER_REPEAT, 700);

/*#ifdef MTS310
        call Timer.start(TIMER_REPEAT, 1024);
#else
  call Timer.start(TIMER_REPEAT, timer_rate);
#endif
 */     return SUCCESS;
  }

/****************************************************************************
 * Stop the component.
 *
 ****************************************************************************/
  command result_t StdControl.stop() {
    call BattControl.stop();
    call TempControl.stop();
    call PhotoControl.stop();
    call MicControl.stop();

#ifdef MTS310
    call AccelControl.stop();
    call MagControl.stop();
#endif

    return SUCCESS;
  }
/****************************************************************************
 * Measure Temp, Light, Mic, toggle sounder
 *
 ****************************************************************************/
```

```
 /**
  * Timer.fired
  *
  * This function is called each time the timer is fired.  We have made
  * modifications to this code to allow the mote to "tick" until it reads
  * a value.  We have also modified the code to allow our display function
  * to run when a value is ready to be displayed.
  *
  */
  event result_t Timer.fired() {

#ifdef MTS310
    // Zero out the accelerometer, chrl@20061206
    if (accel_ave_points >0)
    {
     //call Leds.greenOn();
     if (accel_ave_points == 1)
     {
       call Timer.stop();
       call Timer.start(TIMER_REPEAT, timer_rate);
     }
    }
#endif

     if (sending_packet)
            return SUCCESS;             //don't overrun buffers
     if(main_state==BUSY)
        return SUCCESS;

            // Begin TinyBrainz addition

            // If we are not displaying a number via the LEDs, we
            // should be counting down (ticking) until our next read
            if (isDisplaying == FALSE){

                    ticks = ticks + 1;

                    // The following code sounds the buzzer for the last
                    // three ticks before taking a reading
                    if (ticks == 1)
                            call Sounder.stop();
                    if (ticks == ticksUntilRead - 5)
                      call Sounder.start();
                    if (ticks == ticksUntilRead - 4)
                      call Sounder.stop();
                    if (ticks == ticksUntilRead - 3)
```

```
            call Sounder.start();
          if (ticks == ticksUntilRead - 2)
            call Sounder.stop();
          if (ticks == ticksUntilRead - 1)
            call Sounder.start();
          if (ticks == ticksUntilRead)
            call Sounder.stop();

          // Flash the yellow LED as a 'tick'
          if (isSounding) {
                call Leds.yellowOff();
                isSounding = FALSE;
          }
          else {
                call Leds.yellowOn();
                isSounding = TRUE;
          }

          // After 'ticksUntilRead' passes, we are ready to read the
          // temperature
          if (ticks > ticksUntilRead) {
                isDisplaying = FALSE;
                hundreds=0;
                tens=0;
                ones=0;
                isSounding = FALSE;
                //ticks = 0;
             start();
          atomic main_state = BUSY;
                if (!sensinginsession){
                      call ADCBATT.getData();
                      atomic sensinginsession = TRUE;
                      }
          }
    }

    // If we are still displaying a number on the LEDs, continue
    // to do so
    else{
          displayNumber();
    }
    // End TinyBrainz Addition

  return SUCCESS;
}
```

```
/*****************************************************************************
* Battery Ref  or thermistor data ready
*****************************************************************************/
 async event result_t ADCBATT.dataReady(uint16_t data) {
    if (!sensinginsession) return FAIL;
    readings.vref = data;
    atomic sensinginsession = FALSE;
    post battstop();
    //call TempControl.start();
    call Temperature.getData();
    return SUCCESS;
 }


/*****************************************************************************
 * Temperature ADC data ready
 * Read and get next channel.
 *****************************************************************************/
 /**
  * Temperature.dataReady
  *
  * This function is called once the sensor board has successfully read a
  * temperature value.  Our modification will turn all of the LEDS to the
  * on position to signify that the mote is in the process of taking a
  * reading.
  *
  * We set displayValue = data here, meaning that our displayNumber function
  * will display the data of this function (temperature).
  *
  */
 async event result_t Temperature.dataReady(uint16_t data) {
    readings.thermistor = data;

    call TempControl.stop();

    call Leds.yellowOn();
    call Leds.greenOn();
    call Leds.redOn();

          // We wish to display this function's data (temperature)
          displayValue = data;

          //call PhotoControl.start();
    call Light.getData();
    return SUCCESS;
 }
```

```
/*****************************************************************************
 * Photocell ADC data ready
 * Read and get next channel.
 *****************************************************************************/
  async event result_t Light.dataReady(uint16_t data) {
     readings.light = data;
     call PhotoControl.stop();
     call MicADC.getData();
     return SUCCESS;
  }

/*****************************************************************************
 * MicroPhone ADC data ready
 * Read and toggle sounder.
 * send uart packet
 *****************************************************************************/
  async event result_t MicADC.dataReady(uint16_t data) {
    if(miccnt<50)
    {
            atomic miccnt=miccnt+1;
            TOSH_uwait(1000);
            if(val<data)
            {
                 atomic val=data;
            }
            call MicADC.getData();
            return SUCCESS;
    }
    else
    {
            atomic miccnt=0;
    }
    readings.mic = val;//data;
    post Micstop();
    val=0;

#ifdef MTS310
    call AccelX.getData();
#else
    // This is the final sensor reading for the MTS300...
    atomic {
          if (!sending_packet) {
             sending_packet = TRUE;
             post send_radio_msg();
          }
```

```
    }

#if FEATURE_SOUNDER
    if (sound_state) call Sounder.start();
    else call Sounder.stop();
    atomic {
          sound_state = SOUND_STATE_CHANGE;
    }
#endif
#endif
    return SUCCESS;
 }


/****************************************************************************
 *  ADC data ready
 * Read and toggle sounder.
 * send uart packet
 ****************************************************************************/
  async event result_t AccelX.dataReady(uint16_t data) {
     // Zero out the accelerometer, chrl@20061207
     if (accel_ave_points>0)
     {
       accel_ave_x = accel_ave_x + data;
       call AccelY.getData();
       return SUCCESS;
     }

     readings.accelX = data - accel_ave_x;

     call AccelY.getData();
     return SUCCESS;
  }

/****************************************************************************
 *  ADC data ready
 * Read and toggle sounder.
 * send uart packet
 ****************************************************************************/
  async event result_t AccelY.dataReady(uint16_t data) {
     // Zero out the accelerometer, chrl@20061207
     if (accel_ave_points>0)
     {
       accel_ave_y = accel_ave_y + data;
       accel_ave_points--;
       //call Leds.greenOff();
```

```
      call StdControl.stop();
      if(accel_ave_points == 0)
      {
        accel_ave_x = accel_ave_x / ACCEL_AVERAGE_POINTS - 450;
        accel_ave_y = accel_ave_y / ACCEL_AVERAGE_POINTS - 450;
      }
      main_state = START;
      return SUCCESS;
    }

    readings.accelY = data - accel_ave_y;

    post Accelstop();
    call MagX.getData();
    return SUCCESS;
  }

 /**
  * In response to the <code>MagX.dataReady</code> event, it stores the
  * sample and issues command to sample the magnetometer's Y axis.
  * (Magnetometer B pin)
  *
  * @return returns <code>SUCCESS</code>
  */
  async event result_t MagX.dataReady(uint16_t data){
    readings.magX = data;

    call  MagY.getData(); //get data for MagnetometerB
    return SUCCESS;
  }

 /**
  * In response to the <code>MagY.dataReady</code> event, it stores the
  * sample and issues a task to filter and process the stored magnetometer
  * data.
  *
  * It also has a schedule which starts sampling the Temperture and
  * Accelormeter depending on the stepdown counter.
  *
  * @return returns <code>SUCCESS</code>
  */
  async event result_t MagY.dataReady(uint16_t data){
    readings.magY = data;
    atomic {
          if (!sending_packet) {
             sending_packet = TRUE;
```

```
            post send_radio_msg();
        }
    }

#if FEATURE_SOUNDER
    if (sound_state) call Sounder.start();
    else call Sounder.stop();
    atomic {
        sound_state = SOUND_STATE_CHANGE;
    }
#endif
    return SUCCESS;
  }

 /**
  * Handles all broadcast command messages sent over network.
  *
  * NOTE: Bcast messages will not be received if seq_no is not properly
  *       set in first two bytes of data payload.  Also, payload is
  *       the remaining data after the required seq_no.
  *
  * @version   2004/10/5   mturon     Initial version
  */
  event result_t XCommand.received(XCommandOp *opcode) {

    switch (opcode->cmd) {
        case XCOMMAND_SET_RATE:
            // Change the data collection rate.
            timer_rate = opcode->param.newrate;
            call Timer.stop();
            call Timer.start(TIMER_REPEAT, timer_rate);
            break;

        case XCOMMAND_SLEEP:
            // Stop collecting data, and go to sleep.
            sleeping = TRUE;
            call Timer.stop();
            call StdControl.stop();
            call Leds.set(0);
        break;

        case XCOMMAND_WAKEUP:
            // Wake up from sleep state.
            if (sleeping) {
                initialize();
                call Timer.start(TIMER_REPEAT, timer_rate);
```

```
                call StdControl.start();
                sleeping = FALSE;
            }
            break;

        case XCOMMAND_RESET:
            // Reset the mote now.
            break;

        case XCOMMAND_ACTUATE: {
            uint16_t state = opcode->param.actuate.state;
            if (opcode->param.actuate.device != XCMD_DEVICE_SOUNDER) break;

            // Play the sounder for one period.
            sound_state = state;
            if (sound_state) call Sounder.start();
            else call Sounder.stop();
            atomic {
                sound_state = SOUND_STATE_CHANGE;
            }
            break;
        }


        default:
            break;
    }

    return SUCCESS;
  }

#if FEATURE_UART_SEND
 /**
  * Handle completion of sent UART packet.
  *
  * @author    Martin Turon
  * @version   2004/7/21      mturon       Initial revision
  */
  event result_t SendUART.sendDone(TOS_MsgPtr msg, result_t success)
  {
    //      if (msg->addr == TOS_UART_ADDR) {
    atomic msg_radio = msg;
    msg_radio->addr = TOS_BCAST_ADDR;

    if (call Send.send(BASE_STATION_ADDRESS,MODE_UPSTREAM,msg_radio,
sizeof(XDataMsg)) != SUCCESS) {
```

```
            atomic sending_packet = FALSE;
            // call Leds.yellowOff();
        }

        if (TOS_LOCAL_ADDRESS != 0) // never turn on power mgr for base
            call PowerMgrEnable();

        //}
        return SUCCESS;
    }
#endif

    /**
     * Handle completion of sent RF packet.
     *
     * @author    Martin Turon
     * @version   2004/5/27      mturon       Initial revision
     */
    event result_t Send.sendDone(TOS_MsgPtr msg, result_t success)
    {
        atomic {
            msg_radio = msg;
            main_state = START;
            sending_packet = FALSE;
            call StdControl.stop();

        }
        call Sounder.stop(); //Leds.yellowOff();

#if FEATURE_UART_SEND
        if (TOS_LOCAL_ADDRESS != 0) // never turn on power mgr for base
            call PowerMgrEnable();
#endif

        return SUCCESS;
    }

    event result_t XEEControl.restoreDone(result_t result)
    {
                if(result) {
                        call Timer.stop();
                    call Timer.start(TIMER_REPEAT, timer_rate);
                }
            return SUCCESS;
    }
}
```

```
/**
 * Configuration file for the
 * Temperature-enabled Sensor Application with LED Display
 * Part of the TinyBrainz Project
 *
 * Modifications allow more specific control over LEDs.
 *
 * Original code by Crossbow Technology, Inc.
 * @author Brandon Ardiente, Steven Arth, Claire DuPont
 *
 */

/*
 * Copyright (c) 2004-2007 Crossbow Technology, Inc.
 * All rights reserved.
 * See license.txt file included with the distribution.
 *
 * $Id: XMTS310.nc,v 1.3.4.4 2007/04/26 20:18:03 njain Exp $
 */

/**
 * XSensor multi-hop application for MTS310 sensorboard.
 *
 * @author Martin Turon, Alan Broad, Hu Siquan, Pi Peng
 */

#include "appFeatures.h"
includes sensorboardApp;

configuration XMTS310 {
// this module does not provide any interface
}
implementation
{
   components Main,
            TimerC,
                  GenericCommPromiscuous as Comm,
                  MULTIHOPROUTER,XMTS310M, QueuedSend,
                  Voltage, MicC, PhotoTemp, Accel, Mag, Sounder, LedsC,

                  XCommandC,
         HPLPowerManagementM,
         LEDS_COMPONENT
         Bcast;

   Main.StdControl -> XMTS310M;
```

```
    Main.StdControl -> QueuedSend.StdControl;

    Main.StdControl -> MULTIHOPROUTER.StdControl;
    Main.StdControl -> Comm;
    Main.StdControl -> TimerC;

    LEDS_WIRING(XMTS310M)

    // Wiring for UART msg.
    XMTS310M.PowerMgrDisable -> HPLPowerManagementM.Disable;
    XMTS310M.PowerMgrEnable -> HPLPowerManagementM.Enable;
#if FEATURE_UART_SEND
    XMTS310M.SendUART -> QueuedSend.SendMsg[AM_XDEBUG_MSG];
#endif

    XMTS310M.Timer -> TimerC.Timer[unique("Timer")];

    // Wiring for Battery Ref
    XMTS310M.BattControl -> Voltage;
    XMTS310M.ADCBATT -> Voltage;

    XMTS310M.TempControl -> PhotoTemp.TempStdControl;
    XMTS310M.Temperature -> PhotoTemp.ExternalTempADC;

    XMTS310M.PhotoControl -> PhotoTemp.PhotoStdControl;
    XMTS310M.Light -> PhotoTemp.ExternalPhotoADC;

    XMTS310M.Sounder -> Sounder;

    XMTS310M.MicControl -> MicC;
    XMTS310M.MicADC -> MicC;
    XMTS310M.Mic -> MicC;

    XMTS310M.AccelControl -> Accel;
    XMTS310M.AccelX -> Accel.AccelX;
    XMTS310M.AccelY -> Accel.AccelY;

    XMTS310M.MagControl-> Mag;
    XMTS310M.MagX -> Mag.MagX;
    XMTS310M.MagY -> Mag.MagY;

    XMTS310M.XCommand -> XCommandC;
    XMTS310M.XEEControl -> XCommandC;

    // Wiring for RF mesh networking.
    XMTS310M.RouteControl -> MULTIHOPROUTER;
```

```
    XMTS310M.Send -> MULTIHOPROUTER.MhopSend[AM_XMULTIHOP_MSG];
    MULTIHOPROUTER.ReceiveMsg[AM_XMULTIHOP_MSG] -
>Comm.ReceiveMsg[AM_XMULTIHOP_MSG];
    XMTS310M.HealthMsgGet -> MULTIHOPROUTER;
    XMTS310M.health_packet -> MULTIHOPROUTER;

        // TinyBrainz addition
        XMTS310M.Leds -> LedsC.Leds;
}
```