

A WIRELESS SENSOR DATA FUSION FRAMEWORK
FOR CONTAMINANT DETECTION

by
Joshua Kiepert

A thesis
submitted in partial fulfillment
of the requirements of the degree of
Master of Science in Computer Engineering
Boise State University

Summer 2009

© 2009

Joshua Kiepert

ALL RIGHTS RESERVED

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the thesis submitted by

Joshua Kiepert

Thesis Title: A Wireless Sensor Data Fusion Framework for Contaminant Detection

Date of Final Oral Examination: 25 June 2009

The following individuals read and discussed the thesis submitted by student Joshua Kiepert, and they also evaluated his presentation and response to questions during the final oral examination. They found that the student passed the final oral examination, and that the thesis was satisfactory for a master's degree and ready for any final modifications that they explicitly required.

Sin Ming Loo, Ph.D.

Chair, Supervisory Committee

Robert Davidson, Ph.D.

Member, Supervisory Committee

Arvin Farid, Ph.D.

Member, Supervisory Committee

The final reading approval of the thesis was granted by Sin Ming Loo, Ph.D., Chair of the Supervisory Committee. The thesis was approved for the Graduate College by John R. Pelton, Ph.D., Dean of the Graduate College.

For My Wife

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Dr. Sin Ming Loo for both his guidance and friendship throughout the course of this research and academic pursuits.

Additionally, I would like to thank all of the people who have been involved with this research, as it would not have been possible without their significant contributions: Jon Bills, Mike Owen, Mike Pook, Derek Klein, Arlen Planting, Mike Martin, and Dereck Rasmussen.

Finally, I would like to thank my family for their love and support during this process and always, and in particular, my wife Araya, who has supported and cared for me despite the long hours I have been locked away working on my academic career. My thanks seem hardly sufficient to convey my heartfelt appreciation for her love and support, which, in no small part, have been responsible for allowing me to complete this thesis.

This work is funded by FAA Cooperative Agreement No. 04-C-ACE-BSU and 07-C-RITE-BSU¹.

¹ Although the FAA has sponsored this project, it neither endorses nor rejects the findings of this research. The presentation of this information is in the interest of invoking technical community comment on the results and conclusions of the research.

ABSTRACT

A Wireless Sensor Data Fusion Framework for Contaminant Detection

Joshua Kiepert

Master of Science in Computer Engineering

In the search for more effective instruments to collect data for the identification of threats to security, health, and safety, new tools must be designed to meet the challenges of a diverse set of possible applications. The extensive range of potential applications raises the need for a general purpose system capable of addressing a wide variety of deployment environments. This thesis focuses on a wireless sensor network framework for collecting environmental data in an effort to develop a sensing solution that fits within many design spaces. The framework includes reconfigurable wireless sensor node hardware, firmware, and software for interfacing sensor networks for upstream data aggregation and sensor data fusion. The wireless sensor modules utilize mesh network architecture to allow low power radios to be effective even with low sensor module dispersion density, or in environments that have obstructions which prevent line-of-sight communications. In the current implementation, the software is designed to allow a computer to be used to monitor all sensor module activities as data is collected, request information as needed, and forward collected data to a database system for further analysis. It also supports software modules to allow different sensor data fusion and analysis algorithms to be applied to the collected data in real-time.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	v
ABSTRACT.....	vi
LIST OF FIGURES	x
LIST OF ABBREVIATIONS.....	xii
CHAPTER 1: INTRODUCTION	1
1.1 Wireless Sensor Networks	1
1.2 History of Sensor Data Fusion	2
1.3 General Purpose Hardware	3
1.4 Software for Sensor Data Fusion	3
1.5 Contributions.....	4
1.6 Overview.....	6
CHAPTER 2: PREVIOUS WORK AND EXISTING TECHNOLOGY.....	7
2.1 Previous Research.....	7
2.2 Existing Technology.....	9
2.2.1 Crossbow Technologies.....	9
2.2.3 Firefly WSN Platform	10
CHAPTER 3: SENSOR DATA FUSION FRAMEWORK.....	11
3.3 Sensor Node (level 1 to 4).....	13
3.4 Communication and Interface (level 5 to 6)	14
3.5 Database, Visualization, and Fusion (level 7 to 9).....	14

CHAPTER 4: HARDWARE DESIGN	16
4.1 General Purpose Sensor Modules	16
4.2 Wireless Capabilities.....	16
4.3 Modular and Reconfigurable Design	17
4.4 Processor	18
4.5 Power	19
4.6 Sensor Integration	19
4.7 Communications	20
4.8 Data Storage and Transmission.....	21
4.9 Time Management.....	23
4.10 Systems Integration.....	23
CHAPTER 5: SOFTWARE DEVELOPMENT	26
5.1 Sensor Module Firmware.....	26
5.1.1 Implementation Details	26
5.2 Computer Software	34
5.2.1 Plotting	34
5.2.2 Data Sink.....	35
5.2.3 Sensor Module Control	35
5.2.4 Real-Time Data Fusion	37
5.2.5 Database	39
5.2.6 Implementation Details	39
5.3 Sensor Network Simulation	42
CHAPTER 6: TIME SYNCHRONIZATION.....	45

6.1 Necessity of Time Synchronization	45
6.2 Network Time Protocol.....	45
6.3 Single-Pulse Synchronization	46
6.3.1 Performance Characteristics.....	47
6.4 Reference Broadcast Synchronization	51
6.5 Summary	53
CHAPTER 7: SENSOR DATA FUSION AND APPLICATIONS	54
7.1 Sensor Data Fusion	54
7.2 Applications	59
CHAPTER 8: CONCLUSIONS AND FUTURE WORK	62
8.1 Summary and Conclusions.....	62
8.2 Future Work	63
8.2.1 Hardware Improvements.....	63
8.2.2 RBS Implementation.....	64
8.2.3 Advanced Sensor Node Operating System	64
8.2.4 Field Testing.....	66
REFERENCES.....	67

LIST OF FIGURES

Figure 1: Wireless Sensor Module Design Space	5
Figure 2: Star Network Configuration	8
Figure 3: Previous Sensor Node Prototype	8
Figure 4: Overall System Architecture.....	11
Figure 5: Wireless Sensor Data Fusion Framework.....	13
Figure 6: Sensor Module System Board	18
Figure 7: Example Identifier and Measurement Strings	22
Figure 8: Original Sensor Module Design, (a), vs. New Design (b).....	24
Figure 9: Latest Sensor Module Hardware Design.....	25
Figure 10: Firmware Flow of Control (Simplified)	27
Figure 11: Firmware Layered Architecture	28
Figure 12: Sensor Table Structure.....	29
Figure 13: Sensor Identify Function	31
Figure 14: Sensor Measure Function	32
Figure 15: Top Level Identify and Measure Functions	33
Figure 16: BSU Sensor Monitor Data Plot	35
Figure 17: Two-Way Communication and Time Synchronization.....	36
Figure 18: Real-Time Sensor Data Fusion Dialog	37
Figure 19: Average Temperature Fusion Algorithm.....	38
Figure 20: Sensor Simulation Console Output	43

Figure 21: Sensor Simulation Graphical View.....	43
Figure 22: Single-Pulse Synchronization Test Configuration.....	48
Figure 23: Single-Pulse Synchronization Timing	49
Figure 24: Time Phase Error between Sensor Nodes.....	50
Figure 25: JDL Data Fusion Model [17].....	55
Figure 26: Characterization of an Entity through Data Fusion	58
Figure 27: Simulator Visualization of a Wave Front.....	60
Figure 28: Sensor Output Patterns during Simulation	61

LIST OF ABBREVIATIONS

ADC	Analog-to-Digital Converter
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BSUSM	Boise State University Sensor Monitor
CO	Carbon Monoxide
CO ₂	Carbon Dioxide
CSV	Comma Separated Values
DoD	Department of Defense
EEPROM	Electrically Erasable Programmable Read-Only Memory
FAA	Federal Aviation Administration
GPIO	General Purpose Input/Output
GPS	Global Positioning System
I/O	Input/Output
I ² C	Inter-Integrated Circuit
IDC	Insulation Displacement Connector
IFFN	Identification-Friend-Foe-Neutral
ISM	Industrial, Scientific, and Medical
JDL	Joint directors of Laboratories
MMSP	Master Synchronous Serial Port
NTP	Network Time Protocol
PDA	Personal Data Assistant
PLL	Phase-Locked Loop
PWM	Pulse Width Modulation
RBS	Reference Broadcast Synchronization
SD	Secure Digital
SPI	Serial Peripheral Interface
SQL	Structured Query Language
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver/Transmitter
WSN	Wireless Sensor Node
WWVB	National Institute of Standards and Technology time synchronization radio station

CHAPTER 1: INTRODUCTION

1.1 Wireless Sensor Networks

As science advances there is a continual need to provide new tools to study the world around us. Wireless technology, sensor data fusion, and microelectronics are just a few of the components that can be used to form tools, which may provide a large range of possible advancements. Whether it is an early warning system for the detection of dangerous chemicals or a portable sensing system for diagnosing problems within complex machinery, the possible applications seems limited only by one's imagination.

Wireless sensor networks are comprised of many individual wireless sensor nodes (WSN). Each node is a small embedded system that includes a microcontroller, sensors, and a radio system that allows the nodes to communicate with each other and the outside world. This thesis focuses on a wireless sensor network framework for collecting environmental data in an effort to develop a sensing solution that fits within many design spaces. It includes the design and implementation of a highly portable, reconfigurable, and wireless sensor network for collecting environmental data over large areas, and in particular, the “back-end” interfacing, delivery, and storage such that different types of sensors can be interfaced to the sensor modules. With the use of different sensors based on different (orthogonal sensing technologies) detection technology, the data collected can be transferred to a central location and provide enough information for characterization and data fusion processing.

This system, additionally, utilizes mesh network architecture to allow low power radios to be effective even with low sensor dispersion density or in environments that have obstructions which prevent line-of-sight communications. The software framework is designed to allow a computer to be used to monitor all sensor activities as data is collected as well as allowing a computer to request information as needed.

1.2 History of Sensor Data Fusion

Sensor data fusion, as it applies to this thesis, is the process by which many sensor inputs are combined and processed by algorithms to provide an improved representation of the data. “Improved” may mean more accurate, more complete, or more reliable. The goal of sensor data fusion is to provide a dataset that can more easily be processed and comprehended by human observers.

Sensor data fusion research was started by the U.S. Department of Defense (DoD) when it began funding research for many different applications. Some of the research projects included: automatic target recognition using many diverse sensors (radar, satellite, etc) in concert for identification-friend-foe-neutral (IFFN) systems, target tracking systems, situation assessment, and others. One of the results of this research was the formation of the U.S. Joint Directors of Laboratories (JDL) Data Fusion Working Group in 1986 to develop common terminology to describe different processes within data fusion applications. This eventually led to the design of a data fusion model that could be used as a common basis for discussion of many aspects of data fusion tools and processes. Work in data fusion has since expanded to include many non-military applications as well, some of which include robotics, condition-based maintenance of structures or machines, medical diagnoses through medical imaging, and environmental

monitoring [1].

There are several ways data fusion can be used in concert with multi-sensor systems. Data fusion algorithms may 1) directly fuse sensor data to provide data aggregation and thus reduce large data sets to a more easily managed collection, 2) fuse sensors to provide a virtual sensor entity on which higher level queries can be leveraged, or 3) utilize a wide range of common sensor inputs to make inferences or decisions that would not otherwise be possible by analyzing any of the sensors individually.

The work of this thesis focuses on the design and implementation of a framework (hardware and software) that can provide a means to apply data fusion algorithms both for data aggregation and for making inferences and decisions based on a wide range of data from wireless sensor networks. With these goals in mind, it is necessary to design the framework to support many different types of sensors and provide as much flexibility as possible with regard to managing the data collected from sensor networks.

1.3 General Purpose Hardware

With universal applications in mind, it was important to build a system that utilizes standard device communication protocols, device power supply voltages, data storage formats, and standard general communication protocols to interact with outside computer systems. In addition to the general purpose electronic design, a layered and highly abstracted design was required in the embedded operating software to enable control of any type of hardware that may be attached to the system.

1.4 Software for Sensor Data Fusion

By its very nature, sensor data fusion may require a large amount of processing to

identify abnormal events and help reduce false positives. The system described in this thesis is designed to store received data in a database, which allows high powered computing systems to analyze the collected data as it becomes available. With the data available and accessible at one location, diffusion and pattern algorithms can process the data in near real-time, which may aid the detection of contaminants.

Aside from the hardware and embedded software, the work of this thesis has also been the development of software for computers to not only serve as a data sink for networked sensor modules, but also provide a means to do real-time data analysis. The software also allows fusion of sensor data to help users of the system to better identify important information the sensor system has collected.

This goal has in turn resulted in the need for simulation software to help better characterize the fusion algorithms employed without setting up experiments, which would be difficult to control particularly when the testing environment must be spatially large.

1.5 Contributions

The primary contribution of this work is the implementation of a general purpose framework for sensor data fusion applications. Much of the scientific community has focused on either the theoretical design of such a system or on creating very small sensor modules that are capable only of managing a limited collection of sensors, the idea being to minimize the cost of the sensor modules to allow for deployment of the maximum number of sensor modules. The other end of the spectrum is to utilize a few highly capable sensor modules that have a limited number of high performance (and thus expensive) sensor modules to accurately monitor a modest area. As was pointed out in

[2], with a sufficient number of low resolution sensors, a larger coverage area is achievable at the same cost, which can offset the benefits of high resolution sensor modules. Our focus, in contrast, has been to look at the middle ground of this design space. Is it possible to provide a sensor module design and supporting framework that offer an inexpensive solution that has good capabilities? We believe, and will show, that it is possible to achieve most of the benefits of both inexpensive and expensive, high resolution sensor modules. Figure 1 shows a graphical representation of the design space we are targeting.

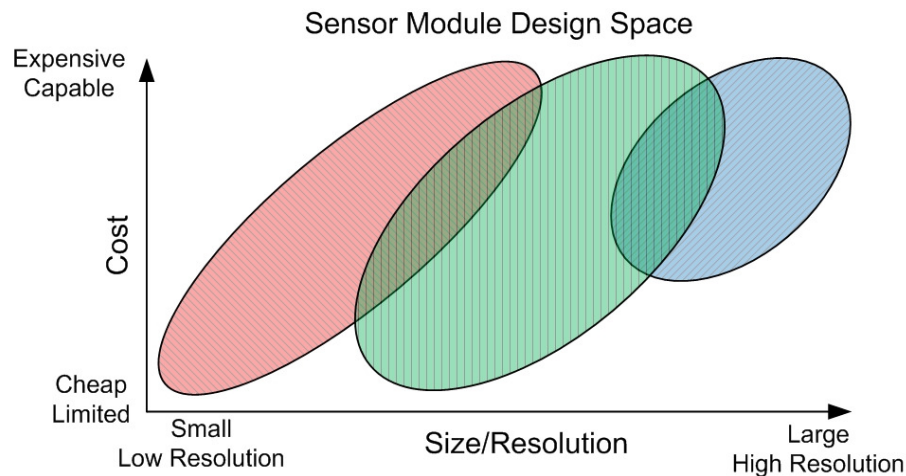


Figure 1: Wireless Sensor Module Design Space

As depicted in Figure 1, the design space of sensor module hardware falls into several categories. In the far left area we have designs that range from limited and inexpensive to expensive and capable, with few sensors. The far right area represents sensor nodes that are expensive, possibly with several high resolution sensors. Our design is targeted to the center area of the figure, in that it is designed to be easily configured for applications that require either only low resolution sensors or high resolution sensors. This provides a means to adapt the set of sensors installed on a sensor

node to meet the specific application requirements without re-engineering the hardware or firmware of the sensor module².

1.6 Overview

In the following chapters we will discuss many of the topics surrounding sensor data fusion platforms as it applies to our work. In Chapter 2 we will describe some of our previous work in sensor network technology as well as some of the existing technologies that are under development in the sensor networking field. In Chapter 3 we will describe our sensor data fusion framework in terms of the design parameters that were considered and the general architecture of the system that we have designed. Chapter 4 outlines the design and implementation of our sensor module hardware in terms of each feature that we believe are important to any general purpose sensor system. In Chapter 5 we discuss the software development both for the embedded firmware that resides on the sensor modules and computer software that serves as a link between sensor networks and the outside world. Chapter 5 also discusses some simulation software that has been developed in an effort to improve our ability to reliably test data fusion algorithms. In Chapter 6 we discuss the need for time synchronization and some of the algorithms that may be used to accomplish synchronization. Chapter 7 discusses how data fusion algorithms can be utilized to manage the large amount of data collected by sensor networks or extract information that could not be otherwise gleaned from individual sensors. Finally, Chapter 8 offers up some conclusions from this work as well as a brief description of some possibilities for future work.

² Note: the terms “sensor node” and “sensor module” may be used interchangeably.

CHAPTER 2: PREVIOUS WORK AND EXISTING TECHNOLOGY

2.1 Previous Research

Previous to this research much work was done to design a modular, flexible, and reconfigurable hardware platform for taking sensor measurements. This took the form of a small battery-powered device which could be configured with a wide range of environmental sensors. The focus was to design a stand-alone sensor module with the capability to reconfigure the set of sensors on a sensor module with minimal re-engineering. The sensor data storage medium was removable Secure Digital (SD) flash memory card. While there was hardware support for wireless communication, this feature was not fully explored [3], [4].

In the previous work, the sensor modules were designed to act as standalone data loggers that could be configured with a wide range of sensors. The data logging task did not require time synchronization or wireless capabilities as each module worked independently and data points collected only needed to be assigned a common time reference with regard to all other measurements taken on a particular logger for the data to easily be correlated. Our task was to investigate how we could leverage the previous low cost sensor module hardware in a more advanced system in which each module was a member of an ad-hoc wireless network, and therefore able to collect information about the environment that would not be possible with an individual sensor module. Figure 2 and Figure 3 show the network architecture and hardware design, respectively, of the

sensor modules that were developed in [3].

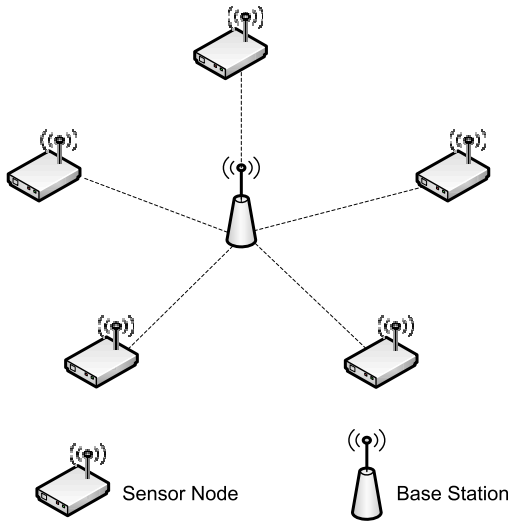


Figure 2: Star Network Configuration

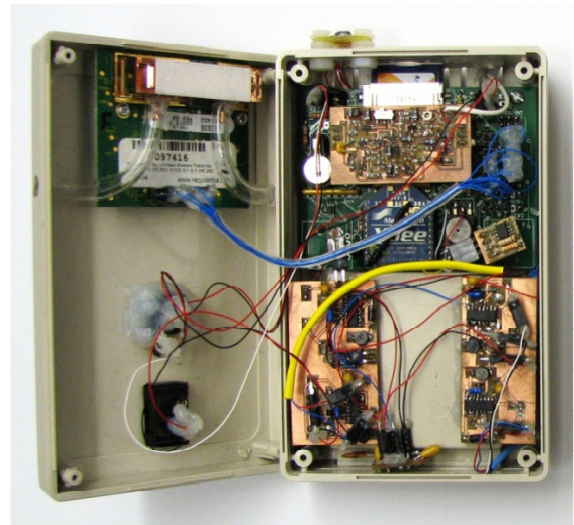


Figure 3: Previous Sensor Node Prototype

In a star network, as depicted in Figure 2, there is a fundamental restriction on how far the sensor nodes may be from the base station. This range is limited by the power of the transmitters and the size and type of obstructions between an individual sensor node and the base station. For our new design, we needed a more robust network architecture that could perform well with obstructions to line-of-sight, and operate with larger distances between sensor nodes. As seen in Figure 3, the previous sensor node design had individual circuit boards for the various components and many point-to-point connections between circuit boards. This architecture, while reconfigurable, still requires re-engineering to change the sensor configuration, and is prone to failure in harsh environments due to the wiring scheme.

2.2 Existing Technology

While there are several commercial and research sensor systems available that provide sensor hardware and networking capabilities, e.g. [6], [7], and [8], there are few systems that provide a complete sensing solution targeted for sensor data fusion applications. There has been development of interface frameworks for sensor networks, e.g. [9], but few with sensor data fusion applications in mind.

Perhaps some of the most notable existing systems are found with Crossbow Technologies Inc [6] and a system that has been developed at Carnegie Mellon University (CMU) [10]. Each of these systems provides some of the functionality that is implemented in our framework. Therefore it is important to look at how these systems compare with the work discussed in this thesis.

2.2.1 Crossbow Technologies

Crossbow Technologies provides a range of wireless sensor modules that are capable of forming wireless mesh networks with a variety of sensors available to be connected to the modules (or motes, as they are called). In general, their systems provide support for one sensor per sensor module. They offer sensor boards with accelerometers, light detectors, pressure, temperature, global positioning system (GPS), sound, and magnetic field sensors [11]. Crossbow offers a software stack for operating their sensor systems and interfacing the sensor networks to a general network infrastructure such as Ethernet. While their systems are capable, they do not offer a framework for managing the data acquired from sensor network with sensor data fusion applications in mind. These systems do not offer a large storage medium for logging data when a module is not connected to a network (though some media may be retrofitted). Additionally, their

systems are cost-prohibitive when compared with our solution.

2.2.3 Firefly WSN Platform

CMU's FireFly platform offers a similar solution in terms of the hardware and sensor options though this system is still targeted toward a limited number of sensors per mote. This system does employ an out-of-band time synchronization mechanism through the land-based atomic clock broadcast signal (WWVB) though it is capable of in-band synchronization. These nodes have been configured with sensor boards carrying several sensors such as light, temperature, audio, passive infrared motion, dual axis acceleration, and battery voltage sensors. The FireFly platform also includes models with mini Secure Digital cards for facilitating local data storage [12]. CMU has also implemented an advanced embedded operating system for the Firefly platform (known as Nano-RK), that manages sensor measurements and time synchronization. Some of the primary differences with this system are the set of sensors that have been integrated and a hardware architecture that is closely tied to the wireless network radio as well as hardware-assisted time synchronization.

The hardware developed in this thesis does not offer capabilities beyond those that are currently available; it does provide a unique set of capabilities that are particularly conducive to sensor data fusion applications. Many sensor systems, such as [6] or [7], approach sensor networks from the perspective of having many small sensor nodes with limited capabilities. Our system employs inexpensive, but capable sensor modules that can be outfitted with many sensors and function as a standalone sensing module or as a member of a larger network of sensor modules.

CHAPTER 3: SENSOR DATA FUSION FRAMEWORK

There are many possible system architectures that may be used to collect and process sensor data. The architecture of our system was chosen based on the idea that it should be flexible enough to allow other systems to utilize the collected sensor data, as well as allow on-site processing of the data (when the algorithms applied do not require significant computational power). Figure 4 shows the overall architecture of the system that our framework has been designed to achieve.

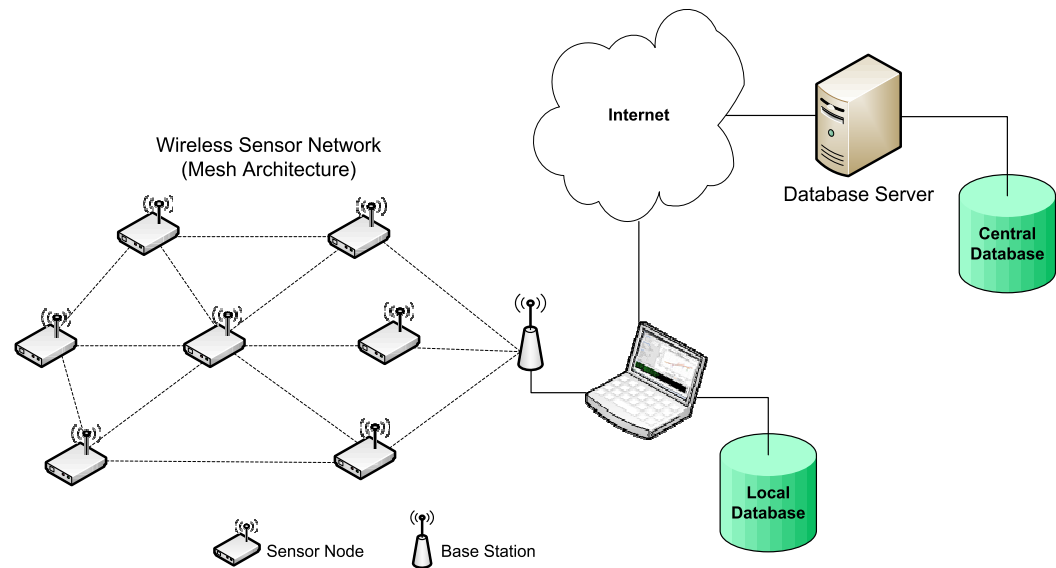


Figure 4: Overall System Architecture

As seen in Figure 4, there are several primary components to this system. At the front end, we have a sensor network that is capable of forming mesh networks. Each sensor node in the network, which may have a variety of sensors, collects data and transmits it to a base receiver station. The base station is connected to a small computer

system that may interact directly with the sensor network, both monitoring collected data, and potentially forwarding the data to a central database for further processing and analysis. The computer system connected to the base station radio provides the gateway for data to be moved from the sensor network to a location that allows data fusion algorithms to be applied. The computer system near the sensor network may also be used to directly control the sensor network and provide application of basic fusion algorithms that do not require significant computational power. Our wireless sensor data fusion framework contains each of these core elements for building a wireless sensor data fusion system. The framework includes both hardware and software, which makes it possible to insert data fusion algorithms to process the sensor data or aggregate data to a central database for high-powered analysis. This framework has a highly flexible design such that different sensors can be integrated into the sensor nodes without re-designing the overall system. The general objective of this design is to provide an implementation of a framework (hardware and software) such that once a sensor has been selected for a particular sensing application the system can be deployed quickly, and sensor data fusion algorithms can be inserted into the framework to analyze the data as desired. The framework is shown in Figure 5. This diagram shows features that may be classified into three categories: (i) sensor node, (ii) communication and interface, (iii) database, visualization, and fusion.

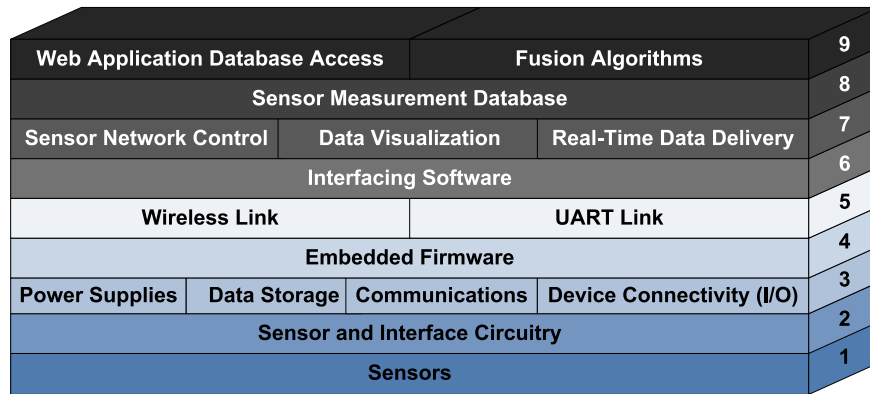


Figure 5: Wireless Sensor Data Fusion Framework

3.3 Sensor Node (level 1 to 4)

Sensor nodes with the contaminant detectors are the frontend of a wireless sensor fusion system. It is the element of the system that passively or actively measures the contaminant levels and reports the findings in a timely manner. As shown in Figure 5, the sensor node provides circuitry to interface to sensors as well as power sources and power regulation for sensors. Once data is collected from a sensor, it is processed, stored, and transmitted. The data transmission requires connectivity to the server (or sink node) through a wireless link.

To be effective, the sensor modules were required to meet many design constraints. The primary objectives were to maintain a small physical size, have limited power requirements, and be highly reconfigurable. Additionally, the sensor modules needed to provide high connectivity for connections with many sensors, and be visually inconspicuous. The sensor modules were required to not only send collected data to a remote location wirelessly, but also to enable the storage of collected data locally in case of network failure or applications that require limited network activity and thus send data only after long periods without wireless connectivity.

To meet the connectivity and re-configurability constraints, the hardware was required to have many input/output (I/O) ports and support for many communications protocols, to enable connections with numerous sensors as well as providing internal power supplies to meet a variety of sensor requirements.

As with the hardware, the sensor module firmware was required to be highly reconfigurable in that it needed to have an architecture that minimizes re-engineering when adding or removing sensors from the sensor module. To achieve this objective, the firmware design was required to exhibit data coupling and a functionally cohesive architecture.

3.4 Communication and Interface (level 5 to 6)

The communication link and interface are the critical infrastructure that delivers the sensed data to the proper destination. For sensor data fusion (residing in the backend to characterize and analyze the data) to work effectively, determining when the data arrives is critical. The delivery of sensor data depends on reliable wireless communication channels. The wireless hardware was required to have a reliable, redundant, and robust network architecture formed between sensor modules (e.g. mesh network architecture) to meet wireless connectivity objectives. The base station might not be reachable directly (one hop away), but the data could be delivered through intermediate nodes. More importantly, having multiple intermediate nodes will guarantee delivery of the data to the base station no matter what happens to any single node.

3.5 Database, Visualization, and Fusion (level 7 to 9)

Database, visualization and data fusion is the backend – where the heavy duty

processing happens. The sensed data (usually in significant quantities) is stored on a database server such that algorithms can be applied to “make sense” of the data. The raw data can be visualized, but with its intrinsic volume, visualization may be difficult; that is one area where sensor data fusion algorithms can help reduce the data set, allowing further attention to be placed on the reduced set. If effective analysis is to be done on the collected data, it must be organized such that relationships can easily be determined.

Both centralized and distributed database architectures have benefits. It is our belief that the backend should have flexibility, i.e., new fusion algorithms can be written to process the data without fundamental changes to the system. The system should provide “hooks” such that a new algorithm can be used to analyze the data. The timely arrival of data to the database server is important; however, one will need to define the “real-time” expectation of delivery. The greater the responsiveness needed, the greater the hardware and design costs to implement the system.

Once the data has been analyzed, the system will need to deliver the results to someone in a timely manner. The result could be a decision (e.g. yes, there is contaminant; or no, nothing is out there) or a series of plots and graphs for human analysis.

The remaining portion of this thesis describes the prototype and implementation of this framework. In particular, Chapter 4 discusses the design and implementation of the hardware associated with levels 1-4 of the framework. Chapter 5 discusses the design and implementation of the software, which encompasses levels 5-9 of the framework.

CHAPTER 4: HARDWARE DESIGN

4.1 General Purpose Sensor Modules

For this research, the sensor module hardware was further refined and the wireless communication capabilities were expanded to include mesh network architecture. Computer software was also developed to allow coordination of data collection and provide a facility to fuse the data collected across the network of sensor modules. This software was also designed to store the collected data in a centralized database for post processing.

4.2 Wireless Capabilities

There are many cases where it is difficult or impractical to effectively determine the state of an environment from a single measuring unit. When the environment is large or its conditions vary greatly over space or time, it becomes necessary to use multiple measuring units to provide enough sensor density to gain a full perspective of the environment in question. It is in these cases where a network of measuring units becomes important. A network allows the measured data to be correlated with each measuring unit in the environment and transmitted to a centralized database for detailed analysis. With the prevalence of low-power and inexpensive wireless communication devices, the creation of high density sensor networks is more easily achieved than it has been historically.

There are two main network architectures employed by our sensor system: star

and mesh. The star architecture requires that all nodes connect directly to a single master node. This means that there is a fundamental limitation on how many members may be part of the network, as well as the maximum spacing between nodes. This also requires that there be an unobstructed “view” to the master node. Our system will form this type of network provided that all remote nodes are within range of the master node; however, it may dynamically change to form a mesh architecture, if obstructions or distance begin to interfere with a remote node’s communication with the master node.

Mesh architecture, in contrast, has fewer constraints with regard to the layout of the remote measurement units. This architecture allows for multi-hop communication; thus the master node may be located anywhere among the remote units and need only be within range of any one of the remote units. Any messages addressed to the master will be relayed as required to deliver the message to the master. This architecture is also far more robust in constrained environments where line-of-sight communication to all nodes directly is not possible. The dynamic formation of the different network types is controlled by the radio hardware. The radio used in our system has an embedded processor paired with a microcontroller, which manages all of the low level communications with the radio hardware, including network formation.

4.3 Modular and Reconfigurable Design

Utilizing the previous work done in [3] and [4], the wireless sensor units were redesigned with a more modular and reconfigurable design. This was accomplished by the design of a general purpose system board that provides a microcontroller, real-time clock, Secure Digital flash memory card, three to four step-up/down power supplies, and

many digital and analog I/O pins. Figure 6 shows the main system board design.

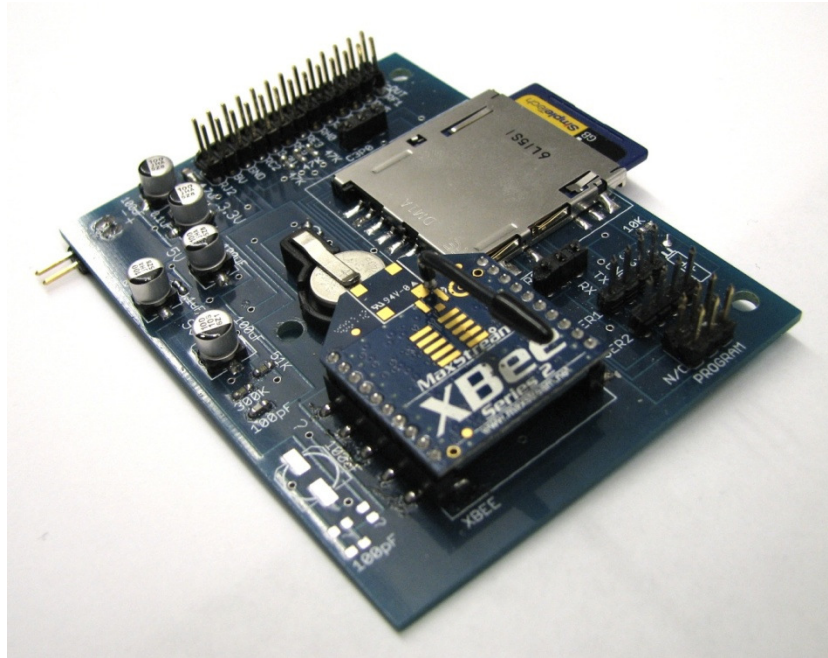


Figure 6: Sensor Module System Board

4.4 Processor

The processor used in this system is a Microchip PIC18F8722 8-bit microcontroller. This platform offers a generous amount of program and data space for embedded applications with 128 KB Flash, 4 KB SRAM and of 1 KB EEPROM. In addition it offers 70 I/O pins, sixteen of which can function as inputs to a 10-bit analog-to-digital converters (ADC). For communication with various devices the unit has several facilities including two RS232 universal asynchronous receiver/transmitters (UART), and two master synchronous serial ports (MSSP) that support 2/3/4 wire serial peripheral interface (SPI) and inter-integrated circuit (I²C) master/slave functionality. It also offers three capture/compare/pulse width modulation (PWM) modules and four hardware timers [5]. All of the features described above give the microcontroller significant versatility so that it may easily be adapted to a wide range of sensing

applications.

This microcontroller not only provides a wide range of protocols and I/O options, but reasonable computation power as well. On this system, the microcontroller was set up to run at 8 MHz, but the microcontroller has an internal phase-locked-loop (PLL), which allows it to use the 8 MHz external crystal and internally run at four times the external crystal frequency. The microcontroller is rated to run up to 40 MHz by use of a 10 MHz external crystal and the internal PLL [5].

4.5 Power

There are many common voltages for sensors. Some of the most common are 5 V and 3.3 V; however others may be necessary. The system board (motherboard) of the module provides four power supplies running at 3.3 V, 5 V, 9 V and a custom supply that may be configured at build time. The system uses two types of DC-DC converters: a Maxim MAX642 and MAX710. The MAX642 is rated to output up to 18 V at 450 mA, whereas the MAX710 is rated to output up to 11.5 V at 700 mA. The system board has space for two MAX710s and two MAX642s. Both of these supplies have efficiencies of over 80% [13], [14]. In the current hardware configuration, the two MAX710 supplies are set up to output 3.3 V and 5 V. One of the MAX642 chips is configured to output 9 V, whereas the fourth supply is not currently used.

4.6 Sensor Integration

To provide a means to easily change the sensor set configured on the system, a secondary board or “breakout board” was designed for sensor integration with the main system board. The breakout board currently provides an interface for seven sensors:

carbon dioxide (CO₂), carbon monoxide (CO), temperature, relative humidity, barometric pressure, GPS, and sound intensity. The hardware has also been adapted to other form factors, which allow additional sensors to be connected externally to the enclosure. The current sensor set was chosen as a means to test the overall framework, as the suite provided several standard sensors for general applications.

4.7 Communications

While the microcontroller supports many communication protocols, the primary protocol used for external communication is UART. This provides a standard protocol that interfaces with computers as well as other devices. The microcontroller used in the system offers two UARTs. One UART is used for controlling software system configuration via a computer while the second is used for wireless communication through use of a ZigBee® modem. The second UART has also been used with a Bluetooth communication module to add link capabilities with PDAs to display sensor measurements.

Wireless communication is achieved with Digi XBee® ZigBee® modems that use the Industrial, Scientific & Medical (ISM) 2.4GHz band and support both the IEEE 802.15.4 standard and proprietary DigiMesh™ protocols [15]. These units provide a simple UART modem interface to the microcontroller, and help offload much of the communications processing by managing nearly all of the network formation and routing needed for wireless communication. The XBee® modems automatically create ad-hoc star or mesh networks at power-up and dynamically reroute packets when a destination node becomes out of range for direct communication.

4.8 Data Storage and Transmission

Local data storage is accomplished with an SD card reader built into the main system board. This media was chosen based on its availability, compatibility, and form factor. The small size of SD cards results in minimal space requirements on the system board. Additionally, media card readers and laptop computers commonly support SD media. The current system firmware supports SD cards up to 2GB, which would allow for approximately four years of data collection without removing the card (assuming measurements are taken every 30 seconds).

The data from sensor measurements is stored on the SD card in a human readable text format. This not only allows users to easily view the data collected, but it results in simple programming to load data files into databases or generate plots. The same format is also used in wireless transmission of the sensor measurements. Each sensor measurement string is a collection of key-value pairs containing information such as the identification number of the sensor module on which the measurement was collected, the sensor identification number within the sensor module, the raw sensor reading, converted sensor reading, and a time stamp of when the measurement was taken. Additional strings stored in the data files identify the type of sensors, each sensor's measurement units, model number, and description. These strings provide a way to limit how much data is stored for each measurement. By separating out the sensor information from the measurement data we prevent repeatedly transmitting or storing sensor information with each subsequent sensor measurement. Figure 7 shows the data format used by the sensor modules.

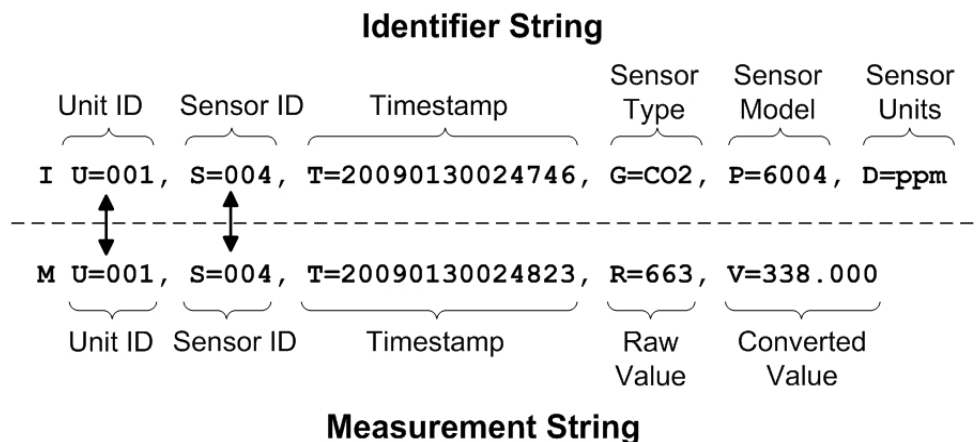


Figure 7: Example Identifier and Measurement Strings

As seen in Figure 7, the identifier string contains information about the sensor on a particular sensor module, whereas the measurement string contains information about a measurement from a sensor on a sensor module. The use of two packet types helps reduce the amount of data sent with each sensor measurement. The identifier strings are sent only at startup or when requested. This allows the measurement strings to remain small, and only contain the data unique to the measurement. The common sensor node global unique identification (GUID) number and sensor identification (SID) numbers are identical between a measurement strings and the identifier string for a respective sensor. This allows the two (identifier and measurement) to be related, and it avoids the need to send the sensor information (type, units of measurement, etc) with every measurement.

The data transmitted has the same format as the data stored to the flash memory on the system. The plain text format requires more data to be transmitted than if formatted into binary packets, but it offers significant advantages in terms of versatility. The string structure of key/value pairs allows for only minor modification to the transmitter/receiver code to change the data fields transmitted or stored.

4.9 Time Management

Time keeping is an important part of any data logging device. It is particularly important for a system that must correlate measurements among distinct, independent modules, such as a multi-sensor data fusion system. While it is not strictly necessary to have perfect synchronization among the sensor modules, it is necessary to have the system self-consistent in that all sensor modules agree on the ordering of the events recorded by the system. The sensor system described in this paper pushes most time management control to software. We will discuss time synchronization in more detail in Chapter 6.

As for the time-keeping hardware, the system utilizes a Maxim DS1339 real-time clock. This chip utilizes an external 32.768 KHz crystal oscillator and is controlled through an I²C interface. It offers very low current operation (~450 μ A) and accuracy which depends on the crystal used [16]. The typical crystal oscillators used have an accuracy of +/- 40 ppm which means that the crystal has a potential error that could result in up to +/- 10 minutes per year depending on the temperature variations.

While the DS1339 maintains real time, an internal hardware timer is used to manage system events. The internal timer utilizes an additional external 32.768 KHz crystal to maintain system time. This clock is synchronized with the DS1339 at startup and every 24 hours to insure accuracy.

4.10 Systems Integration

There were many areas where the current sensor modules were improved over the original design. Most of the changes were related to minimizing the amount of re-engineering required for adding or reconfiguring the suite of sensors attached to the

sensor modules, but other changes were made to make the sensor system more robust and modular. Figure 8 shows a comparison between the original sensor module (left) and an early prototype (right) of the latest sensor module. Both of the sensor modules shown in Figure 8 have the same sensor suite and functionality.

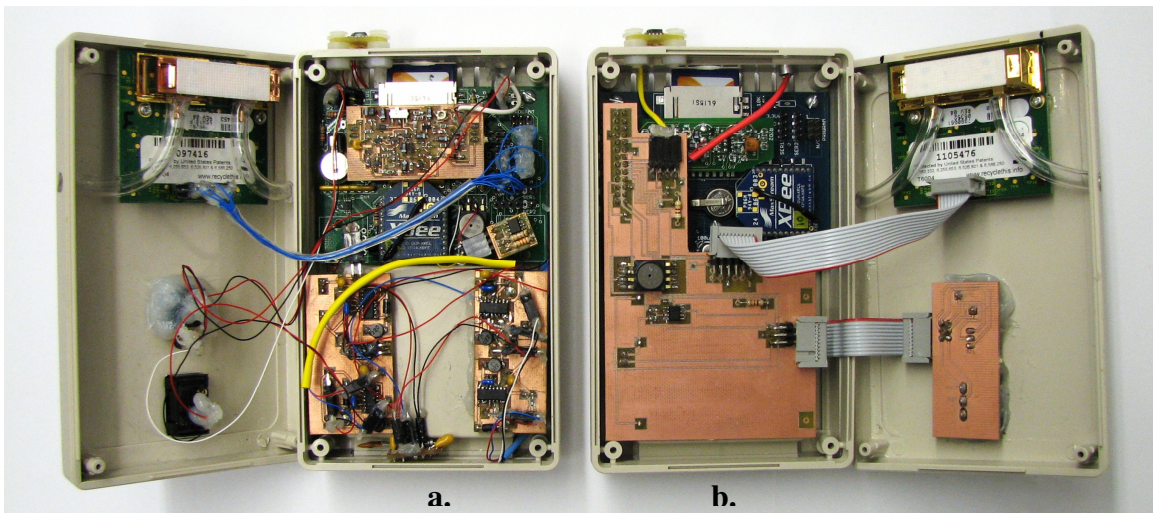


Figure 8: Original Sensor Module Design, (a), vs. New Design (b)

From Figure 8 we see that the system cabling was substantially improved by using insulation displacement connector (IDC) cables and integrated circuit boards (b). The original prototype, (a), has significantly more point-to-point wiring, and individual circuit boards for each power supply. The various power supplies were consolidated to reside on the main system board rather than relying on three discrete power supply boards. Additionally, as discussed previously, the sensors are now consolidated to a modular circuit board that provides all of the available voltages and communications lines from the main system board. Figure 9 shows one of the latest versions of the updated sensor module.

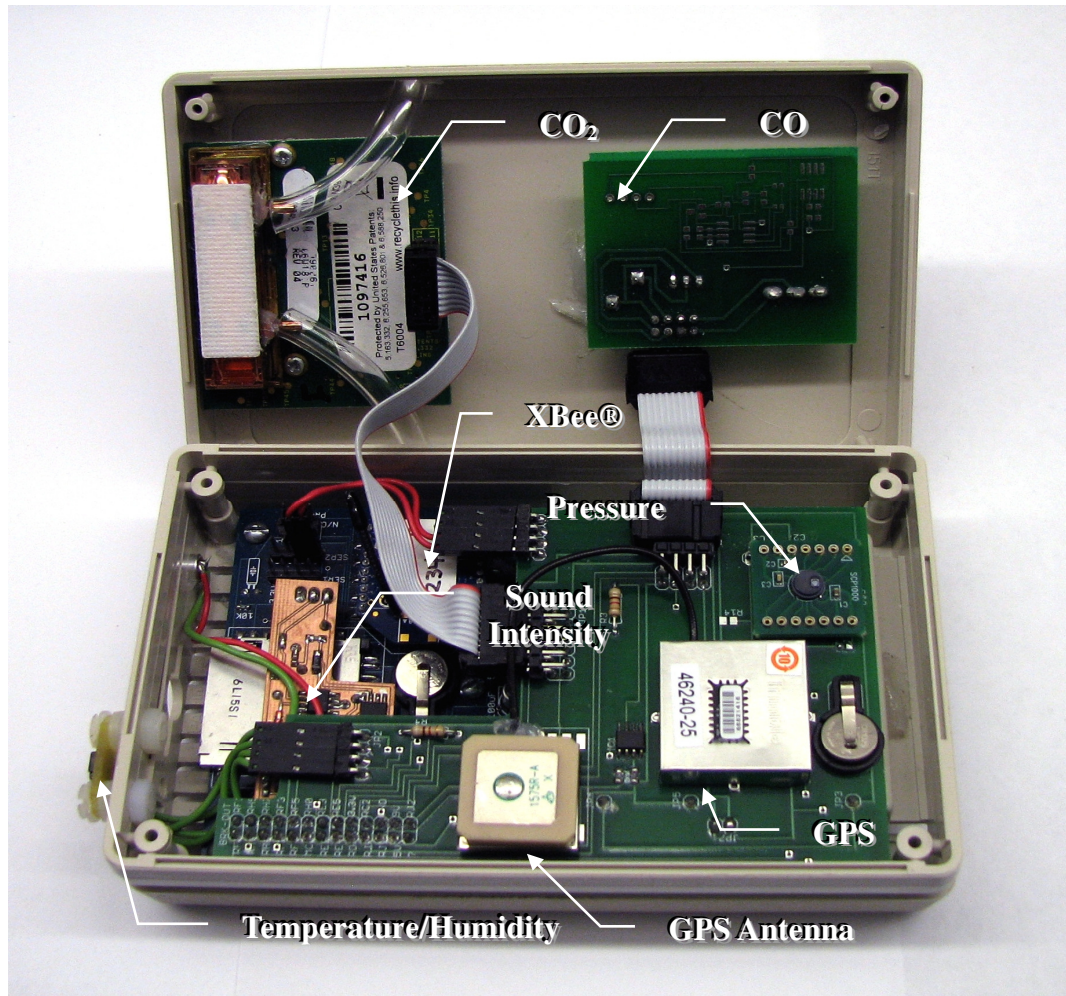


Figure 9: Latest Sensor Module Hardware Design

As seen in Figure 9, the latest version of the sensor module takes advantage of the extra space garnered by the more modular design. The newer breakout board was designed to allow the addition of a more accurate barometric pressure sensor (which contains an integrated temperature sensor) as well as a GPS unit. The sound intensity sensor was also improved, redesigned to include a wider range of operation and a better form factor for integration with the system board.

CHAPTER 5: SOFTWARE DEVELOPMENT

5.1 Sensor Module Firmware

As the sensor modules must be battery-powered and have limited resources, it is important to limit the amount of computation that occurs locally. In addition, it is important that the software be designed such that it is easily reconfigurable to allow for a wide range of sensors to be connected to the system. This was generally accomplished by a layered and modular design. At the lowest level, the code for each sensor must provide a consistent application programming interface (API) that “hides” the low level hardware communication from the higher levels of the software. It is this area that will need to be created to add new sensors to the system, while only the sensor configuration table will need to be modified to include the new sensors at the top level.

5.1.1 Implementation Details

The software architecture of the firmware includes many different modules. It provides a simple system to manage facilities, including sensors, time keeping, data storage, and communications. All system resources are abstracted to provide a hardware agnostic view of the system. This allows for underlying hardware changes with little to no modification of the application level software. Figure 10 shows a flow chart that summarizes the basic flow of control for the firmware.

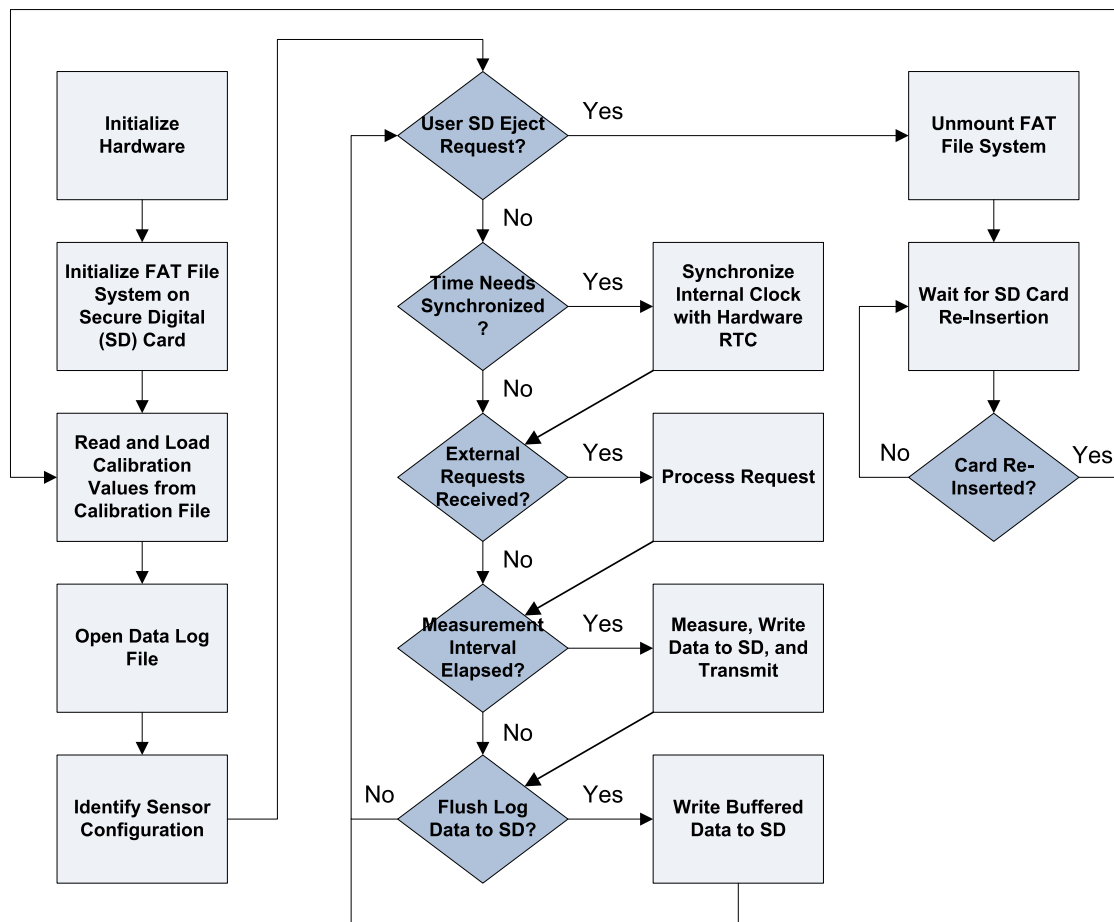


Figure 10: Firmware Flow of Control (Simplified)

As seen in Figure 10, the main execution loop of the firmware has four tasks. It must 1) keep the internal clock synchronized with the external real-time clock, 2) process unsolicited requests received over the network, 3) take measurement data storing to SD and transmitting to the data sink, and 4) insure that the SD contains the latest data in case of power loss.

The sensor related firmware is organized into five layers. These layers include: low-level hardware communications, sensor specific drivers, sensor configurations, general sensor operations, and finally the application that utilizes the sensors. The

architecture is layered as shown in Figure 11.

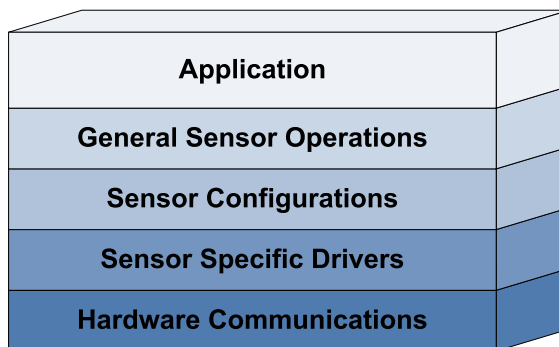


Figure 11: Firmware Layered Architecture

The hardware communications layer provides the lowest level interaction with the sensor devices. This layer provides communications protocol code that may be used by many different sensors. The protocols used by the various sensors could be I²C, SPI, UART, or some other proprietary protocol. Re-configurability is maintained in that the hardware communications layer is unchanged when adding a new sensor. This assumes that the protocol is already supported. If the hardware protocol used by a sensor is not already implemented in the hardware communications layer, a software module must be written to manage this type of protocol. The benefit of this architecture is that after a module is added to the hardware communications layer, it is available for use by any other sensor that utilizes the same protocol.

The sensor specific driver layer provides the interpretation of the data being accessed by the hardware. This layer provides a unified interface for the upper levels of the system to work with and interacts with the hardware communications layer to access the hardware associated with a sensor. A sensor driver software module must be designed for every sensor that is added to the system. Each sensor driver provides three functions:

identify, *measure*, and *convert*. This set of functions provides a means to request an identification string, take a measurement, and convert a raw sensor measurement to its corresponding unit value. The identification string describes a sensor's type, measuring units, and model number that is included in the identify packet that is sent on initialization.

With a unified interface to sensors available, the sensor configurations layer can easily table the function calls and manage many sensors with only a small amount of controlling code. Figure 12 shows the sensor table used to configure the sensors that are active in the system.

```

struct sensor
{
    char desc[25];
    void (*identify)(char buff[]);
    long (*measure)(void);
    float (*convert)(long);
    float calibration;
};

extern rom struct sensor      sensors[];
extern rom unsigned char     sensor_count;

...

rom struct sensor sensors[] = {
    { "Fibonacci", fib_identify, fib_measure, fib_convert, 0.0 },
    { "Temperature", SHT75_temp_identify, SHT75_temp_measure, SHT75_temp_convert, 1.0 },
    { "Humidity", SHT75_hum_identify, SHT75_hum_measure, SHT75_hum_convert, 2.0 },
    { "Pressure", pressure_identify, pressure_measure, pressure_convert, 4.0},
    { "CO2", CO2_identify, CO2_measure, CO2_convert, 5.0 },
    { "Sound_Level" , sound_identify, sound_measure, sound_convert, 4.0 },
    { "Altitude", altitude_identify, altitude_measure, altitude_convert, 3.0 },
    { "Case_Temperature", temperature_identify, temperature_measure, temperature_convert, 3.0},
    { "Battery_Level", BTest_identify, BTest_measure, BTest_convert, 6.0 }
};

rom unsigned char sensor_count = sizeof(sensors)/sizeof(struct sensor);

```

Figure 12: Sensor Table Structure

As seen in Figure 12, each sensor in the system requires *identify*, *measure*, and *convert* functions that are registered in a sensor structure object. A registry of sensor

structures is maintained to provide a simple means to both manage the sensor that are connected to the system and directly access the sensor driver interfaces. This table must be modified manually when adding or removing a sensor from the system. The size of the table is calculated dynamically so any sensor can be deactivated in the system by simply commenting the sensor entry in the sensor table code and recompiling.

The general sensor operations layer provides high-level access to the sensors. This layer provides a simple interface for acquiring data from the sensors and requesting sensor details for identification purposes. This code provides a way to form measurement and identifier packets for storage and transmission by simply referring to a sensor by its index. Figure 13 and Figure 14 show some of the basic functions provided, which utilize the sensor structure objects in the sensor table to call the various sensor specific functions in each sensor driver.

```

1 void Sensor_Identify(int index, char strIDM[])
2 {
3     const char IStart[] = "I U=";
4     char SIndex[4];
5     char Group[45];
6
7     extern char UnitID;
8
9     strcpypgm2ram(strIDM, "I U="); // "I U="
10    if (UnitID < 10)
11        strcatpgm2ram(strIDM, "00");
12    else if (UnitID < 100)
13        strcatpgm2ram(strIDM, "0");
14    itoa(UnitID,SIndex);
15    strcat(strIDM,SIndex); // "I U=xxx"
16
17    strcatpgm2ram(strIDM, ", S=");
18    if (index < 10)
19        strcatpgm2ram(strIDM, "00");
20    else if (index < 100)
21        strcatpgm2ram(strIDM, "0");
22    itoa(index,SIndex);
23    strcat(strIDM, SIndex); // "I U=xxx, S=yyy"
24
25    strcatpgm2ram(strIDM, ", T=");
26    getTime(Group);
27    strcat(strIDM, Group); // "I U=uuu, S=sss, T=yyyymmddhhmmss"
28
29    strcatpgm2ram(strIDM, ", ");
30    (*sensors[index].identify)(Group);
31    strcat(strIDM, Group); // "I U=uuu, G=sss, T=yyyymmddhhmmss, G=<group>, P=<part>, D=<units>"
32 }

```

Figure 13: Sensor Identify Function

As seen in Figure 13, the sensor identify function forms the ASCII identify string containing sensor module and sensor information. Line 30 in Figure 13 shows how the identify function, which is part of the sensor driver, is called by accessing the sensor structure table by the sensor index. This loads a character buffer with the associated sensor information from the sensor driver.

```

1 void Sensor_Measure(int index, char strIDM[])
2 {
3     const char SIndex[4];
4     char TimeStamp[20];
5     long Raw;
6     float VData;
7     char Value[11];
8     extern char UnitID;
9
10    strcpypgm2ram(strIDM, "M U=");
11    if (UnitID < 10)
12        strcatpgm2ram(strIDM, "00");
13    else if (UnitID < 100)
14        strcatpgm2ram(strIDM, "0");
15    itoa(UnitID, (char*)SIndex);
16    strcat(strIDM, (char*)SIndex); // "M U=uuu"
17
18    strcatpgm2ram(strIDM, ", S=");
19    if (index < 10)
20        strcatpgm2ram(strIDM, "00");
21    else if (index < 100)
22        strcatpgm2ram(strIDM, "0");
23    itoa(index, (char*)SIndex);
24    strcat(strIDM, SIndex); // "M U=uuu, S=sss"
25
26    strcatpgm2ram(strIDM, ", T=");
27    getTime(TimeStamp);
28    strcat(strIDM, TimeStamp); // "M U=uuu, S=sss, T=yyyymmddhhmmss"
29
30    strcatpgm2ram(strIDM, ", R=");
31    Raw = (*sensors[index].measure)();
32    ltoa(Raw, Value);
33    strcat(strIDM, Value); // "M U=uuu, S=sss, T=yyyymmddhhmmss, R=xxx"
34
35    strcatpgm2ram(strIDM, ", V=");
36    VData = (*sensors[index].convert)(Raw);
37    floatToString(Value, VData, 3);
38    strcat(strIDM, Value); // "M U=uuu, S=sss, T=yyyymmddhhmmss, R=xxx, V=yyy.yyy"
39 }

```

Figure 14: Sensor Measure Function

The sensor measurement ASCII strings are formed by the code shown in Figure 14. As with the *Sensor_Identify* function, the *Sensor_Measure* function leverages the sensor structure table to gather the raw sensor measurement and convert the raw measurement to its corresponding unit value. Lines 31 and 36 show the calls to the sensor driver associated with the specified sensor index in the sensor structure table.

Finally, the application layer is the top-level code that could be considered to be the main operations as described previously in Figure 10. Figure 15 shows the sensor

systems top level function calls, which identify all of the sensors and take measurements for all of the sensors.

```

void Identify(int writefile){
    int i;
    for(i=0; i<sensor_count; i++)
    {
        Sensor_Identify(i,(char*)buff);
        strcatpgm2ram((char*)buff, (const far rom char *)"\r\n");

        if(writefile)
            FSfwrite((void*)buff, strlen((char*)buff), 1, file_w);

        if (USE_XBEE)
            sendMSG_XBee2(NO_BROADCAST64, server, buff, strlen((const char*)buff));

        printf("%s", buff); //Print to debug
    }
}

void Measure(int writefile){
    int i;
    for(i=0; i<sensor_count; i++)
    {
        Sensor_Measure(i,(char*)buff);
        strcatpgm2ram((char*)buff, "\r\n");

        if(writefile)
            FSfwrite((void*)buff, strlen((char*)buff), 1, file_w);

        if (USE_XBEE)
            sendMSG_XBee2(NO_BROADCAST64, server, buff, strlen((const char*)buff));

        printf("%s", buff); //Print to debug
    }
}

```

Figure 15: Top Level Identify and Measure Functions

As seen in Figure 15, identifying or taking measurements from all of the active sensors on the system is as simple as looping through the sensor table. The *Sensor_Identify* and *Sensor_Measure* functions load character buffers with the identify or measurement strings, respectively. Once the identify or measurement string is collected, it is both stored to the SD card and sent through the XBee® radio (if the radio is enabled).

5.2 Computer Software

As it was a goal to minimize the computation at each sensor node, computer software was required to both store and process the collected data. The processing responsibilities range from graphically representing the data to make it simple to view the real-time state of the monitored variables, to applying algorithms to the received data.

The primary computer-based application developed as part of this thesis was the BSU Sensor Monitor (BSUSM) application. This program was designed using the C# programming language as a general test-bed to provide an example of the types of interactions possible with the sensor system. This software offers a wide range of capabilities, including data plotting, data sinking, sensor module device control, and sensor data fusion.

5.2.1 Plotting

The program's principle function is to allow a network of sensor modules to stream collected data to the application using a USB-XBee adapter connected to the computer. This allows the data to be plotted as the data is received in real-time. Each of the active sensor modules has a unique identifier that the BSUSM can use to allow users to select the data of interest. Figure 16 shows the BSUSM as it plots data received from sensor modules.

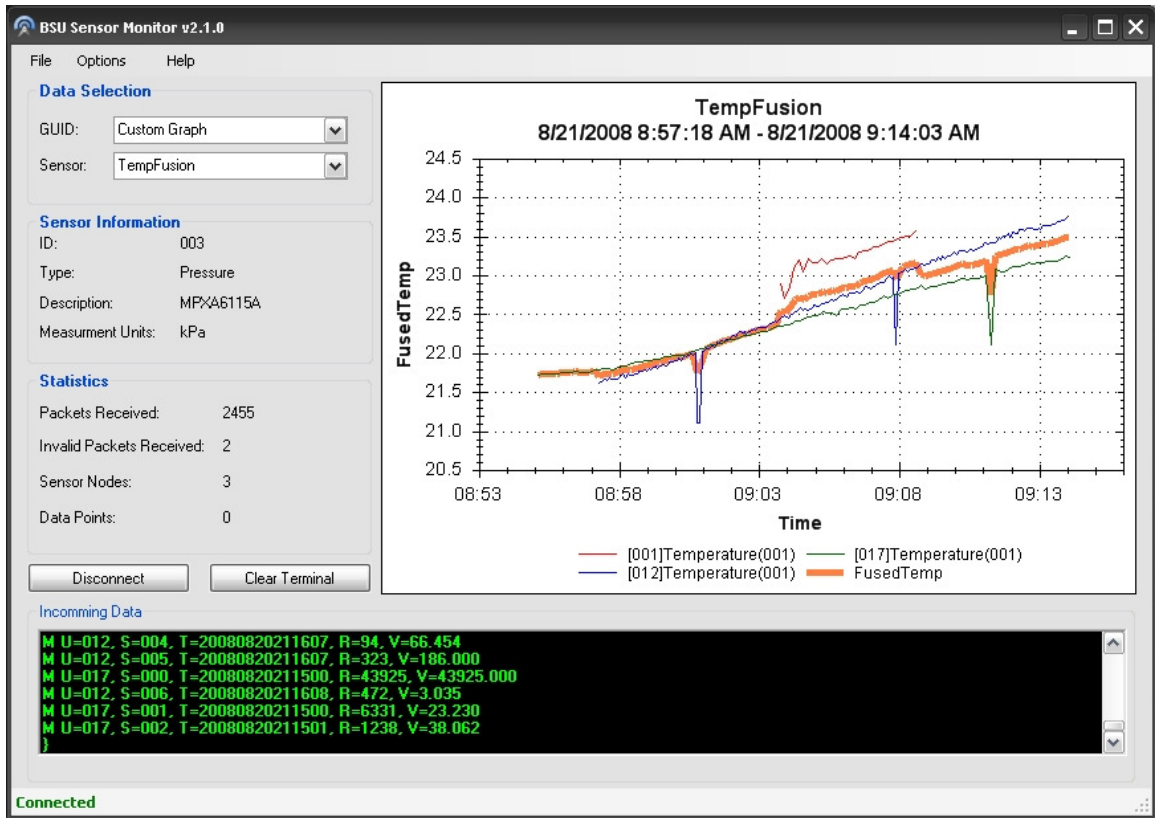


Figure 16: BSU Sensor Monitor Data Plot

5.2.2 Data Sink

In addition to plotting data, the BSUSM was designed to store collected data in a structured query language (SQL) database and subsequently export in a comma separated values (CSV) format compatible with common spreadsheet software. Two versions of the software have been developed: one that relies on a SQL server service to be available and another that uses an internal database to manage the data that is presented in plots and data exportation. Utilizing a computer to provide a data sink for a network of sensor modules allows the data to be stored in a central location for real-time or post processing.

5.2.3 Sensor Module Control

The BSUSM application was designed not only to have data “pushed” to it by the

sensor modules, but also to directly request data from any sensor unit within the network. This offers many possibilities for the sensor network configuration as the computer may be used to request specific sensor measurements before the sensor module would otherwise have provided it, or to remotely control the sensor module for some other purpose such as time synchronization. Figure 17 shows the basic communication interface implemented.

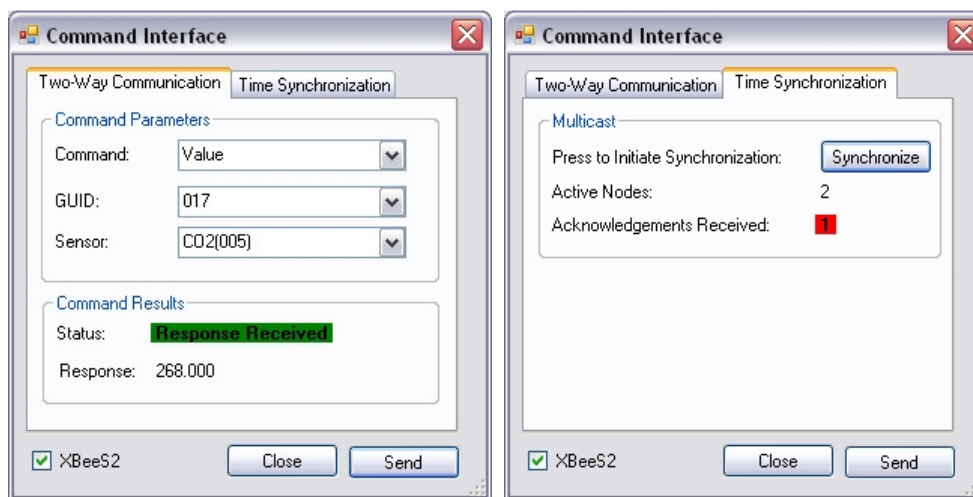


Figure 17: Two-Way Communication and Time Synchronization

As seen in Figure 17, the command interface allows users to specify a command to send to a particular sensor module. The commands currently implemented include *get-value*, *get-raw-value*, *get-time*, *set-time*, *self-identify*, and *synchronize-time*. The *get-value* and *get-raw-value* commands allow the user to request the value of any sensor on the module, whereas *get-value* retrieves the converted user-readable sensor reading and *get-raw-value* retrieves the raw sensor reading. The *get-time* and *set-time* commands, respectively, allow users to retrieve and set the time, while the *synchronize-time* command attempts to synchronize all sensor modules that are monitored by the BSUSM

program by multicasting a *set-time* command to all sensor modules connected.

5.2.4 Real-Time Data Fusion

Real-time sensor data fusion tools implemented in the BSUSM software allows various data fusion algorithms to be applied as data is received from each of the sensor modules connected. The software was designed in a layered architecture to make the addition of new fusion algorithms straightforward. Currently the software implements two fusion/data processing algorithms: averaging and peak detection. Figure 18 shows the graph options associated with the sensor data fusion features of the BSUSM.

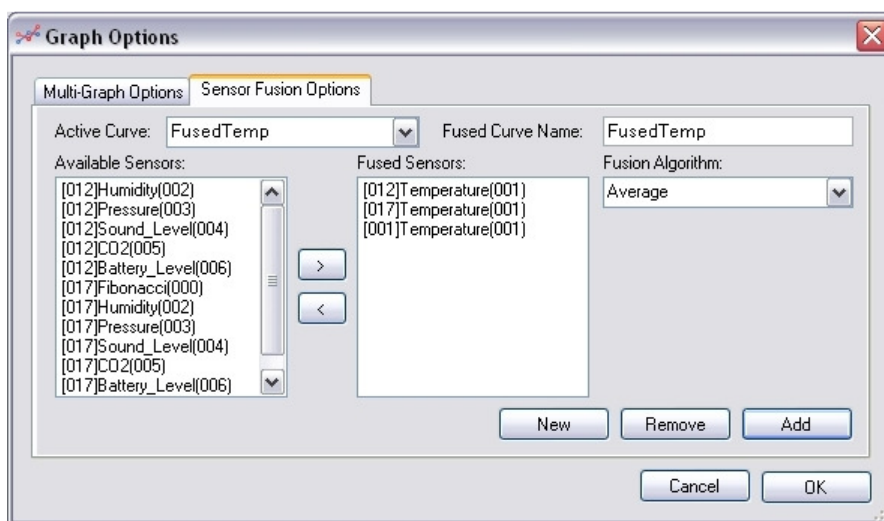


Figure 18: Real-Time Sensor Data Fusion Dialog

As seen in Figure 18, the software allows users to select the sensors to be fused, select a name to represent the curve, and choose the fusion algorithm to be applied to the selected sensors. In this example three temperature sensors were selected (one from sensor module 012, 017, and 001, respectively), the averaging algorithm is applied, and the name of the curve generated is “FusedTemp.” Figure 19 shows a plot utilizing the averaging fusion algorithm with the three sensor modules.

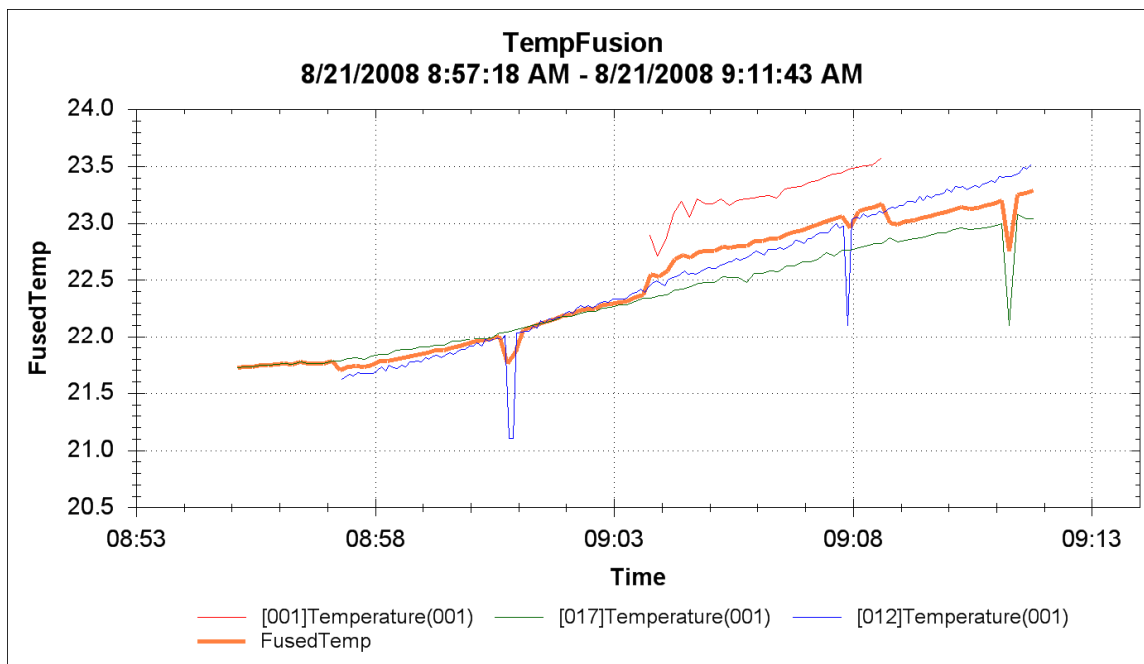


Figure 19: Average Temperature Fusion Algorithm

It is clear from Figure 19 that each of the temperature readings from the various sensor modules have equal weight in the average curve, and sensor measurements are only included in the average when the measurements are defined over the same time interval.

There are many possible fusion algorithms; however, data fusion typically requires very specific questions to be answered, particularly if dissimilar sensors are to be used in concert. The simple average fusion algorithm was implemented, because it is a general purpose tool that can be applied to any group of like sensors. It also serves as an example for testing integration of algorithms into the BSUSM. Only the averaging algorithm was implemented, as the focus of this research was providing the framework to manage algorithms rather than designing specific algorithms that depend greatly on the application. This concept will be explored further in Chapter 7.

5.2.5 Database

In addition to the software created to interact with the sensor network, a relational database was designed using SQL to provide a simple means of both storing and analyzing sensor data collected by the sensor modules. The database created has been utilized both for local and web-based applications to manage sensor data.

5.2.6 Implementation Details

There are many internal components associated with the functionality presented in the BSUSM. To provide the most reconfigurable implementation, the architecture was split into a layered and modular design. The primary components include: user interface, graphing utilities, communications, database manager, data fusion processor, and CSV file generator. As the graphical interface has already been discussed in some detail, the following will focus on the remaining components.

5.2.6.1 Graphing Utilities

The graphing features of the BSUSM are accomplished with the use of the ZedGraph graphing library [17]. This is an open source library that provides the base graphing mechanics that is driven by the database system, whether it be a local or remote database.

5.2.6.2 Communications

The communication layers are perhaps the most complex part of the BSUSM software components. There are several communication layers that provide the ability to connect with the XBee® wireless networks, and receive data as it is collected as well as providing a means to control the individual sensor modules within the sensor networks.

The BSUSM utilizes a double buffering system to manage all of the data received

on the UART. This helps to prevent inadvertent loss of data that could arise if the computer was unable to service the UART while it is processing previously received data. The processing of the data is a fairly significant task, as each packet of data that is received must be parsed to identify a wide range of details. Some of the details within the data packets are the originating address of the sensor node, the packet length, the packet type, and the packet payload. There are two primary communication layers: ZigBee® XBee®, and payload processing.

The XBee® radio's communication protocol provides many different packet types, which aid in the identification and processing of the packets as they move through the sensor network. These packets range from XBee® modem status indicators (e.g., "modem is associated with the network" or "modem has been reset") to actual data packets which carry sensor measurement data. Thus, for managing this area of identification and processing the BSUSM utilizes a packet handler layer.

The packet handler assembles packets by "pulling" data off of the UART buffer and testing for packet type and packet integrity. Once a packet has been formed, an event is raised to notify upper layers of the packet and allow the packet payload to be processed. Some packets (such as status packets) may not cause events to be raised if the packet handler has a known response to such a packet. Receipt of corrupt packets is also handled by raising an event to notify higher layers of the problem.

An additional responsibility of the packet handler is to extract and store the active association of sensor identification numbers and sensor node network addresses. This allows other entities to request the network address of a sensor module without having to broadcast a request on the network. This is accomplished by maintaining a hash table

that maps sensor module identification numbers to their respective address.

The packet payload processing layer parses the ASCII string packets in the format identified in Figure 7. The data processor extracts the key-value pairs utilizing the regular expressions library provided in C#. Once a packet payload has been identified as either an identifier packet or a measurement packet, an event is raised to pass the data to the database and plotting layers.

The final communications module resides in managing two-way messaging with the sensor nodes in the network and the BSUSM. To improve communications reliability, all messages sent by the BSUSM require the target sensor module to return an acknowledgement in response. The response may be the requested data, or it may be simply an acknowledgement if no data was requested from the sensor node.

The two-way communications module is event driven. When a message is sent out to a sensor module the calling module registers a callback. At the time an acknowledgement is received from a sensor node, the callback is activated to notify the original software module that registered it. The packet handler works closely with the two-way communications module to allow processing of packets that are related to two-way communications. The messages are tracked in the two-way communications module by a packet identification number that is assigned when the packet is created and echoed by the sensor module in reply to the BSUSM. The callback registration uses the packet identification number as a key to a hash table containing all outstanding callbacks. Thus, when a response is received from a sensor node, it can be tied to the callback function that was registered.

5.2.6.3 Database Manager

The database manager module provides database connectivity by supplying a generalized API for the BSUSM to interact with database systems. This allows the underlying database to be easily modified without changing any other part of the application. Thus, the BSUSM can be configured to utilize an online database or a local one by simply making changes within the database manager module. The database manager module and the graphing utilities closely communicate to provide plots of data as it is added to the database.

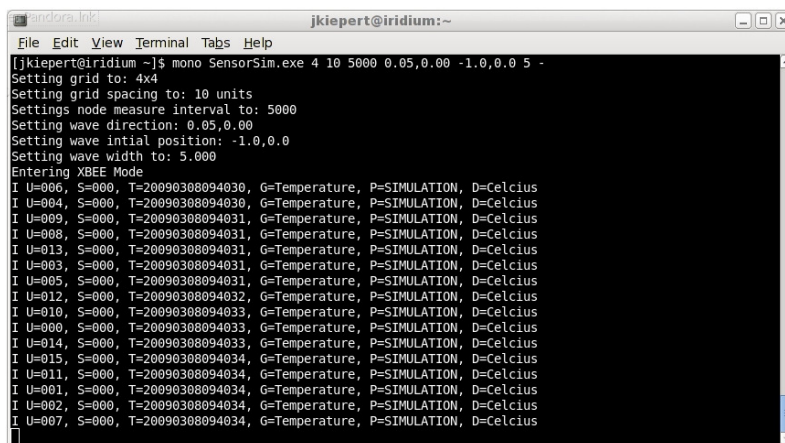
5.2.6.4 CSV File Generator

The CSV file generator module provides a way to export database data to a comma separated value file format. This format was chosen as it is easily imported to many spreadsheet applications, including Microsoft Office. One of the primary benefits of this module is that it allows for fast, in-memory modification of all the data before committing to the file system.

5.3 Sensor Network Simulation

In an effort to better understand the detection of patterns within large sensor grids, a sensor network simulator was created. This provided a consistent way to test algorithms within the BSUSM software as well as a means to test very large sensor networks, which would be impractical to create either due to cost or setup complexity. The primary focus of the research done under the FAA funding has been environmental sensing, and as such, the simulator was designed principally to simulate environmental contamination diffusion through an area. The simulator allows users to specify many parameters such as the contamination wave vector, wave properties, sensor grid size and

spacing. The output of the simulator is sensor measurement packets of the same type normally generated by the sensor modules as well as a visualization tool to allow the user to view the progress of the simulation. The simulated sensor measurement outputs can be directed to a file, a console window, or an XBee® radio. Figure 20 and Figure 21 show the output of a simulation with an impulse wave of a finite width moving through a four-by-four grid of sensor modules.



```

jkiepert@iridium:~
File Edit View Terminal Tabs Help
[jkiepert@iridium ~]$ mono SensorSim.exe 4 10 5000 0.05,0.00 -1.0,0.0 5 -
Setting grid to: 4x4
Setting grid spacing to: 10 units
Settings node measure interval to: 5000
Setting wave direction: 0.05,0.00
Setting wave initial position: -1.0,0.0
Setting wave width to: 5.000
Entering XBEE Mode
I U=006, S=000, T=20090388094030, G=Temperature, P=SIMULATION, D=Celcius
I U=004, S=000, T=20090388094030, G=Temperature, P=SIMULATION, D=Celcius
I U=009, S=000, T=20090388094031, G=Temperature, P=SIMULATION, D=Celcius
I U=008, S=000, T=20090388094031, G=Temperature, P=SIMULATION, D=Celcius
I U=013, S=000, T=20090388094031, G=Temperature, P=SIMULATION, D=Celcius
I U=003, S=000, T=20090388094031, G=Temperature, P=SIMULATION, D=Celcius
I U=005, S=000, T=20090388094031, G=Temperature, P=SIMULATION, D=Celcius
I U=012, S=000, T=20090388094032, G=Temperature, P=SIMULATION, D=Celcius
I U=010, S=000, T=20090388094033, G=Temperature, P=SIMULATION, D=Celcius
I U=000, S=000, T=20090388094033, G=Temperature, P=SIMULATION, D=Celcius
I U=014, S=000, T=20090388094033, G=Temperature, P=SIMULATION, D=Celcius
I U=015, S=000, T=20090388094034, G=Temperature, P=SIMULATION, D=Celcius
I U=011, S=000, T=20090388094034, G=Temperature, P=SIMULATION, D=Celcius
I U=001, S=000, T=20090388094034, G=Temperature, P=SIMULATION, D=Celcius
I U=002, S=000, T=20090388094034, G=Temperature, P=SIMULATION, D=Celcius
I U=007, S=000, T=20090388094034, G=Temperature, P=SIMULATION, D=Celcius

```

Figure 20: Sensor Simulation Console Output

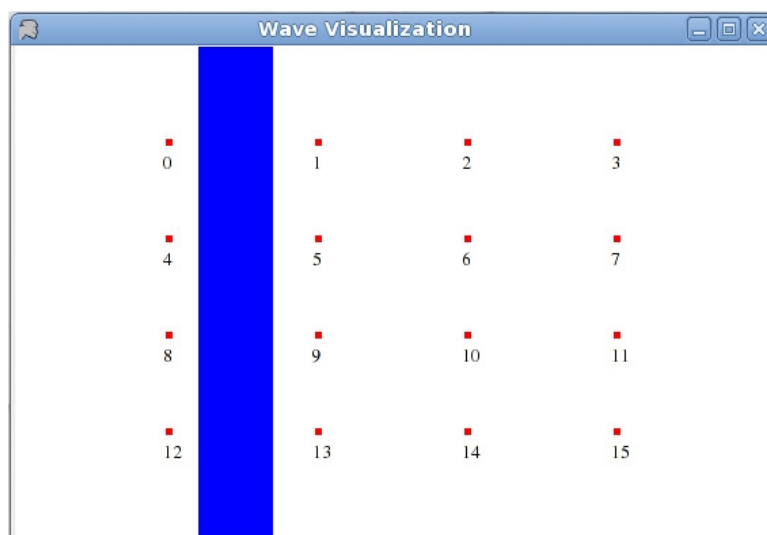


Figure 21: Sensor Simulation Graphical View

The “Wave Visualization” window shown in Figure 21 indicates the progress of the wave (vertical bar) as it moves from left to right through the sensor nodes (numbered squares), while the console shown in Figure 20 displays some of the simulated sensor packets as they are generated. Each virtual sensor module operates within its own execution thread to provide a measure of realistic autonomy. The simulation can also be configured to independently adjust each virtual sensor node’s operating properties such as the measurement interval. The simulation software is discussed in further detail in Chapter 7.

CHAPTER 6: TIME SYNCHRONIZATION

6.1 Necessity of Time Synchronization

Time synchronization of computer systems has been of interest since computers were first connected to one another. It is an old problem, but one of great importance none the less. While time synchronization within a collection of computing systems connected via wires has been addressed at great length, many of these techniques are not applicable when the computing systems are connected via wireless links [18]. This is because the network topology is not as uniform or as reliable. In this chapter, some of the various time synchronization techniques for wireless sensor networks will be explored. Additionally, the currently implemented clock synchronization technique in our system will be described and analyzed.

6.2 Network Time Protocol

Perhaps one of the most pervasive time synchronization protocols in use today by computer systems is Mills' Network Time Protocol (NTP). Whereas NTP is highly effective in typical computer networks, it does not have features that are conducive to wireless sensor networks. This is because NTP and similar algorithms make assumptions about the network and hardware environment that are not necessarily true in wireless sensor networks, such as:

- The network can be continuously monitored for time data
- The CPU is generally free to manage time synchronization utilities

- The network is continuously available for sending time data

The primary issue with these assumptions is the energy required to maintain a continuous network connection on a likely battery powered-device [19]. Since the CPU and radio in wireless sensor systems may be put into a sleep mode for energy savings, neither the network nor the CPU can be guaranteed to be available to manage time synchronization.

6.3 Single-Pulse Synchronization

One of the simplest techniques for time synchronization is implemented by periodically broadcasting a time reference to all nodes in the network. This technique, known as single-pulse synchronization, can effectively provide synchronization in wireless networks in a star configuration. Assuming that the latency of the nodes connected to the coordinator is similar (a safe assumption due to the homogeneous nature of the sensor nodes in the network), the resulting synchronization error will also be comparable. This technique also requires the wireless network to support multicasting, because the synchronization pulse must be received by all members of the network simultaneously for it to be effective. We expect that latency in the receipt and processing of the time broadcast at each of the sensor nodes will result in phase error in the clock equal to the total latency and processing time. On the other hand, since we are dealing with homogeneous hardware and software on each of the sensor nodes, the latency is likely very similar between nodes. This feature allows the sensor nodes to maintain time agreement with significant accuracy.

The single-pulse synchronization technique has limited effectiveness when the network involved has a mesh (and thus multi-hop) architecture. If there are nodes in the network that are more than one hop away, the latency will be greater for those nodes, and

the time associated with the indirect path must be accounted for to achieve low synchronization error.

Despite the limitations of this technique, its primary benefit is that it requires little processing power or special hardware to be implemented. Also as described above, node-to-node time synchronization can be achieved with a small portion of phase error when similar hardware and software is present on each of the sensor nodes.

6.3.1 Performance Characteristics

As single-pulse synchronization was implemented in our system, we were able to directly characterize its performance. The synchronization performance characterization required a specialized configuration of the sensor nodes and computer time reference. Capturing precise timing between events that occur in software can be a tricky prospect, especially when the events to be measured occur on multiple autonomous entities. In this case, we needed to capture the time from the transmission of a packet from a computer to the receipt of the packet at multiple sensor nodes. Additionally, it was important to characterize the time required to process the packets at the sensor node to gain an understanding of all contributors to latency. To accomplish effective timing of such varied events on different entities, a collection of digital outputs were configured on each entity and monitored via a single oscilloscope to provide relative time differences on a common time scale. Each of the sensor nodes utilized two digital outputs. The first would be toggled high when a packet was received and toggled low when an acknowledgement was sent back in response to the packet received. The second digital output was configured to be toggled high when a time packet was recognized and low when the new time was successfully stored to the real-time clock. Figure 22 shows the

test configuration.

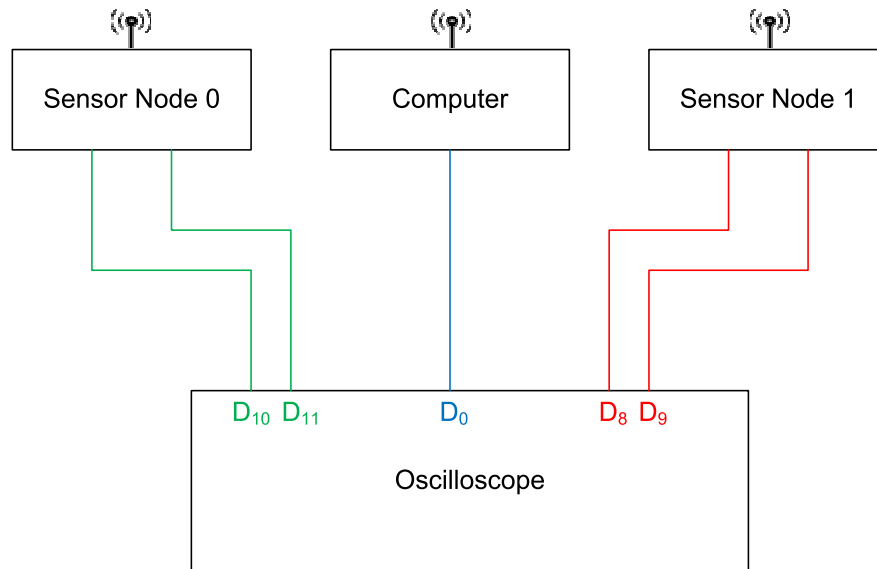


Figure 22: Single-Pulse Synchronization Test Configuration

The outputs of the sensor nodes were configured from unused GPIO pins. For the computer (which does not normally have user accessible GPIO) a parallel port was configured to provide a user software controlled GPIO. Controlling code for both the sensor nodes and the computer was added to adjust the state of the I/O pins when the software reached the states of interest. It should be noted that since the computer system employed for this testing was utilizing a non-real-time preemptive operating system (Windows XP), we cannot depend on the control timing of the I/O pin to be deterministic. While this does not invalidate the test, it does potentially affect the measured initial response time, i.e., the time taken for the sensor nodes to acknowledge the receipt of a time packet following the transmission of the time packet from the computer. Figure 23 shows the result of the test.

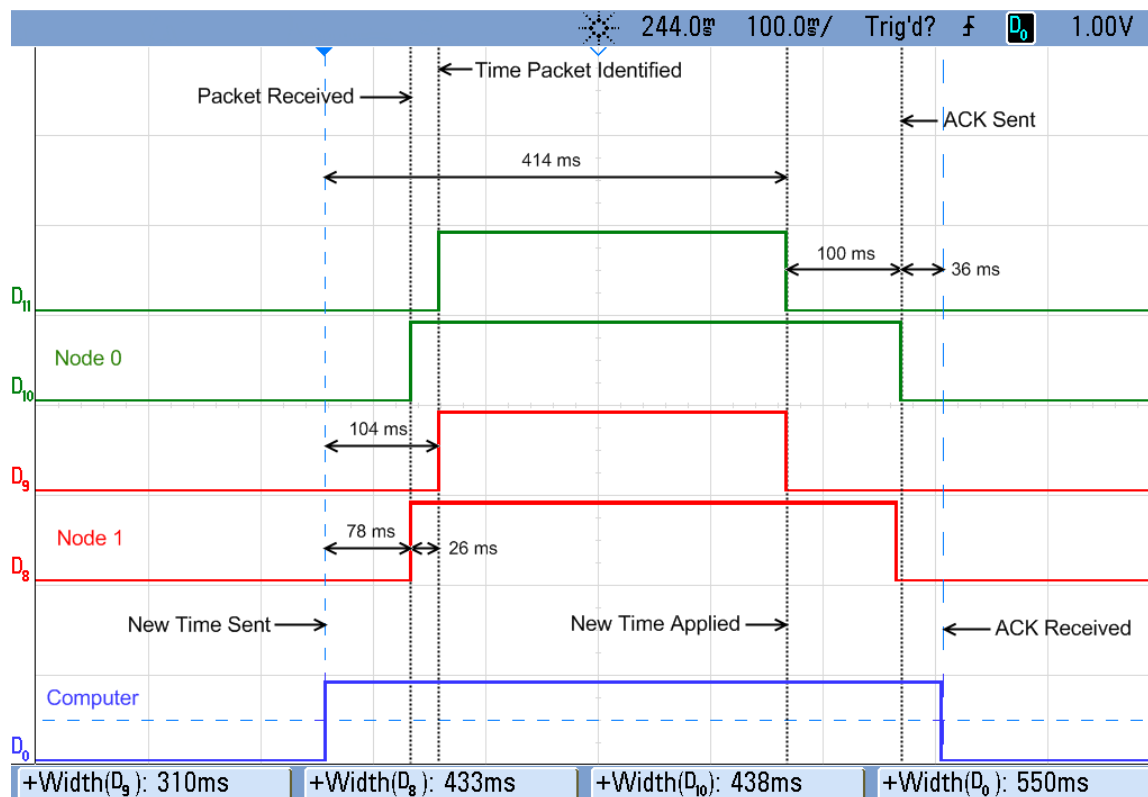


Figure 23: Single-Pulse Synchronization Timing

As expected, it was found that there is potentially significant (depending on the application) phase error with respect to the time reference. In this case, the time from when a time synchronization packet is transmitted at the computer to when the sensor nodes apply the new time is 414 ms. Since our current hardware is capable of one second resolution, the measured phase error is smaller than what can be represented on the sensor nodes' real-time clock hardware. While phase error was present with respect to the time reference, there was good agreement between sensor nodes. Figure 24 shows the measured phase error between sensor nodes.

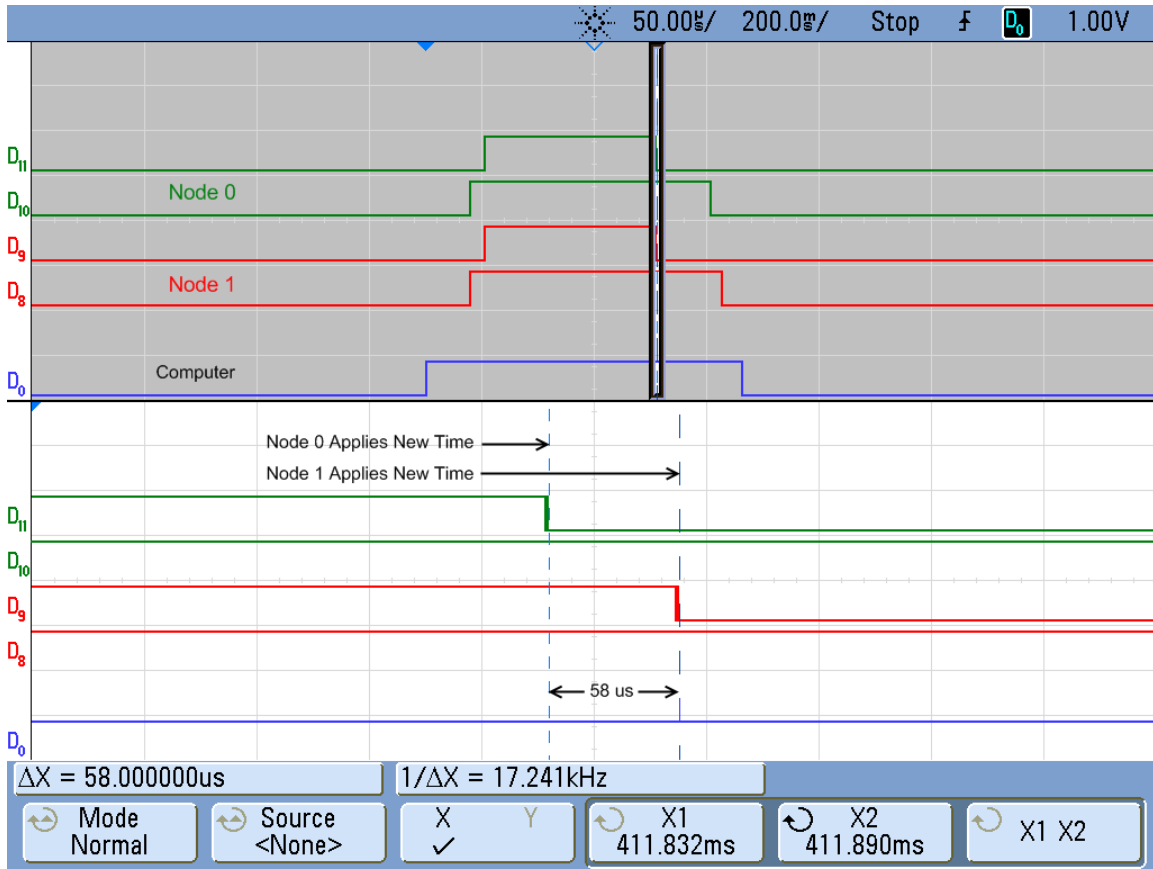


Figure 24: Time Phase Error between Sensor Nodes

As seen in Figure 24, the phase error between sensor nodes within the network was found to be 58 μs . This raises the point that knowing the true wall time may not be as important as knowing the ordering of the events in the system. When there is small phase error between sensor nodes, there can be general consensus about the ordering of events. In some applications, absolute time may be completely unimportant if the relative time between all events is known. An extreme implementation of relative time can be achieved through Lamport's virtual clock system in which the clock state does not refer to time as we know it, but ticks only as events occur with respect to the system [20].

One final area that single-pulse synchronization does not address is the inherent differences in the time keeping oscillator frequency on each of the sensors. Even if the

clocks are synchronized after a reference pulse, they are unlikely to remain so for any significant duration. This synchronization lifetime issue comes from the inherent irregularities in crystal oscillators. Temperature and vibration can affect the frequency generated by crystal oscillators. It is important to note that there are algorithms that can manage this issue and adjust to compensate for any frequency skew. As a result of the skew, successful time synchronization would depend on how often the clocks are synchronized.

When accurate relative time between sensor nodes is not enough and oscillator skew must be managed, there are several time synchronization techniques available for use in a wireless sensor network that can provide better time synchronization than the single-pulse method that we have implemented. These techniques may require more powerful hardware or, at the very least, high resolution time keeping hardware. In the following section we will discuss another potential algorithm that would result in significantly better synchronization: reference broadcast synchronization.

6.4 Reference Broadcast Synchronization

Reference Broadcast Synchronization (RBS) is an algorithm developed by Elson et al. in [21] and others. This algorithm provides an interesting divergence from the most commonly used algorithms in that it does not require individual nodes to change their respective clocks during synchronization, but rather requires only that the nodes maintain the relative time scales associated with other members of the network. In this way, a sensor node may convert its local time to any other timescale within the network. This idea could greatly benefit the time keeping in our sensor system as a large portion of the processing time required to synchronize the clock on a particular sensor node was related

to changing the physical time on the RTC hardware.

RBS works with multiple messages passed between every node within the network. The algorithm is a multi-setup process as follows:

1. A sensor node or other device (perhaps with an absolute timescale) broadcasts m packets each of which is received nearly simultaneously at each node in the network (assuming single-hop network architecture)
2. Each of the n receiving nodes records the time according to its local internal clock
3. Each receiving node exchanges time of arrival with all other nodes in the network
4. Each node i calculates its offset to node j by taking the average of the time differences associated with each of the m reference broadcasts as shown in

Equation 6.1:

$$\forall i \in n, j \in n: \text{Offset}[i, j] = \frac{1}{m} \sum_{k=1}^m (T_{j,k} - T_{i,k}) \quad (6.1)$$

This algorithm does not account for clock skew as discussed earlier. Oscillator clock skew can be compensated for by using a least-square linear regression rather than an average for calculating the offsets [19].

Multi-hop network architectures can be accommodated by having multiple beacon transmitters that are within range of different domains (areas of the network that require at least one additional hop to be reached). In this way, the algorithm is essentially the same in that each beacon node broadcasts and all nodes within range of the broadcast and each other exchange information. Since some nodes will receive a broadcast from multiple beacon nodes, each logical domain will have enough information to build a timescale that relates it to all other domains.

6.5 Summary

As a first degree solution, we have implemented a single-pulse synchronization system. While this system does not provide the best synchronization possible, it does fit well with the current hardware precision. In future work, higher performance synchronization will need to be implemented to allow for applications that must correlate measurement data across the network on an absolute time scale. As we have shown, node-to-node synchronization is on the order of 58 μs , which is an accuracy that cannot be truly utilized due to the low resolution nature of the timekeeping hardware on the system. It should also be noted that while it has not yet been implemented, it would be possible with the current hardware to utilize a virtual real-time clock by using an internal hardware timer on the microcontroller and extracting the wall time from the hardware real-time clock. This scheme would offer a significant performance improvement over the currently implemented method, because there would not be latency associated with sending commands to the real-time clock hardware. As soon as a time packet is received and identified, there would be little latency associated with updating a register within the microcontroller.

CHAPTER 7: SENSOR DATA FUSION AND APPLICATIONS

7.1 Sensor Data Fusion

While a complete discussion of sensor data fusion is beyond the scope of this thesis, it is important to outline some of the issues involved when attempting to combine large quantities of sensor data to make inferences about an event or phenomenon. Sensor data fusion can be leveraged in several ways; it may be used for data aggregation to reduce large data sets into a more manageable or concise set, to improve the accuracy of a single measurement variable utilizing a large set of sensor information that is known to be measuring the same phenomenon, or to provide a means to utilize multiple types of sensors to provide a more complete picture of the environment being monitored. In this section, we will discuss some of the issues involved in leveraging sensor data fusion.

As discussed in Section 1.2, much research has been applied to formalizing the terminology and processes involved with multi-sensor data fusion. This was originally inspired by funding from the Department of Defense (DoD) to aid in the use of multi-sensor data fusion in military systems, such as target tracking systems that must utilize a variety of tracking sensors in concert. With the need for military researchers to communicate effectively with regard to data fusion processes, the Joint Directors of Laboratories (JDL) Data Fusion Working Group was commissioned to develop a process model that could be used to effectively outline the various aspects of data fusion. Figure 25 shows a graphical representation of the JDL model.

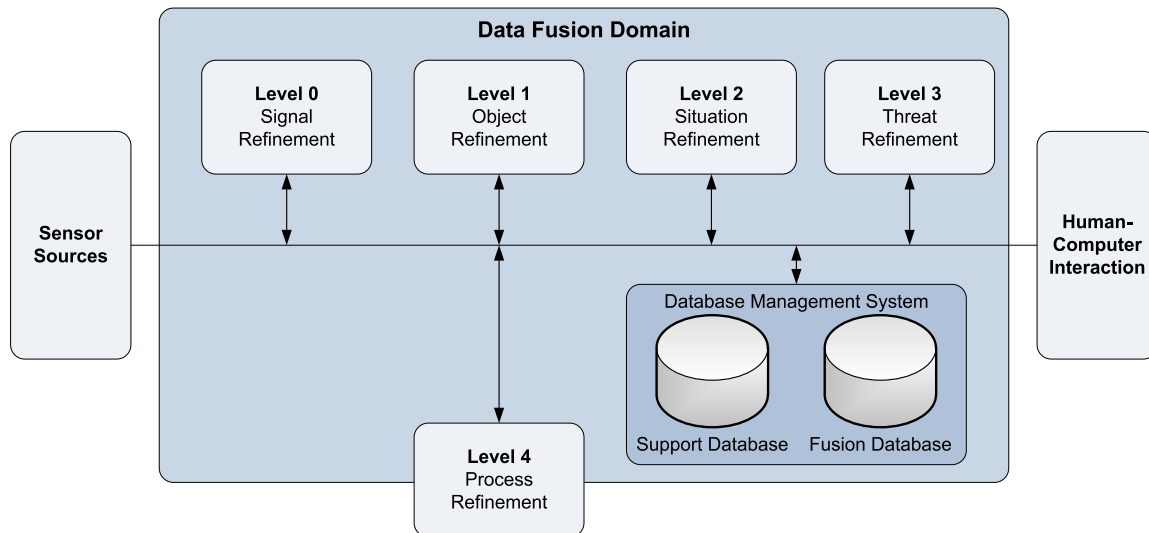


Figure 25: JDL Data Fusion Model [1]

As seen in Figure 25, the model outlines four levels (or processes) of data fusion as well as database resources utilized by each of these processes. As we move across the figure from left to right we see the translation of the data from the raw sensor inputs to an interface for human interaction with the system through application of each fusion process. In the JDL model, as outlined in the *Handbook of Multisensor Data Fusion* by Liggins et al. (2009), these processes are defined as follows:

Level 0 processing (sub-object data association and estimation) is aimed at combining pixel or signal level data to obtain initial information about an observed target's characteristics.

Level 1 processing (object refinement) is aimed at combining sensor data to obtain the most reliable and accurate estimate of an entity's position, velocity, attributes, and identity (to support prediction estimates of future position, velocity, and attributes).

Level 2 processing (situation refinement) dynamically attempts to develop a description of current relationships among entities and events in the context of their environment. This entails object clustering and relational analysis in such as force structure and cross-force relations, communications, physical context, etc.

Level 3 processing (significance estimation) projects the current situation into the future to draw inferences about enemy threats, friend and foe vulnerabilities, and opportunities for operations (and also consequence prediction, susceptibility, and vulnerability assessments).

Level 4 processing (process refinement) is a meta-process that monitors the overall data fusion process to assess and improve real-time system performance. This is an element of resource management.

Level 5 processing (cognitive refinement) seeks to improve the interaction between a fusion system and one or more user/analysts. Functions performed include aids for visualization, cognitive assistance, bias remediation, collaboration, team-based decision making, course of action analysis, etc [22].

What is apparent by the definition above (though the definition is somewhat focused on military applications), is that data fusion is a multi-layered process of refining the sensor inputs, first by collecting the raw data, then characterizing that data in the context of the some defined parameters, and finally applying algorithms to make judgments about the characteristics that were identified. With so many aspects deeply related to specific applications, it is clear that providing a general purpose framework for utilizing data fusion is no small undertaking.

There are many ways to approach the characterization level of data fusion. Each method has a different processing requirement to analyze the data. Thus, in a wireless sensor system that is typically battery-powered, only minimal processing is possible at the sensor nodes. Some of the techniques or methods of data fusion include: Bayesian and Dempster-Shafer inference, pattern recognition through signal processing, fuzzy logic, and many others [23]. The choice of algorithm would be based on their fit with the application in question.

For sensor data fusion of different sensor types, there must be some objective or

query to be answered for which knowing the values of the various sensors in combination will allow the question to be answered. For instance, in a sensor system monitoring air quality, “is the average particulate matter concentration abnormal, thus requiring an alarm?” This question can not necessarily be answered by looking only at the particulate matter concentration of a single sensor or even the combination of all sensors in the area. Has a vehicle been detected near the sensor that is reading a high value? Is the measured average higher than is typical during this period of the day? Questions of this nature can be leveraged by smart fusion algorithms to answer the original query with good certainty and a low false positive rate. As has been said, the nature of the algorithm chosen to deal with such a query is highly dependent on the characteristics of the application. In the above example, an algorithm would need to take into account the average concentration associated with a particular time of the day, compare the inputs from vehicle tracking with the time of the increased concentration, and then make a decision. This process could be implemented with a neural network or fuzzy logic.

Sensor data fusion may require the spatial and/or temporal relationship of the sensors to be known to a good certainty. With enough sensors located across an area, it becomes possible to infer the spatial or temporal dependence of a variable, and thus form gradient plots or scalar field visualizations to aid human observers in understanding the behavior of the variable being monitored. Again, sensor data fusion can provide a means to identify information that would not be possible with any single sensor, if the phenomenon is sufficiently large. Suppose a large network of sensors is spatially distributed over such an area. Each sensor provides a binary yes or no reading of a phenomenon. With the spatial relationships and the sensor readings known for each

sensor at a particular instant, it is then possible to determine the shape of the detected phenomenon – a task that would not be possible with a single sensor. Figure 26 is an illustration that outlines the power of fusing multiple sensor inputs and its ability to characterize an entity in a way not possible with a single sensor.

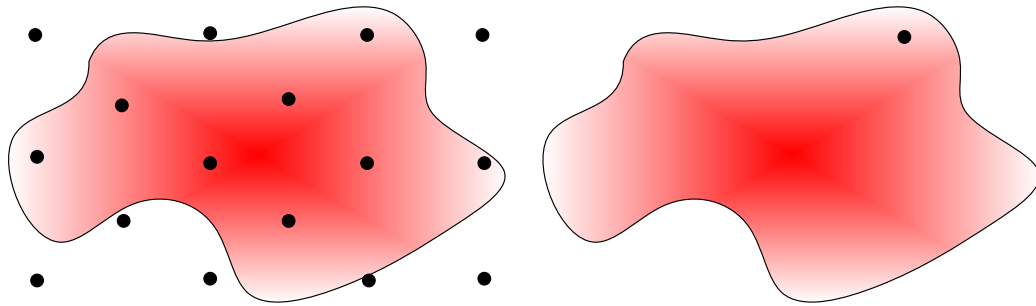


Figure 26: Characterization of an Entity through Data Fusion

As is seen in Figure 26 (left), even if the sensors (represented as black dots) were only capable of a binary, yes/no detection of a phenomenon (irregular shape), the combination of all of the sensor inputs provides a more complete view of the phenomenon, namely its size, shape, position and velocity. With a single sensor configuration (right) the sensor is able to detect the phenomenon, but the concept of size, shape, position, and velocity cannot be directly extracted.

Coordination of the sensor modules requires tight time synchronization to allow identification of events. As each sensor module detects a phenomenon, it is possible that more than one module will “see” the same change, and having time agreement among the sensor nodes allows events to be associated. This is particularly important when we are interested in tracking a dynamic phenomenon and determining its velocity, etc.

In summary, each application must be examined to determine what type of information is required to be collected by the sensor network, and how that information

should be leveraged by fusion algorithms to extract the needed decision or notifications based on the collected information. Our focus in this work has not been on the algorithms employed for data fusion, but rather on how the design of a system may lend itself to providing the information needed to apply such algorithms, and making an analysis system capable of incorporating fusion algorithms as the application dictates.

7.2 Applications

One of the primary applications that our hardware design has been tested with is the environmental monitoring of airliner cabins. Security issues and potential passenger concerns related to conspicuous unattended electronics within the airliner cabin environment have, thus far, prevented the use of the networking capabilities. However, due to the local flash storage, sensor nodes have been used for single point measurements. There are many potential applications for a complete sensor data fusion framework. Of particular interest for security applications, is the deployment of large sensor networks to monitor environmental contamination. Such systems could be used to determine parameters such as threat-level to health, contaminant concentration, and point of origin of contaminants. Additionally, such systems may provide an early warning system that causes an alert if measured parameters exceed a defined level. The flexibility and portability of the hardware and firmware allow a wide range of contaminant sensors to be fitted to the system without re-engineering the sensor modules. The interfacing software and network connectivity allow large sensor networks to be managed both from a data processing standpoint and sensor module control.

One additional aspect of this research has been to develop a means to characterize how this sensor framework could be used to determine the point of origin for diffusive

contaminants as in [24]. As experiments with diffusive contaminant sources over large sensor networks can be difficult to implement, particularly with repeatability, some work has been done to develop a sensor network simulator. This allows repeatable experiments to test the framework for identification of diffusive sources. Figure 27 shows a visualization produced by the simulator as a wave front moves through a sensor network of 16 sensor nodes. Figure 28 shows the output of the BSUSM while it monitors sensor nodes 0, 5, 10, and 15 during the simulation. As seen in Figure 28, there is a distinctive detection peak associated with each sensor node as an environmental change “moves” across the sensor network from top left to bottom right. The magnitude, spatial, and temporal relationships of the measurements would allow a data fusion algorithm to project the source and direction of the change. While this research is in early stages, it may provide a means to characterize the effectiveness of data fusion algorithms before they are deployed in the field.

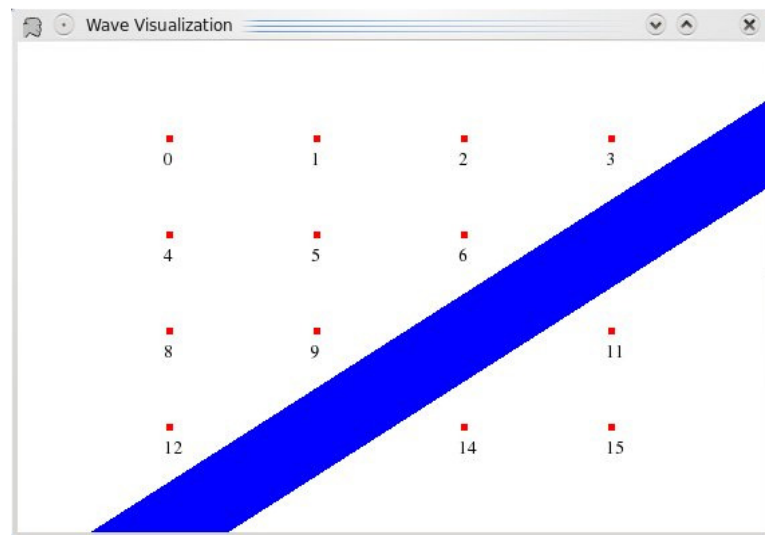


Figure 27: Simulator Visualization of a Wave Front

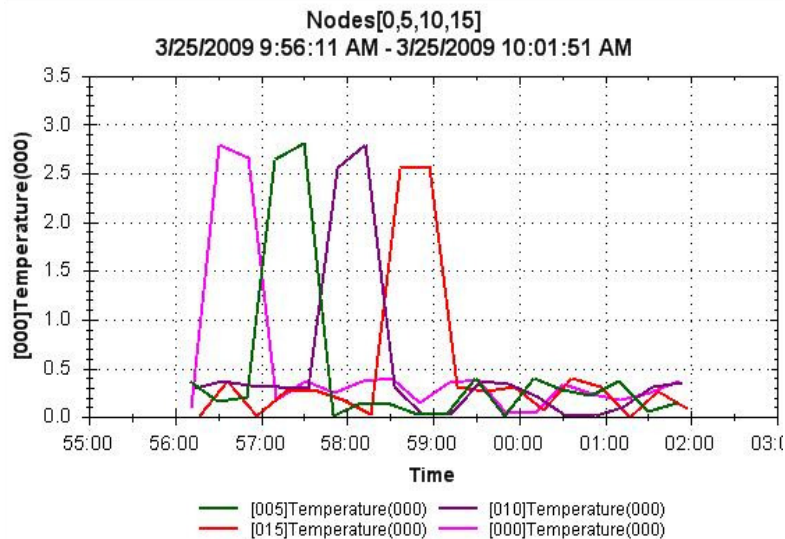


Figure 28: Sensor Output Patterns during Simulation

Clearly, there are many aspects of data fusion and its application. Choosing algorithms on a case-by-case basis is necessary to take advantage of data fusion. Our framework has been designed to provide the base level data collection components, data centralization through database connectivity, and some software extensions that allow algorithms to be applied as the data is collected. In terms of application, much work is still needed to enable the complete characterization of environmental contaminants; however, many of the tools required are provided with the framework developed.

CHAPTER 8: CONCLUSIONS AND FUTURE WORK

8.1 Summary and Conclusions

Many parameters were considered during the design and implementation of a sensor data fusion framework. Each component of the framework was designed with modularity and re-configurability as primary objectives to better enable its use in a wide range of applications. The sensor module hardware was developed to reduce re-engineering and simplify sensor changes or application retargeting. This inexpensive hardware was leveraged to provide data collection capabilities normally only possible with more costly systems. The developed software provides many of the necessary components for both raw data collection and the application of real-time analysis or fusion algorithms. Additionally, the software provides data centralization through database connectivity to aid in applications that require significant data processing schemes that would not be possible on a single computer connected to the sensor network. Time synchronization implemented in the system provides a first degree solution for smaller networks in which mesh architecture is not needed. Further work in this area is planned. The framework outlined by this thesis has been developed to provide a solution to many sensing applications, such as those for security or scientific research.

8.2 Future Work

Many aspects of a general purpose framework implementation were addressed within this work; however, there are other aspects that have stood out during the course of this research that seem important to pursue. These areas include the need for some further hardware refinements, expansion of time synchronization capabilities, and further field testing.

8.2.1 Hardware Improvements

While the sensor module hardware developed provides significant functionality, there are areas that could be improved or extended. Improvements of particular note include the ZigBee® radio platform and microcontroller performance.

As discussed in Section 4.2, the current radio utilized in the system uses a self contained radio system that offloads much of the processing required to maintain the network communications. While this offers benefits to low power microcontrollers (and is needed for the microcontroller currently used in our system), it also creates a significant contribution to added communications latency as the data moves through a buffer on the ZigBee® module, and then is transferred via UART to the microcontroller. When high accuracy time synchronization is required, this scheme does not provide an ideal solution. Perhaps a better solution would be to have the microcontroller directly control the radio chipset as has been done in other systems, discussed in Section 2.2. The radio issue goes hand-in-hand with the limitations of the microcontroller platform.

The radio system was, in part, chosen based on its ability to completely manage the networking tasks. However, if a higher performance microcontroller was used in the system, these network tasks could easily be managed along with the normal CPU

responsibilities. Many new microcontrollers (particularly 32-bit versions) offer peripheral direct-memory-access (DMA) controllers, which allow the microcontrollers to multi-task during peripheral I/O, even though there is only a single CPU execution thread [25], [26]. These processors provide significantly greater raw processing power at similar or lower power consumption than our current microcontroller. While this seems counter-intuitive, it is simply a result of the progression of technology since our system was originally designed. Moving to a 32-bit processor in our system would provide several benefits. It would enable direct management of the network stack, provide improvements in timekeeping resolution (higher CPU clock speeds), provide improved raw computing power for sensor node centric data processing, and maintain or improve the power consumption requirements of the CPU.

8.2.2 RBS Implementation

As discussed in Section 6.3, the single-pulse synchronization technique implemented in our system is not scalable or accurate enough for some applications, particularly those that rely on precise reference to wall time or require significant use of multi-hop network architecture. Some of the latency issues could be solved by some changes in the way time is managed on the system, e.g. internal hardware timers. Nevertheless, multi-hop synchronization needs to be more fully supported, a task (as outlined in Section 6.4) that can be addressed by reference broadcast synchronization.

8.2.3 Advanced Sensor Node Operating System

While the firmware developed for our sensor module hardware is modular and reconfigurable, there are some areas that could be improved. Perhaps most notable

would be the addition of a real-time multitasking environment. There are many ways to implement multitasking on a microcontroller; however, each has varied requirements in terms of the microcontroller's performance and characteristics. Our current system manages multiple tasks by establishing interrupts that notify the main execution thread to service another tasks. This process works without problems when the number of tasks is small and each task takes a small amount of time. However, outside of those conditions, it is possible for a single task to require too much time of the main execution thread, and thus prevent other tasks from being serviced in a timely fashion. These issues can be solved, in part, by using a real-time preemptive operating system.

One such operating system that meets many of our design requirements is Nano-RK, which was developed by Carnegie Mellon University [27]. According to their documentation, the operating system provides a “reservation based real-time operating system (RTOS)” “...with multi-hop networking support for use in wireless sensor networks.[28]” This operating system provides many features specifically targeted to wireless sensor networks including:

- Classical Preemptive Operating System Multitasking Abstractions
- Real-Time Priority Based Scheduling
- Built-in Fault Handling
 - Task Timing Violations
 - Stack Integrity
 - Unexpected Node Restarts
 - Resource Over-Use
 - Low Voltage Detection
 - Watchdog Timer
- Energy Efficient Scheduling based on a-prior task-set knowledge
- Small Footprint (<2K RAM, 16K ROM, including link layer) [28]

Implementation of this operating system in our sensor modules would require a different microcontroller platform that supports software control of the global stack pointer to

provide task context switching (a feature that is lacking in the PIC18F8722, which uses a hardware stack) [5].

The Nano-RK operating system supports power management, an area that we would also like to improve in our system. While our current system supports powering down sensors when needed, the architecture does not provide significant power savings techniques with regard to the communications system. This is mainly a software issue, as both the microcontroller and the ZigBee® radio are capable of entering into a low-power sleep mode. As pointed out in [12], smart deactivation of the radios requires close time synchronization among sensor nodes to schedule when radios should be enabled or disabled. This insures that there are not collisions with data packets and that the sensor nodes continue to have high availability.

8.2.4 Field Testing

It seems clear that there is no substitute for real-world testing. After making some of the aforementioned improvements, we hope to utilize our system in various applications. Initially, this system is slated to be applied to a new study on the characterization of airflow within airliner cabins. Another application of interest is an early warning system for detection of biological or chemical contaminants. As discussed in Section 7.2, this application provides many issues to be solved that fit well with the research we have already started, e.g., the simulator that is being developed for testing the identification of contaminant diffusion patterns and locating point of origin. Further work must be done to enable characterization of contaminants that would allow application of high-level data fusion algorithms for notifying authorities of potential problems.

REFERENCES

- [1] Martin E. Liggins, David L. Hall, and James Llinas, *Handbook of Multisensor Data Fusion: Theory and Practice*, 2nd ed. Boca Raton: CRC Press, 2009.
- [2] A. Norige, "Distributed sensing for chemical and biological defense," presented at IEEE Homeland Security Technology Conference, Waltham, MA 2009.
- [3] M. Owen, "Portable Wireless Multipurpose Sensor System for Environmental Monitoring," M.S. thesis, Boise State University, Boise, ID, 2007.
- [4] Sin Ming Loo, Mike Owen, and Josh Kiepert, "Modular, Portable, Reconfigurable, and Wireless Sensing System," *Journal of ASTM International*, Vol. 5, No. 4, May 2008.
- [5] Microchip Technology Inc., Microchip PIC18F8722 Family Data Sheet, 2008.
- [6] Crossbow Technology, "Wireless Module Portfolio," March 2009, <https://www.xbow.com/Products/productdetails.aspx?sid=156>.
- [7] Dust Networks Inc., "SmartMesh XT 2.4 GHz", March 2009, http://www.dustnetworks.com/products/SmartMesh_XT_2_4_GHz.
- [8] Jason Hill, Mike Horton, Ralph Kling, and Lakshman Krishnamurthy. The platforms enabling wireless sensor networks. 2004. *Communications of the ACM* 47, no. 6: 41-46.
- [9] Xianghui Cao, Jiming Chen, and Youxian Sun, "An interface designed for networked monitoring and control in wireless sensor networks," *Computer Standards & Interfaces*, Volume 31, Issue 3, Industrial Networking Standards for Real-time Automation and Control, March 2009, pg. 579-585.
- [10] R. Mangharam, A. Rowe, R. Rajkumar, "FireFly: A Cross-Layer Platform for Wireless Sensor Networks", *Real Time Systems Journal*, Special Issue on Real-Time Wireless Sensor Networks, Nov. 2006.
- [11] Crossbow Technologies Inc., "Product Feature Reference Chart," June 2009, https://www.xbow.com/Support/Support_pdf_files/Product_Feature_Reference_Chart.pdf

- [12] Carnegie Mellon University. "FireFly WSN Platform," June 2009, <http://www.nanork.org/wiki/FireFly>
- [13] Maxim Integrated Products, Inc. Technical Staff, *MAXIM Fixed Output 10W CMOS Step-Up Switching Regulators*, 1990.
- [14] Maxim Integrated Products, Inc. Technical Staff, *MAXIM 3.3V/5V or Adjustable, Step-Up/Down DC-DC Converters*, 1997.
- [15] Digi International Inc. Technical Staff, *XBee/XBee PRO DigiMesh 2.4 OEM RF Modules*, Digi International Inc., 2008.
- [16] Maxim/Dallas Semiconductor Technical Staff, *MAXIM DS1339 I²C Serial Real-Time Clock*, 2008.
- [17] ZedGraph. "ZedGraph Wiki," June 2009, http://zedgraph.org/wiki/index.php?title=Main_Page
- [18] F. Sivrikaya, and B. Yener, "Time synchronization in sensor networks: a survey," *Network, IEEE* , vol.18, no.4, pp. 45-50, July-Aug. 2004.
- [19] Jeremy Elson, "Time synchronization in wireless sensor networks," Ph.D. dissertation, University of California, 2003.
- [20] Leslie Lamport. "Time, clocks, and the ordering of events in a distributed system." *Communications of the ACM*, 21(7):558-65, 1978.
- [21] Jeremy Elson, "Time synchronization in wireless sensor networks," presented at the International Parallel and Distributed Processing Symposium (IPDPS), San Francisco, CA, April 2001.
- [22] Martin E. Liggins, David L. Hall, and James Llinas, *Handbook of Multisensor Data Fusion: Theory and Practice*, 2nd ed. Boca Raton: CRC Press, pg 8-9, 2009.
- [23] Lawrence A. Klein, *Sensor and Data Fusion: A Tool for Information and Decision Making*. SPIE, July 2004.
- [24] Tong Zhao, Nehorai, A., "Distributed Sequential Bayesian Estimation of a Diffusive Source in Wireless Sensor Networks," *Signal Processing, IEEE Transactions on*, vol.55, no.4, pp.1511-1524, April 2007.
- [25] Microchip Technology Inc. PIC32MX3XX/4XX Family Datasheet, June 2009.
- [26] Atmel. AVR@32 32-bit Microcontroller Preliminary, March, 2009.

- [27] Carnegie Mellon University. "Nano-RK: A Wireless Sensor Network Real-Time Operating System," June 2009, <http://www.nanork.org/>
- [28] Carnegie Mellon University. "Nano-RK," June 2009, <http://www.nanork.org/>