Winter 12-10-2014

Developing a Wireless Sensor Network Programming Language Application Guide Using Memsic Devices and LabVIEW

Xiao Xie
Bowling Green State University

Developing a Wireless Sensor Network Programming Language Application Guide Using Memsic Devices and LabVIEW

Xiao Xie

A Major Project Report

Submitted to the Graduate College of Bowling Green State University in partial fulfillment of the requirement for the degree of

Master of Technology Management

December 2014

Committee:

Dr. David Border, Chair

Dr. Sri Kolla

Dr. Todd C. Waggoner

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Acknowledgement

I am using this opportunity to express my gratitude to everyone who supported me with my MTM graduation project. I would never have been able to complete my project without the help of my committee members, help from friends and my family.

I would like to express my thanks to my project advisor, Dr. David Border, for his great patience and guidance for my research work. I am thankful to Dr. David Border to provide me valuable study materials, constructive criticism and advice when I was developing my project.

I would like to thank my committee members, Dr. Sri Kolla and Dr. Todd C. Waggoner for their patience and advice throughout my project development. I also would like to thank Dr. Alan Atalah and Ms. Heidi for their help to organize my defense and graduation plan.

Finally, I would like to thank my parents and my friends for their support and encouragement with best wishes.

Abstract

The principal objective of this project is to develop a wireless sensor network (WSN) programming language application guide for junior and senior undergraduate students in College of Technology, Architecture and Applied Engineering in Bowling Green State University. Memsic device, MoteWorks and LabVIEW software are used to conduct experiments in developing WSN applications after both software and hardware platform are verified to be usable with experimental and statistical analysis. The guide is divided into six chapters including both theoretical knowledge and practical experiments in WSN area. Programs, both in nesC language and LabVIEW, are improved from previous work, tested to run successfully and noted in detail.

Chapter I Introduction

Context of the Problem

With the rapid development of Micro-Electro-Mechanical System (MEMS), System on Chip (SOC), Wireless Communication and Low-Power Embedded System Technology, Wireless Sensor Network (WSN) became prominent in recent years. As one type of sensor networks, WSN has been applied in many industries like health care, agriculture, military and environmental monitoring. The ON World's survey of 216 industrial automation professionals (Hatler, 2013), in collaboration with ISA, Hart Communication Foundation, and the Wireless Industrial Networking Alliance (WINA), shows the market of industrial WSN had doubled from the year 2010 to the year 2012. In the next following years, the installed wireless industrial field device will still increase as shown in Figure 1.1.

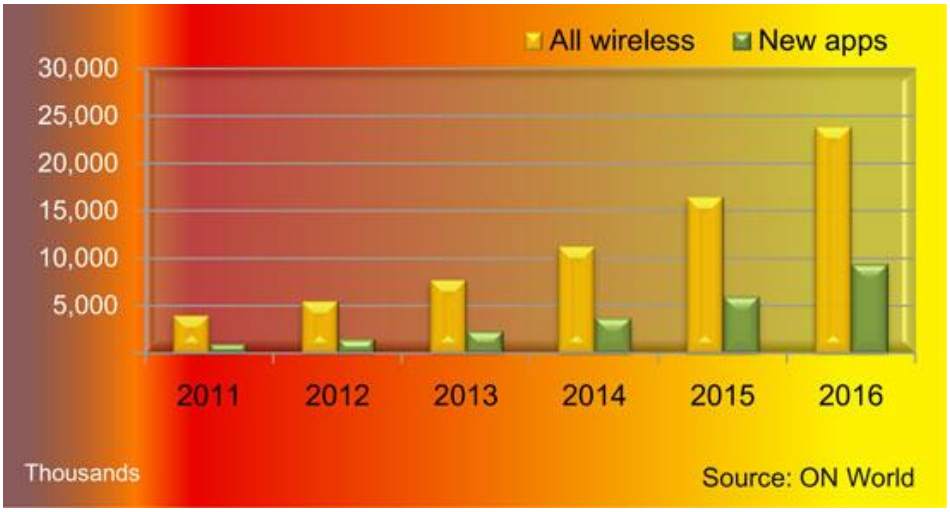

Figure 1.1 Global installed industrial Wireless Sensing Points (2011-2016)

Since the marketplace is booming and expanding, the demand of engineers and specialists in the WSN area has been increasing. The education in the WSN field becomes pressing for the students in colleges, who aspire to working in WSN industry.

Memsic manufactured wireless motes and sensors provide an easy path for the students to learn the physical environment of WSN, since the development kits are easy to obtain and can be available in

most Electrical and Computer Engineering labs; Its Crossbow Wireless Sensor Network Kit provides for the installations of TinyOS, MoteWorks and MoteView, which allow programming of motes and observation of data from sensors; LabVIEW, as a graphical programming language application, can be linked to a WSN and be used to view the output of the sensors on a computer display. Researchers have developed some tutorials to introduce WSN programming application but not a comprehensive and detail guide for students who are not so familiar with WSN environment. In allusion to practical difficulties with the actual learning situation of college students, this project developed a programming guide to help students better understand the fundamentals of WSN environment and programming.

NesC (Network Embedded System C) language programming is the basic programming skill when people learn WSN. However, the novice programmers always face difficulties when learning the nesC, C or other programming languages. The Study of the Difficulties of Novice Programmers of the Tampere University of Technology (Lahtinen, Ala-Mutka, & Järvinen, 2005) shows the current situation that students have problems and lack of interests to programming in the universities. The survey shown in Table 1.1, indicates that the different aspects of learning difficulties for the university students.

Question	Code	Students			Teachers		
		N	Avg	Std	N	Avg	Std
THE COURSE CONTENTS							
What kind of issues you feel difficult in learning programming?							
Using program development environment	I1	553	2,43	0,99	33	2,61	0,90
Gaining access to computers/networks	I2	536	2,11	0,95	32	1,97	0,78
Understanding programming structures	I3	556	2,92	1,02	33	3,27	0,67
Learning the programming language syntax	I4	555	2,75	1,01	33	2,70	0,73
Designing a program to solve a certain task	I5	555	3,12	0,98	33	3,97	0,73
Dividing functionality into procedures	I6	543	3,10	1,09	31	4,06	0,63
Finding bugs from my own program	I7	549	3,28	1,03	33	3,91	0,77
Which programming concepts have been difficult for you to learn?							
Variables (lifetime, scope)	C1	541	2,10	0,97	34	2,41	0,70
Selection structures	C2	552	1,98	0,90	34	2,38	0,70
Loop structures	C3	551	2,09	0,97	34	2,79	0,91
Recursion	C4	512	3,22	1,03	31	4,06	0,96
Arrays	C5	526	2,79	1,15	33	3,24	0,71
Pointers, references	C6	518	3,59	1,04	32	4,44	0,56
Parameters	C7	513	2,60	1,09	32	3,47	0,76
Structured data types	C8	496	2,90	1,03	31	3,45	0,81
Abstract data types	C9	499	3,02	1,10	31	4,06	0,81
Input/output handling	C10	519	2,96	1,04	32	3,75	0,88
Error handling	C11	481	3,33	1,01	32	4,13	0,79
Using language libraries	C12	465	3,04	1,09	32	3,88	0,71
LEARNING AND TEACHING PROGRAMMING							
When do you feel that you learn issues about programming?							
In lectures	S1	543	3,01	1,01	33	3,21	1,02
In exercise sessions in small groups	S2	510	3,44	1,10	32	3,84	0,99
In practical sessions	S3	514	3,77	1,03	31	4,35	0,75
While studying alone	S4	546	3,79	1,06	31	3,42	0,72
While working alone on programming coursework	S5	539	3,98	1,09	33	4,00	0,79
What kind of materials have helped/would help you in learning programming?							
Programming course book	M1	515	3,35	1,03	33	3,30	0,88
Lecture notes/copies of transparencies	M2	539	3,39	1,05	34	3,47	0,71
Exercise questions and answers	M3	523	3,33	1,07	34	3,62	1,02
Example programs	M4	551	4,19	0,86	34	4,24	0,65
Still pictures of programming structures	M5	490	3,15	1,00	30	3,70	0,75
Interactive visualizations	M6	315	3,33	1,03	27	4,07	0,87

Table 1.1 Results on sections course contents and learning and teaching.

The survey asked the students' and teachers' response on a five-point scale and gave the average scores and the standard deviation of scores to see the score distribution. From the Learning and Teaching Programming part, it shows while working alone on programming coursework, students learn issues about programming. In that condition, reference materials are particularly important to give instructions to students. Among all the materials that help in learning programming, the survey shows example programs are the most useful materials. These findings in the survey lead this project to focus on the right direction when considering how to structure the content of the guide.

Statement of Problem

This project is aimed to develop a WSN language application programming guide for junior/senior level undergraduate students and graduate students. The project is achieved by using Memsic's wireless hardware devices, Memsic software development platform and LabVIEW. The ability to program wireless nodes and gateway boards helps future students to gain skills needed in the growing WSN marketplace. It will motivate the students to better understand the logic and algorithm of programming. The programming interface includes the use of virtual instruments (VIs), since LabVIEW is acknowledged as a strong tool to supervise the status of sensor boards and to obtain real-time sensor data.

Statement of Objectives

This project has seven main objectives to achieve. They are described by procedures as below.

1. Study the history and trend of WSN development.
2. Investigate the features of existing WSN hardware/software platform.
3. Survey the weakness and limitations of the existing WSN guide for both educational and business use.
4. Obtain and verify environmental variable data.
5. Explain and note existing sample programming applications.
6. Develop new applications and assess the performance.
7. Integrate above objectives and other findings into the guide.

Description of the Product

This project is aimed to develop a WSN programming guide. Examples and sample programs are given in the guide to help students to clarify the concepts and give students an opportunity to learn how to program by themselves. The structure of this guide is planned as shown in the table below.

Contents of the Guide	
Chapter 1	Introduction to Computing Basics
Chapter 2	Wireless Sensor Network Technology History and Features
Chapter 3	Wireless Sensor Network Hardware Features: Memsic
Chapter 4	Wireless Sensor Network Software Features: Tiny OS & MoteWorks
Chapter 5	NesC Language Programming
Chapter 6	Sensor Data Display in LabVIEW
Appendix	

Table 1.2 Contents of the project guide

Chapter 1 covers computer memory and C programming basics. It helps the students who have no knowledge of these topics or help those who need to refresh their memory.

Chapter 2 briefly presents the development of history of WSN technology. The application fields in the use of education, business and industries are briefly discussed in general. It also presents the current research situation and future trends.

Chapter 3 focuses on the Memsic WSN development kit, including Iris motes, environmental sensor boards and USB interface boards. It gives a brief specification explanation of the development kit. The detailed specifications of each part are provided in the Appendix section.

Chapter 4 introduces the environment of software platform MoteWorks and Tiny OS including Cygwin (a Unix-like command language interface), a programming complier, etc. It involves in the installation, parameter settings, other general introductions and preparations before utilizing the software platform.

Chapter 5 explains how to use nesC language to program and compile and install codes into motes. Sample programs provided in the MoteWorks are explained and noted. Other programming examples are verified and explained in this chapter.

Chapter 6 explores the linking of WSN to LabVIEW VIs and displays the sensor data.

The appendices are designed to provide all reference material required for the topics covered in the guide.

Product Performance Specification

This guide is intended to be used for college students in junior/senior or graduate level, who should have taken some prerequisite courses and build the basic knowledge of programming and networking. It not only provides the fundamental principles of programming but also guide the students how to program by themselves and observe the results by visual interfaces. It is be a valuable tutorial for college-level students who are interested in WSN.

User Specification

Readers of this guide should have had taken some prerequisite courses like Instrumentation, Digital Communication and Networking, and Digital Electronic Components and Systems. Knowledge of WSN would be helpful but is no necessary. Although this guide is written for the students who have no experience in WSN. For the chapter of nesC language programming, a basic knowledge of programming in C or any other languages is required. For the chapter of WSN application linked to LabVIEW, a basic knowledge of LabVIEW is also required. There are plenty of sources available online to help understand programming and LabVIEW.

Significance of the Project

In the reality of rapid development of WSN technology and increasing of WSN products, the demand of educated talents in WSN field has been rising. There is no specific course available in BGSU related to WSN and there is no well-organized and integrated study guide for the students to understand

WSN structure and programming skills. This current situation makes it difficult to train people for work in WSN industries. This project is to give a possible solution by introducing a valuable WSN programming application guide. By comparing with existing WSN study materials, the project finds their weakness and makes the supplements into its own guide.

Definitions of Terms

MEMS- Micro-Electro-Mechanical System is a technology that utilizes small devices that can combine electrical and mechanical components.

Algorithm- In programming, algorithm is a step by step procedure for calculations.

Memory- In computing, memory stores programs and data for a temporary or permanent use.

Instrumentation- Measure and control process variables in the production and manufacturing field.

Cygwin- An operating system environment like UNIX and provides a command line interface that can be used in Window systems.

Chapter II Literature Review

Introduction

This chapter covers the theoretical background of WSN, the current research situation, the future development trend and existing WSN guides. Firstly, it gives a brief introduction of history of WSN development. Then the existing products of hardware device and software platform are discussed. Also the application areas, current research and development trends are presented. Since this project is to develop a WSN programming guide, an exploration of existing study materials about WSN is introduced by developing a comparison list with the developing valuable guide. Overall, the literature review is to give general WSN knowledge to educate the readers, to learn methodology and gain data from previous research and to find support resources for the project.

History of Wireless Sensor Network Development

On the basis of "An Overview on Wireless Sensor Networks" (Nack, 2010) at the Institute of Computer Science, Freie University Berlin, the history of WSN development can be divided into four stages. WSN technology can be traced back to some projects in Cold-War Era in the United States. For instance, the Sound Surveillance System (also known as SOSUS) was aimed to track Soviet submarines by placing acoustic sensors underwater at key locations as listening posts by US Navy. In the early 1980s, the Distributed Sensor Network (DSN) program was initiated at Defense Advanced Research Projects Agency (DARPA). DSN consists a set of sensors, which are intelligent and distributed into different areas to obtain and analyze environmental variables from data collected. All the sensors were supposed to operate autonomously and collaborate with each other. Since personal computers and workstations were not popularized and the size of sensors is quite large during that period, the development of many potential DSN projects was limited. However DARPA's efforts, contributions and achievements in DSN drew the interests of US military due to warfare purpose in the late 1980s. The large replenishment of a fund gave the scientists more possibilities to develop sensor network technology. Therefore, WSN technology made a huge and fast progress in the early 1990s. The latest stage of WSN development lasts

till present. With the rapid development of computing, Micro-Elecrtro-Mechanical System (MEMS) and other technologies, the sensors are becoming smaller in size and cheaper in price. This advancement provides WSN the opportunities for commercial use in more areas. Companies like Memsic and Crossbow Technology begins to produce wireless motes, sensors and software support. The standardization of protocols also becomes more and more mature. The standards like Zigbee, 802.15.4 and 6LoWPAN are built and commonly used in WSN communications.

Survey on Wireless Sensor Network Hardware and Software Platform

i. Wireless Sensor Node (Mote)

A sensor node (also known as a mote) is the fundamental unit of a WSN. It is used to collect information from sensors, process commands and communicate with other sensor nodes. A sensor node usually has five components. They are the controller, transceiver, memory, power source and one or more sensors.

Referring to the "Mini Hardware Survey" (Bokareva, 2014) and "Embedded WiSeNts Platform Survey" (WSN Research Group, 2014) maintained by the Imperial College, London, there are currently numerous available sensor nodes for the use of education, research and commerce. From the previous study, information is provided about BTnode, COOKIES, EPIC mote, Telos, SunSPOT and the others. Since the limitation of available sources in the Electronics and Computer Engineering Technology (ECET) lab is limited, the project mainly focuses on the Iris mote, which is accessible in the ECET lab.

Compared with other sensor boards like MICA2, MICAz and TelosB, Iris has an improved radio range. The outdoor range is over 300 meters and indoor range is more than 50 meters. Iris is a 2.4 GHz mote coming with the processor/radio board of the XM2110CA. XM2110CA is built on the low-power microcontroller ATmega1281. It can be used to run sensor application and process network or radio communications stack simultaneously. The Iris has a 51-pin expansion connector which supports

interfaces, such as, Analog Inputs, Digital I/O, Serial Peripheral Interface and others. It provides the ability to connect the Iris motes with a large number of external devices. For example, in the Crossbow product, the sensor board MTS420 is used with its Light, Temperature, Humidity, Barometric Pressure and Seismic Sensors from Memsic. By utilizing the sensor boards, environmental variable data can be monitored with the help of MoteWorks software, which was developed by Crossbow Technology. Also, Iris can be performed as a base station by plugging with MIB510 or MIB520 USB interface board. The MIB510/MIB520 provides the functions of data transfer and in-system programming to Iris node. The detail specification of Iris, MTS420 and MIB520 is provided in the appendix.

ii. Operating System

The Operating System of WSN is aimed to manage WSN hardware resource by providing a collection of software application. The Operating System of WSN is not as complex as Windows, Android, Mac OS and other general Operating Systems because of the typical requirements and constraints of WSN hardware. The current sensor nodes can be run in some specific WSN OSs such as Contiki, LiteOS, Nano-RK, TinyOS and so forth. The Memsic Wireless Sensor Network Kit available in the ECET lab in BGSU provides the installation of TinyOS as the software platform to support the Iris node.

TinyOS was firstly developed by UC Berkeley as part of DARPA-sponsored Neural Engineering, Science and Technology (NEST) program, and initially released in 2000. It is in support of WSN as a free and open-source operating system ran on the computer. “TinyOS features a component-based architecture, which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks.” (Crossbow, 2007). It contains four main components including network protocols, distributed services, sensor drivers and data acquisition tools.

NesC is a component-based, event-driven programming language. It is used to develop applications for TinyOS. A nesC application consists of one or more components linked together. These

components form an integrated and executable application. NesC has two types of components, module and configuration. Module provides application codes and realizes one or more interfaces. The provider of the interface must declare two groups of functions, one is “command” and the other one is “event”. If one component wants to use the command in an interface, it must realize the event in the interface. Configuration is a component to wire other components, connect the interfaces used by different components. Simply, the programmers develop a nesC application by creating a group of modules and linking them together through a configuration.

NesC defines the concurrency model for tiny OS. TinyOS executes only one application one time. The components which form an application are from both system itself and custom components for specific application users. When running an application, there are two threads of execution: one is task operation and the other one is hardware event handler. Tasks are delayed functions and once they are scheduled, they will run till end and are not allowed to preempting each other in the execution. Hardware event handler is used to process the hardware interruption. Although hardware event handlers also need to be completed, they can preempt the executions of other tasks and hardware event handlers. If a command or an event wants to be executed as part of hardware event handler, the programmer must declare that by using the keyword of “async” (Crossbow, 2007).

Application, Research and Development Trend

WSN is currently a growing research field which involves in multi-disciplinary, highly cross and highly integrated knowledge. It combines sensor technology, embedded computing technology, modern network and wireless communication technology, distributed information processing technology, etc. It enables real-time monitoring, perception and gathering of all kinds of environment or object information through integrated microprocessors. This information can be sent wirelessly to the user terminal. WSN has a very broad application prospect in military defense, biological and medical industries, agriculture, urban management, environmental monitoring, disaster relief, anti-terrorism and remote

monitoring/controlling in dangerous areas. It draws the attention to the academic and industrial fields in many countries (Buratti, Conti, & Verdone, 2009).

Survey on Existing Wireless Sensor Network Study Materials

This project is aimed to develop a relevant WSN programming application guide. To strengthen the superiority of the guide, the existing and reachable WSN study materials are read and investigated, including books, lectures, journals and other online resources. By comparing the materials with each other and summarize the main content, the general views and emphasizes was established which are discussed in my project. This section provides the summaries of different materials below and attaches a comparison list in the appendix.

- "Wireless Sensor Networks Research and Application" (Wang) contains four main parts, including the general introduction, current research situation, research trends and typical application. It talks about the history of development and features of WSN, current WSN products like "smart dust" and motes. When discussing research trends, the guides explain in several aspects, including protocols, network management, data management and application support service. To show the application of WSN, this guide illustrates the examples of moisture sensor system, "Sensicast Art" to monitor and protect valuable works of arts, "Senera" to monitor the transportation infrastructure and so on.
- "Wireless Sensor Network Programming Using TinyOS" (He, 2012) firstly introduce the architecture of WSN. It also talks about the components and interfaces, tasks and concurrency, compilation and tool chain in the programming. It gives two programming examples; one is Anti-Theft module and the other one is Radio Message module. It provides the codes without a detailed explanation. At last, it shows the tutorial how to install TinyOS and gives some sample exercises without the solutions.

- "Wireless Sensor Networks: Motes, NesC and TinyOS" (Schonwalder & Harvan, 2007) firstly introduces the general information of WSN, such as, definitions, application, hardware devices, research topics and constraints. Secondly, it introduces the TinyOS and nesC with the simplest application of "Blink". At last the guide shows how to connect WSN to the internet and it involves lots of prospective knowledge of network administration.
- "Wireless Sensor Networks: Technology Roadmap" (Desai, Jain & Merchant) explains WSN theories, including the history of WSN, current and future research and development trends and applications. It provides a good reference for students to learn WSN background knowledge but lacks practical uses.
- "Crossbow: MoteWorks Getting Started Guide" (Greene & Khamphavong, 2007) mainly presents the installation of MoteWorks, concepts of TinyOS and nesC. It also gives the simple application of "Blink". It uses XSniffer to view sensor data through the network. At last the guide gives the data logging application with some descriptions by words.
- "TinyOS Tutorial" (Fok, 2004) introduces MICA2 mote, MTS300CA/MTS400/420 sensor board and MIB510 programming board. These devices are available in ECET lab in BGSU. Then it also shows the installation and configuration of TinyOS. NesC programming is discussed a little bit by giving some examples. The Network Communication part mainly explains how to send and receive a message.

In addition, another project was accomplished by a graduate named Omar El Aridi with the help of Dr. David Border in Bowling Green State University. "Developing and Designing Undergraduate Laboratory Wireless Sensor Network Exercises" (Aridi, 2010) mainly included five labs to help students familiar with WSN starter kit, understand the basic skills how to program the nodes and to obtain the sensory data.

This project mainly helps the student understand the coding logic and algorithm before utilizing the programming applications. It has some overlapping parts with Mr. Aridi's work but the core

significance of this project is to develop a valuable study guide of WSN not a WSN lab manual and the core content of this project is the programming part with detail codes (Aridi, 2010) (Border, 2012).

In conclusion, the guides existed currently are mostly fragmentary and not well-organized. Most guides are focusing either on the introduction to the hardware platform or software environment. Some guides introduce the hardware devices which are not available in the ECET lab in BGSU. Some guides are aimed to train students the programming skills but do not come with adequate examples. This current situation causes the inconvenience to start learning WSN from fundamentals. Students will easily get confused by moving the study of one guide to another and feel difficult to understand the programming principles without practical application. This project is trying to provide a possible solution to this issue to help students study Wireless Sensor Network more effectively.

Chapter III Methodology

Introduction

This chapter has the restatement of problems and objectives, which were firstly introduced in Chapter 1. With the clarification of problems and objectives, data collection and experimental procedures in this project are further explored. Experimental environment installation is introduced firstly. In the data collection part, environmental variables including temperature, pressure, and humidity and so on, are measured by wireless sensors and observed in the data acquisition tool, MoteView and LabVIEW. The cost, time and the amount of data needed are determined. Some explanations about how data is analyzed and interpreted are also included in the data collection section. Furthermore, principles of nesC programming are presented. The sample program provided in MoteWorks is explained with the use of helpful notes. The sample applications are debugged and ran and the results are observed and verified on the motes.

Restatement of Problem

This project is aimed to develop a WSN language application programming guide for junior/senior level undergraduate students and graduate students. The project is achieved by using Memsic's wireless hardware devices, Memsic software development platform and LabVIEW. The ability to program wireless nodes and gateway boards helps future students to gain skills needed in the growing WSN marketplace. It will motivate the students to better understand the logic and algorithm of programming. The programming interface includes the use of virtual instruments (VIs), since LabVIEW is acknowledged as a strong tool to supervise the status of sensor boards and to obtain real-time sensor data.

.

Restatement of Objectives

This project has seven main objectives to achieve. They are described by procedures as below.

1. Study the history and trend of WSN development.
2. Investigate the features of existing WSN hardware/software platform.
3. Survey the weakness and limitations of the existing WSN guide for both educational and business use.
4. Obtain and verify environmental variable data.
5. Explain and note existing sample programming applications.
6. Develop new applications and assess the performance.
7. Integrate above objectives and other findings into the guide.

Experimental Environment Installation

The Crossbow's MoteWorks CD-ROM provides the installation of the experimental environment. It is compatible with Microsoft Windows XP. The main software packages, which are included in the CD-ROM and are used in the project, are Cygwin, Programmer's Notepad, and MoteView. Other packages are recommended to be installed to keep the integrity of the environment. Further installation guide advice can be found in the "MoteWorks Getting Started Guide".

Sensor Data Collection

i. MoteView

In this project, MoteView is used to monitor data from wireless sensors. By comparing different data from different sensors, MoteView's environmental sensor readings can be verified. MoteView can automatically discover the wireless nodes when connecting the live sensor network to a local PC. The node list shows with ID and name at the left side of the interface. It has seven tabs in the main window.

- The data tab includes the column of voltage, humidity, pressure and other parameters from the sensor boards. By right clicking the column, the unit of the values can be converted.

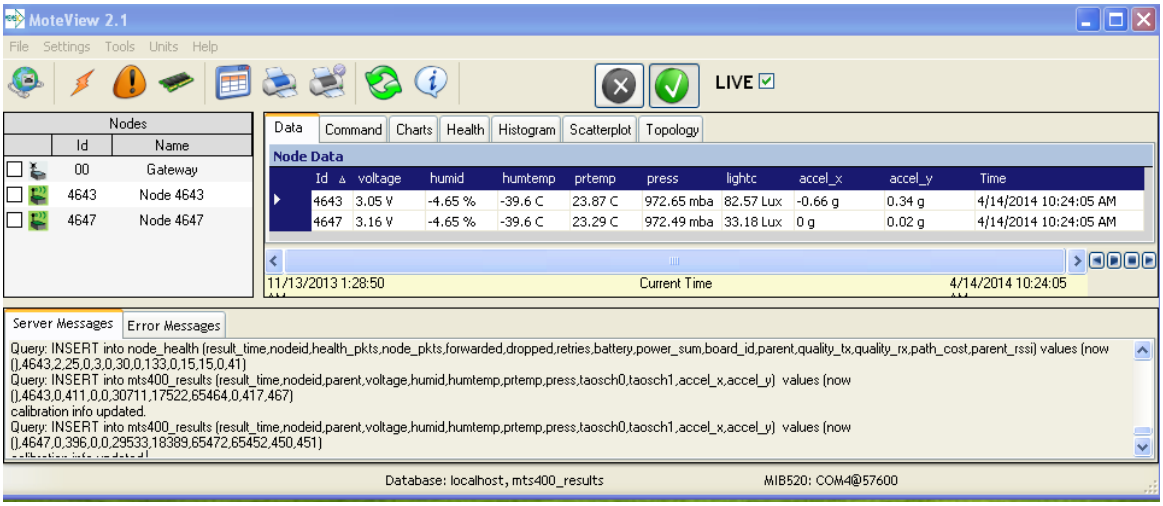

Figure 3.1 MoteView data tab

- The command tab can modify the data rate and obtain the 64-bit ID of a node. It can also change the LED status on the node.

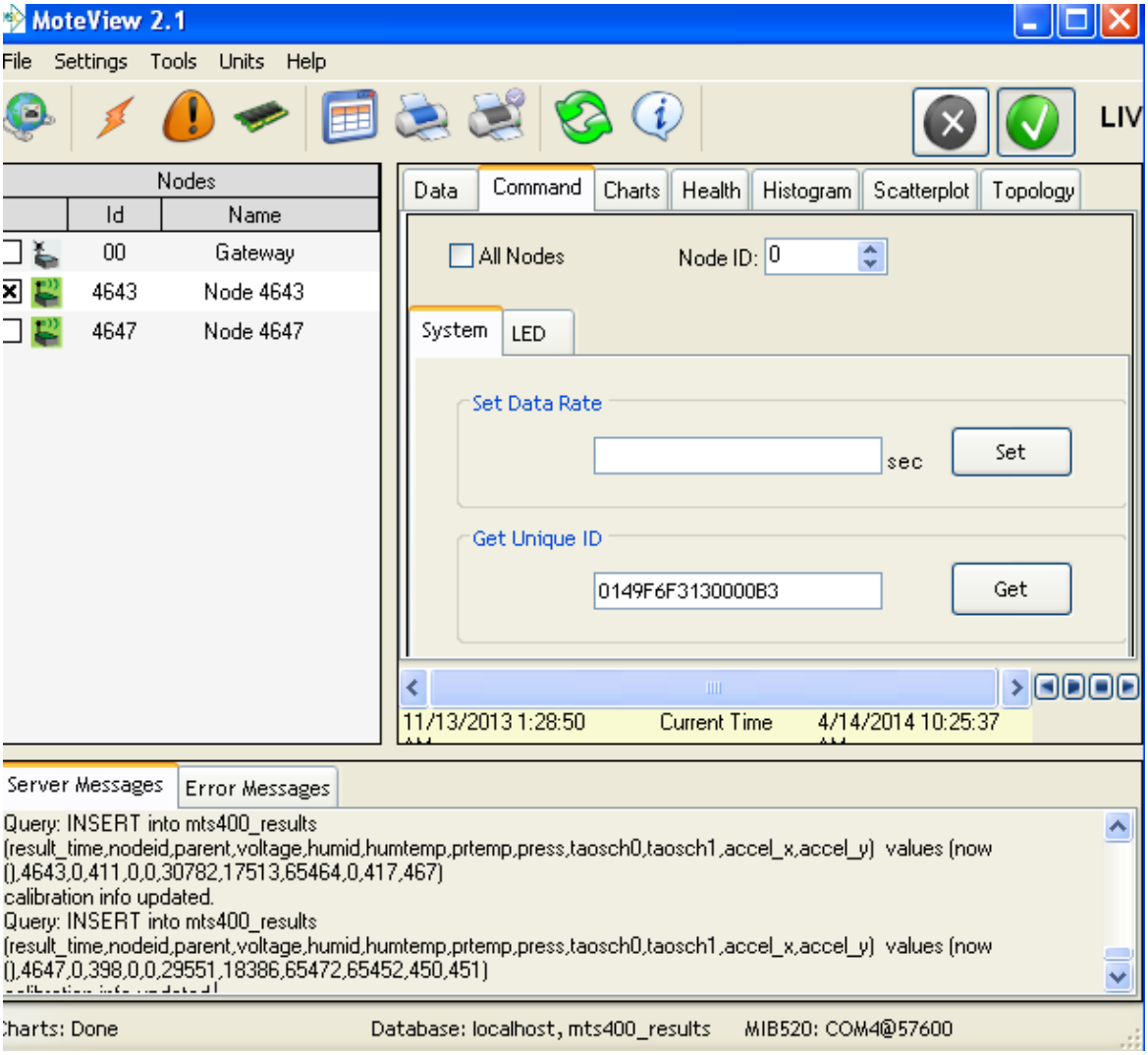

Figure 3.2 MoteView command tab

- Chart tab shows graphs of data with the time changing.

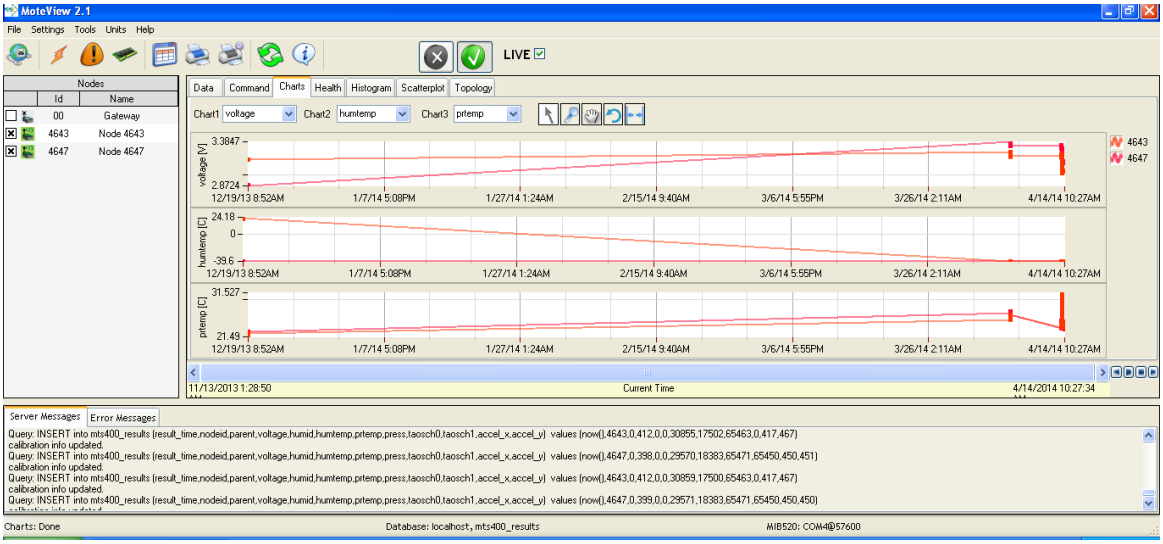

Figure 3.3 MoteView chart tab

- Health tab shows health packets readings from the node.

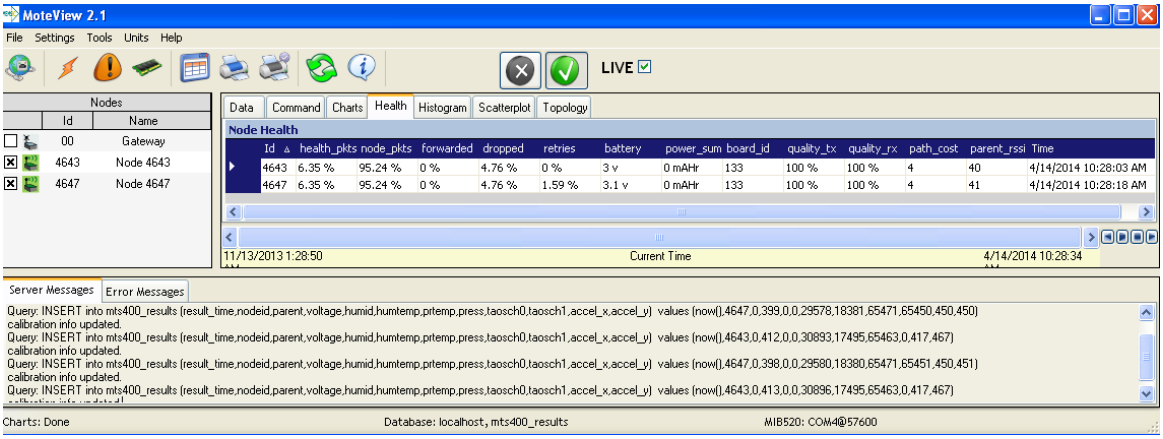

Figure 3.4 MoteView health tab

- Histogram tab shows the data by the bar chart. It is an easy way to observe the distribution of sensor values.

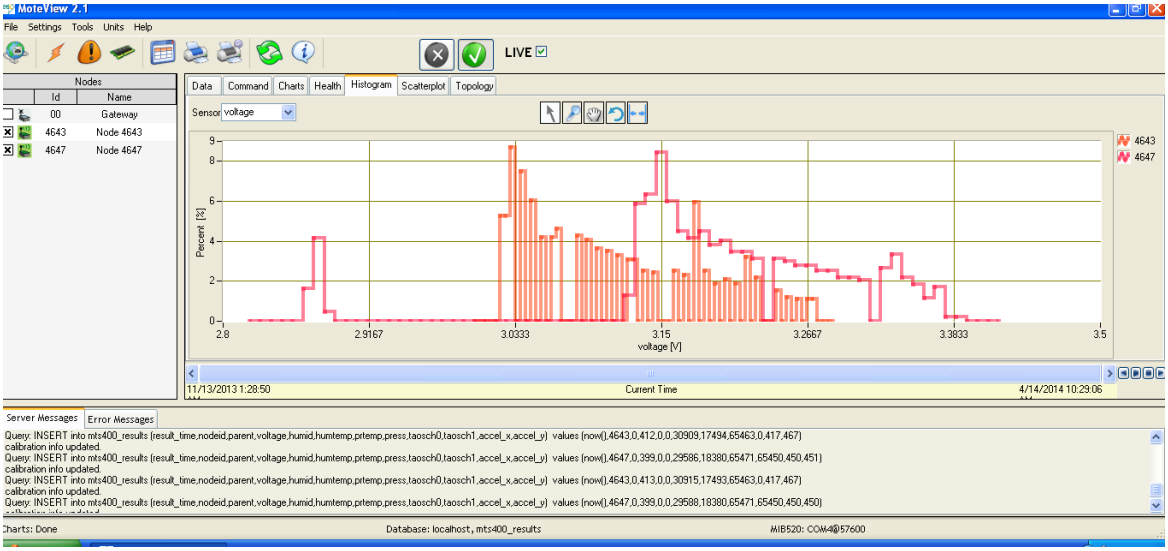

Figure 3.5 MoteView histogram tab

- Scatterplot tab shows the comparison of two different data types. It provides the ability to determine the correlation of two data types.

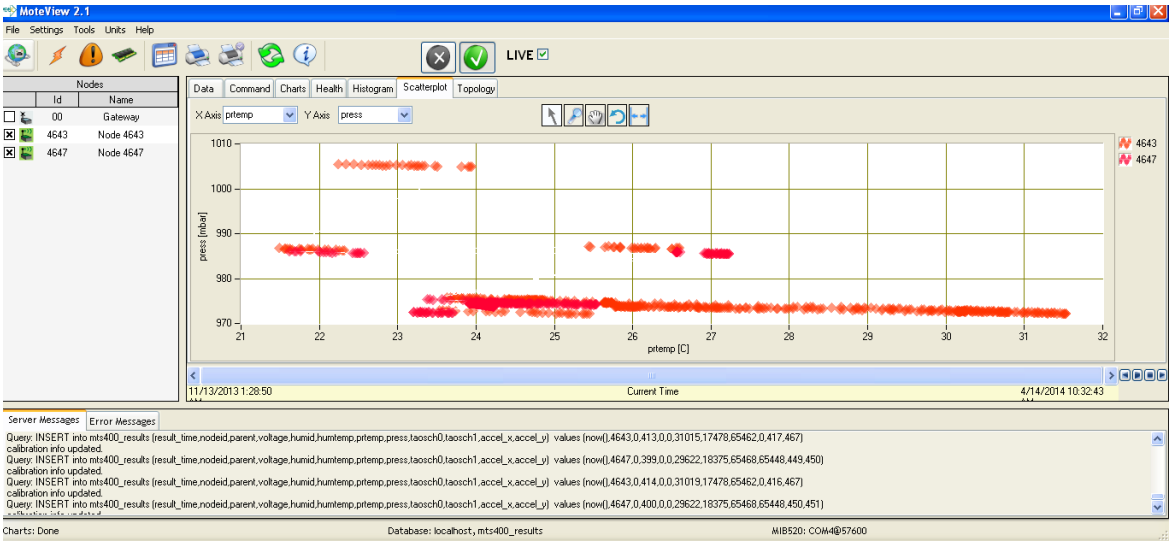

Figure 3.6 MoteView scatterplot tab

- Topology tab allows the users to observe and modify the location of nodes on the network map.

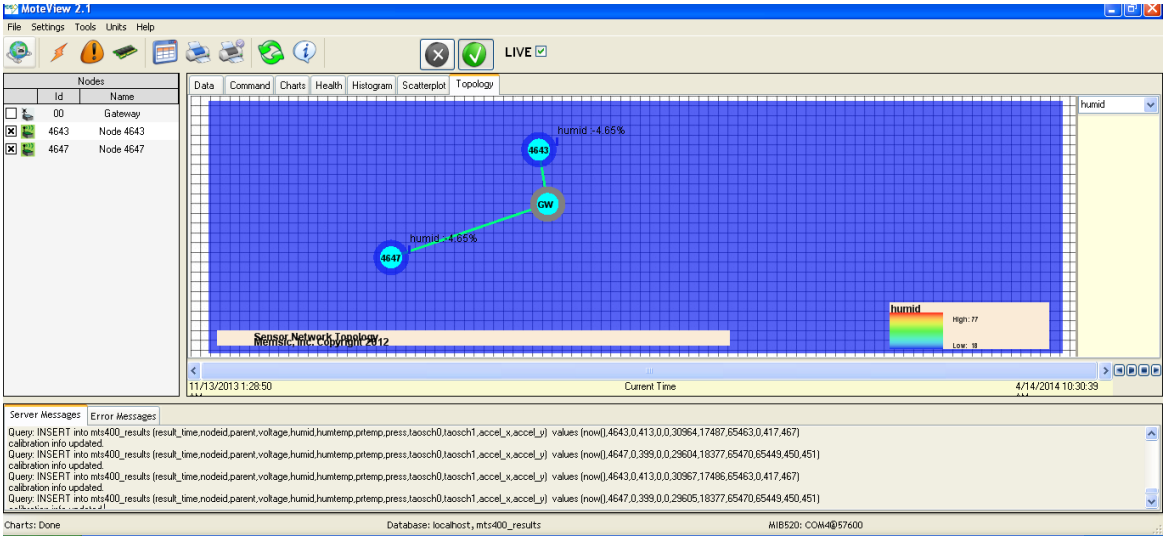

Figure 3.7 MoteView topology tab

ii. LabVIEW Plug and Play Instrument Driver

LabVIEW, as a graphical programming language, provides the specific driver to allow users link a WSN application to LabVIEW and view the output data from the sensor. It provides a series of sample applications with different functions like reading data and display health, WSN Check Timeout, WSN Check Packet Type. By viewing the data obtained from LabVIEW, the project demonstrates if the results match the data obtained in the MoteView. Since the LabVIEW program provided in the drive is not particularly used for Memsic Device, programs needed to be revised and improved to fit into the project data collection (National Instruments, 2013).

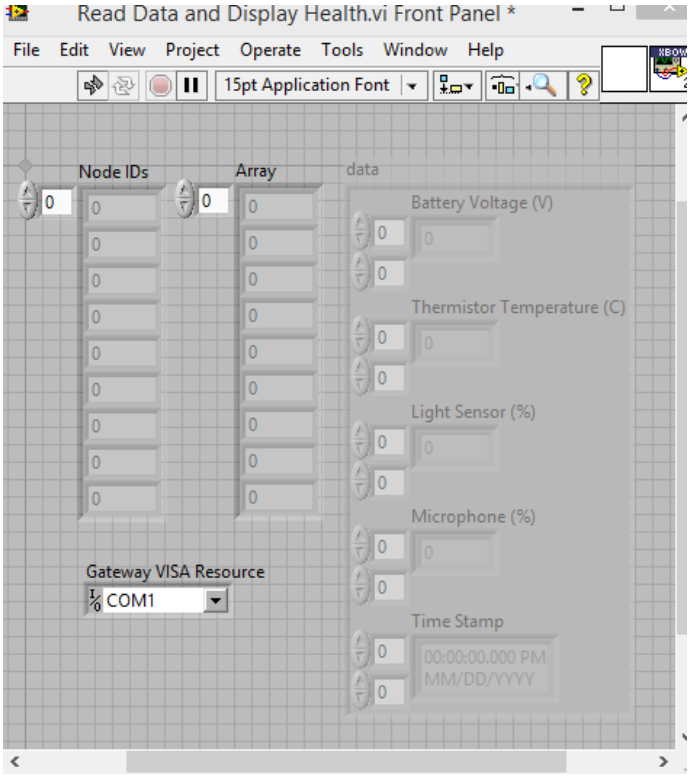

Figure 3.8 LabVIEW read data and display health interface

For industrial and commercial work, in the data collection process, the cost, time and the amount of data need to be determined. In this project, the hardware/software platform and its cost is not of interest. Because data from sensor boards can be obtained immediately, the project can set up different groups of the time in different dates to sample real-time environmental variables with some placebo groups measured by instruments.

Sample variables from different sensor boards are compared based on their mean, standard deviation and correlation. This data analysis process verifies if the sensor boards work well or not, demonstrate if LabVIEW programming revised is correct or not, and obtain reliable data when running nesC applications.

The formulas which are used in the analysis are provided below (Lane, 2014).

- Sample Mean

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i. \quad (1)$$

- Sample Standard Deviation

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}. \quad (2)$$

$\{x_1, x_2, ..., x_N\}$ are the sample values and $\overline{x}$ is the sample mean, N stands for the size of the sample.

Other statistics like median, skewness and regression can be calculated by using Microsoft Excel or Statcrunch. Visual graphs like bar chart, scatter plot and histogram can also be either viewed in MoteView or created in Statcrunch.

NesC and TinyOS Programming

Memsic Sensor Network Kit comes with a plenty of sample nesC programming applications. Programmer's Notepad, the IDE (Integrated Development Environment) of nesC programming, shows the directory of applications provided. By simply expanding one of the application's directories, five files including Makefile, Makefile.component, "nc." of application's configuration, "nc." of application's module and README are shown.

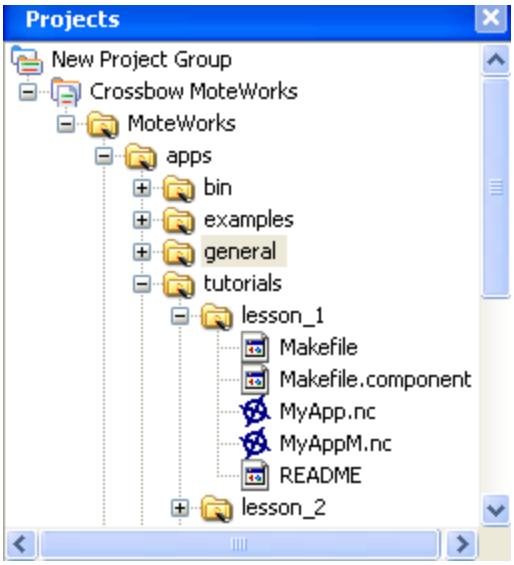

Figure 3.9 MoteWorks sample programs directory

The general steps to create a simple nesC program are provided as below:

1. To create the Makefile: Makefile is to define how to compile and connect the source files to general an executable file, and to define the dependencies between the source files.
2. To create Makefile.component: Makefile.component is to define the sensor board which will be used and the application component.
3. To create a configuration: The configuration is to link the different modules.
4. To create a module: The module is to type in the programming codes and provide the application's function.
5. To compile and install the program into a mote: The way to compile the codes in the Programmer's Notepad is to select "Tools" then "make iris" and the output will be displayed in the "Output" window. The way to install the codes into a mote with programming board is to select "Tools" then "shell".

The "Blink" application is a basic application to toggle the LED on the mote on each clock interrupt of 1 second. It has three main files. The detail codes are provided in the appendix (Levis, 2006).

“Blink.nc” is the configuration of the “Blink” application. It starts with the keyword “Configuration” and the real content of the configuration is followed by “implementation”. In this application, “Main”, “BlinkM”, “SingleTimer” and “LedsC” are the components used. “Main” is the first component executed in the TinyOS application and it must be included in a TinyOS application. The “StdControl” is used to initial and launch a common interface of the TinyOS components. The little arrows in the “Blink.nc” mean to combine interfaces of components at both sides. “BlinkM.Timer -> SingleTimer.Timer;” means combine the interface “Timer” of “BlinkM” with the interface “Timer” of “SingleTimer”. “Blink.Timer” uses the interface “Timer” and “SingleTimer.Timer” achieves the interface “Timer”. “BlinkM.Leds -> LedsC;” means “BlinkM.Leds -> LedsC.Leds;”.

“BlinkM.nc” is to achieve the function that toggles the red LED when a Timer fires.

“SingleTimer.nc” is to achieve the function as a timer.

For this project, the possibility to improve some sensing applications is discussed. Firstly the application to read sensor data from the sensor board is modified and created. The application to send the message that contains sampling sensor data to programming board through the port needs to be developed. Finally, the project is trying to display the sampling data message on PC.

In conclusion, this chapter restated the problem statement and objectives of the project. It also gives a guide about how to set up the experimental environment. The data gained from sensor boards is analyzed and interpreted by using statistical tools. The tools to collect data are MoteView and LabVIEW; the tools to analyze data are Excel and Statcrunch. This chapter also provided an overview of the procedures to develop a simple nesC application and how to compile and install the application.

Chapter IV Results and Findings

Data Collection and Statistical Analysis

To verify the accuracy of the sensor boards for the future uses in the applications, the data is collected by choosing different data types within two different nodes. The unit of each type of data can be modified as well. All the data is obtained every 6 minutes in a total one hour in the same place sharing the same environment.

Figure 4.1 Place where the two nodes locate

The data acquisition tool is MoteView provided by Crossbow. MoteView also provides visual tools to directly observe data floating over a specific period. Further statistical analysis within or between sensor boards is processed after the data collection is completed.

Id	voltage	humid	humtemp	prtemp	press	lightc	accel_x	accel_y	Time
4643	2.84V	-4.65%	-39.6C	24.7C	984.3mbar	31.97Lux	-0.4g	0.44g	7/16/2014 6:13:17 AM
4647	2.88V	59.91%	24.53C	24.69C	983.58mbar	31.97Lux	-0.28g	-0.08g	7/16/2014 6:13:14 AM

Figure 4.2 MoteView interface to display node data

Data by ID / Time	Temperature/C		Light Intensity/Lux		Voltage/V		Pressure/Mba	
	4643	4647	4643	4647	4643	4647	4643	4647
6:20AM	25.4	25.39	33.81	33.81	2.82	2.85	983.99	983.19
6:26AM	25.81	25.77	35.65	35.65	2.81	2.85	983.87	983.12
6:32AM	26.12	26.05	41.17	43.01	2.8	2.83	983.88	983
6:38AM	26.34	26.24	64.17	64.17	2.79	2.83	983.76	983.1
6:44AM	26.52	26.42	71.53	75.21	2.79	2.83	983.75	983
6:50AM	26.64	26.55	75.21	78.89	2.78	2.82	983.74	982.93
6:56AM	26.79	26.65	86.25	89.93	2.78	2.81	983.69	982.9
7:02AM	26.87	26.69	93.61	100.97	2.78	2.81	983.72	982.96
7:08AM	26.94	26.74	100.97	108.83	2.77	2.81	983.83	983.11
7:14AM	26.98	26.8	104.65	113.85	2.76	2.81	983.8	982.94
7:20AM	26.92	26.82	104.65	108.33	2.76	2.81	983.99	983.05

Table 4.1 Data from sensor node 4643 & 4647 created by July 16th, 2014

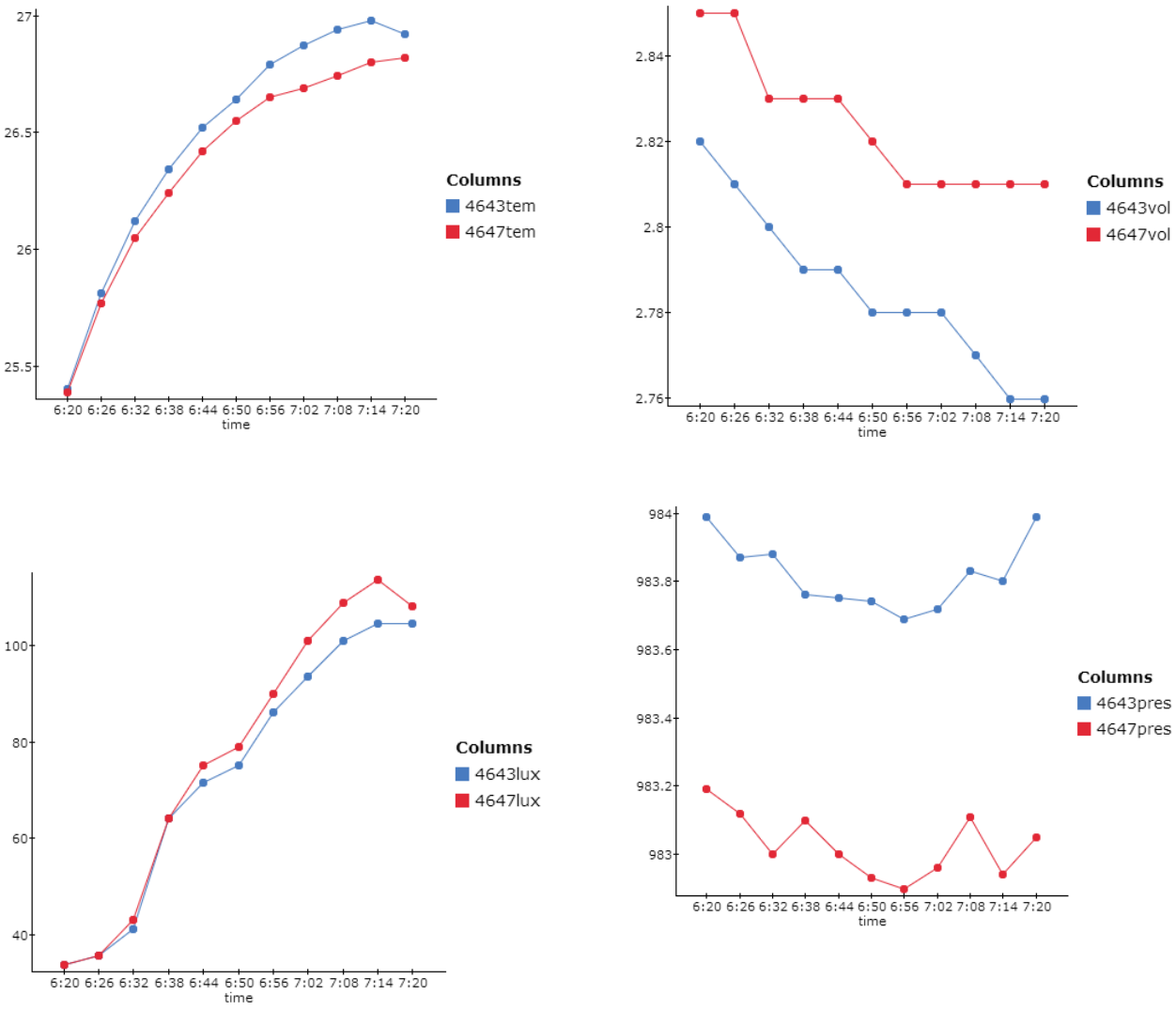

Figure 4.3 Charts of data from node 4643 & 4647 created on July 16th, 2014

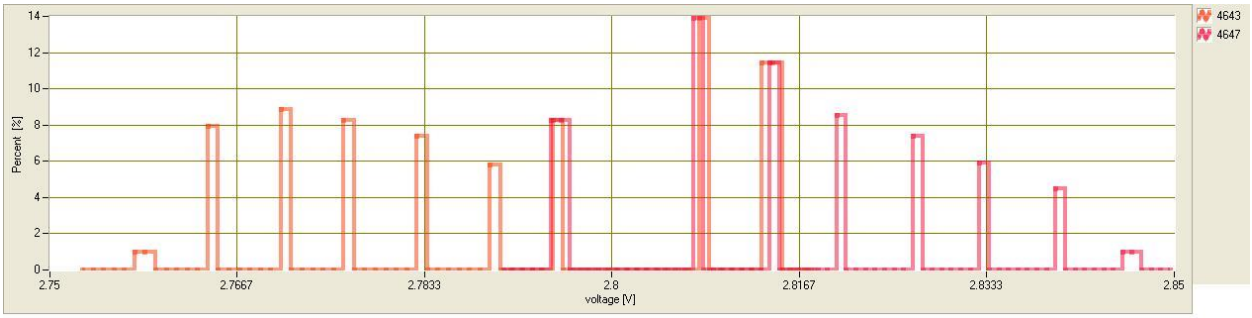

Figure 4.4 Histogram of voltage from node 4643 & 4647 created on July 16th, 2014

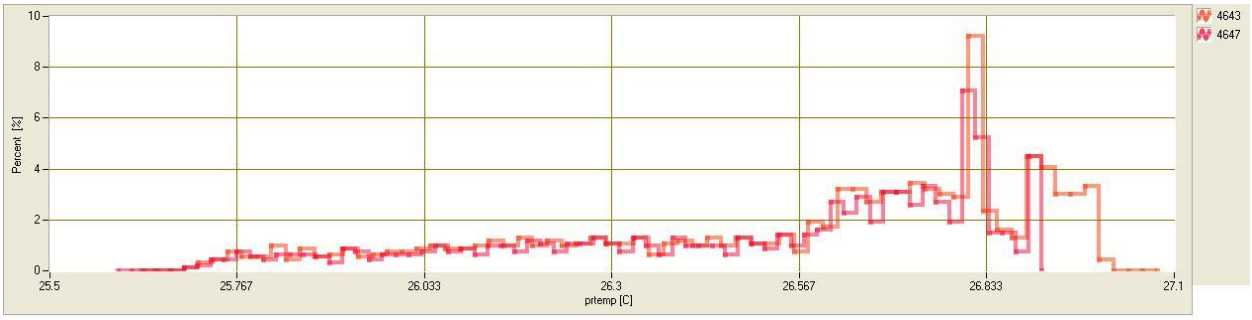

Figure 4.5 Histogram of temperature from node 4643 & 4647 created on July 16th, 2014

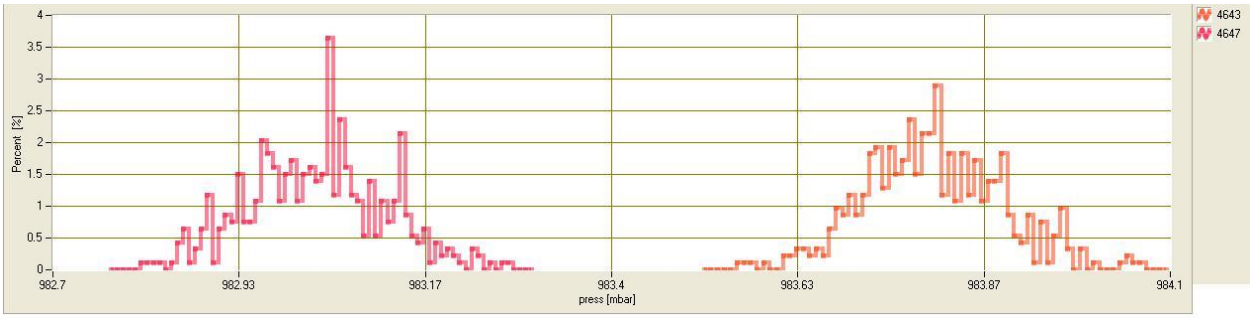

Figure 4.6 Histogram of pressure from node 4643 & 4647 created on July 16th, 2014

The line chart is the scatterplot connected by lines and the data is observed in a specific order (followed by time in this case). The histogram is to represent the data distribution. The X axis represents the range of the values. The Y axis represents the percentage that a certain value occurred. The histograms above are generated by MoteView Histogram tab. They are generated form the continuous data, which is not exactly the same as the eleven sampling data in Table 4.1. From the visual graphs above, the shape of the line chart and the distribution of the histogram of two-sensor nodes' data are almost the same with an acceptable tolerance. Some of the line charts look like that the data floating is large just because the interval of Y axis is large. For light intensity, the graphs are in a positive skew because the sunrise caused the room to brighten.

A more accurate demonstration of sensor node accuracy is shown by the statistical analysis that includes the calculations of the range, the sample mean, the sample standard deviation, and so forth. The

data processing tool used is Microsoft Excel. It provides a number of functions to generate the statistics automatically by creating the data sheet and selecting the processing data. Some formulas of calculations are provided in the previous chapter as a reference. The mean, standard deviation and the range between Min and Max of temperature, voltage and pressure are nearly the same when comparing the same statistic between two nodes. The standard deviations of temperature, voltage and pressure are low. That indicates that each value is pretty close to the mean. For light intensity, the differences of statistical results are not quite small but the overall performance of light sensors keeps the same. In conclusion, the performances of these two nodes are effective and parallel.

Column/Unit	Mean	Std. dev.	Range	Min	Max	Variance	n
4643tem/C	26.484545	0.51912164	1.58	25.4	26.98	0.26948727	11
4647tem/C	26.374545	0.46684823	1.43	25.39	26.82	0.21794727	11
4643vol/V	2.7854545	0.019164361	0.06	2.76	2.82	0.00036727273	11
4647vol/V	2.8236364	0.015666989	0.04	2.81	2.85	0.00024545455	11
4647light/lux	77.513636	29.988978	80.04	33.81	113.85	899.33879	11
4643light/lux	73.788182	27.218208	70.84	33.81	104.65	740.83084	11
4643pres/Mba	983.82	0.10305338	0.3	983.69	983.99	0.01062	11
4647pres/Mba	983.02727	0.093283536	0.29	982.9	983.19	0.0087018182	11

Table 4.2 Summary of statistics generated by Excel

NesC Language and TinyOS Resources

Investing the available resources for educational use is a crucial process before conducting this project and writing the guide. There are plenty of books, journals, guides or websites giving references of nesC programming. Some of them are segmented and not so organized; some of them are not easily

understandable because of the lack of detail explanations about terms and commands. That is the reason this project is initialized and that makes the difficulty to understand nesC codes when writing the explanations of applications. The following content provides several effective resources to learn nesC/TinyOS to make a supplement of this project.

TinyOS official website, tinyos.net, provides the instruction how to install TinyOS and programming manual.

Wikipedia provides some explanations about terms and some basic concepts of nesC.

Memsic Company provides the data sheet of motes/sensors and the download of WSN software.

A number of PowerPoint slides are available online with the explanation of nesC concepts and simple applications.

Chapter V Summary, Discussion and Recommendations

Summary of Guide Development

This project to develop a nesC language application guide is firstly proposed and planned with the instruction of Dr. David Border. Before the writing of this study guide, the first stage to investigate the current development and potentials related to the project topic is necessary. The development of WSN history, the current research trends, applications, available hardware/software platforms and available educational resources are investigated through all kinds of resources. The hardware/software platforms have to be verified to be accessible to students in ECET laboratory, College of Technology, Architecture & Applied Engineering, BGSU. Then the guide outline is firstly established with 6 chapters.

Chapter 1, Introduction to Computing explains the memories used in the Iris mote, which is applied when learning the WSN hardware platform and OTAP. It also explains C programming basics. This content is not available in most WSN guides but necessary to refresh the users' memory to get prepared before learning WSN

Chapter 2 introduces some background knowledge of WSN developments. The related knowledge can be found in many online articles about WSN introduction. So this chapter is briefly written and the readers can obtain more by searching on Google.

Chapter 3 covers WSN hardware platform and get readers familiar with the Iris mote and other boards. The hardware features are summarized in this chapter. The detailed specifications are available in Memsic website and they are attached in the appendix.

Chapter 4 covers WSN software platform and get readers familiar with TinyOS/nesC. Some of the concepts are firstly introduced in the guide like module, configuration, interface, which is applied in chapter 5 for programming. Chapter also briefly introduces MoteWorks Starter Kit without the detailed explanations how to install and use the software packages, such as MoteView, MoteConfig, Cygwin, etc. Since MoteWorks Getting Started Guide provided by Crossbow explains how to install MoteWorks and

software's functions. In addition, Mr. Omar's project (Aridi, 2010) did a great work about how to set parameters, utilize user interfaces, or run MoteView, MoteConfig, Programmer Notepad and Cygwin. It also covers the WSN protocols and standards. Therefore, the readers can refer to the above two materials to better familiar with WSN software environment. For Xmesh networking, the Crossbow Xmesh Manual gives detailed explanations.

Chapter 5, as the main part of this guide, covers nesC programming fundamental rules. NesC 1.3 Language Reference Manual (Gay, Levis, Culler & Brewer, 2014) is a comprehensive manual to introduce all the nesC unique grammars and some grammars shared with C language. It describes many terms and conceptions not easily understood because the readers may not have a good understanding in C language. That makes difficulties to learn nesC programming for readers. This chapter summarizes the basic programming rules and avoids using abstruse descriptions to explain the terms or conceptions which may be confusing. Combined with example programs, this chapter covers programming codes with detailed notations. Following the notations, the readers can understand how the programming rules are implemented into the real programs. It introduces some library modules which are not easily understood and develops the diagrams of the relationship between components to help readers clearly understand the interactions of different components and interfaces.

Chapter 6 covers the introduction of LabVIEW and example programs to display sensor data. The default program provided with the Crossbow Xmesh Driver is not exactly suitable for MTS400CC sensor boards. This chapter provides the revision of the original programs and two additional programs found in the National Instruments forum. All the programs share the same logic from data acquisition and processing to data display. This chapter gives the detailed procedures of data flow in the block diagram for the readers' future improvement and development of WSN sensor data monitoring system.

During the whole period of this guide development, keeping reviewing and revising the drafts are important to make the guide better. The following table shows the important WSN concepts and features included in the guide or not.

WSN Guide Potential Content		Included in Guide or not/Location	Reason for Existence or Absence
Introduction of Computing		Yes/Chapter I	Preparation before learning programming
WSN History, applications, trends and issues		Yes/Chapter II	Necessary knowledge to familiar with WSN
WSN hardware platform	Iris, MTS400CC, MIB20	Yes/Chapter III	Preparation before learning programming
	Other hardware	No	Not available for use
WSN software platform	Operating System	Yes/Chapter IV	Preparation before learning programming
	Xmesh Networking	No	Introduced in detail in Crossbow Xmesh User Manual
	Software package: Cygwin, PN, MoteView, MoteConfig,etc.	No	Introduced in detail in Mr. Omar's guide
	Protocols, standards	No	Introduced in detail in Mr. Omar's guide
NesC programming	Programming rules	Yes/Chapter V	Essential content to learn programming
	Example applications	Yes/Chapter V	Essential content to learn programming
	Codes compiling, installation	Yes/Chapter V	Introduced briefly
Crossbow Xmesh Driver	Downloading, installation	No	Introduced in detail in Mr. Omar's guide
	Example applications	Yes/Chapter VI	Detailed programs to display data
Lab activities development with procedures		No	Introduced in detail in Mr. Omar's guide
Hardware specifications		Yes/Appendix	Introduced to be a reference

Table 5.1 Summary of WSN guide potential content

Guide Potential Limitations and Recommendations

Considering the length of the guide, it cannot fully contain the programming rules of nesC language. There are still a number of commands which are be useful in other nesC applications. The users can find other educational resources as reference. Also, because of the limitations of hardware/software platforms available in ECET lab, nesC programming cannot be applied to non-Memsic wireless nodes. Other trials of nesC programming in other devices are helpful to practice programming skills. The Crossbow WSN Starter kit is just compatible with Windows XP. However, Windows XP is no longer supported from Microsoft. Considering the documents and system safety, seeking other WSN software platforms which are compatible with Window 7 or 8 will be recommended. A previous guide written by Mr. Omar El Aridi can be considered as a supplement of this guide. Unlike this guide, which is

more suitable for junior/senior students who have programming background knowledge, Mr. Aridi's work concentrates on conducting non programming lab activities.

References

1. Aridi, E. O. (2010). *Developing and Designing Undergraduate Laboratory Wireless Sensor Network Exercises.* Bowling Green, Ohio: Bowling Green State University.

2. Bokareva, T. (2014). *Mini Hardware Survey*. Retrieved 2014, from http://www.cse.unsw.edu.au/~sensar/hardware/hardware_survey.html

3. Border, D. (2012). Developing and Designing Undergraduate Laboratory. *Proceedings of American Society for Engineering Education Annual Conference.*

4. Buratti, C., Conti, A., & Verdone, R. (2009, 8 31). An Overview on Wireless Sensor Networks Technology. *Open Access Sensors*, pp. 6870-6872.

5. Crossbow. (2007). In *MoteWorks Getting Started Guide* (p. 32).

6. Desai, U. B., Jain, B. N., & Merchant, S. N. (2007). *Wireless Sensor Networks: Technology Roadmap.* Retrieved 2014, from https://www.iith.ac.in/~ubdesai/WSN_Roadmap_Final_%20Report.pdf

7. Fok, C.-L. (2004, 9). *TinyOS Tutorial.* Retrieved 2014, from http://www.cs.bme.hu/~sl/mitmot/files/tos_tutorial.pdf

8. Gay, D., Levis, P., Culler, D., & Brewer, E. (2014). *nesC 1.3 Language Reference Manual.* Retrieved 2014, from https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=0CDYQFjAC&url=http%3A%2F%2Fwww.tinyos.net%2Fdist-2.0.0%2Ftinyos-2.0.0beta1%2Fdoc%2Fnesc%2Fref.pdf&ei=EyXLU-rfEdGNyASWxILoBw&usg=AFQjCNEIvL41UX0Joag146qbLHewLCgAOg&sig2=QxJR

9. Greene, C., & Khamphavong, B. (2007). *MoteWorks Getting Started Guide.* Retrieved 2014, from http://www.radford.edu/nsrl/creu1011/PowerPoints/MoteWorks.pdf

10. Hatler, M. (2013). *Industrial Wireless Sensor Networks:Trends and developments*. Retrieved 11 14, 2013, from International Society of Automation: www.isa.org/InTechTemplate.cfm?template=/ContentManagement/ContentDisplay.cfm&ContentID=90824

11. He, J. (2012, 3). *Wireless Sensor Network Programming Using TinyOS.* Retrieved 2014, from http://web.cse.ohio-state.edu/~heji/TinyOSTutorial_Mar2013.pdf

12. Lahtinen, E., Ala-Mutka, K., & Järvinen, H. (2005). A study of the difficulties of novice programmers. *ITiCSE '05 Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pp. 14-18.

13. Lane, D. (2014). *Introduction to Statistics.* Retrieved 1 10, 2014, from Online Statistics Education: An Interactive Multimedia Course of Study: http://onlinestatbook.com/2/index.html

14. Levis, P. (2006). *TinyOS Programming.* Retrieved 2014, from http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf

15. Nack, F. (2010). An Overview on Wireless Sensor Networks. Institute of Computer Science (ICS), Freie Universität Berlin.

16. National Instruments. (2013, 10 30). *Developing LabVIEW Plug and Play Instrument Drivers*. Retrieved 12 22, 2014, from National Instruments: http://www.ni.com/white-paper/3271/en/

17. Schonwalder, J. (2007, 4). Wireless Sensor Networks: Motes,NesC and TinyOS.

18. Wang, W. (n.d.). Wireless Sensor Networks Research and Application. Shanxi, China.

19. WSN Research Group. (2014). *Sensor Network Hardware Systems*. Retrieved 1 22, 2014, from The Sensor Network Museum: http://www.snm.ethz.ch/

Appendix (A)

Guide Name.	Author, date, website, etc.	Guide Content
WSN Programming	Olaf Landsiedel	TinyOS component, “Blink”, other examples.
TinyOS Tutorial	Chien-Liang Fok ,Fall 2004	WSN hardware, TinyOS installation and configuration, nesC syntax, network communication, data acquisition, debugging
WSN Research and Application	Wenyong Wang	Current WSN research trends and hot pots, applications
WSN Programming Using TinyOS	Jin He, Mar 2012	WSN/TinyOS architecture, installation, example codes
WSN Technology, Protocols and Applications	Kazen Sohraby,etc., 2007	WSN history/trends, application, protocols, network management, OS
WSN Technologies for the Information Explosion Era	Takahiro Hara, etc., 2010	Scheduling, Data management, Networked Sensing Sys, Implementation, development support
WSN Designs	Anna Hac, 2003	Routing in WSN, Smart dust, Clustering techniques, protocol,

		applications
WSN: Motes, NesC, and TinyOS	Jurgen Schonwalder, April2007	Application, hardware, constraints/challenges, nesC/TinyOS, Internet/Protocols
Cygwin User's Guide	http://cygwin.com/cygwin-ug-net/cygwin-ug-net.html	Setting up, how to use Cygwin, programming
WSN	Sebastian Buttrich, Nov. 2011	History, applications, hardware, OS, challenges.
Crossbow: MoteWorks Getting Started Guide	Catherine Greene, etc.	Installation of MoteWorks, TinyOS/nesC, sensing application and Xmesh
TinyOS Programming	Philip Levis, Oct. 2006	Interfaces/Modules, Configurations/Wiring, design

Appendix (B)

Blink.nc

```
configuration Blink {

}

implementation {

components Main, BlinkM, SingleTimer, LedsC;

Main.StdControl -> SingleTimer.StdControl;

Main.StdControl -> BlinkM.StdControl;

BlinkM.Timer -> SingleTimer.Timer;

BlinkM.Leds -> LedsC;

}
```

BlinkM.nc

```
/**

* Implementation for Blink application. Toggle the red LED when a

* Timer fires.

**/

module BlinkM {

provides {

interface StdControl;

}

uses {

interface Timer;

interface Leds;

}

}
```

```
implementation {
/**
* Initialize the component.
*
* @return Always returns <code>SUCCESS</code>
**/
command result_t StdControl.init() {
call Leds.init();
return SUCCESS;
}
/**
* Start things up. This just sets the rate for the clock component.
*
* @return Always returns <code>SUCCESS</code>
**/
command result_t StdControl.start() {
// Start a repeating timer that fires every 1000ms
return call Timer.start(TIMER_REPEAT, 1000);
}
/**
* Halt execution of the application.
* This just disables the clock component.
*
* @return Always returns <code>SUCCESS</code>
**/
```

```
command result_t StdControl.stop() {
return call Timer.stop();
}
/**
* Toggle the red LED in response to the <code>Timer.fired</code> event.
*
* @return Always returns <code>SUCCESS</code>
**/
event result_t Timer.fired()
{call Leds.redToggle();
return SUCCESS;
}
}
```

SingleTimer.nc

```
Configuration SingleTimer {
provides interface Timer;
provides interface stdControl;
}
implementation {
components TimerC;
Timer = TimerC.Timer [unique("Timer")];
stdControl = TimerC;
}
```

Appendix (C)

Wireless Sensor Network Programming Language Application Guide

Chapter I Introduction to Computing Basics

This chapter covers the introduction of computer memory, such as RAM, ROM, EEPROM, flash memory. It also provides some C programming language basics. This chapter is designed for the students who have not taken courses about computing or who need to refresh their memory.

Section 1.1 Semiconductor Memories

In computing, the memory capacity is the number of bits that be addressed and the semiconductor memory chip can store. A group of 8 bits is called byte. For higher storage units, 1 Kilobyte (KB) = 1024 Bytes, 1 Megabyte (MB) = 1024KB, 1 Gigabyte (GB) = 1024MB, etc.

Section 1.1.1 ROM

ROM (Read-only memory) is the memory which does not lose any data when the power is off. The data stored in ROM is stable but cannot be modified as quickly as the data in RAM (Radom access memory). ROM has different types such as PROM, EPROM, EEPROM, flash memory, etc.

EPROM (Erasable programmable read-only memory) uses high voltage to write data into memory and erase data many times. An older, now less common type of EPROM is Ultraviolet ERPOM (UV-EPROM), which erases content through exposure under ultraviolet rays. However, the process to erase data in UV-EPROM usually takes up to 20 minutes. EEPROM (Electrically erasable programmable read-only memory) is a successor technology to EPROM. EEPROM erase times are typically in microseconds instead of minutes for EPROMs. The XM2110CA multi-chip module (MCM) has a 4K byte EEPROM (See Chapter 3).

Flash memory is also a type of memory that can store data when power is off. The main difference between EEPROM and flash memory is that flash memory erases data by block and EEPROM erases data by bytes. The XM2110CA MCM has a 128K bytes program flash memory and a 512K bytes serial flash memory (See Chapter 3).

Section 1.1.2 RAM

RAM (Random access memory) is the internal memory that exchanges data with CPU. It is usually used as temporary data storage media for operating systems or other running applications. However, RAM cannot keep data when power is off, which is different from ROM. (Mazidi & Causey, 2009). The XM100CA MCM has an 8K bytes RAM (See Chapter 3).

Section 1.2 C Programming Basics

This section briefly explains some basics of C language programming, which shares the same grammar with nesC language used in programming on WSNs. The concepts discussed this section will be applied to the example applications in Chapter 5.

In the C programming language, there are four types of data as shown in the table 1.3.1. Data types are used to determine the storage space of a variable and how a variable is interpreted.

Types	Explanations	Example
Basic	Integer type or floating-point type (Applied in Chapter 5)	int a, float b

Enumerated	Define a data to be a set of predefined constants	enum cardsuit {
		CLUBS = 1,
		DIAMONDS = 2,
		HEARTS = 4,
		SPADES = 8
		};

Void	No value available	void f (void) ;
Derived	Pointer, array, structure, union and function types	*ptr = 8;

Table 1.3.1 C Programming Language Data Type

In the C programming language, there are six types of operators as shown in the table 1.3.2. Operators are used to perform mathematical or logical operations. The detailed examples of operations can be found at http://www.tutorialspoint.com/cprogramming/c_operators.htm (Tutorialspoint, 2014).

Types	Explanations	
Arithmetic	+, -, *, /,%	Add, subtract, multiply, divide, remainder
	++, --	Increase, decrease values by one
Relational	==, !=	Check if two values are equal or not equal
	>, <	Check which value is greater or less
	>=, <=	Check which value is greater or equal, less or equal
Logical	&&	Logical AND operator
	||	Logical OR operator
	!	Logical NOT operator
Bitwise	&,|, ^	Binary AND, OR, XOR operators
	~	Binary Ones complement operator
	<<, >>	Binary left, right shift operators
Assignment	=	Assignment operator
	+=, -=, *=, /=, %=	Add, subtract, multiply, divide, modulus AND assignment operators
	<<=, >>=	Left, right shift assignment operators
	&=, ^=, |=	Bitwise AND, exclusive OR, inclusive OR assignment operators
Misc	Sizeof()	Return the size of an variable

	&	Returns the address of an variable
	*	Pointer to a variable
	?:	If Condition is true? Then value X : Otherwise value Y

Table 1.3.2 C Programming Language Operator Type

In the C programming language, there are three basic types of program design structure as shown in the Table 1.3.3.

The statement of if…else will be used in Chapter 5. Its syntax is:

```
if(boolean_expression)
{
   /* statement(s) will execute if the boolean expression is true */
}
else
{
  /* statement(s) will execute if the boolean expression is false */
}
```

Type	Explanations
Sequence structure	Program in sequential order
Decision making structure	If, if…else, nested if, switch, nested switch
Loop structure	While, for, do…while, nested loops

Table 1.3.3 C Program Structure

Chapter II Wireless Sensor Network Technology History and Features

Section 2.1 History of Development of WSN Technology

The history of development of WSN technology can be divided into four stages by time (Nack, 2010).

1. Cold - War Era: Sensor: Sensor Networks for military use were applied. For instance, the Sound Surveillance System (SOSUS) was developed to track Soviet submarines through placing acoustic sensors underwater at key locations. The air defense radar network was developed to defense the territorial air space of United States.
2. Early 1980s: Distributed Sensor Network program was started to monitor environmental variables through a set of sensors in different locations by Defense Advanced Research Project Agency. DSN development was still limited by the size and cost of sensors at this period.
3. Late 1980s: The contributions and achievements of DARPA drew the attention of US military. The large replenishment of a fund gave the scientists more opportunities to develop sensor network technology. That encouraged a great progress of sensor network technology.
4. Present research: Great development of computing, Micro-Electro-Mechanical System and other technologies led to the sensors in smaller size and cheaper price. WSN technology is applied into commercial and educational uses. More companies which produce wireless sensors and software support are built. The standardization of WSN protocols is becoming mature.

Section 2.2 Applications of WSN Technology

WSN technology can be applied into many fields. It has the features of rapid deployment, self-organization, strong concealment and high fault tolerance. Therefore, it is very suitable for the applications in the military. Agriculture environment automatic monitoring system can also be built by using WSN. Use of a shared network allows the data acquisition and environmental control of wind, light,

electricity, heat and chemicals, which can effectively improve the degree of intensive agricultural production, simplify the system complexity and reduce the equipment costs.

In addition, WSN can play an important role in the detection of human physiological data, the elderly health, hospital drugs' management and remote medical treatment. Use of appropriate sensors, such as piezoelectric sensor, acceleration sensor, ultrasonic sensor, humidity sensor, etc. can effectively allow design of a multi-dimensional protection network. The system can be used for monitoring a bridge, viaduct, and highway and road environments. With the help of the spacecraft distributed sensor nodes, achieving wide range and long-term close monitoring and exploration on the surface of a planet is feasible (Ren & Yang, 2010).

Section 2.3 Research Trends and Issues of WSN Technology

Section 2.3.1 WSN Research Trends

With the researchers' indefatigable work in different fields over years, WSN technology applications have found use in the military, fine agriculture, security monitoring, environmental monitoring, construction, medical care, industrial control, intelligent transportation, logistics management and intelligent household. The research trends of WSN technology are stated below.

1. Simulation platform: There are many existing WSN simulation platforms. However, they still have some limitations. Therefore, standard simulation technology and tools is one of the hot research topics.
2. Development of sensor nodes: Different application fields need to apply different types of sensor nodes. The research and development of new, low cost and low power consumption sensor nodes are still an importance in the development of WSN technology.
3. Node localization algorithm and evaluation model: Further localization algorithm research will mainly focus on how to use the local information provided by a few nodes and other nodes'

communication constraints to estimate the unknown nodes, especially mobile nodes, under a low cost and high precision.

4. Cross-layer design: The goal of cross-layer design is to achieve designed interaction and balanced performance between non-adjacent protocol layers. That will optimize energy management and low power consumption design of WSN.
5. Network fusion research: WSN with the functions of data acquisition, preprocessing and transmission need to be integrated with the existing internet, mobile communication network to transmit information and innovate applications by using sensing information.
6. Mature industry application: Seamless connection between WSN and existing systems is an important foundation for sustainable development of WSN technology. It is also the key to the WSN further industrialization and marketization.

Section 2.3.2 WSN Issues

Wireless Sensor Networks, however, still face some practical problems (Baidu, 2014).

1. The problem of network communication: In the WSN communications, the signal may be affected by some obstacles or other electronic signal interferences. How to make safe and effective communications is a problem to be studied in the future.
2. Cost problem: Wireless sensor networks need to use a large number of micro sensors, so cost will restrict its development.
3. System energy supply problem: The current main solutions are to use high-energy batteries and reduce power consumption. In addition, there are sensor networks self-energy collection technology and wireless battery charging technology.
4. Efficient wireless sensor network structure: There are many forms and ways to structure wireless sensors. A reasonable wireless sensor network can maximize the use of resources. This includes the problems of network security protocols and large-scale sensor network nodes mobility management.

In short, the wireless sensor network application prospect is very attractive. Wireless sensor network (WSN) is considered to be one of the important technologies affecting human future life; this emerging technology provides people with a new access to information and a way of processing information.

Chapter III Wireless Sensor Network Hardware Features: Memsic

A wireless sensor network is composed of a large number of low-cost micro sensor nodes deployed in the monitoring areas. It forms a multiple hop and self-organizing network through the wireless communication mode. A sensor node (a mote) is the fundamental unit of WSN. It is used to collect information from sensors, process commands and communicate with other nodes. A mote usually has five components, including a controller, transceiver, power source, memory and sensors. There are currently numerous available sensor nodes for educational, research and commercial uses, such as BTnode, COOKIES, EPIC mote, Telos, etc. This guide focuses on the IRIS mote available in the ECET lab.

Section 3.1 IRIS Mote

The IRIS has a MCM XM2110CA and it consists of several components as shown in the table below.

Integrated Circuit Type	Model Number	Functions
Microcontroller Unit	ATmega1281	A low power microcontroller unit (MCU) designed for embedded applications
RF Transceiver	AT86RF230	A Radio Frequency (RF) module for communication over the Wireless Personal Area Network (WPAN)
External Serial Flash Memory (512K)	AT45DB041D	Store code images through Over The Air Programming (OTAP) to serial flash

Table 3.1.1 XM2110CA Integrate Circuits

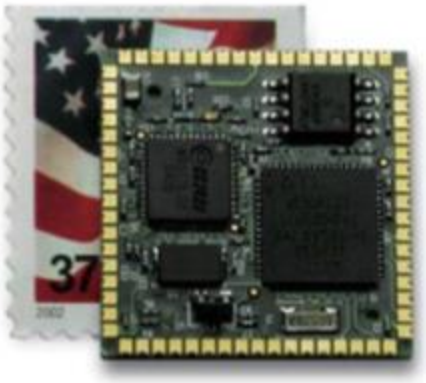

Figure 3.1.1 XM2110CA Module (Crossbow, 2007)

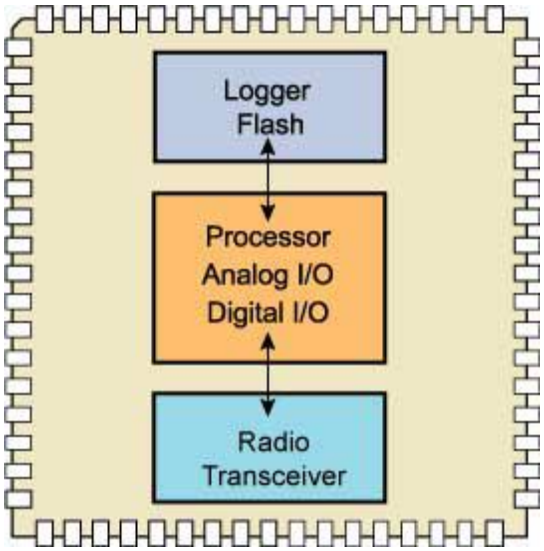

Figure 3.1.2 XM2110CA Block Diagram (Crossbow, 2007)

The IRIS also has a 51-pin expansion connector (shown in figure 3.1.4) which supports interfaces. This provides the ability to connect the IRIS motes with a large number of external devices. Overall, the IRIS mote has the following features:

1. It has an optimized processor and an improved radio range (as far as 500 meters) compared with previous MICA motes.

2. Its radio operates on the 2.4GHz globally compatible ISM (Industrial, Scientific and Medical) band.
3. It is supported by MoteWorks software platform which allows WSN management and development.
4. It has a variety of software interfaces which support standard or custom sensing devices.
5. It supports Memsic sensor boards, data acquisition boards, gateway, etc.

Figure 3.1.3 An IRIS Mote (Crossbow, 2007)

Figure 3.1.4 A 51-pin Expansion Connnector (Crossbow, 2007)

Section 3.1.1 ATmega1281 Microprocessor

3.1.1(a) Memory

The ATmega1281 microprocessor has an 8 bit AVR CPU with a 16MHz maximum operating frequency. It has several types of memory. The following table shows the usage of different memories. The basic concepts of memory were introduced in Chapter 1.

Memory Type	Capacity	Uses
Program Flash Memory	128K Bytes	Store application codes through serial ports or Over The Air Programming (OTAP).
EEPROM	4K Bytes	Store persistent values such as mote ID, group ID, radio channel and other mote configuration data
RAM	8K Bytes	Store user application parameters, Xmesh and TinyOS variables, and the stack

Table 3.1.2 ATmega1281 Memory

3.1.1(b) USARTs

The ATmega1281 has two USARTs (Universal Synchronous/Asynchronous Receiver/Transmitter) devices. They perform as UARTs (Universal Asynchronous Receiver/Transmitter) in the Memsic MCM. UART0 is used to communicate with the client device when the IRIS mote functions as a base station. It is the default I/O serial communication port. UART1 can be used to communicate with another serial device for users, for example, external serial flash memory.

3.1.1(c) ADC

The ATmega1281 has a 10 bit analog to digital converter. It has eight channels and uses the battery voltage as a full scale reference. For instance, if the battery voltage is 3.2V, the ADC will measure a full scale voltage; if the battery voltage is 2.5V, the ADC will measure the corresponding full scale voltage.

3.1.1(d) Other Interfaces

The ATmega1281 has additional interfaces. The I²C (Inter-Integrated Circuit) provides a synchronous interface for lower speed serial devices. The SPI Bus (Serial Peripheral Interface Bus) provides a synchronous interface, for operating at speeds greater than the I²C standard (Wikipedia, 2014).

Section 3.1.2 External Serial Flash Memory

The Iris mote has a 512K bytes external flash memory which is interfaced to the MCU. It is connected to one of the USART (Universal Synchronous/Asynchronous Receiver/Transmitter) devices on the ATmega1281. It is used to store data, measurements and other user-defined information as a pseudo disk drive because of the memory capacity and other limitations of the internal flash memory. It is also used for OTAP described below.

The 128K bytes program flash memory is logically divided into two sections (Figure 3.1.5). The application section is used to store the application codes. The boot loader provides a Read-While-Write Self-Programming mechanism to download and upload program codes for the application software updates without the need of a second microcontroller that was found in previous motes. The code in boot loader section can process read-write operations to the entire flash including the boot loader section. Thus, the boot loader can modify or erase itself.

The external serial flash memory has 4 logical slots. When programming over the air, an incoming code image is broken into page fragments by OTAP protocols to reduce the traffic required for the download

process. A page is the amount of the image that can be transmitted in a single TOS packet. The selected mote receives the code page and stores them into the serial flash memory (Memsic, 2010). The boot loader provides the ability to load programs (a copy of the bootable image in slot 1, 2 or 3) and reprogram the microprocessor from serial flash memory into the program flash.

The slot to store bootable image can be assigned (shown in Figure 3.1.6) by MoteConfig (a graphical user interface for mote programming and OTAP). The use of MoteConfig can be referred to "MoteConfig User's Manual" by Crossbow. The boot loader codes are loaded when over-the-air-programming the mote initially. Once the programming is completed, the boot loader can be erased via the UISP command.

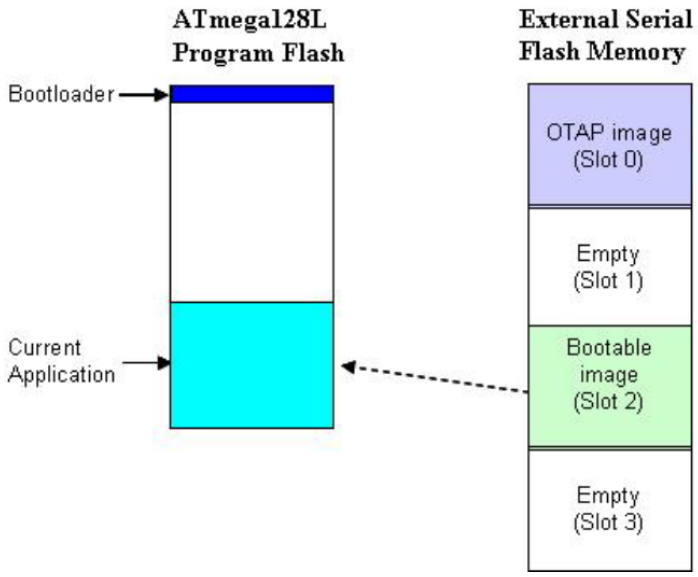

Figure 3.1.5 OTAP Image Transfer (Memsic, 2010)

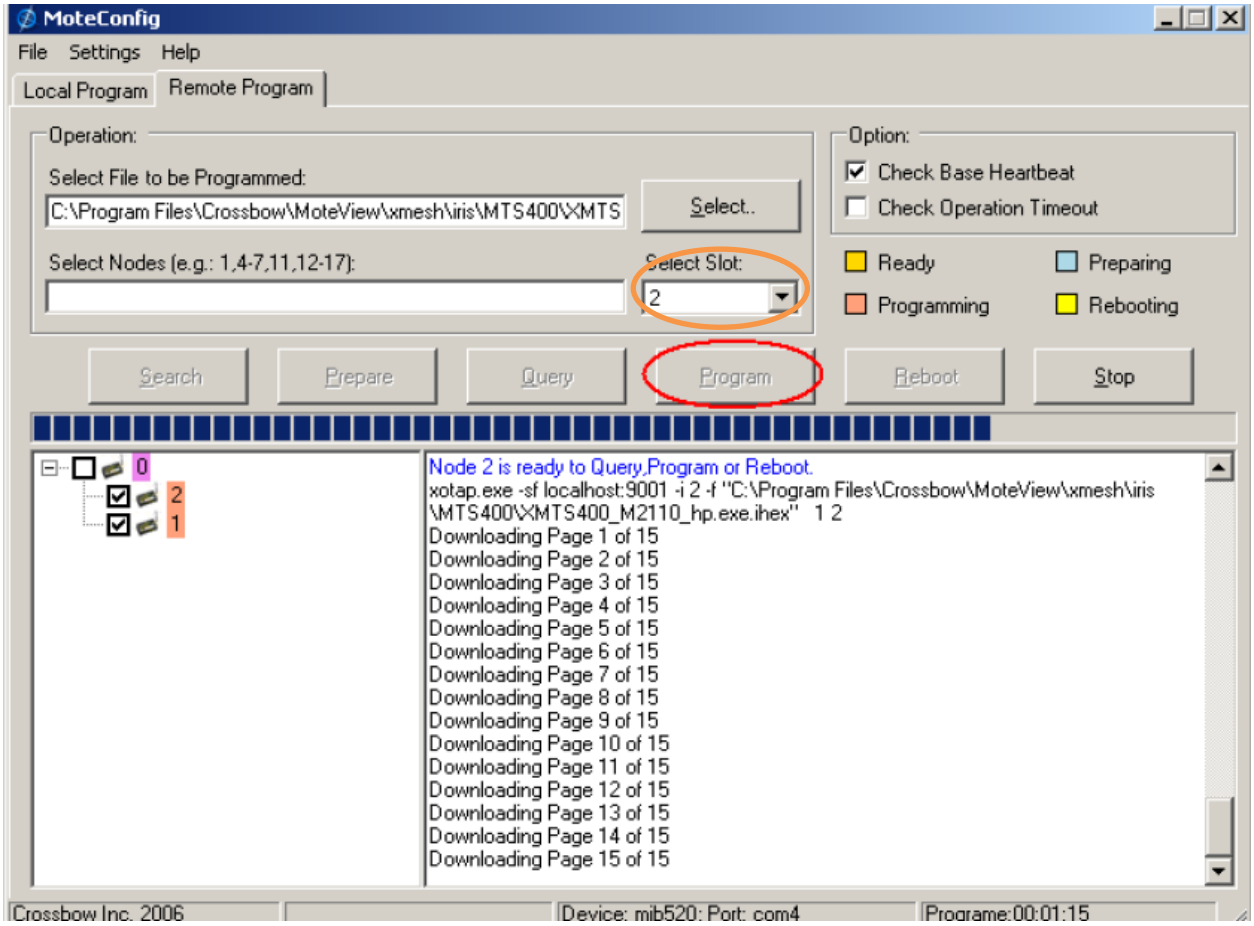

Figure 3.1.6 OTAP with the selection of the slot in MoteConfig

Section 3.1.3 RF Transceiver

The Iris mote has a 2.4GHz frequency band, IEEE 802.15.4 compliant RF transceiver. IEEE 802.15.4 is the standard to specify the physical layer and media access control for low data rate wireless personal area networks (Wikipedia, 2014). The RF transceiver frequency range is 2405MHz to 2480MHz. The maximum transmit data rate is 250Kbps with a RF power of 3dBm. The RF transceiver has more than a 50 meters indoor range and more than a 200 meters outdoor range.

The radio messages can be transmitted between the base station and nodes through the Xmesh network. The Xmesh network also allows different nodes to communicate with each other. The Xmesh supports OTAP operation. It also provides the network services that enable network self-organizing and self-

healing. The Xmesh is a multi-hop, ad-hoc, mesh networking protocol developed for wireless networks by Memsic and it features as an improved radio coverage and an improved reliability when compared to other mote network protocols. The Xmesh supports a network that consists of a PC, a base station and motes. See Figure 3.1.6 below (Memsic, 2010).

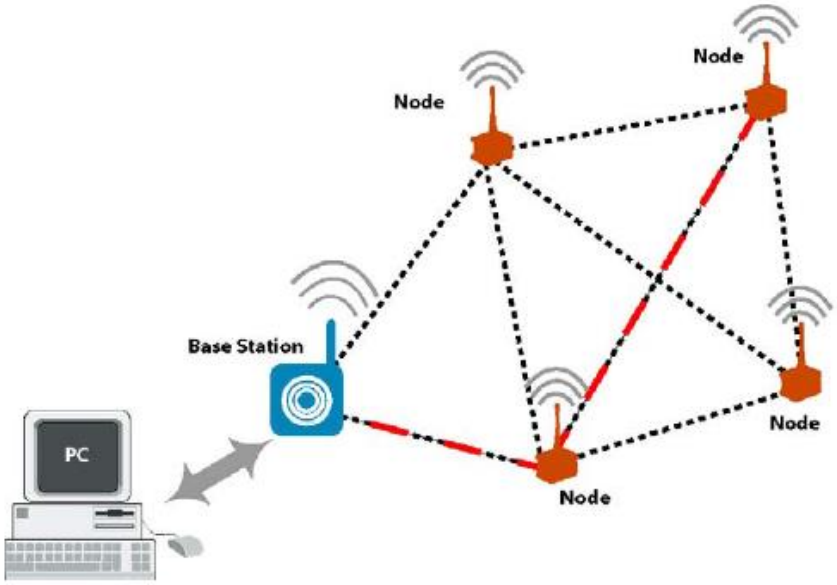

Figure 3.1.7 An Xmesh Network Structure (Memsic, 2010)

The PC receives data and sends commands into the Xmesh network. The base station sends messages to the PC through serial communications and communicates with other nodes over the radio. The nodes communicate with each other over the radio.

Section 3.1.4 Power Supply

The Iris mote uses two AA batteries with the external power of 2.7V to 3.3V. The power is connect through the 2-pin Molex connector. Each ATmega1281 operation consumes current. Typical values are shown in table 3.1.2 (Memsic, 2010). However, the processor and radio can invoke sleep mode and the current can be reduced to micro-amps instead of milli-amps to extend battery life.

Operation	Operational currents (mA)
Microprocessor, full operation	6
Microprocessor, sleep	0.010
Radio, receive	16
Radio, transmit	17
Radio, sleep	0.001
Serial flash memory, write	15
Serial flash memory, read	4
Serial flash memory, sleep	0.002

Table 3.1.3 IRIS Mote Operational Currents

Section 3.2 MIB520 USB Interface Board

MIB520 USB interface board has a baud rate of 57.6k. It has a 51-pin connector, red, green and yellow indicators. In the MIB520, the USB interface board is mated to an IRIS mote using the 51 pin connector. The pair of boards comprises a mote network "base station" (Memsic, 2014). When the base station connects to the PC computer, it has two virtual ports: com(x) and com(x+1) that use the single physical USB connection. The low virtual port number provides the programming channel from the PC to the MIB520. The high virtual port number provides the data communication channel from the MIB520 to the PC.

.

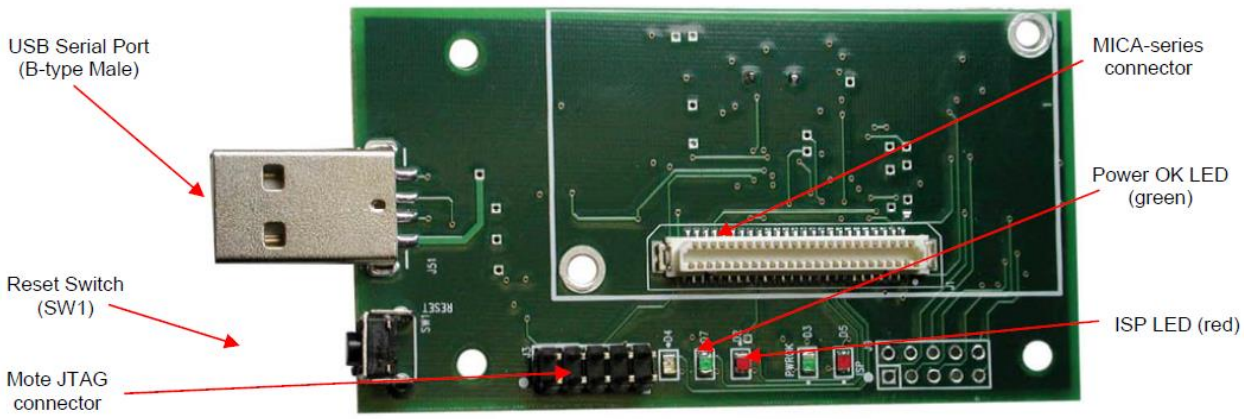

Figure 3.2.1 MIB 520CB USB Interface Board (Crossbow, 2007)

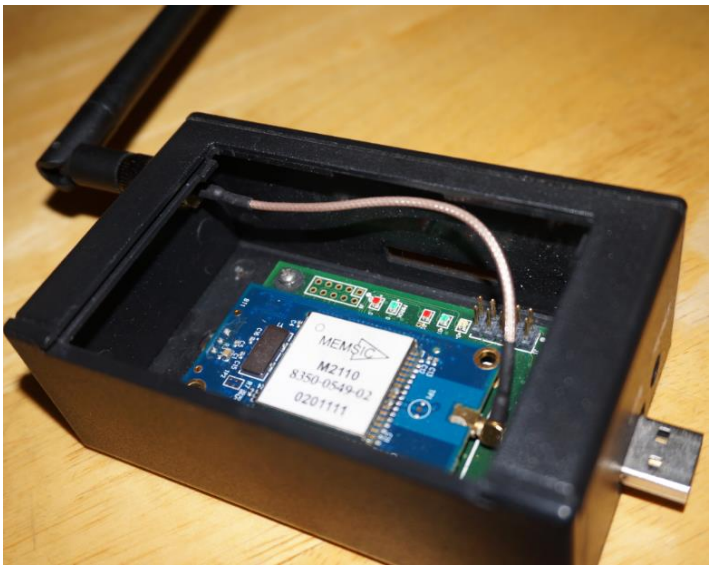

Figure 3.2.2 An IRIS Mote Functions as a Base Station

Section 3.3 MTS400CC Sensor Board

Memsic provides a variety of sensor and data acquisition boards for the IRIS mote. These boards are each compatible to the 51-pin connector and therefore can directly connect to the IRIS board. The MTS 400CC

sensor boards have the ability to monitor environmental conditions through the sensors shown in table 3.3.1.

Sensor Type	Specifications
Ambient light sensor	Range: 400-1000nm
Temperature sensor	Range: -40 to +80 Celsius, temperature accuracy: 2 Celsius
Relative Humidity sensor	Range: 0-100%RH, accuracy: 3.5%RH
Barometric pressure sensor	Range: 300-1000 mbar, accuracy: 1.5% @25 Celsius
Dual-axis accelerometer	Range: -2g to +2g

Table 3.3.1 MTS400CC sensors specifications

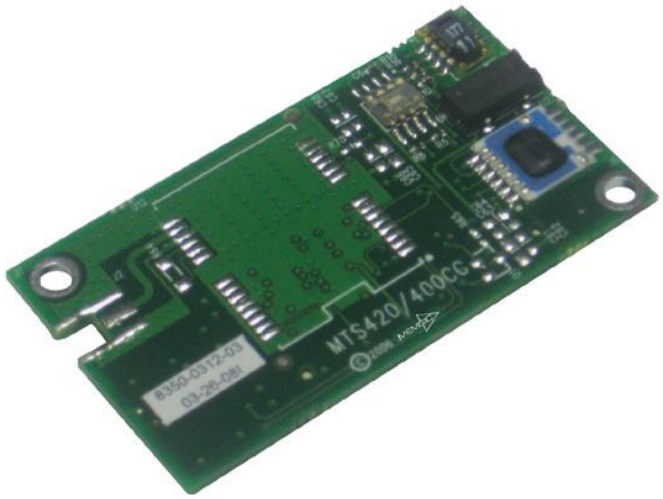

Figure 3.3.1 MTS400CC Sensor Board (Crossbow, 2007)

Figure 3.3.2 A Wireless Node

Detail specifications of IRIS mote, MTS400/420 sensor boards and MIB520 USB interface boards are provided in the appendix.

Chapter IV Wireless Sensor Network Software Features: TinyOS & Moteworks

Section 4.1 Introduction to TinyOS

The operating system of a wireless sensor network manages WSN hardware resources by providing a collection of software applications. Chiefly, this involves creating and managing network traffic, and data acquisition. The operating system of WSN is not as complex as Windows, Android, Mac OS and other general operating systems because of the typical requirements and constraints of WSN hardware. There are many WSN OSs including Contiki, LiteOS, and Nano-RK. TinyOS is one of the WSN OSs and was developed by UC Berkeley and initially released in 2000. It is an open-source operating system, which can communicate and take commands from a Linux-based device. It also supports Windows 2000/XP through the software application Cygwin. TinyOS provides a set of services such as power management, security, messaging, etc. It also supports a variety of hardware devices including the IRIS mote.

TinyOS has a component-based architecture, which allows for rapid innovation and code size minimization, both of which are helpful when working with small computing devices that must service a variety of sensor applications. The TinyOS architecture can be divided into four parts; these are network protocols, distributed services, sensor drivers and data acquisition tools. Its libraries and applications are written in nesC (Network Embedded System C), which is one type of component-based, event-driven programming languages (See Chapter 5).

Figure 4.4.1 shows TinyOS components communicate through commands and events. There are three types of components in TinyOS, including high level components, synthetic hardware and hardware components. The high level components are used to control other components, routing, data calculation and aggregation. Synthetic hardware components are used to simulate advanced hardware. Hardware components are used to provide commands and events to control the specific hardware. The scheduler controls the tasks posted by commands and events (Lang, 2006). The concepts of commands and events will be further discussed in Chapter 5. Figure 4.4.2 shows the link between TinyOS and Mote Hardware.

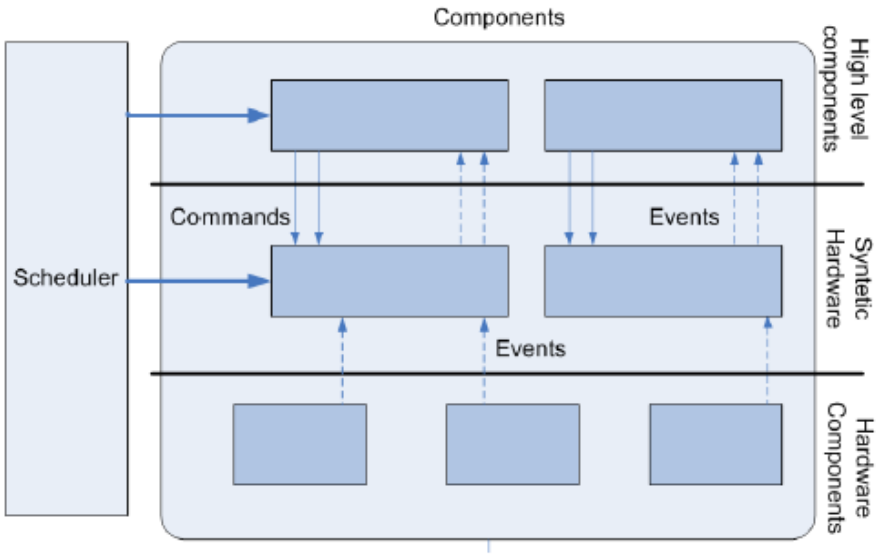

Figure 4.1.1 TinyOS Structure (Lang, 2006)

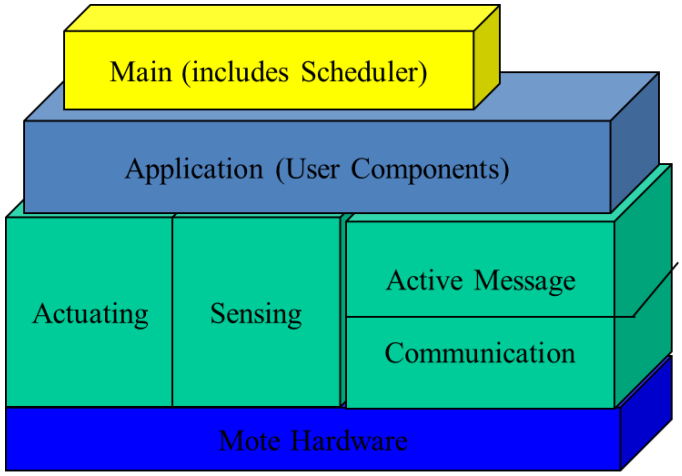

Figure 4.1.2 TinyOS Structure (Chen, 2008)

Section 4.2 Introduction to MoteWorks

MoteWorks is the first industry-used, open source and standards based software platform which supports OEM equipment and system development. It is based on TinyOS and provides the Xmesh networking, OTAP operations, server middleware and client monitoring and managing user interfaces. This software platform supports a variety of wireless sensors. Its powerful flexibility helps developers to choose the best network topology, power management mode and bandwidth of applications, especially for low power operation network. In addition, MoteWorks advanced hardware design makes it possible for users to develop hardware.

In MoteWorks, there are three tiers including the Mote Tier, Server Tier and Client Tier. MoteWorks optimized for the low-power network provides the full end-to-end support for each layer application of sensor networks (IMALEX, 2014).

1. The Mote Tier is composed of nodes running in the Xmesh network (See Chapter 3).
2. In the Server Tier, Xserve play as the bridge between sensor networks and traditional Internet. It also links the communications between sensor networks and enterprise applications. Xserve supports the files, databases, XML (Extensible Markup Language) data parsing and forwarding. The data communication with sensor networks can be completed through databases, custom ports and XML.
3. In the Client Tier, the users can utilize visual software and graphical nodes to manage and monitor the network, sending commands and set alarms. MoteView is one of the client software provided by Crossbow.

The Crossbow's MoteWorks CD-ROM provides the installation of the MoteWorks software platform. It is compatible with Windows XP. The CD-ROM offers software packages shown in the table below. Further installations and instructions can be found in the "MoteWorks Getting Started Guide" (Crossbow,

2014) and “Developing and Designing Undergraduate Laboratory Wireless Sensor Network Exercises” (Aridi, 2012).

MoteWorks Software Package	Introduction
Programmer’s Notepad	Integrated development environment to compile nesC codes and debug (introduced in Chapter 5)
Cygwin	Linux-like environment which supports TinyOS compatible in Windows 2000/XP
AVR Tools	Collection for AVR microcontrollers application development
MoteConfig	Graphical user interface for mote programming and OTAP
MoteView	Graphical user interface for sensor data logging and display
XSniffer	Networking monitoring tool
Graphviz	Tool to view nesC program diagrams
PuTTY	Terminal emulator, serial console and network file transfer
TortoiseCVS	Client server software revision control system

Table 4.2.1 MoteWorks Software Package

Chapter V NesC Language Programming

Section 5.1 Introduction

NesC is an extension of C programming language and mainly used for sensor network programming development. TinyOS is written in nesC and designed for the use of embedded wireless sensor network by UC Berkeley. A nesC program is composed of "components". All the components are wired together to form the entire program. There are two types of components: one is "module" and the other one is "configuration". A module provides the programming codes and realizes interfaces. A configuration links different module and configuration components together and connects interfaces used by component. The syntax of the configuration component is unusual and unlike program codes found in C language programming.

The settings of interfaces explain the component's functions. Interfaces can be set as "provided" or set as "used" by components. The "provided" interfaces provide components' functions to users; the "used" interfaces provide components with needed functions. Interfaces are bidirectional. They describe a set of functions provided by the interface's provider (commands) and another set of functions achieved by the interface's user (events). For instance, an interface can force a component to "send data" by command and force "send data done" by event. That is because the commands in TinyOS are non-blocking and the events send signal to complete the commands (Mitra, Chakraborty, Mondal, & Naskar, 2014).

The rest of this chapter will discuss the nesC programming rules with examples, such as nesC scope, name spaces, definitions of interfaces and components, and how to implement module and configuration. It will also cover the nesC applications coding, compiling and execution with notes

Section 5.2 Interface Definition and Specifications

The interface is used to describe the bi-directional interaction between two different components through the functions of commands and events. The interface provider implements commands to the interface user. The interface user implements events to the interface provider.

The interface definition follows the syntax below:

interface identifier type-parameters$_{opt}$ {declaration-list}

Therefore, the interface definition has an identifier (a unique name), optional type parameters (the same parameter types as C) and a declaration list for the functions of commands and events. A type parameter is the generic labels to refer unknown data type, data structure or class. A declaration list of the interface definition is used to explain TinyOS commands and events.

If the interface definition has type parameters, the interface will be a generic type. There are different types of interfaces according to different types of type parameters. The type parameters cannot be incomplete, function or array type. The components can only be connected using interfaces of the same type.

Example interface definitions provided in "NesC 1.3 Language Reference Manual" are explained (nesC 1.3 Language Reference Manual, 2014):

```
interface SendMsg {

command result_t send(uint16_t address, uint8_t length, TOS_MsgPtr msg);

event result_t sendDone(TOS_MsgPtr msg, result_t success);

}

interface Init<t> {

command void doit(t x);   //Define a command with a void function named as doit

}

module Simple {

provides interface Init<int> as MyInit;

uses interface SendMsg as MyMessage;

} ...
```

There are two interface definitions in this code segment.

The first non-generic interface SendMsg has two functions, one is the command to send the message and the other one is the event to signal the message sending done.

The second interface definition Init<t> is a generic interface definition.

In last section of the code, the module Simple, the “provides” interface which is an Init<int> type and named as MyInit; the “uses” interface which is a SendMsg type and named as MyMessage implements the call to send the message and the event to signal the message sending done.

If the name MyMessage of the interface is not stated, the program will be “uses interface SendMsg”. “Use interface SendMsg” is interpreted as “use interface SendMsg as SendMsg”. Therefore, the interface name will be the same as the name of the interface definition specified by the interface type.

```
provides interface Init<int> as MyInit;

uses interface SendMsg as MyMessage;
```

MyInt and MyMessage are interface names. Int<int> and SendMsg are the names of the interface definitions specified by the interface types.

Hardware Event Handler

By adding the keyword async, the command or the event can be performed as a hardware event handler not a task. The task and hardware event handler are two types of threads of execution when running a nesC application. The task will run to completion once it is scheduled. The hardware event handler can preempt the executions of other tasks and hardware event handlers, and run to completion. (See section 5.4).

Section 5.3 Component Definition, Specifications, and Implementation

A nesC program is built by components. The components provide and use interfaces. A nesC component definition follows the syntax below:

```
component-kind identifier    component-parameters_opt    component-specification implementation_opt
```

There are five specific component kinds as shown below in table 5.3.1:

Module	Configuration	Binary component	Generic module	Generic configuration

Table 5.3.1

There are two kinds of the component implementations: Module implementation and Configuration implementation.

The component name must be different from other components and interfaces. Similar to the interface definition, a component definition with parameters is called a generic component. The difference between a generic component and a component without parameters is that a generic component must be instantiated in the configuration before used. The binary component is a component in a binary form, which has no implementations.

The component specification provides and uses the interfaces (Section 5.2). The component can also contain bare commands and events, typedefs and tagged type declarations.

Module Implementation

The module implements components specifications with the following syntax:

```
implementation {translation-unit}
```

The translation unit contains the codes to implement all the provided commands and events of the module. The commands and events can be non-parameterized or parameterized. The keyword to execute the commands is “call” and the keyword to execute the events is “signal”.

The keyword atomic is used to conditionally prevent simultaneous computations. The following example shows how atomic avoids do_something to execute concurrently (nesC 1.3 Language Reference Manual, 2014):

```
bool busy

void f( ) {

bool available;

atomic {

available = !busy;

busy = TRUE;

}

if (available) do_something;

atomic busy = FALSE;

}
```

Configuration Implementation

The configuration is to define the connection between components. The configuration is implemented by the following syntax:

implementation {configuration-element-list$_{opt}$}

The configuration elements include components, connection and declaration.

The configuration is built by components. Components can be non-generic components, or instantiations of generic components in the configurations. For instance, in the following configuration AWiring:

```
configuration AWiring { }

implementation {

components J, new Grade (2, int);

components new Grade (4, float) as SecondGrade;

}
```

There are one non-generic component J, and two instantiations of generic components that are Grade and SecondGrade.

The wiring statement is defined by the connection (nesC 1.3 Language Reference Manual, 2014):

1. endpoint = endpoint : if any specification elements from any endpoints are external, that makes both sides of the endpoints equivalent.
2. endpoint -> endpoint : this means this connection is from a used specification element specified by the left side endpoint to a provided specification element specified by the right side endpoint.
3. endpoint <- endpoint : this means this connection is from a used specification element specified by the right side endpoint to a provided specification element specified by the left side endpoint.

The codes below (Crossbow, 2007) follow the wiring statements:

Example 1:

```
configuration SingleTimer {
  provides interface Timer;
  provides interface StdControl;
}
implementation {
  components TimerC;
  Timer = TimerC.Timer[unique("Timer")];
  StdControl = TimerC;
}
```

Example 2:

```
  Main.StdControl -> SingleTimer.StdControl;
```

Example 3:

```
SingleTimer.StdControl  <-  Main.StdControl;
```

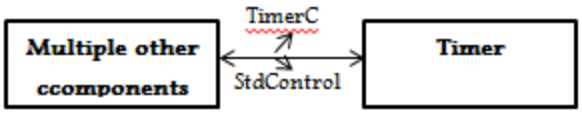

Figure 5.3.1 Example 1

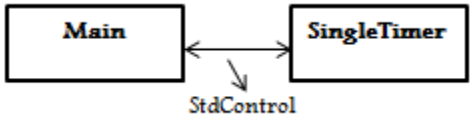

Figure 5.3.2 Examples 2&3

The first example uses wiring statement 1 (endpoint = endpoint) to implement the calling between Timer and multiple other external components. Interface TimerC is the interface provided by Crossbow CD ROM. StdControl is a system interface. (Discussed in "A Deeper Look at Blink Application" part of section 5.7.)

The second example wires "uses" interface StdControl in Main component and "provides" interface StdControl in SingleTimer component. (Main component will be introduced in detail in section 5.7).

The third example accomplishes the same goal, but uses a different syntax.

Synax

The specification elements from both endpoint sides must be the same as commands, events or interfaces. The commands or events must have the same function signature. The interfaces must have the same type.

Section 5.4 NesC Concurrency Model

NesC concurrency model has two threads of execution. Tasks are the functions in the run to completion mode; once tasks are scheduled, they are not preempted or preempt others. Preemption is defined to

interrupt a task temporally with the intention to resume the task later. The tasks can be executed in any order, but still need to follow the run to completion rule. Task can perform longer processing operations like background data processing. One task can be executed till the previous task is completed or be preempted by hardware event handlers. Hardware event handlers perform a small amount of work. They are also running to completion in response to the hardware interrupt. However, hardware event handlers can preempt other tasks and hardware event handlers.

Section 5.5 Summary of NesC Programming Rules

NesC is an extension of C programming language. It was developed as a result of working on the embedded sensor networks. The grammar of nesC has no differences with the standard C. A nesC application is built by components. One component is an "nc." file extension. An application typically has a component "Main" (Similar to the main function of C). "Main" invokes other components to achieve application functions. "Main" invokes other components and one component invokes other components through interfaces. Interfaces can be understood as an encapsulation of function declarations. The content of the interface is a set of function declarations, but without function definitions. Therefore, the interface can be understood as the attribute of the component, the functions can be understood as the attribute of the interface.

Components are divided by two types, module and configuration. The configuration is used to link components and the module is used to achieve how components work. NesC defines two special functions, command and event. In a component, the command is implemented by "call" in a provided interface; the event is implemented by "signal" in a used interface. The keyword "async" declares the command or event can be executed as a part of a hardware event handler to be preempted. Hence "async" can make a command or event executed at any time by preempting other commands or events. Therefore, commands or events declared as "async" can complete small workload quickly. Tasks are used to perform large workload operations. One task can be postponed by using keyword "post", which places the task to

a FIFO order queue. For the coordination of tasks and hardware event handlers, the keyword “atomic” indicates a segment of codes that cannot be interrupted. There is no priority between tasks. However, tasks can be interrupted by hardware event handlers (TinyOS Tutorial, 2003).

Section 5.6 Coding Examples and Explanations

The nesC application is composed of five parts, which are Makefile, Makefile.component, module, configuration and README. Makefile is used to define how to compile and connect source files to generate an executable file, and define the dependencies between files. Makefiles are commonly used by programmers. Makefile.component is used to describe the names of top level application component the sensor board.

The module and configuration function are used to implement and link components. README provides some introductions of the application.

Codes Compiling and Installation

There are two methods to compile and install the codes, one is through Programmers Notepad and the other one is to use Cygwin. To use Programmers Notepad, the program can be compiled through selecting tools and then make IRIS, and be installed through selecting shell and type command “make IRIS reinstall mib520, com(x)”. To use Cygwin, the user must find the directory which the application locates, and then type command “make IRIS” to compile the program and type command “make IRIS reinstall mib520, com(x)” to install the program. The sensor data can be viewed by typing command “Xserve –device=com(x+1)”. The message “writing TOS image” in both methods indicates that the compiling is successful. TOS represents TinyOS. After compiling successfully, a file called “build” will be generated which includes the main,exe application of blink under the directory of blink file.

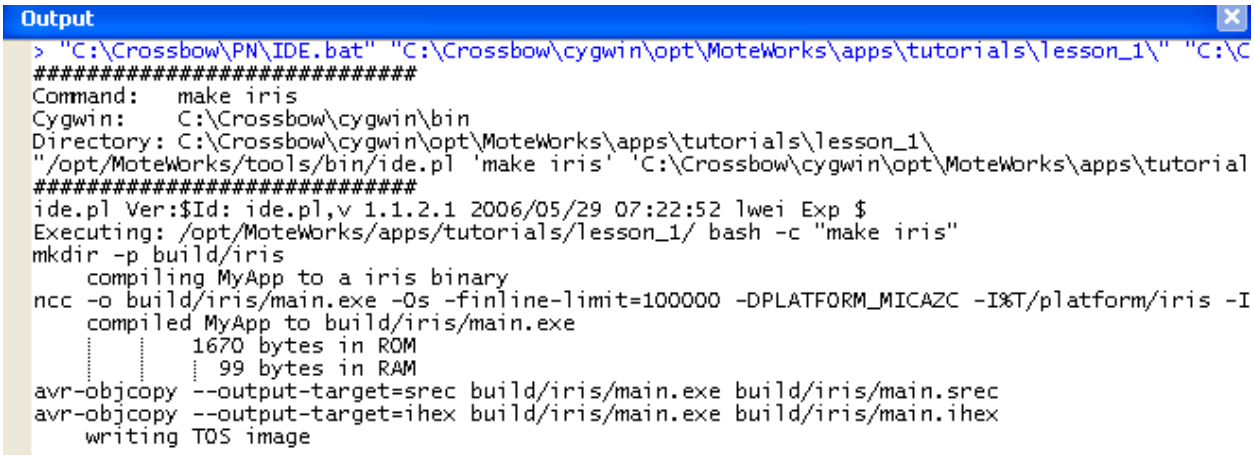

Figure 5.6.1 Compiling in Programmers Notepad

```
Shawn@xiao /cygdrive/c/crossbow/cygwin/opt/moteworks/apps/general/blink
$ make iris
mkdir -p build/iris
    compiling Blink to a iris binary
ncc -o build/iris/main.exe -Os -finline-limit=100000 -DPLATFORM_MICAZC -I%T/plat
form/iris -I%T/lib/Queue -I%T/sensorboards/mts310  -I%T/lib/Broadcast -I%T/lib/X
Lib    -DROUTE_PROTOCOL=0x90 -I%T/radio/rf230 -I%T/lib/internal/XMesh -DMULTIHOP
ROUTER=XMeshRouter -Wall -Wshadow -DDEF_TOS_AM_GROUP=0x7d -Wnesc-all -target=iri
s -fnesc-cfile=build/iris/app.c -board=mts310 -DIDENT_PROGRAM_NAME="Blink" -DIDE
NT_PROGRAM_NAME_BYTES="66,108,105,110,107,0" -DIDENT_USER_ID="Shawn" -DIDENT_USE
R_ID_BYTES="83,104,97,119,110,0" -DIDENT_HOSTNAME="xiao" -DIDENT_HOSTNAME_BYTES=
"120,105,97,111,0" -DIDENT_USER_HASH=0x93cf19e3L -DIDENT_UNIX_TIME=0x53be0180L -
DRF230_DEF_CHANNEL=11  -DRF230_TXPOWER=TXPOWER_MAX Blink.nc -lm
    compiled Blink to build/iris/main.exe
            1670 bytes in ROM
              99 bytes in RAM
avr-objcopy --output-target=srec build/iris/main.exe build/iris/main.srec
avr-objcopy --output-target=ihex build/iris/main.exe build/iris/main.ihex
    writing TOS image
```

Figure 5.6.2 Compiling in Cygwin

```
Shawn@xiao /cygdrive/c/crossbow/cygwin/opt/moteworks/apps/general/blink
$ make iris reinstall mib520,com3
cp build/iris/main.srec build/iris/main.srec.out
    installing iris binary using mib520
uisp -dprog=mib520 -dserial=com3 --wr_fuse_h=0xd9 -dpart=ATmega1281 --wr_fuse_e=
ff  --erase --upload if=build/iris/main.srec.out
Firmware Version: 1.8
Atmel AVR ATmega1281 is found.

Fuse High Byte set to 0xd9

Fuse Extended Byte set to 0xff

Uploading: flash
rm -f build/iris/main.exe.out build/iris/main.srec.out
```

Figure 5.6.3 Installing in Cygwin

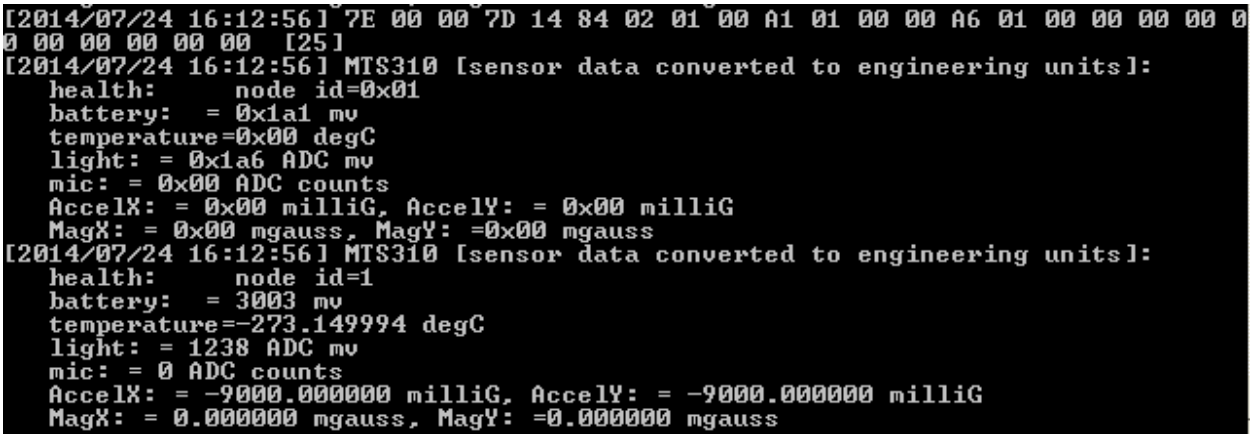

Figure 5.6.4 Sensor Data reported in Cygwin

The structure chart can be constructed to help understand an application visually (Applied in the following examples).

A Deeper Look at Blink Application

The basic application Blink is included on the CD-ROM installation and used to toggle the LED on the IRIS mote. Blink.nc (853 Bytes) implements the wiring between components. The relationship between four different application components is defined in Figure 5.6.5. BlinkM.nc (1.46 KB) implements the application function. SingleTimer.nc (826 Bytes) provides a single timer. LedsC.nc (3.23 KB) is a library module component that provides LEDs.

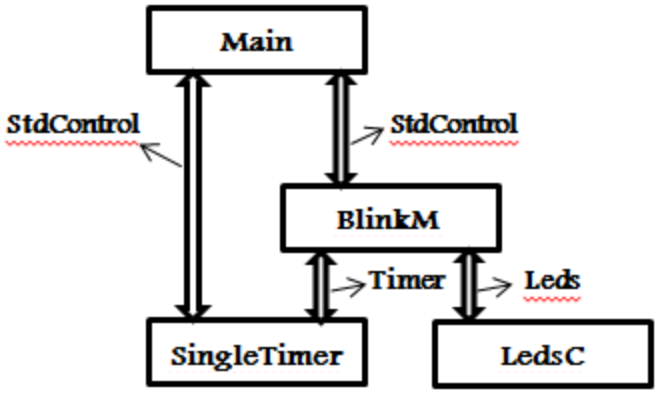

Figure 5.6.5 Blink Structure Chart

As shown in Figure 5.6.5, Main is the first component executed in the application, TinyOS has a "smart" Main component to control the operations of external and internal components directly or indirectly. Main component is used to initialize, start and stop other program components through StdControl interface. In the Blink application, the configuration file (Blink.nc) declares that Main component wires SingleTimer and BlinkM components for control purposes through StdControl directly. It also declares two separate wire connections: BlinkM and SingleTimer, BlinkM and LedsC from the module file (BlinkM.nc).

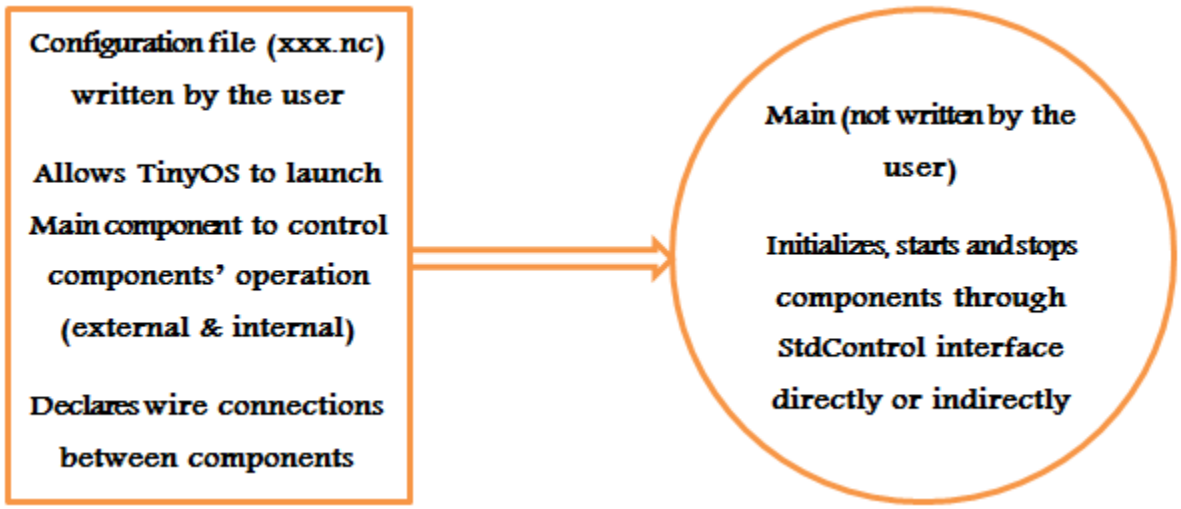

Figure 5.6.6 Main component operation

Blink.nc

```
configuration Blink {

}

implementation {

  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> SingleTimer.StdControl;  //Wire StdControl in Main and StdControl in SingleTimer

  Main.StdControl -> BlinkM.StdControl;

  BlinkM.Timer -> SingleTimer.Timer;     //BlinkM.Timer uses Timer provided by SingleTimer

  BlinkM.Leds -> LedsC.Leds;

}
```

The keyword "configuration" declares this is a configuration file. "Configuration Blink {}" declares the name of this configuration is Blink as the same as Module's name. The real content in the configuration is implemented in the brace after the keyword "implementation". Main, BlinkM, SingleTimer and LedsC are the components which this configuration uses after the keyword “component”. The rest part of the implementation is to wire the interfaces that these components use and the components that provide these interfaces.

BlinkM is the module component to implement application function. The interfaces Timer is implemented by component SingleTimer. The interface Leds provided with the Crossbow CD-ROM installation is implemented by component LedsC.

StdControl.nc

```
interface StdControl

{

   command result_t init();      //Initialize the component and return if the operation is successful.

   command result_t start();    //Start the component and return if the operation is successful.

   command result_t stop();    //Stop the component and return if the operation is successful.

}
```

The interface StdControl is a public interface used to initial and launch other TinyOS components. For instance, Main.StdControl -> SingleTimer.StdControl means that SingleTime.StdControl.int() is called by Main.StdControl.init(). It shows the interface StdControl define three commands: init(), start() and stop(). Init() can be called many times. However, it cannot be called once start() or stop() are called. Result_t is a data type for the status value returned by a command or event and it can either succeed or fail.

BlinkM.nc

```
module BlinkM {
  provides {
    interface StdControl;
  }
  uses {    //uses the interfaces provided by library modules
    interface Timer;
    interface Leds;
  }
}
implementation {
  command result_t StdControl.init() {     //Initialize the component.
    call Leds.init();
    return SUCCESS;
  }
  command result_t StdControl.start() {    //Start the component.
       return call Timer.start(TIMER_REPEAT, 1000);  // Set the repeat time as 1000ms.
```

BlinkM.nc, continued

```
 }

 command result_t StdControl.stop() {    //Stop the component.

  return call Timer.stop();

 }

 event result_t Timer.fired()    //Turn the red LED on when the timer fires defined by the interface timer.

 {

  call Leds.redToggle();

  return SUCCESS;

 }

}
```

BlinkM module provides the interface StdControl and that means it must implement this interface including the command StdControl.init(), StdControl.start() and StdControl.stop(). It also implements Timer.fired() event in the interface Timer.

Init() is implemented to initialize the Leds by calling Leds.init(). Start() is implemented to create the timer by call Timer.start(). Stop() is implemented by calling Leds.redToggle() when Timer.fired() is triggered.

SingleTimer.nc

```
configuration SingleTimer {

  provides interface Timer;

  provides interface StdControl;

}

implementation {

  components TimerC;

  Timer = TimerC.Timer[unique("Timer")];

  StdControl = TimerC;

}
```

Configuration SingleTimer implements its calling through the provided interfaces of TimerC and StdControl. Since multiple timers are allowed in nesC, the Unique("astring") is used to make this timer unique from others in use. The string generates an 8-bit identifier as an argument. Multiple components which use timer [unique(“Timer”)] are each guaranteed to get a signal associated with their specific timer settings. In this program, “Timer” is used as the “astring”.

Example Application 1

This application is used to obtain the value of light intensity from the sensor board and display the lowest 3 bits to the three LEDs. Sense.nc (323 Bytes), the configuration file, defines the wire connections between components. The relationship between five different application components is defined in Figure 5.7.6. SenseM.nc (1.80 KB) implements the application function. TimerC.nc (1.02 KB) and LedsC.nc (3.23 KB) are library modules. Photo.nc (1.08 KB) and PhotoM.nc (6.02 KB) are used to activate the light sensor.

Interface Timer.nc (1.80 KB), interface ADC.nc (1.40 KB) and interface Leds.nc (2.29 KB) are modified from the system interfaces.

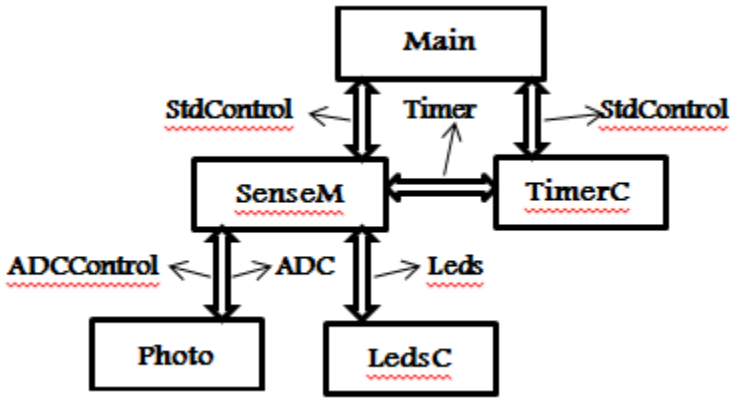

Figure 5.6.7 Example Application 1 Structure Chart

Sense.nc

```
configuration Sense {

}

implementation

{

  components Main, SenseM, LedsC, TimerC, Photo;

  Main.StdControl -> SenseM;

  Main.StdControl -> TimerC;

  SenseM.ADC -> Photo;                    //Same as SenseM.ADC -> Photo.ADC

  SenseM.ADCControl -> Photo;       //Same as SenseM.ADCControl -> Photo.StdControl

  SenseM.Leds -> LedsC;

  SenseM.Timer -> TimerC.Timer[unique("Timer")];  //Parameterized interface

}
```

Parameterized interface allows the component provides multiple instantiations of an interface by giving parameters. It is similar to give different names to the interface instantiations. For instance, in TimerC:

```
provides interface Timer[uint8_t id];   // uint16_t is unsigned 16-bit integer from 0 to 255
```

It provides 256 instantiations of the Timer. Each instantiation is corresponding to each value of uint8_t.

Since this application needs to create and use different timers that can be managed independently.

The component Photo just provides two interfaces ADC and StdControl without interface ADCControl. In fact, ADCControl is a new name of the instantiation of interface StdControl.

SenseM.nc

```
module SenseM {
  provides {
    interface StdControl;  //Define interface StdControl
  }
  uses {
    interface Timer;
    interface ADC;
    interface StdControl as ADCControl;  //Use multiple instantiations of an interface by giving  names.
    interface Leds;
  }
}
implementation {
  result_t display(uint16_t value)     // uint16_t is unsigned 16-bit integer from 0 to +65535
```

SenseM.nc, continued

```
 {
if (value &1) call Leds.yellowOn();
/**Assume value is 13 in decimal and 1101 in binary, 1101&0001 is 0001. **/
   else call Leds.yellowOff();                //Define yellow LED as the least significant bit.
if (value &2) call Leds.greenOn();
/**Assume value is 13 in decimal and 1101 in binary, 1101&0010 is 0010.* */
   else call Leds.greenOff();                 //Define green LED as the second bit.
if (value &4) call Leds.redOn();
/**Assume value is 13 in decimal and 1101 in binary, 1101&0100 is 0100. **/
   else call Leds.redOff();                   //Define red LED as the most significant bit.
   return SUCCESS;
 }
 command result_t StdControl.init() {          //Initialize ADCControl, Leds.
   return rcombine(call ADCControl.init(), call Leds.init());  //Return value is logic value "and" of 2
                                                                commands result using function rcombine
}
```

SenseM.nc, continued

```
  command result_t StdControl.start() {         //Start Timer and 0.5 second clock.
    return call Timer.start(TIMER_REPEAT, 500);
}
  command result_t StdControl.stop() {          //Stop Timer.
    return call Timer.stop();
}
  event result_t Timer.fired() {    //Read sensor data when Timer fires.
    return call ADC.getData();     //Return result from calling ADC.getData().
}
  async event result_t ADC.dataReady(uint16_t data) {     //Display upper 3 bits sensor readings to LEDs.
display(7-((data>>7) &0x7));
/**ADC converts the sensor readings to 10 bits value. The expected activity of photo sensor is to turn off LED when a node is in the lightness and turn on LED when in the darkness. Therefore the upper 3 bits of the 10 bit value should be complemented. **/
    return SUCCESS;
}
```

SenseM provides and uses the interface StdControl, Timer and Leds. In addition, it uses another interface, ADC (Analog to Digital Converter). This module also uses the component TimerC to allow using multiple timers.

Example Application 2

This application included in Crossbow CD-ROM is used to sample the light intensity from the sensor board every one second and send the data back to the base station. The yellow LED on means the data has been sampled and the green LED on means the data has been sent to the base station. MyApp.nc (848 Bytes) implements the wiring between components. The relationship between six different application components is defined in Figure 5.7.7. MyAppM.nc (3.22 KB) implements the application function. Photo.nc (1.08 KB) and PhotoM.nc (6.02 KB) are used to activate the light sensor. TimerC.nc (1.02 KB) and LedsC.nc (3.23 KB) are library modules. The library module GenericComm.nc (1.93 KB) causes the radio transmission.

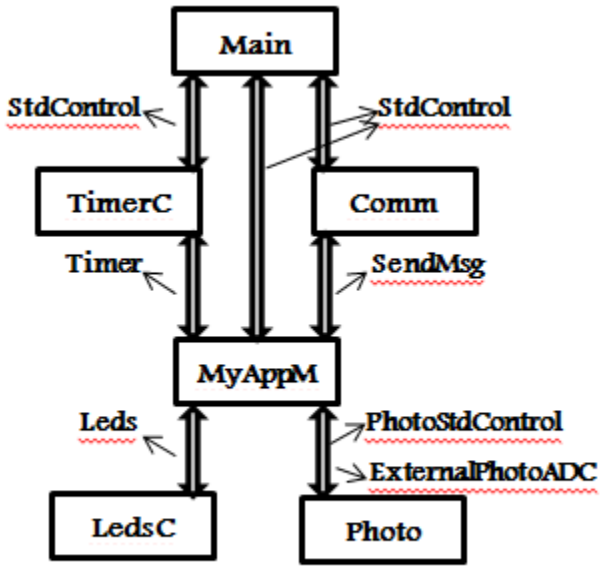

Figure 5.6.8 Example Application 2 Structure Chart

MyApp.nc

```
configuration MyApp {

}

implementation {

  components Main, MyAppM, TimerC, LedsC, Photo, GenericComm as Comm;

  Main.StdControl -> MyAppM.StdControl;

  Main.StdControl -> Comm.Control;



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

  MyAppM.Leds -> LedsC.Leds;

  MyAppM.PhotoControl -> Photo.PhotoStdControl;

  MyAppM.Light -> Photo.ExternalPhotoADC;

  MyAppM.SendMsg -> Comm.SendMsg[AM_XSXMSG];

/**AM_XSXMSG is the type of message. Wire Xsensor channel of GenericComm into application's
send interface. **/

}
```

MyAppM.nc

```
module MyAppM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer; //Define interface Timer named as Timer
    interface Leds;
        interface StdControl as PhotoControl;  //Define interface StdControl named as PhotoControl
        interface ADC as Light;
        interface SendMsg;
  }
}
implementation {
  bool sending_packet = FALSE;
  TOS_Msg msg_buffer;  //Define message packet
  XDataMsg *pack;
```

MyAppM.nc, continued

```
command result_t StdControl.init() {              //Initialize the component
  call Leds.init();
        call PhotoControl.init();
  atomic {                                        //Initialize the message packet with default values
    pack = (XDataMsg *)&(msg_buffer.data);
    pack->xSensorHeader.board_id = SENSOR_BOARD_ID;
    pack->xSensorHeader.packet_id = 2;
    pack->xSensorHeader.node_id = TOS_LOCAL_ADDRESS;
    pack->xSensorHeader.rsvd = 0;
        }
  return SUCCESS;                                 //Return the code Success always
}
command result_t StdControl.start() {          //Start the component
  return call Timer.start(TIMER_REPEAT, 1000);    //Set 1 second as repeating time
}
command result_t StdControl.stop() {          //Stop the component
```

MyAppM.nc, continued

```
  return call Timer.stop();                    // Call the timer to stop

}

event result_t Timer.fired()                   //The Timer fires

{

  call Leds.redToggle();                       // Call the red LED on

        call PhotoControl.start();             //Light sensor control starts

        call Light.getData();                  //Start to sample the data

  return SUCCESS;

}

void task SendData()                           //Task to build message packet and send data

{

  call PhotoControl.stop();                    //Stop the light sensor control



  if (sending_packet) return;        //The if statement is to decide if sending to serial port is successful.

  atomic sending_packet = TRUE;

        if (call SendMsg.send(TOS_UART_ADDR,sizeof(XDataMsg),&msg_buffer) != SUCCESS)
```

MyAppM.nc, continued

```
                    //Changing TOS_UART_ADDR to TOS_BCAST_ADDR will send the message through
                      radio.

          sending_packet = FALSE;

  return;

}

async event result_t Light.dataReady(uint16_t data) {   //Get the data ready

        atomic pack->xData.datap1.light = data;  // Store the light sensor data in the message packet

        atomic pack->xData.datap1.vref = 417; //A dummy 3V reference voltage, 1252352/3000 = 417

  post SendData();    //Post the SendData task to send a message containing sensor data

        call Leds.yellowToggle();                    //Toggle yellow LED to indicate sampling successful.

  return SUCCESS;

}

event result_t SendMsg.sendDone(TOS_MsgPtr msg, result_t success) {  //Send data to serial port

        call Leds.greenToggle();                    //Toggle green LED to indicate sending successful.

        atomic sending_packet = FALSE;

  return SUCCESS;  }}
```

The interface ADC for light sensors calls the command getData to sample the value of light intensity and signal event dataReady to complete sampling to get data ready. The type of the value of light intensity is uint16_t (16 bits).

Section 5.7 Summary of NesC Programming Features

Through the detailed analysis of the programs above, structured programming concepts can be implemented using nesC language. NesC applications can call interfaces to reduce coding workload, establish the links between components quickly, and reduce unnecessary resource consumption when executing tasks and events. Mastering the grammar of nesC can greatly reduce the complexity to achieve the wireless sensor network operating system and applications. It provides a method of reference to deeply study and research TinyOS and design applications in TinyOS. C language is effective and compatible with a large number of microcontrollers. It has a strong readability and mastered by most programmers. Since nesC is an extension of C language, that helps the popularization of nesC. NesC applications have no dynamic memory allocation, which simplifies coding and debugging.

Chapter VI Sensor Data Display in LabVIEW

Section 6.1 Introduction to LabVIEW and Crossbow XMesh WSN drivers

LabVIEW is a program development environment developed by National Instrument (NI). LabVIEW has a complete and extensive function library, which includes data acquisition, serial port control, data display and storage, data analysis and forth. LabVIEW provides components like the traditional instruments such as oscilloscopes, multimeters. These components can be used to build user interfaces easily. The user interface in LabVIEW is called front panel.

The significant difference between LabVIEW and other computer languages is that LabVIEW uses G language to write programs, a graphical editing language. LabVIEW programs are in the form of block diagrams. Traditional text programming language executes applications according to the order of statements and commands. However, LabVIEW uses data flow programming. The data flow between nodes in the block diagram determines the execution order of Virtual Instruments (VIs) and Functions. It is an organized data acquisition system based on the demands of the instrument (Baidu, 2014).

LabVIEW software is the core of the NI design platform and used to development measurement and control systems. LabVIEW development environment is integrated with all the necessary tools to quickly build a variety of applications. It is designed to help engineers and scientists to solve problems, improve productivity and innovate.

Crossbow XMesh WSN Instrument Driver is one of the third party drivers provided by NI. In addition to the driver, examples are provided including acquiring, displaying data and sub-VIs such as start stream, write, add node to build new programs (National Instruments, 2007). The NI VISA (Virtual Instrument Software Architecture) driver and FTDI (Future Technology Devices International) driver must be installed allow the use of WSN base station com ports.

Figure 6.1.1 Crossbow XMesh WSN Instrument Driver Directory

Section 6.2 A deep look into Front Panel and Block Diagram

The VI used to observe sensor status is revised from the example Read Data and Display Health and LabVIEW-Basic Scenario (Aridi, 2012) (Border, 2012). The original VI is applied to MTS300 sensor by default. There is a pull-down list for users to choose appropriate sensors but the original VI has to be revised for different sensors. Since the MTS400 sensor is set to observe ten values in the default program, the data display window in the front panel of the provided VI should be modified to read and display all ten values. The data output window is found in the WSN Read.vi. The revised front panel and block

diagram are shown in figure 6.2.1 and 6.2.2. Node IDs field reports the detected nodes. Gateway VISA Resource is used to configure the appropriate port number (com x+1) connected to the MIB520 base station by a pull-down list. The components used to process data stream are provided by the Crossbow XMesh WSN Instrument Driver. The other two available sensor monitoring programs are shown in figure 6.2.3 and 6.2.4 (NI Discussion Forums, 2014).

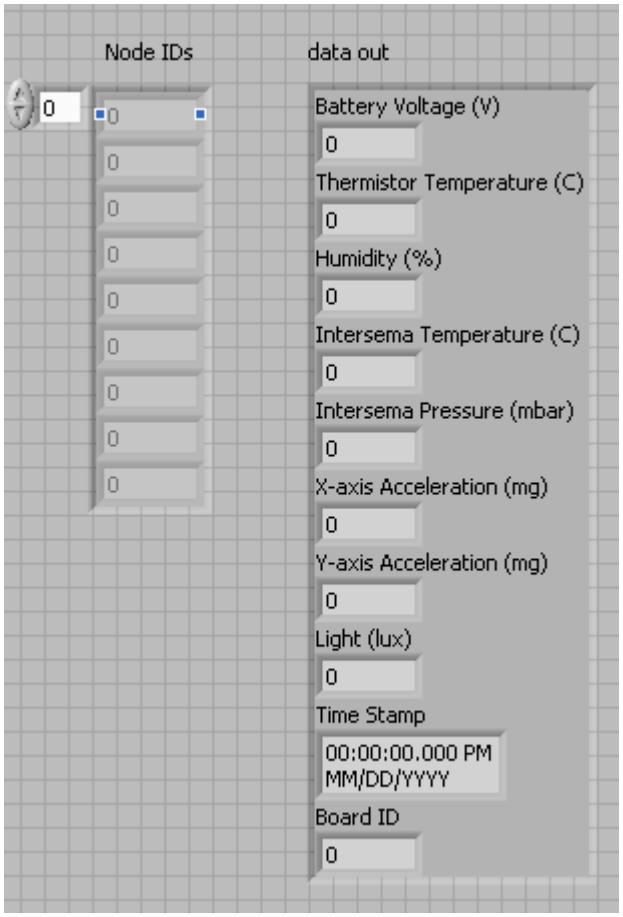

Figure 6.2.1 Revised Front Panel of Read Data and Display Health

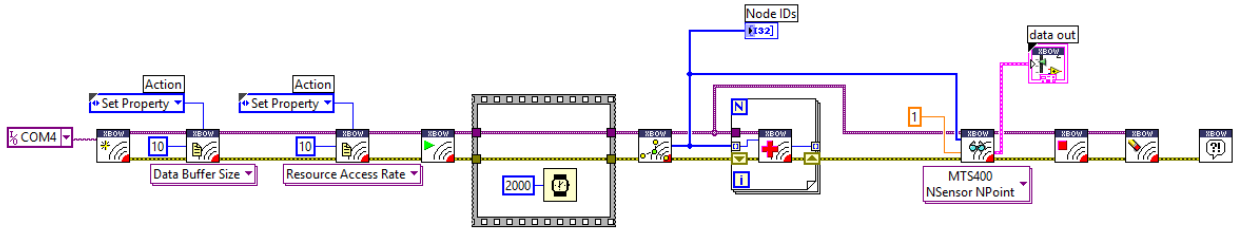

Figure 6.2.2 Revised Block Diagram of Read Data and Display Health

The data flow diagram can be explained step by step:

1. Create stream: allocate resources to connect LabVIEW to base station.
2. Start stream: start listening and retaining packets.
3. Timer to wait for packets. It takes time to receive packets.
4. Get node list: read nodes IDs that packets have been received.
5. Read health: display node IDs and read the sensor data received.
6. WSN read: display the most recent sensor data from each node.
7. Stop stream: close the connection to base station and stop receiving packets.
8. Clear stream: clear the packets received and destroy resources allocated.
9. Error handler: indicate errors with a dialog box.

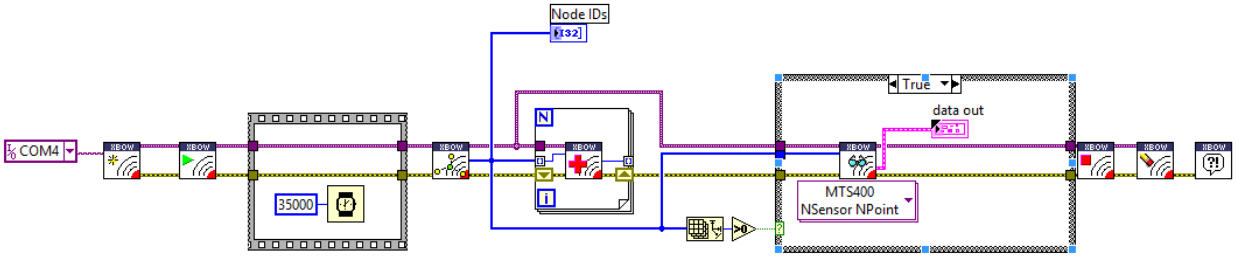

Figure 6.2.3 MTS400 sensor Monitoring Program (NI Discussion Forums, 2014)

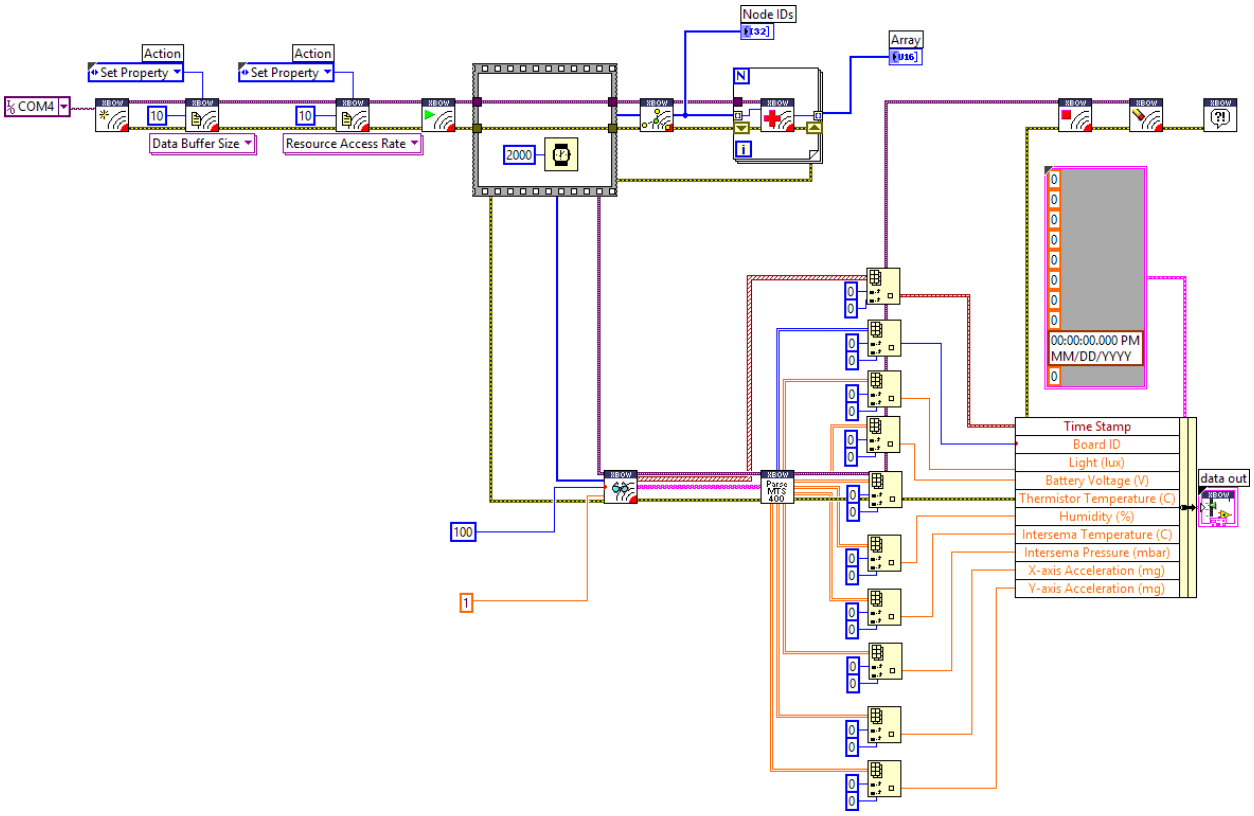

Figure 6.2.4 MTS400 Sensor Monitoring Program (NI Discussion Forums, 2014)

Definitions of Terms

Typedef is a keyword to define a complex type by using basic types. For instance, the codes "typedef int mile_per_hour" mean that mile_per_hour will be a new variable treated as int in programming.

Enum is a keyword to define a variable that can be one of the preset constants. E.g.: enum month {JAN, FEB, MAR, APR, MAY, JUN}

Storage class specifiers define the scope and lifetime of the object and function in a program..

Function signature provides the name, parameters and other information of a function**.**

Debug is a process to find bugs, defects and errors in the program.

uint8_t: stores an unsigned 8 bit number and the range is 0 to +255. Int8_t is from -128 to +127. Uint16_t is from 0 to +65535.

Specifications specify the interaction between components.

References

1. *TinyOS Tutorial*. (2003). Retrieved 2014, from http://www.tinyos.net/tinyos-1.x/doc/tutorial/

2. *National Instruments*. (2007). Retrieved 2014, from Crossbow XMesh WSN Sensor: http://sine.ni.com/apps/utf8/niid_web_display.download_page?p_id_guid=1FB66B354ED149C7E0440003BA230ECF

3. *Baidu*. (2014). Retrieved 2014, from http://baike.baidu.com/subview/140209/5119782.htm?fr=aladdin#10

4. *Baidu*. (2014). Retrieved 2014, from http://baike.baidu.com/view/230451.htm

5. *Memsic*. (2014). Retrieved 2014, from http://www.memsic.com/wireless-sensor-networks/

6. *nesC 1.3 Language Reference Manual.* (2014). Retrieved 2014, from https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=0CDYQFjAC&url=http%3A%2F%2Fwww.tinyos.net%2Fdist-2.0.0%2Ftinyos-2.0.0beta1%2Fdoc%2Fnesc%2Fref.pdf&ei=EyXLU-rfEdGNyASWxILoBw&usg=AFQjCNEIvL41UX0Joag146qbLHewLCgAOg&sig2=QxJR

7. *NI Discussion Forums*. (2014). Retrieved 2014, from http://forums.ni.com/t5/LabVIEW/Labview-Drivers-for-Wireless-Sensor-Networks/td-p/430497

8. *Wikipedia*. (2014). Retrieved 2014, from http://en.wikipedia.org/wiki/IEEE_802.15.4

9. Aridi, E. O. (2010). *Developing and Designing Undergraduate Laboratory Wireless Sensor Network Exercises.* Bowling Green, Ohio: Bowling Green State University.

10. Border, D. (2012). Developing and Designing Undergraduate Laboratory. *Proceedings of American Society for Engineering Education Annual Conference.*

11. Chen, S. (2008). A Glance to Tiny OS. OU-TCOM.

12. Crossbow. (2007). *M2110 Hardware Reference Manual.* Retrieved 2014, from https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CB8QFjAA&url=http%3A%2F%2Fwww.devisersoftware.com%2Fuploads%2Fflink%2F2014320132650447.pdf&ei=n3rcU7HDCoqmyATbl4DACg&usg=AFQjCNEtbO2pZoZtyHFU6O5FVU-UsbK4UA&sig2=6ndzKaIODL

13. Crossbow. (2007). *MPR/MIB User's Manual.* Retrieved 2014, from http://paginas.fe.up.pt/~ee03061/Files/mpr-mib_series_users_manual.pdf

14. Crossbow. (2014). Retrieved 2014, from MoteWorks Getting Started Guide: http://www.memsic.com.cn/index.php?option=com_phocadownload&view=category&download=270%3Amoteworks-getting-started-guide&id=6%3Auser-manuals&Itemid=86&lang=zh

15. IMALEX. (2014). *Wireless Sensor Network Development Environment.* Retrieved 2014, from http://blog.csdn.net/imalex/article/details/9106157

16. Lang, M. (2006). *TinyOS.* Retrieved 2014, from https://koala.cs.pub.ro/redmine/attachments/download/154/ami-report-23_Lang_tinyos.pdf

17. Mazidi, M. A., & Causey, D. (2009). HCS 12 MICROCONTROLLER AND EMBEDDED SYSTEMS. Person Prentice Hall.

18. Memsic. (2010). *XMesh User Manual.* Retrieved 2014, from http://www.devisersoftware.com/uploads/flink/2014320132832286.pdf

19. Mitra, S. K., Chakraborty, A., Mondal, S., & Naskar, M. (2014). *Simulation of Wireless Sensor Networks UsingTinyOS- A case study.* Retrieved 2014, from http://www.academia.edu/438512/Simulation_of_Wireless_Sensor_Networks_Using_TinyOS-A_Case_Study

20. Nack, F. (2010). An Overview on Wireless Sensor Networks. Institute of Computer Science, Freie University Berlin.

21. Ren, X., & Yang, Z. (2010). Research on the key issue in video sensor network. *International Conference on Computer Science and Information Technology.*

22. Tutorialspoint. (2014). *C - Operators*. Retrieved 2014, from tutorialspoint: http://www.tutorialspoint.com/cprogramming/c_operators.htm

23. Wikipedia. (n.d.). Retrieved 2014, from http://en.wikipedia.org/wiki/PuTTY

24. Wikipedia. (2014). Retrieved 2014, from http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus