Wireless Sensor Network for Energy and Environmental Monitoring: Helping Old Buildings Go Green

Jacob Power

Senior Project

Electrical Engineering Department

California Polytechnic State University

San Luis Obispo

Table of Contents

List of Tables and Figures

Acknowledgements

I would like to thank the following individuals who were instrumental in the execution of this project: my adviser, Dr. John Oliver, for providing valuable technical insight; and my parents, Stephen and Elaine Power, for providing moral and practical support.

List of Abbreviations

ADC	Analog-to-digital converter
APS	Application Support Sublayer
BSP	Board Support Package
CS	Configuration Server
CT	Current transformer
GPIO	General Purpose Input Output
HAL	Hardware Abstraction Layer
HVAC	Heating, Ventilation, and Air Conditioning
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IRQ	Interrupt Request
LAN	Local Area Network
LQI	Link Quality Index
MAC	Media Access Control
NWK	Network Layer
PAN	Personal Area Network
RCB	Radio Control Board
RF	Radio Frequency
RP-SMA	Reverse-polarity Subminature version A connector
RSSI	Received Signal Strength Index
SoC	System on Chip
SQL	Structure Query Language
UID	Unique Identifier
WSN	Wireless Sensor Network
ZDO	Zigbee Device Object

Abstract

Today, the rising cost of energy and social trend of “going green” has put increasing pressure on commercial property owners to improve the efficiency of their investments. There is good reason for this, too. In the State of California, office buildings represent the single largest user of energy in the commercial sector. This project consisted of the design, implementation, construction, testing, and deployment of a wireless sensor network for energy and environmental monitoring in retrofit commercial office installations. The analyzed data generated by this system can be used by building management and owners to identify areas of inefficiency within the building’s major energy using systems – namely heating, ventilation, and air conditioning (HVAC) and lighting. While suitable for use in new construction, the system was designed for ease-of-use in retrofit installations in existing construction. The final system consists of a self-expanding mesh network of lightweight, battery- and line-powered sensor nodes which are installed at various locations throughout the building to gather power consumption, environmental, occupancy, and space utilization data for a complete energy audit.

I. Introduction

Today, the rising cost of energy and social trend of "going green" has put increasing pressure on commercial property owners to improve the efficiency of their investments. There is good reason for this, too. In the State of California, office buildings represent the single largest user of energy in the commercial sector (Efficiency Partnership). As Figure 1 shows, the majority of the energy consumed by the typical commercial office building is in heating, ventilation, and air conditioning (HVAC), lighting, and office equipment.

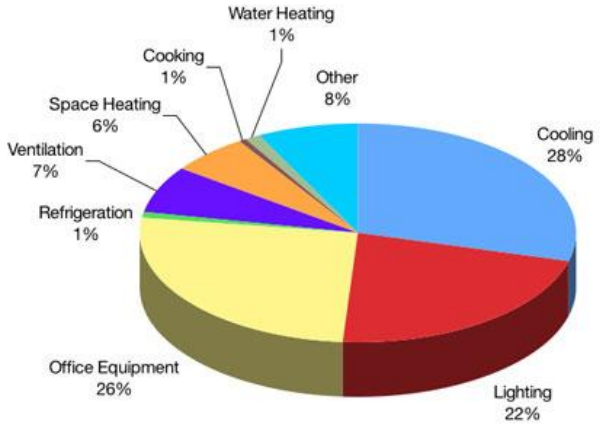


Figure 1: Electricity End-Use Consumption in Office Buildings by Activity

Older buildings tend to be less efficient than newer buildings. In HVAC systems, poorly maintained equipment and dated control systems can yield poor energy performance. The energy efficiency of a building can decrease over time due to deferred equipment maintenance, improperly configured controls, user error, and much more. Worse, even modern "green" buildings can suffer from poor efficiency due to the incorrect installation, configuration, and use of increasingly complex building automation and control systems.

This project consisted of the design, implementation, construction, testing, and deployment of a wireless sensor network for energy and environmental monitoring in retrofit commercial office installations. The analyzed data generated by this system can be used by building management and owners to identify areas of inefficiency within the building's major energy using systems – namely heating, ventilation, and air conditioning (HVAC) and lighting.

While suitable for use in new construction, the system was designed for ease-of-use in retrofit installations in existing construction. The final system consists of a self-expanding mesh network of lightweight, battery- and line-powered sensor nodes which are installed at various locations throughout the building to gather power consumption, environmental, occupancy, and space utilization data for a complete energy audit.

2. Requirements

At the beginning of the design process, the following overall design goals and requirements were established for the final system.

- Monitor a variety of standard building automation and control signals to facilitate the monitoring of existing building equipment.
- Measure the power consumption of a typical large electrical load within a commercial building, such as HVAC, lighting, and office equipment.
- Install a system that will be easy and safe enough for the average non-technical individual to perform with limited instruction. As such, as much of the system as possible should be wireless and battery-powered.
- Install a system that will not interfere with the normal operation of building equipment and controls.
- Provide an easy-to-use interface for the end user to view both real-time and stored data from the sensor network.

3. Design

3.1. Platform Selection

As an embedded systems project, the most fundamental design decision was the selection of a platform for the sensor network. As a wireless sensor networking project, there was the additional constraint that the chosen platform include support for a low-power, low-rate wireless networking protocol. After surveying product offerings from a wide range of embedded device manufacturers, including Atmel, Microchip, Texas Instruments, Freescale, Silicon Labs, Jennic (now owned by NXP), Digi, and Ember, four platforms were chosen for additional investigation (Table 1).

Manufacturer	Product Line	Supported Protocol(s)
Atmel	ATmega with ATRF radios, ATmegaRFA1 SoC	Zigbee Pro
Texas Instruments	MSP430 with CC radios, CC SoC	Zigbee Pro, SimpliciTI
Digi	Xbee	Zigbee Pro, Digimesh

Table 1: Wireless platforms researched

Extensive research was then performed on each of these manufacturers' wireless product offerings.

- Atmel
    - Hardware: Atmel manufacturers the ubiquitous AVR line of 8- and 32-bit microcontrollers – most notably the ATmega line. In addition to their microcontroller offerings, Atmel also provides a line of RF transceiver chips – the AT86RFxxx series. The ATMega128RFA1 is Atmel's current top-of-the-line wireless SoC, and contains the equivalent of an ATmega1281 microcontroller and AT86RF230 2.4GHz wireless transceiver. Atmel also offers the Zigbit line of integrated microcontroller wireless modules.
    - Software: Atmel offers a four software wireless networking software stacks for use with their microcontroller and radio hardware.
        - IEEE 802.15.4 MAC: This IEEE 802.15.4-compliant network stack provides a basic data transport API with support for single-hop star networks.

- RF4Control: A Zigbee RF4CE-compliant network stack for use in consumer electronics remote control applications.
- Bitcloud: A Zigbee Pro-compliant network stack for use in multi-hop wireless networking applications, including sensor networks, security systems, asset tracking, and more.
- Bitcloud Profile Suite: An additional software layer over the Bitcloud stack that provides support for the Zigbee Smart Energy, Home Automation, and Building Automation device profiles.

- Development Tools
  - Development Software: At the time of microcontroller research, AVR Studio 4 was the primary development platform supported for 8-bit AVR development. While this product is somewhat dated when compared to Texas Instruments' new Eclipse-based Code Composer Studio 4, AVR Studio 4 has a large installed user base and is fairly powerful. However, at the time of this writing, Atmel has released the beta version of their next-generation development environment, AVR Studio 5. This environment is based around Microsoft's powerful Visual Studio platform, and is a massive improvement over AVR Studio 4. AVR Studio 5 offers support for both 8- and 32-bit development in the same environment, "Intellisense"-like coding assistance, improved debugging capabilities, and much more.
  - Development Hardware: At the time of this writing, Atmel's wireless development hardware options are rather meager; the AVR Dragon and RZ600 platforms are among the few available directly from Atmel. Several more robust (abeit expensive) platforms are available from third-party manufacturers, including Dresden elektronik, MikroElektronika, and Sparkfun. However, Atmel does offer a flexible and wide range of programming and debugging options for all budgets, including the AVRISP mkII, AVR Dragon (used for this project), and JTAGICE mkIII.

- Texas Instruments
    - Hardware: Texas Instruments offers a full line of 2.4GHz, Zigbee-capable transceivers and system-on-chip ICs. Many of these products are based around the popular MSP430 microprocessor core. Unfortunately, aside from their RF2500 evaluation kit, TI does not offer many economical evaluation or development kits for their wireless offerings; at the time of project research, the cheapest Zigbee-capable wireless development kit was over $500!
    - Software: Texas Instruments provides two free wireless networking stacks for use with their wireless hardware: SimpliciTI and ZStack. SimpliciTI is an IEEE 802.15.4-compliant, lightweight wireless stack targeted at smaller, simpler wireless sensor networking applications. SimpliciTI offers a reduced feature set compared to Zigbee. ZStack is a fully Zigbee Pro-compliant networking stack, but will only run TI's more powerful wireless transceivers and microcontrollers.
- Digi
    - Hardware: Digi manufactures the XBee line of integrated wireless modules in a convienient through-hole package. These modules are available for the Zigbee or Digimesh protocols. Digimesh is Digi's own IEEE 802.15.4-compliant lightweight mesh networking protocol. Each XBee module is based around an Ember wireless SoC running custom Digi firmware. This firmware allows the user to configure each XBee as either a "wireless serial cable" or a simple periodic sampling sensor node.
    - Software: The end user typically does not modify the code running on the XBee module; rather, the module is configured using Digi's X-CTU serial configuration utility. This makes setting up a basic network using XBee modules extremely easy. However, this simplified interface also makes XBee modules unsuitable for more complex applications that require the flexibility of running custom code. Given this simple configuration interface, development software is not required for the XBee modules.

Atmel's STK500 and STK600 development boards are used in several Cal Poly digital design classes; therefore, the author was already familiar with the AVR Studio 4 development environment. Additionally, the faculty advisor for this project had a significant amount of Atmel wireless development hardware available for use – the RZ200 and RZ201 wireless development kits (Figure 2). Although now discontinued, these kits were still able to provide a decent amount of development hardware for preliminary network testing. Based on extensive research and the aforementioned factors, Atmel's AVR platform with the Bitcloud networking stack was chosen for this project.

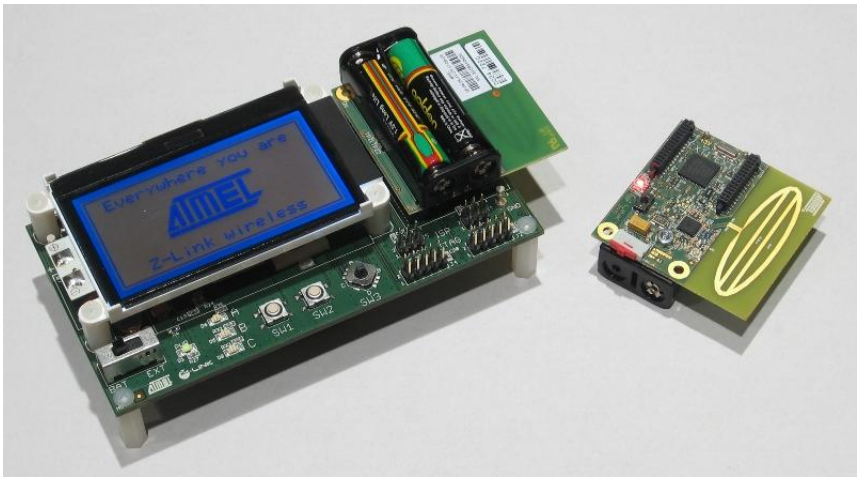


Figure 2: Atmel RZ200 development kit

3.2. Zigbee

As a wireless sensor network designed for actual deployment within existing buildings of a variety of construction types and sizes, the ability of the network to grow via multi-hop mesh networking was imperative. Two of the four wireless network stacks offered by Atmel support multi-hop mesh networking: Bitcloud and the Bitcloud Profile Suite. Both of these stacks implement the Zigbee network specification. Therefore, Zigbee was a clear choice when selecting a wireless networking protocol for this project.

Zigbee, created and maintained by the Zigbee Alliance, outlines a network specification which is based on the IEEE 802.15.4 networking standard. Within the past few years, Zigbee has become widely used for a variety of low-power, low-rate wireless applications, including building automation and control systems, sensor networks, electronic meter reading, and much more. See Figure 3 for a block diagram of a typical Zigbee mesh network. Nodes on a Zigbee

network can take on one of three roles: coordinator, router, or end device. All nodes on the network have a "parent-child" relationship (Atmel Corporation).

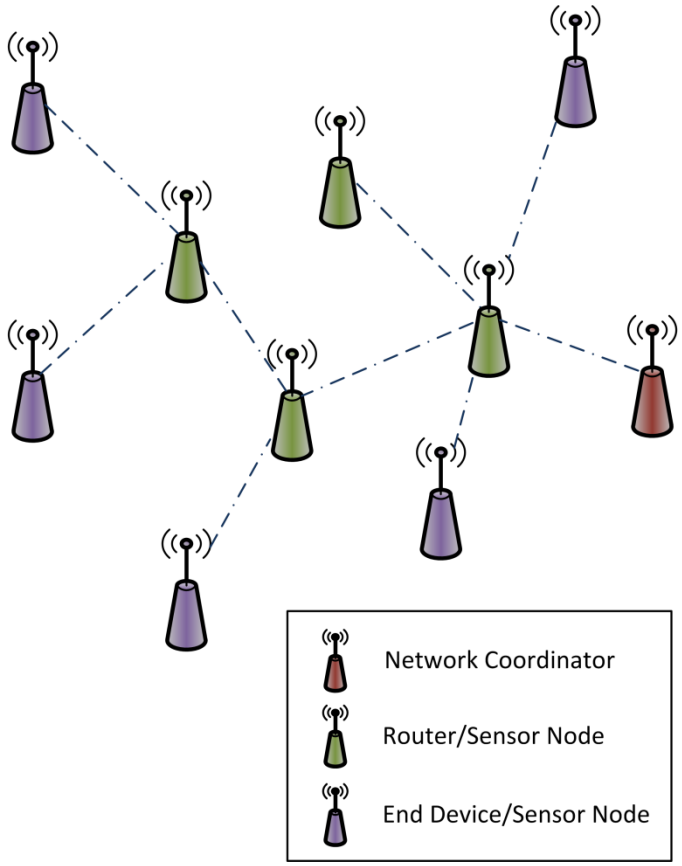


Figure 3: Topology of a typical Zigbee network.

The coordinator forms a Zigbee network. This node holds all configuration and security settings for the network, including the network's channel, identification number, security key, and security level. Since the network coordinator must have a static address of 0x000, a network can only contain one coordinator at a time. When a network is being created, the coordinator is the first parent node. Given the coordinator's critical role in the formation of the network, this node cannot enter a sleep state at any time, and is almost always powered from the wall in an installation.

The router both extends the range of a Zigbee network and acts as a sensor node. Each router holds the same network configuration and security settings as the coordinator, and can act as a parent for new routers and end devices on the network. Additionally, a router can provide all of the functionality of an end device, and perform readings from local sensors.

Given the router's role in network range extension, router nodes are not allowed to sleep and are typically wall powered.

Zigbee end device nodes are "limited functionality" network nodes. An end device is a "limited functionality" network node, and can only act as a child of the coordinator or a router, and cannot act as a parent to any node. Each end device can only send and receive packets that are buffered by its parent. However, as end devices are allowed to enter a low-power sleep state, these nodes consume very little power and are most often used as battery-powered sensor nodes.

3.3. Bitcloud

As this project will not require support for any of the Zigbee Profiles supported by the Bitcloud Profile Suite, the base Bitcloud network stack was selected.

A large portion of Atmel's current microcontroller wireless offerings, including Bitcloud and Zigbit modules, were originally commercial products produced by a Russian company named Meshnetics. Meshnetics specialized in producing hardware and software for Zigbee mesh network sensor and control networks. The company's primary evaluation board, the Meshbean, consisted of a Zigbit module coupled with necessary support hardware and several onboard sensors.

In Feburary 2009, Atmel purchased all Meshnetics intellectual property. At this time, the Zigbit module series was re-branded as an Atmel product line, and the Bitcloud network stack was released as a free product for developers using Atmel microcontrollers and wireless transceivers. The Meshbean evaluation board and all other Meshnetics development kits were effectively discontinued.

Unfortunately, this transfer of intellectual property left the documentation and resources for the Bitcloud stack in a state of confusion. While Atmel has added documentation to Bitcloud regarding the deployment of the stack on newer Atmel hardware, there are still many references to the discontinued Meshbean boards in the sample code and documentation. Therefore, a significant portion of this project consisted of reverse-engineering and generating relevant documentation for Bitcloud.

The Atmel RZ200 and RZ201 evaluation kits shipped with demonstration software running a now defunct Atmel Zigbee stack called Z-Link. Z-Link was discontinued after Atmel's acquisition of the Meshnetics Bitcloud stack. However, since the RCBs used in these kits have almost identical hardware to Atmel's current wireless offerings (the Zigbit modules and ATmega128RFA1 SoC), any development work performed on the RCBs is very easy to port to a more modern platform.

3.4. System-Level Design Overview

After choosing an embedded microcontroller, radio, and network platform, a basic, high-level system block diagram was developed (Figure 4). Aside from minor modifications, this system architecture was preserved throughout the project.

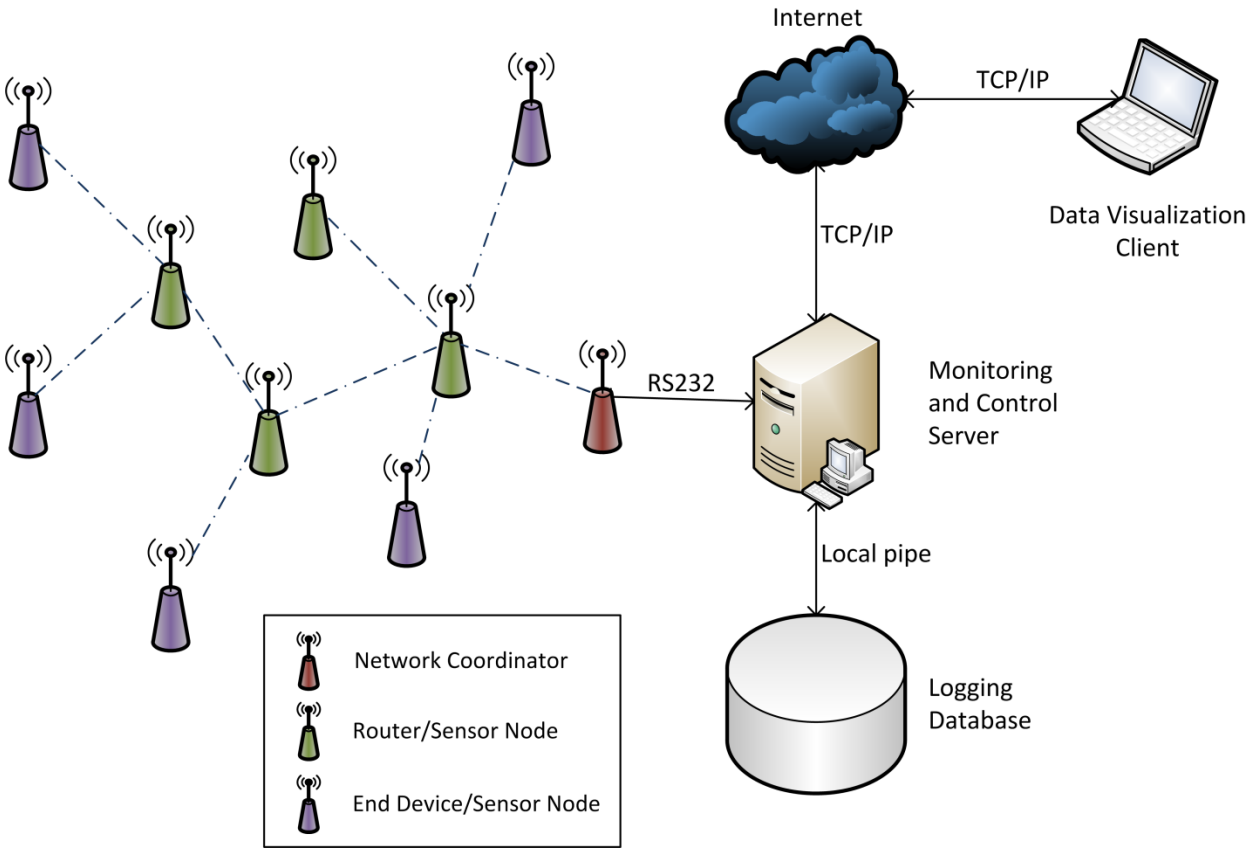


Figure 4: High-level block diagram of proposed wireless sensor network.

As can be seen in this block diagram, this project encompassed the development of numerous pieces of hardware and software. In this document, these various system components will be broken up as follows:

- Sensor node: This flexible device is the heart of the wireless sensor network, and is composed of custom hardware and software. Each sensor node is capable of monitoring both external and internal sensors.
- Coordinator: The coordinator acts as a bridge between the wireless sensor network and the network management and logging server. The coordinator is based around modified Atmel hardware running custom software.
- Network management and logging server: This PC-side application, written in Microsoft Visual C#, manages the addition, removal, and logging for all nodes on the network. Data collected from the network is written to the SQL database.
- SQL database: Running on Microsoft SQL Server 2008 Express, this database holds the settings for all nodes on the network, and all data collected from the network.
- Data visualization application: This PC-side application, also written in Microsoft Visual C#, provides an easy-to-use interface for an end-user to view the node data collected in any local or remote SQL database.

3.5. Sensor Node Hardware

3.5.1. Overall Design

As the fundamental building block of the network, the sensor node hardware and software design occurred first. In an effort to meet the overall system requirements, the following design goals were outlined for the sensor node hardware:

- Non-invasively monitor common building control and automation signals.
- Non-invasively monitor power consumption by building loads.
- Flexibly support the requirements of a wide variety of installations, including various power sources (battery or wall) and modes of operation (router or end device) with minimal effort on the part of the installer.

- Durability to endure the rigors of installation and use in a commercial environment, interfaced with real-world systems.

Since this wireless sensor network was designed specifically for commercial building monitoring applications, one of the major design requirements was that the node hardware support interfacing with the signaling standards commonly used in commercial building control systems. However,there are a large number of signaling levels and protocols present in building control systems; Table 2 presents a sampling of these. To further complicate matters, as this project is designed to interface with existing (and likely out-of-date) controls hardware, the variety of signals is enormous.

Analog	Digital
0-5V	LONWorks
0-10V	BACNet
4-20mA	DALI
Millivolt	Modbus
	DMX
	Wiegand
	Contact Closure
	TCP/IP

Table 2: Typical interfaces used in building control systems.

It would be impossible for any lightweight, low power sensor node to support all of these communication standards – each with very specific hardware and software requirements. The original sensor node design addressed this issue with very generic base node hardware that accepted user-installable input cards. The “mainboard” for each node would hold the microcontroller, radio, RF hardware, power supply, and possibly a temperature sensor and real-time clock. The end user of the system would then connect an appropriate input card to the node to add support for the type of signal from the building control system to be monitored. The input card would hold any specialized hardware necessary for interfacing with a given signal, such as analog amplifiers, additional analog-to-digital converters, bus transceivers, and more. This input card-based node architecture would make the system future-proof, as new input cards could be designed to support newly released communications standards.

Additionally, the end user could easily reconfigure the input capabilities of a system's nodes during the re-deployment of a temporarily installed system.

Power is a major concern when designing a wireless sensor node hardware that is to run from a set of batteries for many months, if not several years. Power consumption and battery lifetime is dependent on a number of factors, including the node sleep/wake patterns, code efficiency, number of external sensors connected, power consumption of the microcontroller and radio, and efficiency of the onboard power supply hardware. All of the Atmel microcontroller and radio offerings are designed to optimally operate from a supply of ~2.7V to ~3.6V – making two series-connected 1.5V AA or AAA cells the perfect power source for the battery-powered nodes on the network.

The easiest solution to powering the sensor node from a nominally 3V battery pack (2xAA or 2xAAA) is directly using the battery pack's output to supply a +3V rail for the sensor node. In fact, Atmel's own RCB development hardware is powered directly from a battery pack comprised of two series-connected AAA batteries, giving the device a nominal rail voltage of +3V. Over the lifetime of the alkaline cells used in the battery pack, the output voltage of the pack will range from approximately 3.2V with brand-new batteries, to approximately 2.5V with fully discharged batteries.

However, although the Atmel RCBs are directly powered from a 2xAAA battery pack, the Meshnetics Meshbean development boards were designed with a more complex power system. While each Meshbean was also powered by a nominally +3V battery pack, the power rail for each board was supplied by a +3.3V switching boost regulator, as shown in Figure 5 (Atmel Corporation). While the design decisions behind this more complex power system are not known, it is likely that the improved battery life offered by a switching regulator was a major factor.

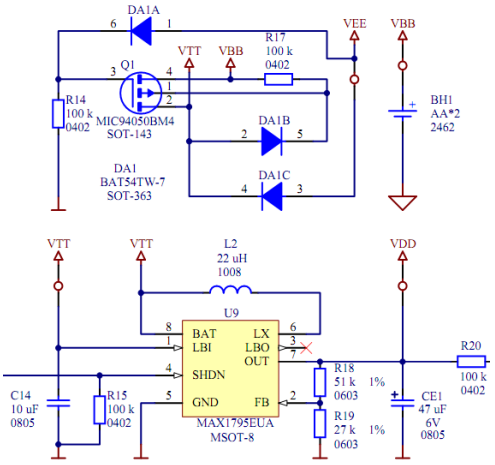


Figure 5: Excerpt from Meshnetics Meshbean schematic showing power supply circuitry.

The Duracell alkaline battery data sheet provides extensive information, shown here in Figure 6 regarding the expected service lifetime of a typical alkaline cell under three types of discharge – constant resistance, constant current and constant power (Proctor and Gamble). As can be seen, a constant resistance discharge offers the shortest usable cell service life, while a constant power discharge offers the longest usable cell service life. Based on the typical Atmel ATmega datasheet, an AVR microcontroller represents a constant resistance load – as supply voltage increases, the controller's current draw also increases, and vice versa. A quick survey of the datasheets for several other parts which were considered for use in the final design revealed that many digital chips represent a constant resistance load. Powering a sensor node (a largely resistive load) directly from unregulated alkaline cells would result in the shortest possible service life from the cells.

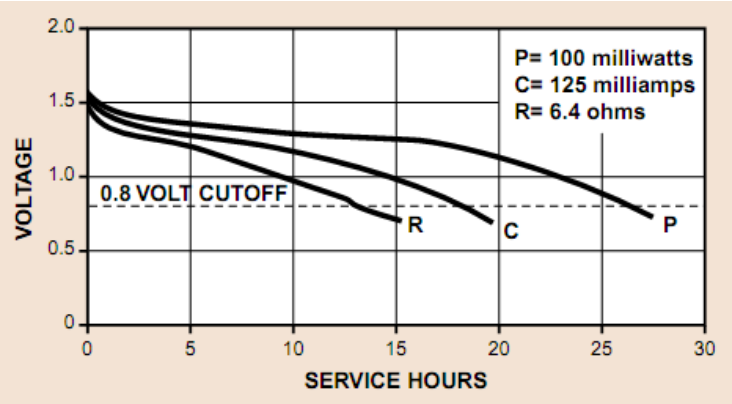


Figure 6: Excerpt from Duracell Alkaline datasheet showing cell performance under different modes of discharge.

Therefore, several switching regulators were considered for use in the power section of the sensor node hardware design. A switching regulator represents a constant power load – as supply voltage drops, current draw increases, and vice versa. As outlined in the Duracell datasheet, a constant power load is capable of getting the longest possible service life out of an alkaline cell. After significant research, the Texas Instruments TPS61025 3.3V 96% efficient boost regulator was selected. In addition to having a very high efficiency, this part also features a built in low battery monitoring comparator. A node power supply design based around this part could provide a constant +3.3V power rail from any source between 0.9V and 6.5V.

The ATmega128RFA1 is Atmel's newest offering for wireless microcontroller-based embedded systems. This part combines many of the features and hardware peripherals of the ATmega1281 microcontroller with the AT86RF230 2.4GHz wireless radio in one compact system-on-chip (SoC) package. The ATMega128RFA1 offers the same performance as this discrete microcontroller and radio pair, but with a small board footprint and simplified signal routing considerations. Therefore, this part was selected as the microcontroller of choice in the original node hardware design.

If each node's sensor data is time-stamped upon arrival at the network coordinator, timing errors may be introduced to the system based on the latency of the multi-hop network and possibility of data packets being buffered for an extended period of time in installations with poor RF connectivity. Therefore, an I2C real-time clock was incorporated into the original node hardware design to provide each node with the ability to accurately timestamp all sensor data at the point of collection.

A system-level block diagram of the original conceptual design for the sensor node hardware can be found in Figure 7. Note that all of the major components outlined above are included in this design, including provisions for input cards, the switching regulator, ATmega128RFA1, and real-time clock.

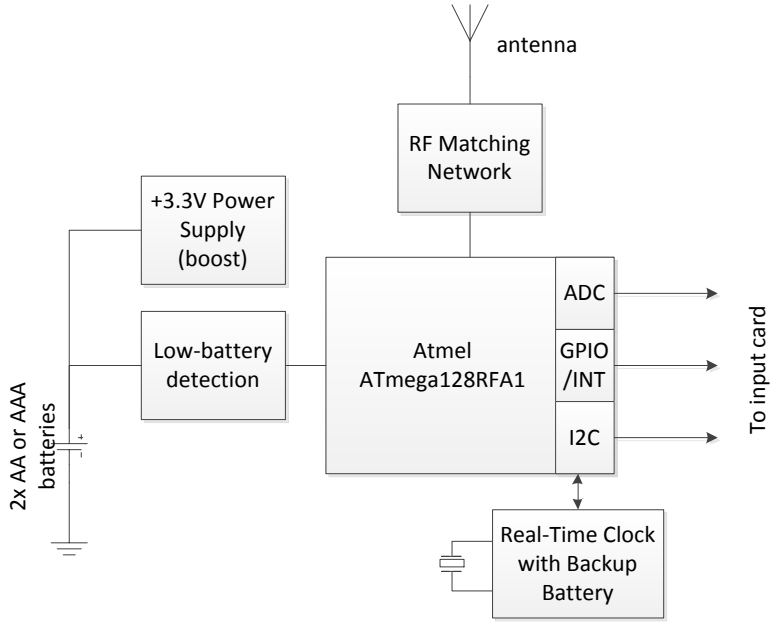


Figure 7: Block diagram of original sensor node hardware design.

This first sensor node hardware design represented an ideal, best-case architecture. Given the time and resource constraints of this project, however, several aspects of this original design were simplified during the design review process.

While the input card system was a novel solution to providing the sensor nodes with a wide range of user-selectable input capabilities, this architecture would also require significant design work. At the hardware level, input card circuitry would need to be designed for each signaling protocol for which the system might support. This design would require significant research into a wide variety of potentially complex protocols and signaling standards. Additionally, a connector interface between the node mainboard and input card would need to be design and specified. Such design work could easily add several weeks, if not months, to the project timeline. Therefore, it was decided to design all nodes with the same input circuitry permanently placed on a single sensor node board. This input circuitry would support several of the most common and universal building control signaling standards. Of those listed in Table 2, 0-5V and contact closure were selected as the two signals that would supported on each sensor node board. The 0-5V voltage interface is commonly used for a wide variety of analog sensors, including pressure, temperature, strain, vibration, and position transducers. The contact closure interface is widely used for monitoring any switching event generated from

motion sensors, magnetic reed switches, glass-break detectors, limit switches, thermostatic controls, and more.

One of the fundamental system design requirements was the ability to measure the power consumption of an electrical load. While it would be possible to connect an external amplified current transducer to a sensor node's 0-5V analog input, the sensor node input collection outlined above would not be able to directly measure current. Therefore, an optional current transformer amplifier was added to the base design. Due to the added cost and complexity of a node with this amplifier on-board, the amplifier circuitry would only be populated as necessary.

During this design review phase, an on-board temperature sensor was also added to the sensor node hardware design. In particular, it was decided that a digital I2C-interface temperature sensor would be the most appropriate temperature sensor style for this project. Many analog temperature sensors are less accurate and have higher power consumption than their digital counterparts.

While the incorporation of a boost regulator-based power subsystem would have likely resulted in excellent node battery life, this portion of the node hardware design was simplified for a number of reasons. Like many boost regulators available today on the market, the TPS621025 has an excellent efficiency rating at 96%. However, this value only applies to a regulator operating under a medium load – when operated with very light or very heavy loads, the efficiency of the device suffers greatly. In a typical installation, the sensor node will spend the majority of the time in a very low-power sleep (~500uA max). Therefore, it is very likely that the boost regulator would actually exhibit very poor efficiency when driving a sleeping sensor node.

Due to this uncertainty regarding the actual benefits of a sensor node in powering a wireless sensor node, the boost regulator-based power subsystem was scrapped in favor of a much simpler direct power configuration (similar to that used in the Atmel RCBs). In this new design, the two AAA batteries would directly supply unregulated power to the node's +3V power rail. A +3.3V voltage regulator would also be included on each node to allow an external power supply to be used.

Unfortunately, while the Atmel ATmega128RFA1 was in full production at the time of the project design phase, this part was unavailable from any of the major electronics parts distributors, including Mouser, Digi-Key, and Newark. Additionally, even though the ATmega128RFA1 is a highly-integrated SoC with very minimal external component requirements, the implementation of this chip in the sensor node board design would require the careful selection and layout of several RF components, including a matching network and some kind of printed or chip antennas. Due to these challenges, the ATmega128RFA1 was replaced with an Atmel Zigbit module.

Each Atmel Zigbit module is approximately the size of a postage stamp, and contains an Atmel microcontroller, wireless radio, and necessary RF support circuitry under a metal shield can. Zigbit modules are available with a number of RF configurations, including two onboard chip antennas, an amplified U.FL external antenna connector, an un-amplified U.FL external antenna connector, and balanced, un-amplified RF output. The ATZB-24-A2 Zigbit module chosen for the sensor node hardware design, shown in Figure 8, consists of an Atmel ATmega1281 microcontroller, Atmel AT86RF230 2.4GHz wireless transceiver, and two on-board chip antennas (Atmel Corporation). This module is functionally equivalent to the ATmega128RFA1 SoC, but reduces the amount of RF design work that would need to performed, as the wireless radio and antennas are all located on the module.

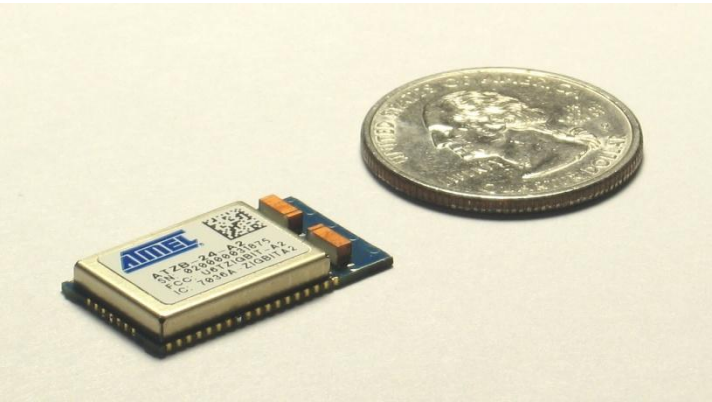


Figure 8: Atmel ATZB-24-A2 Zigbit module with dual chip antennas.

Based on preliminary research done on the deployment of Zigbee networks, a simple local configuration and monitoring interface was added to the node hardware design. This interface will allow the system installer to easily configure node settings, such as node mode

(router vs. end device) and sleep time in the field without having to re-program the firmware image of the node. In an effort to keep this interface simple and easy-to-use, two LEDs and a multi-position DIP switch were chosen.

As will be discussed in greater detail in the software development portion of this report, each sensor node requires a unique address on the network. Since the network server will track and store data based on the node address, it is required that each node have a permanent address on the network. Therefore, a 1-wire interface silicon serial number, the Maxim DS2411, was added to the node design. This part holds a unique 48-bit factory laser-engraved serial number. By basing each node's address on the serial number stored on this chip, each node can be guaranteed a unique address on the network.

Based on the above revisions to the overall sensor node hardware design, the subsystem-level block diagram in Figure 9 was generated.

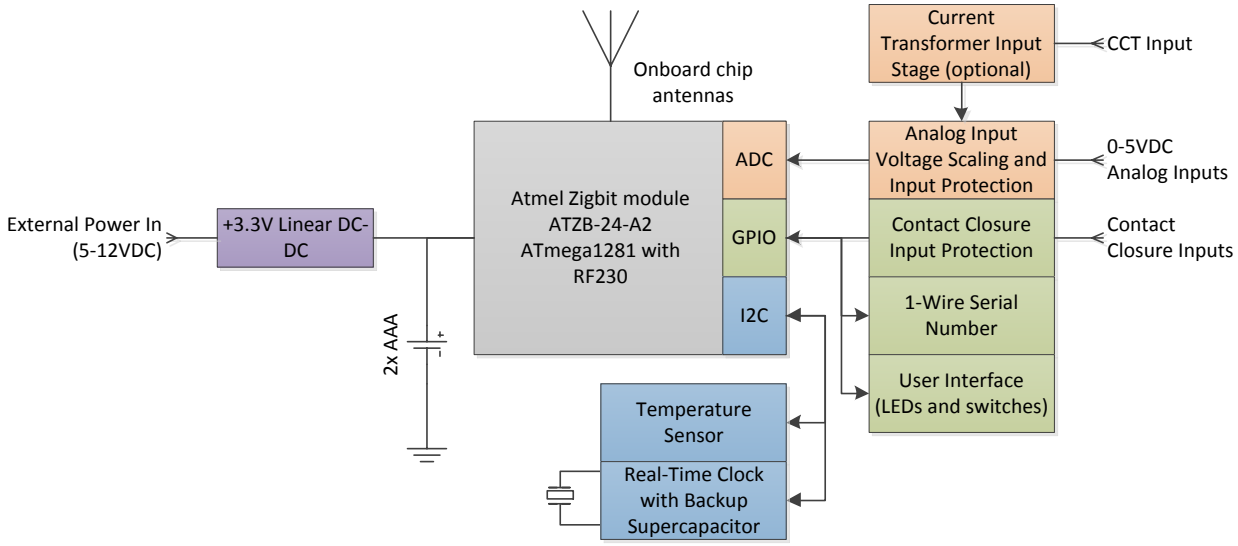


Figure 9: Block diagram of final sensor node hardware design.

3.5.2. Analog Input Stage

As outlined in the overall sensor node hardware design, the analog input stage must be able to accurately measure voltages between 0-5VDC. For the sake of a reduced part count and power consumption, the Zigbit module's on-board analog-to-digital converter will be used to measure the voltage present at the node's analog inputs. In an attempt to improve the accuracy of this ADC, an external, high-accuracy 2.5V voltage reference will be used as the

ADC's reference. As such, the largest magnitude analog signal that can be accurately measured by the ADC is 2.5V.

Originally, the analog input stage was to be buffered with an operational amplifier (op amp) buffer to both provide input protection for the Zigbit module and reduce the effects of any ADC loading on the accuracy of the resistor divider. The Texas Instruments TLV2450 op amp was selected for its ability to run from a single-sided +3V supply rail, low power consumption, and extremely low-current shutdown mode. However, during the design phase, this buffered design was eliminated due to its increased part count, power consumption, and little effect on the accuracy of the analog input stage.

With a 2.5V reference and input range of 0-5V, the input voltage must be passed through a voltage divider. This voltage division was implemented as a simple resistive divider of two 100kΩ resistors. Additionally, since this input stage will only be used to measure the steady-state DC voltage of the input, a 30Hz low-pass filter was implemented as a 0.1uF capacitor in parallel with the bottom resistor of the divider. This high impedance input stage can be safely connected in parallel across existing equipment without the possibility of harmful loading effects.

Since the sensor node was designed for installation in a commercial environment, likely on industrial equipment, input overvoltage and reverse-polarity protection was built into the analog and contact closure inputs and power supply. At the analog input stage, this input protection takes the form of a fast-acting Schottky clamping diode from the input line to VDD. During overvoltage events on the input, this diode should "dump" any voltage over VDD onto the node's positive supply rail. This design was based around material in a Cirrus Logic application note regarding input protection for ADC devices.

3.5.3. Current Transformer Amplifier

One of the major goals of this project was to measure the power consumption of various large electrical loads within a building. In the typical commercial building, HVAC, lighting, and IT/office loads represent the largest energy consumers. Power is defined as the voltage across a given load times the current drawn by that load; therefore, most power measurements require these values.

While HVAC systems range in configuration and power requirements, this project will be designed with the standard electric cooling, gas or electric heat packaged rooftop unit (RTU) in mind. These HVAC units are very common in small- to medium-sized commercial installations, including retail, office, financial, and medical buildings. Depending on the size of the unit and power available at the site, each RTU is usually powered from either a 208V or 480V 3-phase line, and draws between 10-40A. Most commercial building lighting is powered from a single-phase 120V or 277V 20A line, while almost all office and IT equipment loads are run from a single-phase 120V 20A line.

Since this system is designed for easy installation by the end-user without modifications to existing equipment wiring, any "invasive" method of power measurement would be unacceptable. Non-contact forms of AC current measurement are fairly accurate and very commonly used in commercial and industrial load monitoring. Unfortunately, it is very difficult, if not impossible, to implement an accurate non-contact AC voltage measurement system. Therefore, it was decided that all power calculations performed by this network will use the nominal AC line voltage (defined by the system installer) multiplied times the actual, measured AC line current.

One of the most common in-line current measurement techniques is the use of a simple current shunt. A current shunt is a precision, low-resistance load rated for high currents. Based on Ohm's Law, the known resistance of the shunt, and the measured voltage drop across the shunt, the current through the shunt can be found. However, as an in-line current measurement device, the installation of a current shunt requires cutting power to the load. Additionally, the shunt offers no isolation between the measurement circuitry and the high-voltage load; in fact, the measurement circuitry is often directly connected the load's supply! For these reasons, any in-line current measurement techniques were ruled out for this project.

Two of the most predominant non-contact current measurement techniques are the current transformer (CT) and hall-effect sensor. A hall effect-based current transducer measures the magnetic field surrounding a current-carrying wire to determine the current within the line. A current transformer often takes the form of a "clamp" which is placed around a single supply line to the load under measurement. This supply line acts as the primary of the

transformer, while the coil within the current transformer is the secondary. Based on the turns ratio of the current transformer, the AC current flowing in the primary line induces a proportional AC current within the secondary coil. By loading the secondary coil with a known "burden" resistance, a measureable AC voltage can be produced.

Since the system is designed for installation on an existing AC line, the current transformer must be of the "split core" variety. A split core current transformer can open for installation around a wire without the need to cut or disconnect the line. The CR Magnetics 3110-3000 current transformer was chosen for this project due to its split core feature, very high current capacity of 75A, and fairly low cost.

Therefore, a current transformer input stage simply needs to be able to accurately measure the magnitude of an AC input voltage between approximately 0-3VAC. Two AC voltage measurement techniques were considered for this stage: an averaging precision full-wave rectifier and an RMS-to-DC IC. An averaging precision full-wave rectifier design would likely consist of several op amps and diodes with an averaging capacitor. Given the 0.7V "on" voltage of the typical silicon diode, the design of such an input stage must be carefully performed to insure good accuracy even at low-voltage inputs. An RMS-to-DC converter IC is a single-chip solution that outputs a DC voltage that is proportional to the root mean square (or statistical mean) of the magnitude of an AC input voltage.

The design of an averaging rectifier for this sensor node would have been fairly challenging, as the node can only provide a +3V, single-sided supply rail. Many designs for such a rectifier require a double-sided supply, and many op amps require supply rails of at least +/-5V for proper operation. If the design of an averaging precision full-wave rectifier had been attempted, the same Texas Instruments TLV2450 that was selected for the analog input stage voltage buffer would have been used for its low-voltage, single-supply capabilities.

Given the timing constraints of this project, an RMS-to-DC converter IC was chosen as the basis for the current transformer input stage. The Linear Technology LTC1966 was selected as the RMS-to-DC converter as it was the cheapest RMS-to-DC part available on Digikey, at a little over $5 per part in single quantities. Other alternatives on Digikey ranged from $8 to almost $100! Clearly, the RMS-to-DC converter router is not the most economical option; even

the “low cost” LTC1966 is likely more expensive than the combined component costs of a discrete averaging precision rectifier. However, the use of an RMS-to-DC converter offers several key design advantages. Since the LTC1966 is rated for operation from a single-sided supply rail between 2.7V and 5.5V, the part can be easily powered from the node’s single 3V positive supply rail (Linear Technology Corporation). Design using this part is very straightforward – an AC voltage measurement stage can be implemented with just the RMS-to-DC IC and a handful of support components. Most importantly, however, the accuracy of the input stage designed around the LTC1966 is almost guaranteed to be very good – especially when measuring dirty or noisy waveforms.

Per the LTC1966 datasheet, the part can only measure AC input waveforms with a maximum differential magnitude of 1V. Therefore, the burden resistance for the current transformer must be chosen such that the transformer’s output when measuring the maximum measurable current is no more than 1V. The schematic of the typical current transformer application circuit is shown in Figure 10.

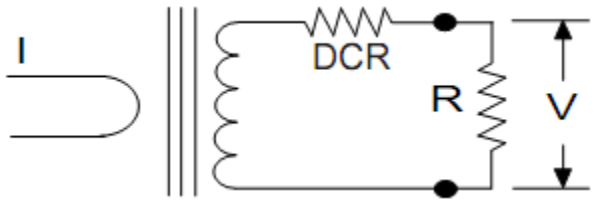


Figure 10: Typical current transformer application circuit

The CR3110’s datasheet provides the following equations regarding the relationship between measured current, burden resistance, and output voltage.

$$V = \frac{I * R}{T_e}$$

$$V_L = Vmax - \frac{I * DCR}{T_e}$$

For best linearity R should be chosen such that $0.8V_L > V$. For the CR3110-3000, T_e = 3100 and DCR = 515Ω. Since the sensor node may be used to monitor the current consumption

circuits of various sizes, burden resistance values were calculated for maximum currents of 15A, 20A, and 30A. An appropriate burden resistance would be installed on the node based on the circuit to be monitored. Table 3, below, shows the calculated resistances, and clearly indicates that the $0.8V_L > V$ criterion for best linearity is easily met with all of these resistor values.

Primary Current (A)	Vout (V)	R burden (Ω)	0.8VL (V)
15	1	206.67	10.00645
20	1	155.00	9.341935
30	1	103.33	8.012903

Table 3: Calculated burden resistances for CR3110

In addition to LTSpice simulations, a small current transformer prototype was constructed to verify the proper operation of this input stage. This prototype, shown in Figure 11 with the CR3110-3000 current transformer, helped verify that the current transformer input stage would work well in practice. Tests on the prototype board revealed that the current transformer design was sound.

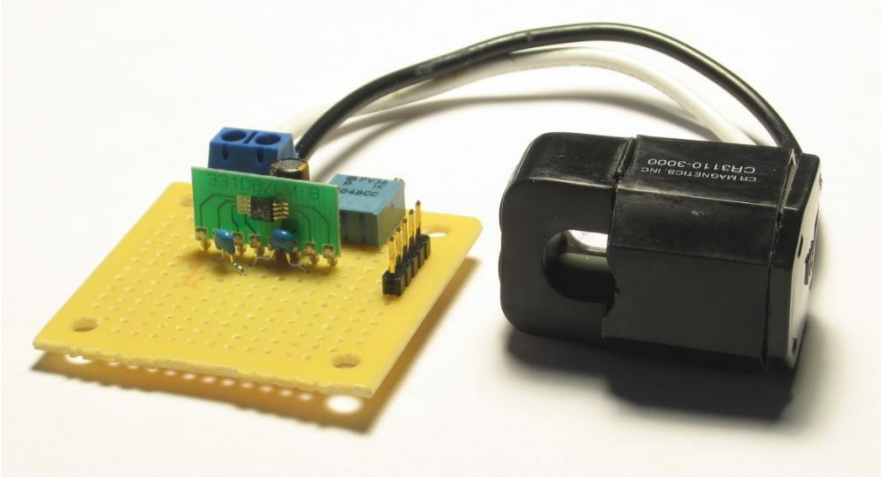


Figure 11: Prototype LTC1966-based current transformer amplifier with CR3110 current transformer.

3.5.4. Contact Closure Input

Each contact closure input channel simply consists of an external IRQ GPIO line of the Zigbit module with added overvoltage input protection. The GPIO line's internal pull-up resistor is used to keep the line in a normally "high" state. A contact closure event between the GPIO line and ground pulls this line low, causing an interrupt on the microcontroller. The input

protection scheme on the contact closure inputs is identical to that employed on the analog voltage inputs; a Schottky diode clamps any excessive input voltage to the node's 3V rail.

3.5.5. Power

With the removal of the switching boost regulator, the design of the power subsystem was quite simple. There are two power connections to the node board – an input for an unregulated, +3V battery pack, and a regulated input for any DC voltage between 5-24V. The LP2950-3.3 voltage regulator was selected for its respectable output current rating of 100mA, low dropout voltage of 40mV, and extremely low quiescent current of 75uA. This regulator was implemented with a ceramic input capacitor and tantalum output capacitor as required by standard linear voltage regulator design practice and the LP2950 datasheet. A SPDT slide switch between these power sources and the rest of the board controls power to the node. As the final component of the node's overvoltage input protection scheme, a +3.9V Zener diode was placed between the +3V rail of the node and ground. This diode was to clamp supply rail to a safe voltage in the event of a transient spike caused by an input overvoltage.

Unless otherwise noted, either a 1uF or 0.1uF bypass capacitor was located on each IC on the node board. Special filtering precautions were taken with the Zigbit module, and both a 10mH inductor and 1uF capacitor were used on the positive supply rail to this part, as outlined in the Zigbit datasheet.

Since the positive supply rail of the node follows the battery pack voltage during battery-based operation, the voltage of the positive supply rail can be measured to determine the current charge state of the battery pack. As previously mentioned, the voltage range of a 2xAAA battery pack can range from approximately 3.2V with brand-new alkaline batteries, to 2.5V with fully discharged batteries. The current voltage of the positive supply rail is measured by the Zigbit module's ADC_INPUT_0 channel through a ÷3 low-pass voltage divider.

3.5.6. Digital Peripherals

Texas Instruments offers a very large range of analog and digital temperature sensors suitable for a variety of applications. From their product line, the TMP102 and TMP175 I2C Celsius temperature sensors were chosen for further evaluation. The TMP175 has a

measurement accuracy of 1.5°C, active current consumption of 100uA, and conversion time of 220mS (Texas Instruments Incorporated). The TMP102 is a much newer and clearly superior part; it has a measurement accuracy of 0.5°C, active current consumption of 15uA, and conversion time of 26mS. However, upon receiving samples of each part, it was discovered that the TMP102's SOT563 package is incredibly small – 1.7mm x 1.3mm! Since this project is to be hand-soldered, this part was rejected due to its tiny size, and the older, SOIC8-packaged TMP175 was selected.

The final node hardware design calls for the implementation of two digital bus protocols – I2C and 1-wire. Additionally, the local user interface utilizes several GPIO lines to drive LEDs and read from the DIP switch.

The I2C bus on each node consists of an I2C master device (the Zigbit module) and two I2C slave devices (the TMP175 temperature sensor and BQ32000 real-time clock). Per standard I2C practice, both the bi-directional serial data (SDA) and serial clock (SCK) lines were pulled up to VDD via 4.7kΩ pull-up resistors.

In addition to the I2C-mandated SDA and SCK lines, the two slave devices also have active-low "alarm" or "alert" lines. On the TMP175, the alert line can be used in conjunction with programmable temperature setpoints to indicate that the sensor's temperature has passed one of these setpoints. On the BQ32000, the alarm line can be used to indicate when a programmable alarm has fired on the real-time clock. These two alert lines were tied together and to a 4.7kΩ pull-up resistor to VDD, and can be optionally connected to the Zigbit module's external IRQ7. This gives the option of having alarm/alert events cause an external interrupt in the code.

The 1-wire bus on each node consists of a 1-wire master device (the Zigbit module) and one 1-wire slave device (the DS2411 silicon serial number). This bus was implemented according to the 1-wire standard, with a 4.7kΩ pull-up resistor to VDD.

The local user interface component of the sensor node hardware design consists of two LEDs and a DIP switch. Two surface mount 0603 LEDs – one red, one green – are connected to GPIO lines 0 and 1 through 560Ω current-limiting resistors. A six-position surface mount DIP-style switch was included to provide an interface for the system installer to configure various

node options without having to re-compile the node software. This DIP switch is connected to GPIO lines 2-7 in an active low configuration.

3.5.7. Manufacturing

From early on in the design phase of the project, it was decided that a custom printed circuit board (PCB) would be designed and fabricated for the sensor node hardware. In addition to giving the project a more professional appearance, the use of a PCB makes the sensor nodes much more durable during testing and deployment. Furthermore, many of the specified components are only available in surface mount packages, making a PCB a requirement for any prototyping and debugging.

The first step of the sensor node manufacturing design consisted of the selection of all physical parts of the sensor node, including case, connectors, and electronic components. All parts selection occurred on Digikey. Since several of these nodes were to be constructed, cost was a major concern during the entire parts selection process. From Digikey's large case offerings, the Hammond 1551KFLBK plastic enclosure was chosen as it was the least expensive case available that could reasonably fit the Zigbit module, a terminal block, and other node components. The PCB was designed to fit this case.

Due to its excellent and easy-to-use feature set, Diptrace was chosen as the design platform for the sensor node hardware. The first step of the PCB design process started with the input of all portions of the sensor node hardware design into Diptrace's schematic capture program. The results of this work can be found in Appendix D: Sensor Node Hardware Schematics. During the schematic capture process, zero ohm resistors were used extensively in the design to provide a means for configuring optional portions of the node board hardware, such as the current transformer input stage and IRQ7 alert line. After this capture process was complete, the schematic was moved into Diptrace's PCB layout program. Due to the PCB's small size and tight component placement, the entire PCB was laid out by hand. 0603 size surface mount components were used throughout the design for a minimum component footprint while still being large enough for hand soldering. A complete parts list for each sensor node board is included in Appendix C. The PCB layout process took approximately one week; the final PCB design can be found in Appendix E.

3.6. Sensor Node Software

3.6.1. Overall Design

The sensor node software was developed concurrently with the sensor node hardware, and had the following design goals:

- Utilize Bitcloud network stack to support all basic functions of Zigbee node, including network form/join, leave, and transmit/receive packets
- Support monitoring of all sensor input channels – voltage, contact closure, and temperature
- Support monitoring of sensor node "health" – battery status and RF link quality

3.6.2. Introduction to Bitcloud

Atmel's Bitcloud network stack provides the services necessary to implement a fully Zigbee Pro-compliant network. The stack is organized per the recommendations of the IEEE 802.15.4 standard and Zigbee Pro specifications. As can be seen in Figure 12, the Bitcloud stack consists of a set of core network services and shared, low-level services. Several of the key components of Bitcloud that were utilized in the development of this project have been outlined below.

- Zigbee Device Object (ZDO) layer – Provides an interface between the user application, Zigbee device profile, and application support sublayer. The ZDO provides a public interface to the user application to control the network discovery, security, and management features of the stack.
- Application Support Sublayer (APS) – Provides a set of services which encapsulate network (NWK) layer functionality for the ZDO and user application.
- Configuration Server (CS) – Also known as the ConfigServer, this component of Bitcloud manages the reading and writing of most publically-accessible stack settings. This project used the CS API extensively to modify Bitcloud's network formation and addressing behaviors.
- Hardware Abstraction Layer (HAL) – Contains a common interface for accessing microcontroller hardware peripherals, including the ADC, USART, and TWI, regardless of the actual microcontroller platform on which the stack is running on. Despite poor

documentation, the hardware drivers provided by the HAL greatly shortened the sensor node software development time.

- Board Support Package (BSP) – Utilizes HAL drivers to provide a board-specific interface to external hardware peripherals. The BSP included with Bitcloud was specifically written for several proprietary Atmel development boards; therefore, the sensor node software developed for this project specifically did not use any components of the BSP.

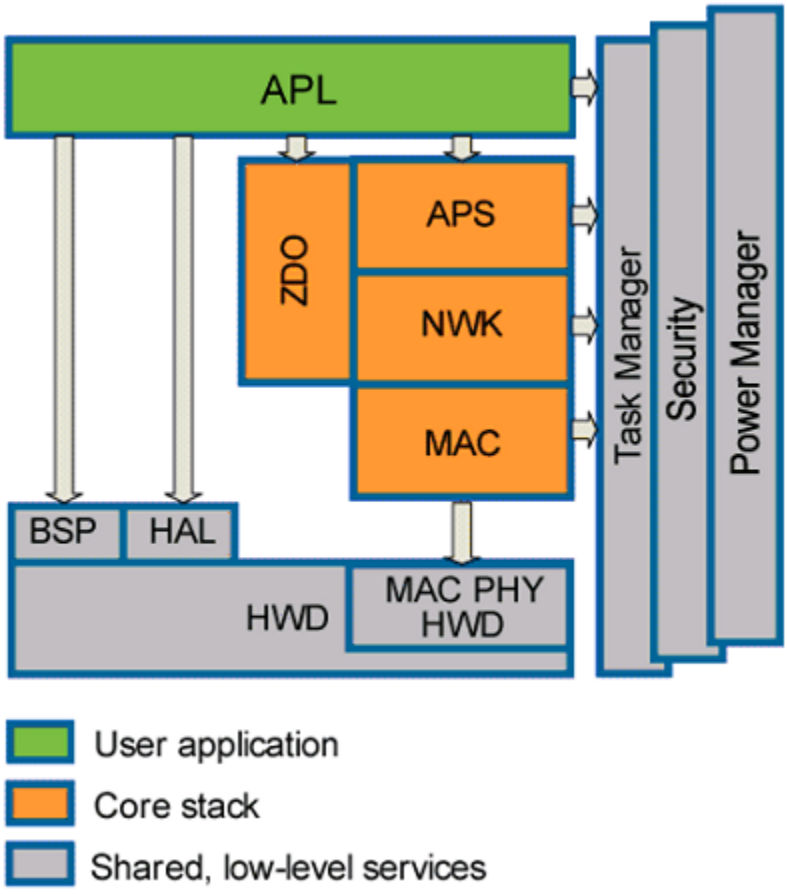


Figure 12: Bitcloud network stack organization

3.6.3. Developing with Bitcloud

As an initial exercise in gaining familiarity with the Bitcloud network stack, Atmel's Zigbee-based WSNDemo wireless sensor network demonstration application was modified to run on the older RCBs. The original Atmel-supplied version of WSNDemo was designed to work almost exclusively on the Meshbean boards, and contained many hardware dependencies in

the code. The processes of getting WSNDemo to properly run on the RCB hardware consisted of modifications to the existing WSNDemo code and the construction of some development hardware.

As with almost any networking standard, all devices on a Zigbee network require a unique 64-bit MAC address, referred to as the UID in Bitcloud and "extended address" in the WSNDemo application. On the RCB platform, each board's ATmega1281 EEPROM held a factory programmed MAC address which was read on runtime by the Z-Link demonstration firmware. On the Meshbean platform, each board contained a Maxim DS2411 1-wire silicon serial number chip, which is factory laser-engraved with a unique 48-bit serial number. The Bitcloud demonstration software on the Meshbean would read this serial number from the DS2411 at runtime, and assign this value to be the board's unique network MAC address. The default build of WSNDemo available from Atmel is still designed for use on the Meshbean boards; as such, the firmware attempts to read from a non-existent DS2411 silicon serial number on boot. The value read from the DS2411 is stored locally as the ConfigServer value CS_UID. WSNDemo reads CS_UID at runtime, and uses it as the node's extended address for network communications.

Since each RCB contained a unique MAC address in EEPROM, these boards do not have a DS2411 on-board. Therefore, this situation left two solutions to giving each RCB a unique MAC address for WSNDemo – Bitcloud could be modified to read the address stored in the RCB's microcontroller EEPROM, or a dedicated firmware image with a hardcoded MAC address could be generated for each RCB. Unfortunately, modifying Bitcloud to read from a MAC address stored in EEPROM could potentially have been very complex. Therefore, it was decided to simply create a firmware image with a unique MAC address for each RCB for this preliminary work with Bitcloud. WSNDemo's Configuration file contains a definition for CS_UID – the default value of 0x0000LL indicates that Bitcloud should attempt to read a unique ID number from an external DS2411, any non-zero value indicates that Bitcloud should use this value as the unique ID number. At this time, it was also decided to add a dedicated DS2411 silicon serial number to the sensor node hardware design, as the Bitcloud stack has existing support for this part.

WSNDemo's original sensor manager and board support package were designed to interface with several Meshbean-specific sensors, none of which are not present on the RCBs. Therefore, the code responsible for reading from these sensors was replaced with code that writes dummy data into the appMessage structure.

Based on the above modifications, five firmware images – one coordinator, two routers, and two end devices – were built for the RCBs with successively numbered, hard-coded UID values and no sensor reading code.

On the Meshbean platform, the WSNDemo network was formed by one Meshbean board that had been configured to act as a coordinator via onboard DIP switches. This coordinator board connected to a Java-based PC application, the WSNDemo visualizer, over an RS232 serial interface. Unfortunately, neither the individual RCB node boards nor the main RCB display board provide easy access to the serial UART interface present on the RCB microcontroller. Therefore, a custom interface board was built on perfboard to allow for easy programming, prototyping, and debugging with an RCB. The photograph of this board connected to an RCB can be found in Figure 13.

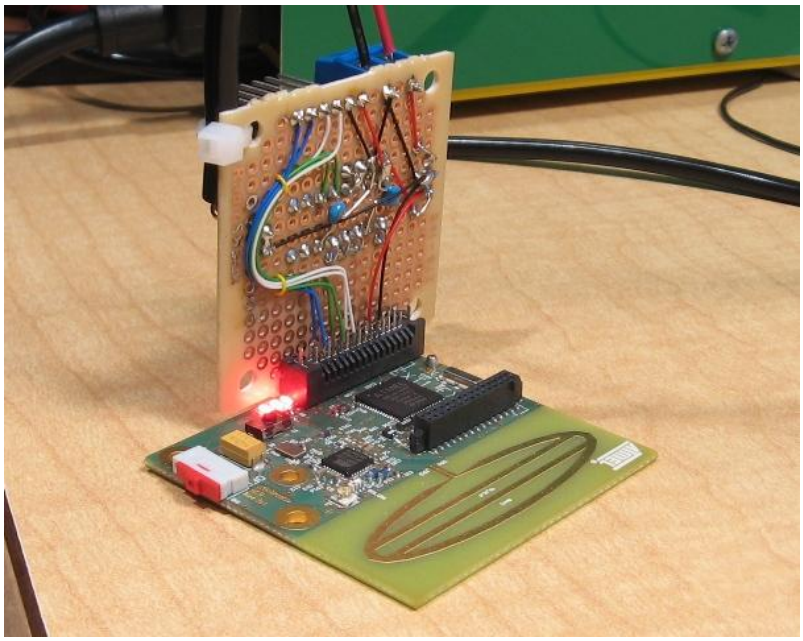


Figure 13: Custom interface board for RCB

This RCB interface board provided an easily-accesible breakout to many of the most-frequently used development pins on the RCB; namely, two external interrupt request pins, two

ADC pins, and the RXD and TXD lines of USART0. The interface board was also designed with a +3.3V regulator and RS232 level shifter onboard to allow for the quick connection between the RCB and a host computer.

Using this board, the RCB programmed with the coordinator image was successfully connected to the Atmel-provided WSNMonitor visualization utility. Using three other RCBs, a simple four-node test network was formed (Figure 14).

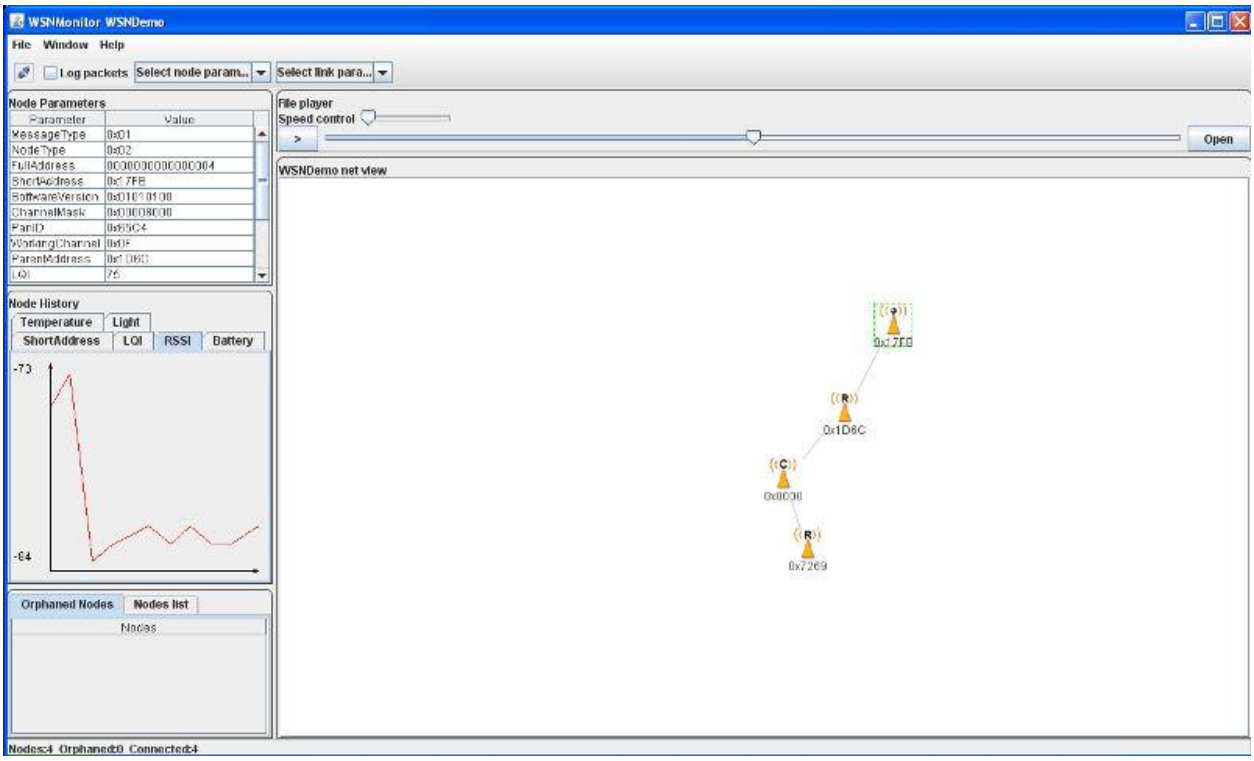


Figure 14: Screenshot of WSNMonitor with four-node test network.

3.6.4. WSNNode Design

Initially, the "user application" component of the sensor node software was to be completely written from scratch. However, the modification of WSNDemo for use on the RCBs revealed that the ground-up development of a Bitcloud user application would be an unnecessary and unreasonably large task given the time frame of this project. Fundamentally, the base functionality of WSNDemo is identical to that required of the sensor node software for this project: WSNDemo forms and joins a network as appropriate and forwards sensor readings down the network to the coordinator. This demo application contains multiple complexly layered state machines which appropriately drive the network stack based on the current

operational mode of the node. A re-written version of this application would simply model much of the existing code structure.

Therefore, it was decided to base the sensor node software, to be called WSNNode, on a heavily modified version of Atmel's WSNDemo application. While WSNDemo contained the source code responsible for node operation as a coordinator, router, or end device all within the same firmware build, WSNNode only contains the source code for node operation as a router or end device. A separate application, WSNCoord, contains the source code for node operation as a coordinator, and will be discussed in the Coordinator design section. Unfortunately, Atmel has provided very little documentation regarding the architecture and design of WSNDemo, meaning that significant reverse-engineering of the code base would be necessary. This reverse-engineering started with a full mapping out of all of the state machines used in the code.

As outlined in the Bitcloud user's guide, the network stack has one of the highest priorities of execution. As such, all user application code must execute within a rather small time frame of 10mSec before suspending to allow network stack operation to continue. Bitcloud implements a "priority queue scheduling" architecture to manage the timing sensitivities inherent in the network stack design. When any operation is completed by the network stack or user application, this "task" is posted to the Bitcloud task manager, which will call the next lower priority task waiting to execute.

Fundamentally, WSNDemo is based around several state machines which advance states based on callbacks performed by the network stack. At the end of each state's execution, the state machine is posted to the task manager to allow program control to return to the network stack. The task manager calls the next state when there is another opening in processor time.

Based on this research, an extensive process of modification and streamlining was performed on the original WSNDemo code to produce the WSNNode application. An outline of the file structure of WSNNode can be found below, along with details regarding the specific modifications made to each source file. When possible, naming conventions established in the existing source code were preserved. The original WSNDemoApp.aps AVR Studio project file

was preserved, along with the very complex Makefiles responsible for building the Bitcloud network stack. The final code for WSNNode can be found in Appendix F.

- WSNNodeApp.c – The main source file for the WSNNode application; it contains the main state machine responsible for basic network operations including forming, joining, and dropping a network connection.
    - As the sensor node hardware will not support the required external EEPROM, all references to the system's over-the-air update capability were removed.
    - All references to the high security mode were removed as this feature is unnecessary for this application and took up a significant amount of code space.
    - All references to the coordinator code were removed.
    - References to structures of type AppMessage_t were updated to reflect changes made in WSNNodeApp.h.
    - The portion of the APP_INITING_STATE that dealt with node mode selection was completely removed and re-written. APP_INITING_STATE now contains the DIP switch read routine, which pulls the installer's defined settings for node mode and sleep time from the DIP switch and stores this data into the appropriate WSNNode variables and data structures. See Figure X for a graphic illustration of the DIP switch settings.
    - CS_NWK_UNIQUE_ADDR is written as true, disabling stochastic addressing.
    - The lower four bytes of CS_UID are read from the ConfigServer in APP_INITING_STATE and stored as the node's short address (CS_NWK_ADDR). Since stochastic network addressing was disabled above, this address will be used as the node's network short address.
    - After the node has successfully connected to a network in APP_STARTING_NETWORK_STATE:APP_NETWORK_STARTING_DONE, external interrupts IRQ6 and IRQ7 are enabled.
- WSNNodeApp.h – The header file for WSNNodeApp.c, this file contains prototypes and definitions for many globally-used functions and data structures.
    - References to all coordinator-specific code were removed.

    - The AppMessage_t data structure was heavily modified to reflect the specific sensor channels and hardware available on the sensor node boards.
    - Removed WAITING_VISUALIZER_STATE from type DeviceState_t, as this state was only relevant for node boards with an on-board LCD.
    - Removed all references to Meshbean-specific BSP.
- WSNEndDevice.c – This source file contains the end device-specific state machine.
    - References to structures of type AppMessage_t were updated to reflect changes made in WSNNodeApp.h.
    - Cleaned up all states dealing with sensor reading.
    - Added code support for disabling external interrupts during application packet transmissions.
    - Removed state WAITING_VISUALIZER_STATE per changes in WSNNodeApp.h.
- WSNRouter.c – This source file contains the router-specific state machine.
    - References to structures of type AppMessage_t were updated to reflect changes made in WSNNodeApp.h.
    - Cleaned up all states dealing with sensor reading.
    - Added code support for disabling external interrupts during application packet transmissions.
    - Modified behavior of HAL_AppTimer_t deviceTimer to reflect re-written initialization code in WSNNodeApp.c.
    - Removed state WAITING_VISUALIZER_STATE per changes in WSNNodeApp.h.
- WSNSensorManager.c – The source file containing the sensor manager, responsible for initializing and reading from all sensor channels. This code is utilized by both the end device and router state machines.
    - Re-wrote entire file to reflect new sensor configuration of sensor node boards. The new code reads from three ADC channels (battery, analog 1, analog 2), two digital GPIO ports (IRQ6 and IRQ7), and the onboard TMP175 temperature sensor. A state diagram for the new behavior of this file can be found in Appendix H.

- WSNVisualizer.c – The source file containing the visualizer, responsible for driving the local LED component of the user interface.
    - Removed all references to the Meshbean-specific BSP, and replaced with direct GPIO commands.
- WSNVisualizer.h – The header file for WSNVisualizer.c.
    - Modified to reflect changes made in WSNVisualizer.c.
- Additionally, all references throughout the application to the source files WSNZclManager.c, WSNUartManager.c, serialInterface.h, and WSNZclManager.h were removed as these files dealt with components of the network stack that would not be utilized for this implementation of the sensor node software.

Since WSNNode is a rather large and complex application, the core components of the code listed below are critical to understanding the basic program flow and execution.

- appTaskHandler(), appEndDeviceTaskHandler(), appRouterTaskHandler(). Task handlers for the main, end device, and router state machines. These functions are called by the Bitcloud task manager as permitted by other stack operations.
- appState. Global variable of enumerated type AppState_t. Holds the current state of the main state machine, but is used globally. Type defined in WSNNode.h.
- appDeviceState. Global variable of enumerated type DeviceState_t. Holds the current state of the device state machine (either the end device or router state machine). Type defined in WSNNode.h.
- event. Local variable of enumerated type AppEvent_t. Holds the next state condition for the device state machine (either the end device or router state machine). Type defined in WSNNode.h.
- appMessage. Global data structure of type AppMessageRequest_t. Contains all data to be sent in the next data packet from node. Appendix G provides a graphic representation of the final packet format for the network.

State diagrams for the three primary state machines in WSNNode can be found in Appendix H. While WSNNode contains a significant amount of original and heavily modified code, the core state machine structure remains almost identical to that found in WSNDemo;

therefore, these state diagrams are also fairly applicable to the original WSNDemo state machines.

3.7. Coordinator

3.7.1. Overview

The coordinator node is responsible for forming the Zigbee network and acting as a bridge between the wireless network and PC server.

3.7.2. Hardware

In an effort to make the system more compact and integrated, it was decided to incorporate the coordinator hardware inside the case of the PC server. However, as the server's case is constructed entirely from metal, it was important that the coordinator be connected to an antenna on the outside of the case to insure a strong RF connection to the rest of the network. This design requirement left the RCB as the best available hardware choice for this application, as each RCB has an optionally enabled U.FL external antenna connector and balun already populated on the board. The sensor node hardware would be acceptable to use as a coordinator within the server case, as each sensor node's Zigbit module only supports the two limited-range chip antennas permanently mounted to the board.

Enabling the U.FL antenna connector on the coordinator RCB required some minor hardware modifications to be made to the board; when shipped from the factory, the printed antenna is enabled, with the U.FL connector electrically disconnected from the RF230. See Appendix I for the schematics of the RCB. Per the recommendations of the Dresden Elektronik technical support staff and the RF230 datasheet, components L1, L2, C18 and C19 were removed from the board, while the pads for AC-coupling capacitors C2 and C3 were populated with 22pF 0402 RF-rated parts. This modification disconnects the printed trace antenna on the RCB, while providing an RF path from the RF230 to the U.FL connector via C2, C3, and the balun.

Figure 15 shows the RCB mounted within the server's case. The U.FL connector was connected to an RP-SMA connector mounted on the case's rear face via a short coaxial pigtail. This RP-SMA connector allows any standard 2.4GHz RP-SMA terminated antenna to be

connected to the internal RCB. Note that the same interface board that was used for preliminary application development on the RCB platform was used, with minor modifications, for this installation inside the server case. When installed in the server, the RCB is powered by the computer's power supply.

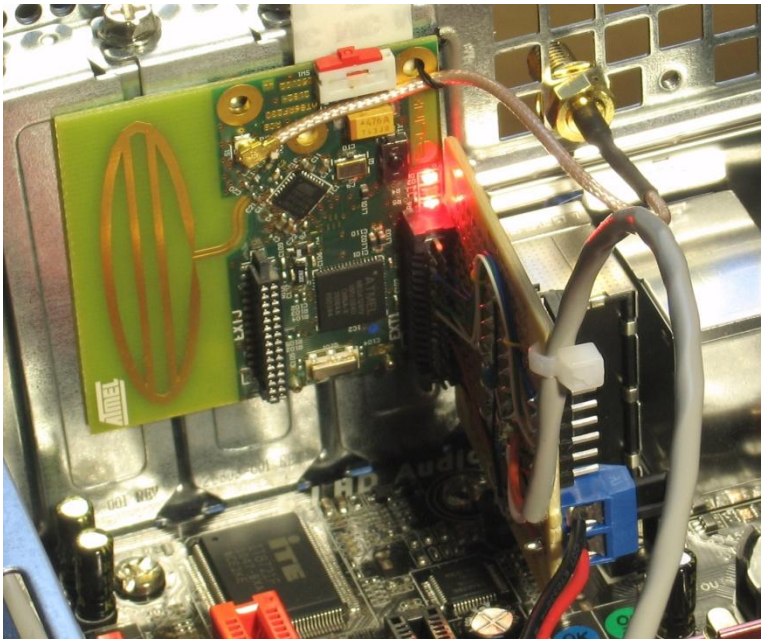


Figure 15: Coordinator RCB mounted in computer case.

3.7.3. Software

Like the sensor node software, the coordinator software package, WSNCoord, is based on a heavily modified version of the original WSNDemo application. The development process of WSNCoord generally followed that of WSNNode. An outline of the file structure of WSNNode can be found below, along with details regarding the specific modifications made to each source file. As with WSNNode, WSNCoord preserved the original WSNDemoApp AVR Studio project and the associated Bitcloud Makefiles. The final code for WSNCoord can be found in Appendix J.

- WSNCoordApp.c – The main source file for the WSNCoord application; it contains the main state machine responsible for basic network operations including forming, joining, and dropping a network connection.
    - As the sensor node hardware will not support the required external EEPROM, all references to the system's over-the-air update capability were removed.

    - All references to the high security mode were removed as this feature is unnecessary for this application and took up a significant amount of code space.
    - All references to the end device and router nodes were removed.
    - References to structures of type AppMessage_t were updated to reflect changes made in WSNNodeApp.h.
    - The portion of the APP_INITING_STATE that dealt with node mode selection was hardcoded to select the coordinator node state.
    - CS_NWK_UNIQUE_ADDR is written as false, enabling stochastic addressing.
    - Since the RCB hardware does not contain a DS2411 silicon serial number, the CS_UID value is hardcoded at 0x1LL.
- WSNNodeApp.h – The header file for WSNNodeApp.c, this file contains prototypes and definitions for many globally-used functions and data structures.
    - References to all router- and end device-specific code were removed.
    - The AppMessage_t data structure was heavily modified to reflect the changes made to the AppMessage_t data structure in the WSNNode code.
    - Removed all references to Meshbean-specific BSP.
- WSNCoord.c – The source file containing the coordinator state machine.
    - Modified the behavior of HAL_AppTimer_t deviceTimer to reduce code dependency on the Configuration.txt file.
- WSNSensorManager.c – The source file containing the sensor manager, responsible for initializing and reading from all sensor channels.
    - Since the coordinator node does not have any local sensor channels to read from; as such, the coordinator's sensor manager was modified to return dummy data as a "heartbeat" status signal for the server application.
- WSNUARTManager.c – The source file containing the UART manager, responsible for writing all received data packets to the server via a serial interface.
    - Removed all code pertaining to the original packet serialization scheme with parity checking; replaced with much simpler implementation using sprintf().

- serialInterface.h – The header file for containing build-specific definitions for WSNUARTManager.c.
    - Updated to reflect changes made in WSNUARTManager.c.
- WSNVisualizer.c – The source file containing the visualizer, responsible for driving the local LED component of the user interface.
    - Removed all references to the Meshbean-specific BSP, and replaced with direct GPIO commands.
- WSNVisualizer.h – The header file for WSNVisualizer.c.
    - Modified to reflect changes made in WSNVisualizer.c.

Additionally, all references throughout the application to the source files WSNEndDevice.c, WSNRouter.c, WSNZclManager.c, and WSNZclManager.h were removed as these files dealt with components of the network stack that would not be utilized for this implementation of the coordinator software.

3.8. Monitoring and Logging Server

3.8.1. Overview

The monitoring and logging server is an always-on PC-side application that provides network services during system installation and operation. This application was coded in Visual C# using Microsoft Visual Studio 2010 Ultimate, and had the following design goals:

- Provide an interface for the system installer to use while adding new nodes to the network.
- Parse all data received from the network and record this data into the SQL database.
- Track all nodes for network drops based on interval between received data packets.

The two primary components of this server application are the SQL database connection and hardware serial port interface.

3.8.2. .NET SQL Database Connection

The SQL database connection is accomplished using the .NET framework data provider for SQL server, the System.Data.SqlClient namespace. Database connections established using

the SqlClient namespace require certain information regarding the database to be connected to. This information is stored in the database "connection string." A typical connection string for a shared memory connection to a local database with integrated Windows security is shown below.

```
"Data Source=jpclaptop\\sqlexpress;Initial Catalog=wsnTestData;Integrated
Security=True;User ID=;Password=;Network Library=dbmslpcn"
```

The data source component of the connection string indicates the path to the SQL server instance to connect to, be it a local or remote path. The initial catalog component of the connection string specifies the name of the database on the SQL server to initially connect to. The integrated security flag indicates if the SQL server login authentication should be performed the Windows login credentials of the currently user. Integrated security is most suitable for local database connections or remote database connections on a system that is managed as part of a Windows domain. Since this connection string example is connecting to a local SQL database with integrated security, the User ID and Password fields are empty.

Since all databases have their own connection specifics, none of the information contained in the connection string is hardcoded into the application. Instead, the database connection string is generated using the SqlConnectionStringBuilder at application runtime based on either user-provided information or stored program settings. Once a connection has been established with a SQL server, standard SQL commands can be executed on the currently selected database via the ExecuteNonQuery() and ExecuteReader() methods.

3.8.3. .NET Serial Port Connection

Serial port connections in Visual C# are established using the System.IO.Ports namespace; specifically, the SerialPort class. The SerialPort class can open a connection to any local serial port with a specified baud, parity, number of data bits, and number of stop bits. Various methods provided by SerialPort allow the program to read data from the serial port one line, byte, or character at a time. Of particular importance in this application is the DataReceived event handler, which fires each time a new line of data is received on an open serial connection. The code contained in the DataReceived event handler is responsible for most of the parsing and storage of newly received data packets into the SQL database.

3.8.4. Application Design and Logic

The final design for the monitoring and logging server application is based around the three forms outlined below. The source code for each of these forms can be found in Appendix K.

mainForm.cs provides the primary user interface for the application and runs the core server logic, including all node data parsing and network drop management. The layout of the form as seen in Windows Forms Designer is shown in Figure 16.

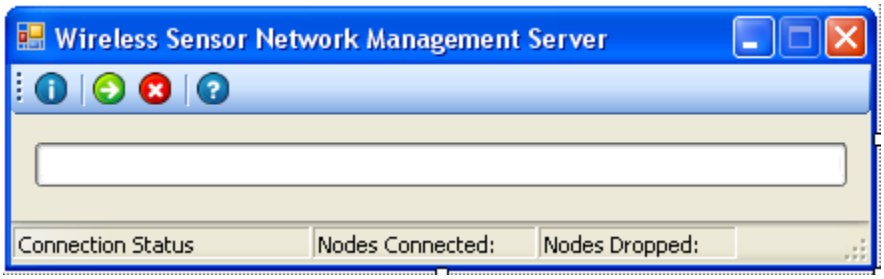


Figure 16: Management and Logging Server mainForm.cs

configForm.cs provides a configuration user interface for selecting connection settings for the SQL database and serial port. Any settings changes made on this form are saved to the config.ini file. The layout of this form as seen in Windows Forms Designer is shown in Figure 17.

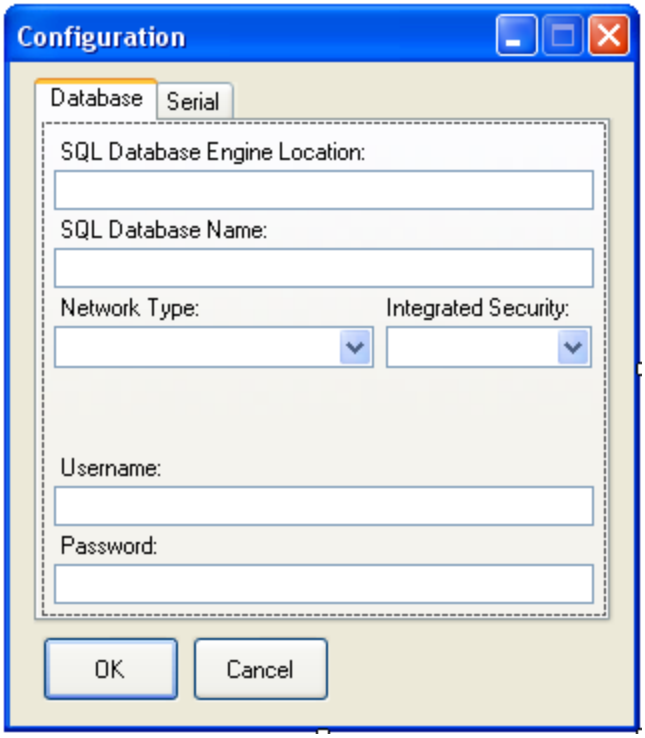


Figure 17: Management and Logging Server configForm.cs

commissioningForm.cs manages the addition of new nodes to the network, and is called when the server logic running on the main form receives data packets from a node not currently contained in the SQL database. The form requests that the user input a name, location, polling interval, and channel settings for the new node. The layout of this form as seen in Windows Forms Designer is shown in Figure 18.

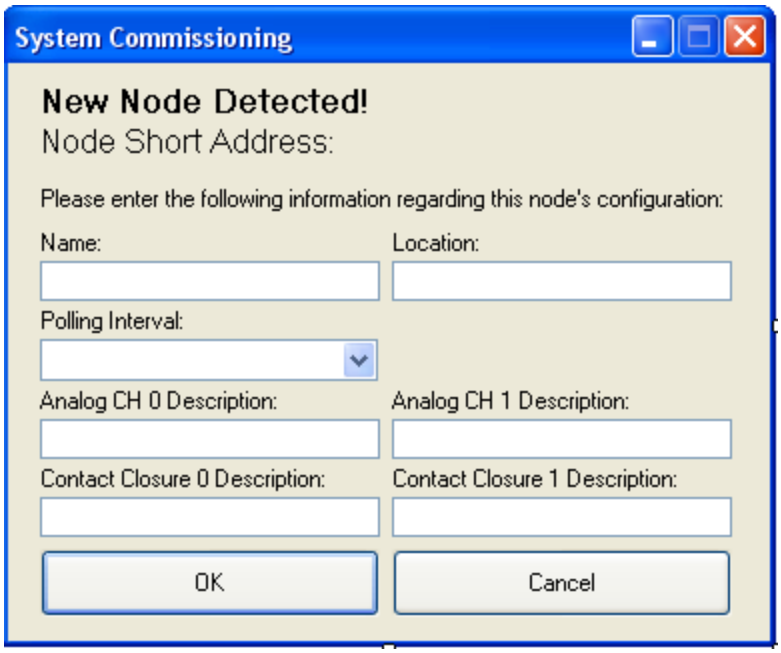


Figure 18: Monitoring and Logging Server commissioningForm.cs

Block diagrams for the major components of the server logic can be found in Appendix K.

3.9. SQL Database

3.9.1. Overview

The data storage component of this project is critical for handling the large volume of data that will be generated by the network. This database had the following two primary design goals:

- Store record of each node that is or has been on the network, including all configuration settings and last seen date and time
- Store all data collected by sensor network

Due to prior experience with SQL databases, SQL was chosen as the database format for this project. Microsoft's free SQL Server 2008 R2 Express edition was chosen as the database server for its excellent feature set and low price point.

3.9.2. Database Schema

The final design of the SQL database sensor network data storage consists of two tables – nodeInfo and nodeData. See Appendix M for the schema of this database. The nodeInfo

table stores configuration information regarding all nodes that have been, or are currently, on the network. A new line is only added to nodeInfo when a new node is added to the network, but existing lines are often read and updated during normal network operation as "known" nodes connect to the network and generate data packets. The nodeData table stores the contents of all data packets generated by the network's nodes. A new line is added to nodeData on the receipt of any data packet. nodeData is only read by the data visualization application.

3.10. Data Visualization Application

3.10.1. Overview

The data visualization application is designed to provide a user-friendly interface for visually interacting with stored network data in the SQL database. Since this application accesses the same common network database, several components of this application are directly based on existing components from the monitoring and logging server codebase.

3.10.2. Application Design

The data visualization application is centered on the visualization form, visForm.cs. The layout of this form can be seen in Figure 19. Each instance of the visForm class is constructed with a specific node's short address passed in as a string parameter. This short address defines the node for which the visForm will provide data visualization for. At a pre-defined interval, the visForm queries the SQL database for the top N rows of nodeData with the appropriate short address. The number of records queried for (N) is defined by the end user via the "Number of records" dropdown box on the visForm. After these records have been returned, they are appropriately scaled and graphed in a Windows Forms Chart control based on the channel selection.

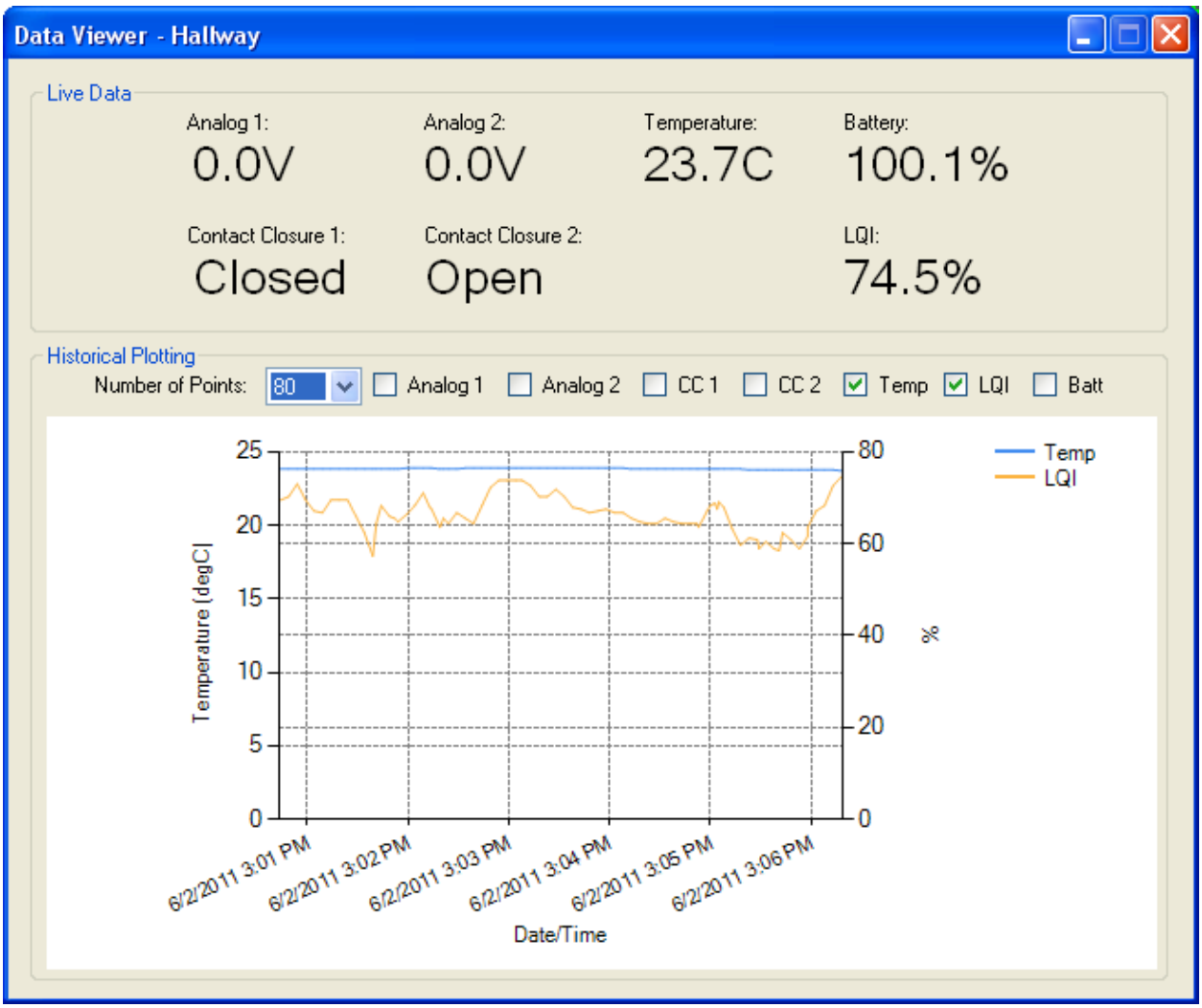


Figure 19: Data Visualization Application visForm.cs

As shown in Figure 20, a map view is included to provide a visual method for selecting which nodes to view data from. Each node graphic's click event opens up a new visForm instance constructed with that particular node's short address.

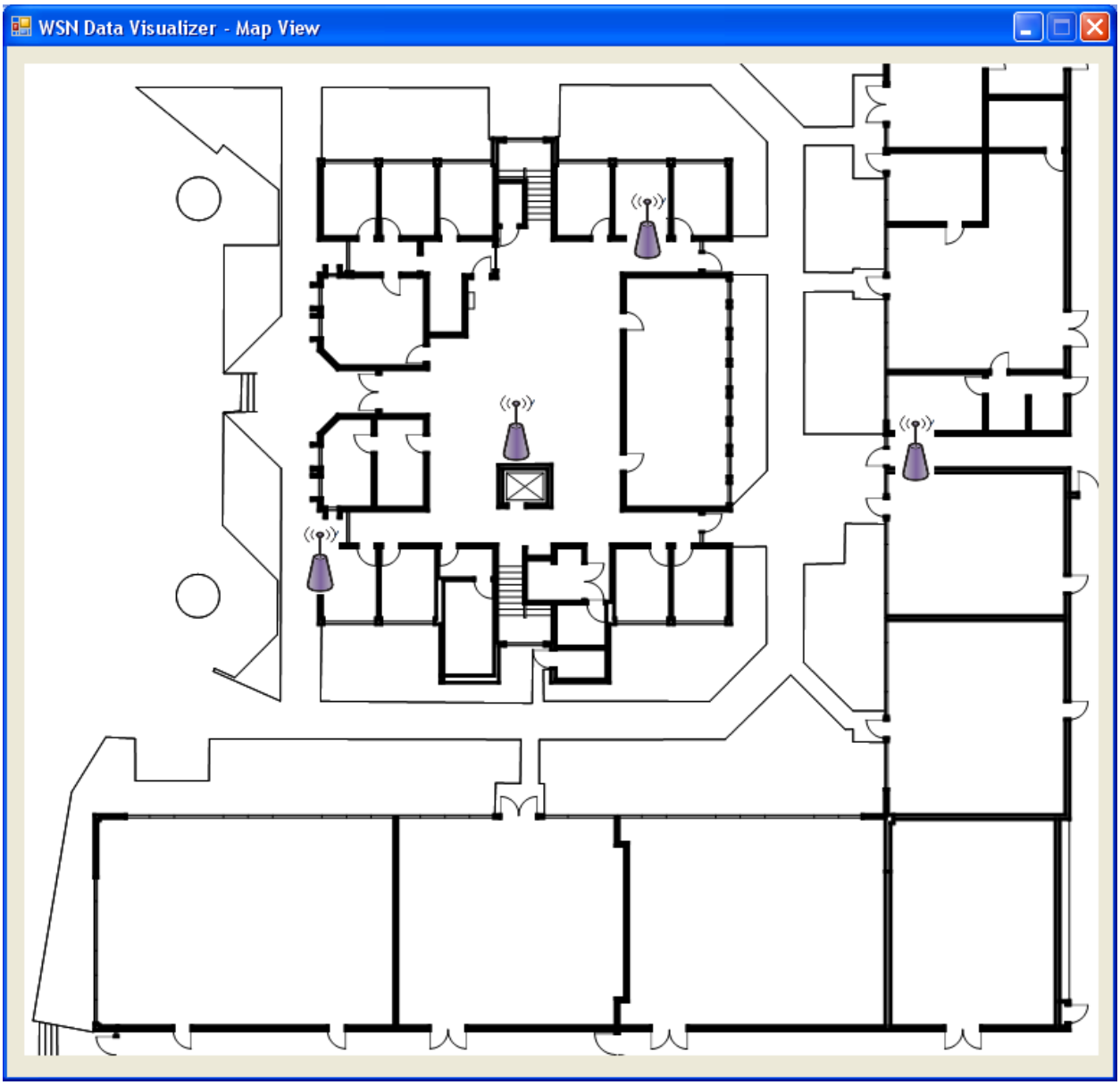


Figure 20: Data Visualization Application mapForm.cs

On program launch, the visualization application prompts the user for the connection settings for the desired SQL database using sqlConfigForm.cs, shown in Figure 21. This form is based on the configForm.cs from the network management and logging server application. Due to the timing constraints of the project, the configuration file component of the server application was not moved over to the visualization application; therefore, the user must input the SQL database connection settings on each launch of the program.

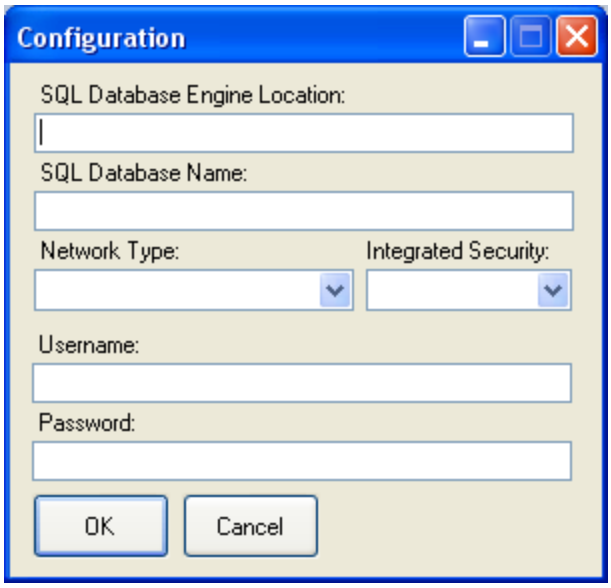


Figure 21: Data Visualization Appliation sqlConfigForm.cs

All code for the aforementioned forms can be found in Appendix N.

4. Development and Construction

4.1. Project Timeline

A Gantt chart was created to provide a rough timeline for the design, build, and testing phases of the project. This Gantt chart, included in Appendix B, was last edited during the first sensor node hardware design phase; therefore, several of the tasks listed on the chart are inaccurate or no longer relevant to the project as it was finally realized. Additionally, the schedule outlined on this chart was very loosely followed, as the project timeline and order of task completion were modified significantly throughout the development process.

4.2. Sensor Node Assembly

As previously mentioned in the Sensor Node Hardware Manufacturing section, the sensor node printed circuit board was designed to fit a Hammond 1551KFLBK case. Therefore, each PCB was laid out to fit within the maximum board size of 74mm x 28mm, as specified in the Hammond datasheet. Having a number of these boards individually fabricated would be very expensive. Instead, this board was duplicated and "panelized" to form a sheet of 14 of these node boards. Advanced Circuits (www.4pcb.com) was able to economically fabricate this sheet of node boards for approximately $100 after shipping and various surcharges.

This sheet of node boards was cut into individual node boards using the "score and snap" technique: the sheet was first deeply scored using an X-Acto knife, then snapped on the edge of a desk. The rough edges of each node board were then cleaned up using a Dremel rotary tool with a fine-grit drum sanding bit.

At a cost of approximately $50 in parts per node, five of these fourteen node boards were populated and tested. The first of these boards was considered a prototype; this board was used to test assembly techniques and discover any errors within the board layout. Several minor issues were found in the board design.

- When the +3.3V voltage regulator, U7, began to smoke during testing, it was discovered that the silkscreen for this part is backwards; the flatted portion of the package should face toward the switch, not away from the switch. This error was due to an incorrect

numbering of the device pins when creating the component's pattern in Diptrace. All subsequent boards have the voltage regulator mounted opposite the silkscreen's orientation.

- The Zigbit module was initially unable to communicate with the DS2411 1-wire silicon serial number chip. This issue was due to a missing 4.7kΩ pull-up resistor on the data line to the DS2411; since this resistor had been omitted on both the schematic and PCB layout, a small 0603 resistor was added directly between the VDD and data pins of the DS2411. This fix was implemented on all node boards.
- The 10-pin male pin header for the Zigbit JTAG programming interface was located too close to the fifth section of the terminal block bank, meaning that a standard female JTAG connector could not physically mate with the node's JTAG header. Since the terminal block had already been soldered into place on the prototype node board when this issue was discovered, this board has a female JTAG connector on a ribbon cable to allow for programming (Figure 22). All subsequently node boards were constructed with the fifth terminal block position left unpopulated, allowing for normal JTAG programming while giving each node only one contact closure input.
- Despite multiple attempts, the Zigbit module could not be programmed using the ISP programming header on the sensor node board. The wiring of the ISP header was verified to be correct; it is thought that the Zigbit module may have some internal wiring preventing the use of the ISP interface. After this issue was discovered on the prototype board, the ISP header was not populated on the remaining boards, and the JTAG interface was used for programming the Zigbit module.
- As discussed in the Node Power Consumption testing section, the power supply Zener diode was discovered to greatly increase node current draw. Therefore, this part was not populated on any of the final node boards.

Aside from the aforementioned issues, the construction of the sensor node boards was fairly straightforward. As shown in Figure 23, each sensor node board was mounted in the Hammond case for protection when installed in a commercial environment.

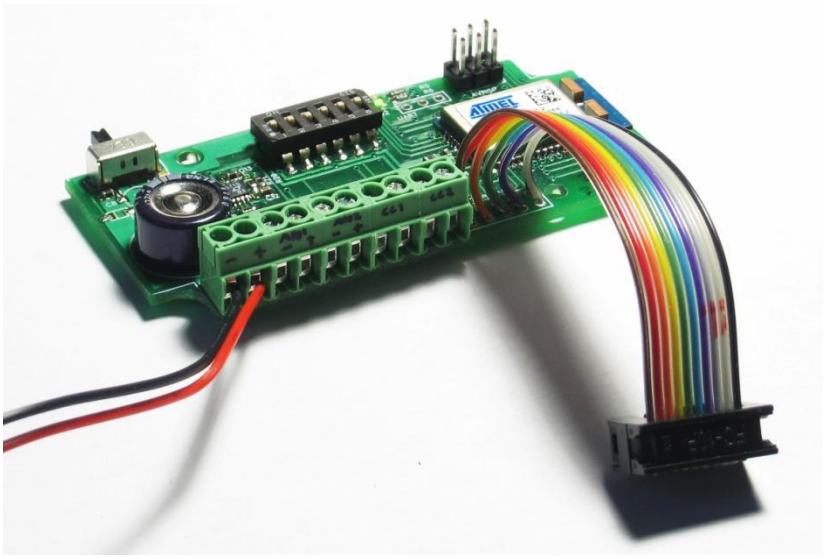


Figure 22: Prototype sensor node board

Figure 23: Final sensor node board installed in enclosure

5. Integration and Testing

5.1. Preliminary Testing

As a full system design project with many components, testing was performed on the individual components and various combinations of components to insure proper system integration. Extensive testing was performed on all portions of the system throughout the design and build phase to insure proper system operation; the details of selected tests have been included below.

5.2. Node Measurement Accuracy

As a wireless sensor network for data acquisition, the accuracy of each input stage of each node is of the upmost importance. Therefore, each completed node board was run through a series of input channel accuracy tests.

- Analog Voltage Input Accuracy Tests. Since the analog input stage is responsible for the accurate measurement of the DC component of a bandwidth-limited input signal, a number of DC voltages were selected to test each node's analog input stages with. This testing revealed that there were no problems in any of the nodes' analog input stages; each stage properly measured DC voltages between 0-5V.
- Current Transformer Input Accuracy Tests. This input stage was tested in conjunction with the design current transformer, the CR Magnetics CR3110-3000. Due to cost and time limitations, only one node board was constructed with the current transformer amplifier circuitry populated; therefore, only one current transformer input stage was tested. For this test, the node's current transformer, along with a current clamp multimeter, was installed around a single supply line to a 120VAC resistive load. The current measured by the node was compared to the current measured by the commercial clamp meter. The results of this testing are shown in Table 4.

Measured Current (A)	Actual Current (A)	% Error
5.44	5.47	0.55%
9.16	9.15	0.11%

Table 4: Current transformer input stage test results

- Contact Closure Input Tests. As a steady-state digital input, there are very few tests that can be run on the contact closure input stage. However, each completed node's contact closure input was verified for proper triggering on the rising and falling edges of incoming waveforms.
- Temperature. The accuracy of each node's onboard TMP175 temperature sensor was measured against a K-type thermocouple of decent accuracy and close proximity to the node at room temperature. The results of these accuracy tests can be found in Table 5. Note that even the least accurate node, Node 1, was still within the TMP175's specified ±1.5°C accuracy.

Node Number	Measured Temperature (°C)	Reference Temperature (°C)	% Error
1	22.69	21.3	6.53%
2	22.19	21.3	4.18%
3	22.31	21.3	4.74%
4	20.94	21.3	1.69%
5	20.69	21.3	2.86%

Table 5: Temperature measurement accuracy test results

5.3. Node Power Consumption

Since the sensor node design was performed to have an extremely long battery life, an extensive power consumption study was performed on the final sensor node hardware and software. This study consisted of the generation of current profiles for a typical sensor node operating in the router and end device modes. Each current profile was produced using a DataQ IN-145 4-channel USB data acquisition interface with 10Ω current shunt resistors. A block diagram of the current measurement setup used for these profiles can be found in Figure 24.

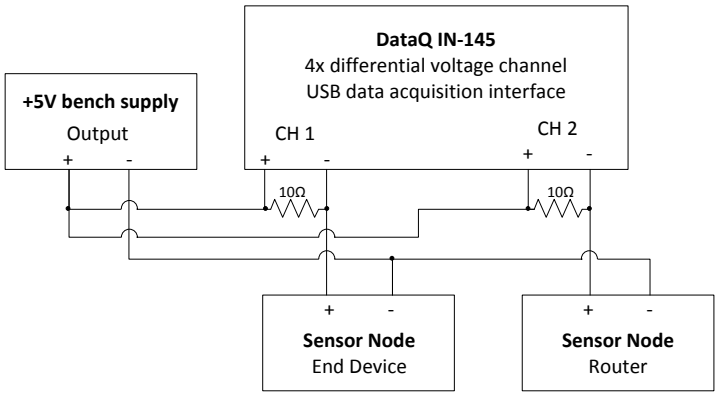


Figure 24: Node power consumption profile test setup

The initial process of generating these current profiles revealed an issue in the hardware design of the sensor node boards. Shortly after setting up and calibrating the data acquisition system, it was noticed that the node board configured as an end device was using over 5mA during the sleep cycle. Based on the specified power efficiency of the Zigbit module in sleep, the node board power consumption should have been in the uA range. After reviewing the hardware schematics and performing various debugging techniques, it was revealed that this excessively high sleep current consumption was caused by leakage in the power supply's protection zener diode, D3. Per the original design, this part was specified as a 3.9V Zener diode. The +3V nominal system positive power supply rail leaves 0.9V between the Zener knee voltage and the node rail voltage; this gap is even smaller when the rail jumps to +3.3V when the node is run from the onboard voltage regulator. This close proximity between the Zener knee and node rail meant that the Zener diode had a significant leakage current of approximately 4.5mA during normal node operation. As this amount of sleep current draw is unacceptable for a low-power sensor node, this Zener diode was removed from all constructed node boards. While this modification did reduce the effectiveness of node hardware's input protection scheme, the current consumption of the node was greatly improved.

After resolving this issue with the sensor node board hardware design, current consumption profiles were successfully captured for nodes operating in the end device and router modes. These profiles can be found in Figure 25.

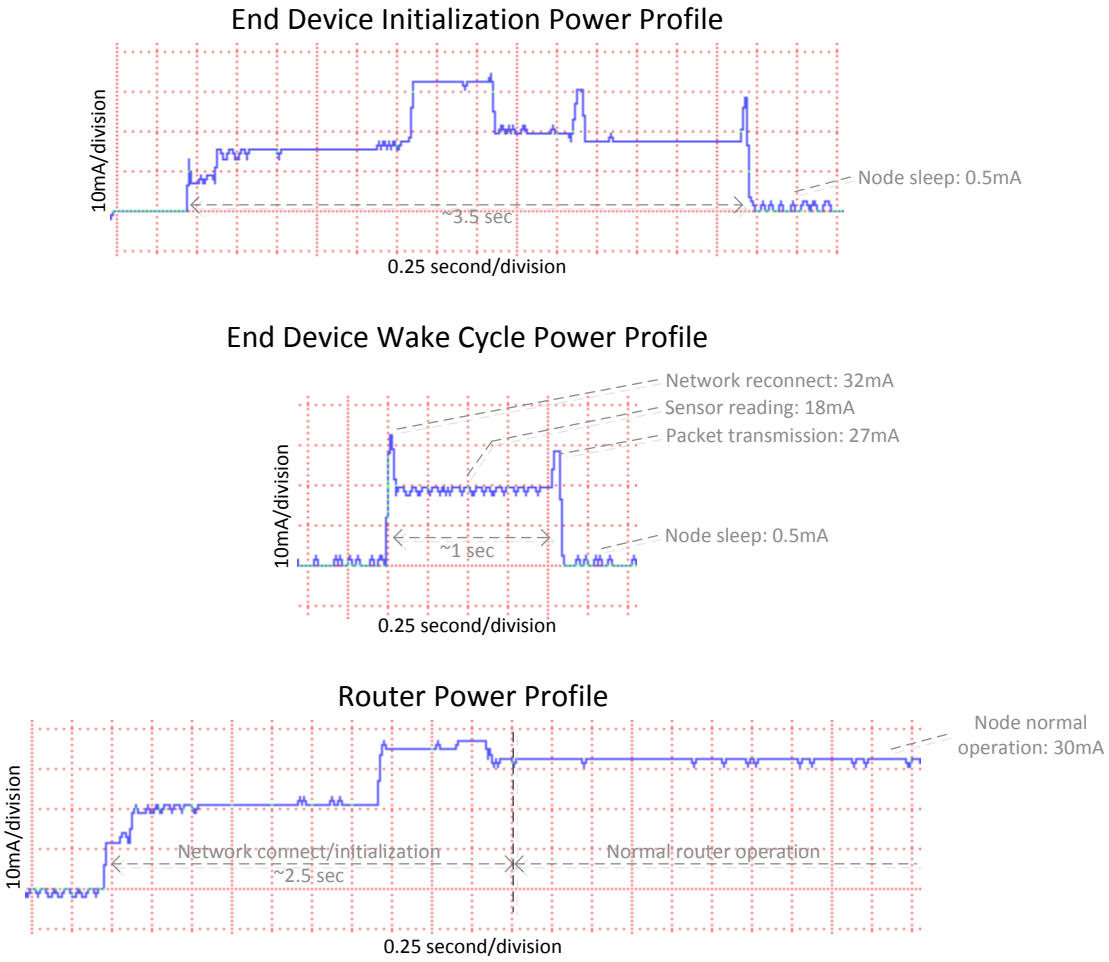


Figure 25: Node power consumption profiles

Both the router and end device initialization and network connection current profiles appear to be almost identical – this process takes approximately 3 seconds, during which the node draws approximately 30mA. After the initial network connection has been established, the router and end device current profiles differ significantly - the router enters a constant-on operation mode that consumes a flat 30mA, while the end device enters a low-power sleep state for a timer-defined interval, drawing a little under 500uA. Each time the sleep timer fires, the end device wakes to perform a sensor reading and data transmission. The end device's wake period lasts approximately 1 second, during which time the node draws about 25mA on average.

Based on these measurements, an end device with a polling interval of 5 minutes should have a theoretical battery life of at least half a year. This battery life could be easily improved

with more effective sensor node software; the current version of the software does not put any of the peripheral hardware to sleep during the node sleep state.

5.4. System Range

An extensive outdoor line-of-sight system range test was performed on the roof of Poly Canyon Village's Canyon Circle parking structure. This space is approximately 400' x 120', and had no vehicles present at the time of testing. Given the remote location of this test site, interference from nearby 2.4GHz 802.11 wireless computer networks was at a minimum. The sensor node-to-coordinator and sensor node-to-sensor node ranges were both tested. A photograph of the test setup can be found in Figure 26, and a graphic of the node test locations and network configuration can be found in Figure 27. Each sensor node's short address and link quality index (LQI) are shown for each test location. LQI is a measure of network connection quality, and ranges between 255 and 0.

Figure 26: System range test coordinator setup

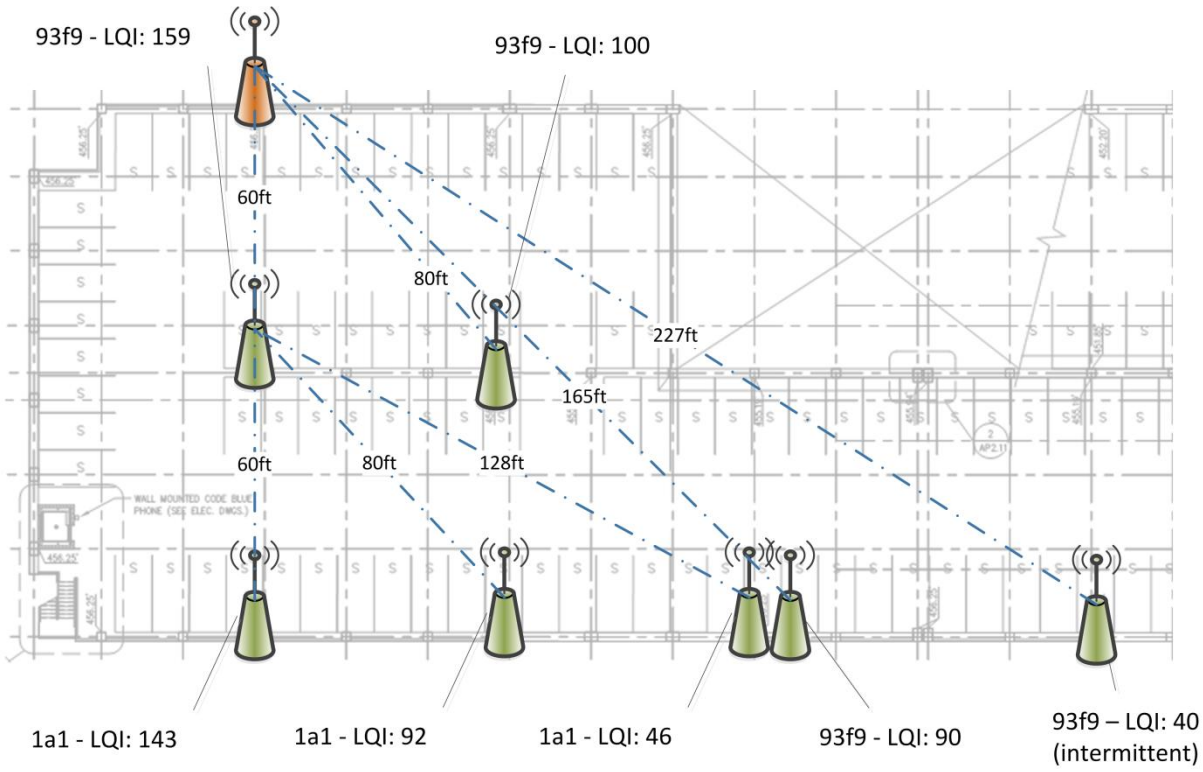


Figure 27: Outdoor network range testing results

The sensor node-to-coordinator connection was found to be significantly stronger than the sensor node-to-sensor node connection. These tests were performed with Node 93f9 connected to only the coordinator; the node was moved to various locations and the average signal strength of the received packets was recorded. This sensor node directly connected to the coordinator was able to successfully and reliably transmit data over 165', line of sight. At a distance of 227', the sensor node's connection to the coordinator became too weak to reliably transmit data packets.

The sensor node-to-sensor node connection was tested using Node 01a1 connecting to Node 93f9 in a multi-hop configuration. With Node 93f9 at a fixed location approximately 60' away from the network coordinator, Node 01a1 was moved to a variety of locations on the roof ranging from 60' to 128' from its parent. Node 01a1 was able to successfully transmit data to 93f9 at distances of up to approximately 128'; past this point, the connection became unreliable.

These differences in network range can be easily explained by the antenna styles used on the coordinator versus the sensor nodes. As previously mentioned in Section 3.7.2,

Coordinator Hardware, the coordinator node installed inside the network server uses an externally-mounted whip antenna, while each sensor node's Zigbit module employs two small PCB-mounted chip antennas. As revealed in these tests, the whip antenna offered significantly better RF performance than the Zigbit's dual chip antennas.

This testing revealed the importance of having a real-time clock on each node. During extended range tests, it was observed that network latency, especially on a multi-hop connection, became quite high. Transmission delays of over 30 seconds were quite common, with longer delays of over 1 minute occurring occasionally. Time stamping each data packet with an accurately set node real-time clock could help mitigate the measurement inaccuracies caused by this high network latency.

5.5. System-Level Testing

After various incremental tests were performed on various components of the system, the entire wireless sensor network was deployed with all final hardware and software for full system testing. Two full system tests were performed: one in a Poly Canyon Village apartment and one at the Electrical Engineering department's Senior Project Expo. During these testing periods, minor glitches were revealed that occurred only when all system components were run together over an extended period of time; these issues were easily fixed with minor code modifications.

The first full system test occurred on the 4th floor of Poly Canyon Village's Estrella apartment building. This test network consisted of two sensor nodes and the network coordinator, installed as shown in Figure 28. The node "Front Door" was installed near the entrance of the apartment, and monitored the indoor air temperature of the apartment using the onboard temperature sensor and activity through the apartment's front door using a capacitive proximity sensor connected to the first contact closure input channel. This node was configured as a router; as such, it was powered from a mains-connected power supply. The node "Hallway" was placed above the ceiling tile in the hallway outside the apartment to monitor the hallway ambient air temperature with its onboard temperature sensor. Since this node was powered by two AAA batteries, it was configured as an end device. Both nodes were configured to have a 5 minute transmission/sleep interval via the onboard DIP switches.

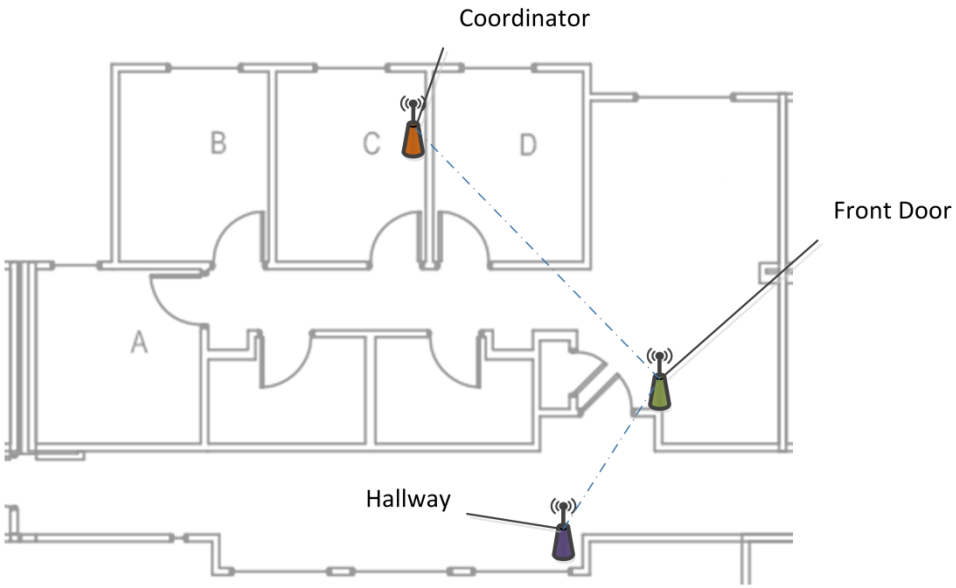


Figure 28: System test installation in Poly Canyon Village

The Poly Canyon Village test was extremely successful; this test demonstrated the full capabilities of the entire wireless sensor network. The multi-hop capabilities of the Bitcloud stack and Zigbee were demonstrated to fully work by the hallway sensor node's ability to connect to the network; this node was outside of the range of the coordinator, and had to connect to the network via the front door sensor node. Additionally, the sensor node software's wake on contact closure event abilities were verified to work on the front door node; this node would wake and transmit a data packet with each opening and closing of the door. This network test was left running and collecting data for the duration of a three-day weekend, during which time the system ran well with very few node disconnects and no fatal crashes.

The second full system test occurred at the Electrical Engineering department's Senior Project Expo event, where the network was installed in and around Building 20A. This installation utilized all four finished sensor nodes, along with the coordinator. The locations of the four nodes can be found in Figure 29. The node "EE Side Door" was installed on the side door of Building 20 with a magnetic reed switch connected to the contact closure input; this node monitored traffic through the door and the ambient air temperature at this location. The

node “Side Hallway” was installed near a side entrance to Building 20A with a motion sensor connected to the contact closure input; this node monitored foot traffic through the hallway and the ambient air temperature at this location. The node “Outdoor” was installed on an outside window of Building 20A with no external sensors, and only monitored the outdoor ambient air temperature. The node “Desk Demo” was a demonstration node with no external sensors connected, and was located near the coordinator on a table in the lobby of Building 20A.

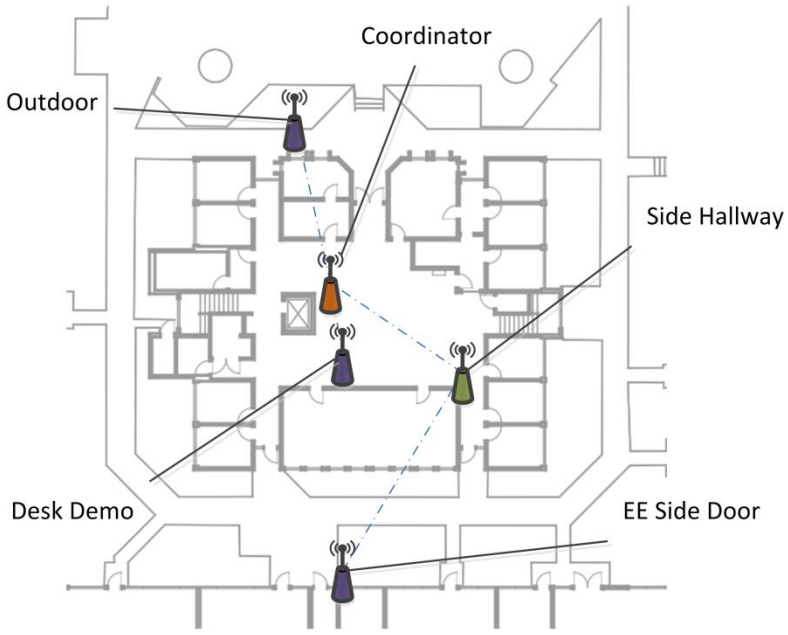


Figure 29: System test installation at Senior Project Expo

Deploying the larger-scale system for the Senior Project Expo event revealed some weaknesses within various portions of the system software. During the system installation and configuration process, it was discovered that the monitoring and logging server’s new node addition process did not work well for large numbers of new nodes coming on the network. Additionally, getting RF coverage throughout Building 20A proved to be more difficult than original thought; it is possible the system was receiving interference from another nearby 2.4GHz wireless network (such as the campus wireless LAN).

6. Conclusion

Overall, this project was a success; all major system design goals were met at the time of project completion. The finished wireless sensor network consists of a number of custom-engineered hardware and software components specifically designed for the energy and environmental monitoring of commercial office building efficiency. The system passed all major tests for accuracy, range, and overall functionality, and will likely be deployed for further testing in an actual commercial environment.

However, although all major system design goals were met, there are many areas for system design improvement and refinement. As previously outlined, several minor errors in the sensor node hardware design were discovered during preliminary testing; these issues can be easily corrected in a future revision of the sensor node boards. The input protection scheme was never fully tested, and should be expanded to include protection for the current transformer input stage. Additionally, improvements can be made to almost all components of the code, including the sensor node firmware, network management and logging server, and data visualization application. Two additions to the sensor node firmware should include improved support for all node peripherals (including real-time clock and peripheral sleep support) and bi-directional network communication.

The skills learned in almost every electrical and computer engineering course were used during the design stage of this project. The construction and testing process was an excellent exercise in time management, system-level design, and real-world engineering. The final work product of this project provides not only a workable sensor network for commercial building monitoring, but an excellent foundation for any number of wireless sensor network applications.

7. Bibliography

Atmel Corporation. "Bitcloud Quick Start Guide." October 2009. MCU Wireless.

—. "Bitcloud User's Guide." October 2009. MCU Wireless.

—. "MCU Wireless." June 2009. Zigbit 2.4GHz Wireless Modules.

—. "Meshbean A2 Schematic." June 2007. Meshnetics / Development Tools.

Cirrus Logic, Inc. "ADC Input Buffer and Protection Techniques." February 1998. Crystal Semiconductor Products Division.

Dresden Elektronik. "Radio Controller Board RCB230 V3.2 Datasheet." August 2009.

Efficiency Partnership. "Best Practice Guide - Commercial Office Buildings." 1999. Flex Your Power.

Linear Technology Corporation. "LTC1966 Precision Micropower RMS-to-DC Converter." 2001. Signal Conditioning: RMS-DC Conversion.

Maxim Integrated Products Incorporated. "DS2411 Silicon Serial Number with Vcc Input." June 2009. 1-Wire Devices.

Proctor and Gamble. "Alkaline-Manganese Dioxide Technical Bulletin." 2005. Duracell.

Texas Instruments Incorporated. "Low Power Digital Temperature Sensor with SMBus/Two-Wire Interface in SOT563." October 2008. Temperature Sensors & Control ICs.

—. "TMP175 Digital Temperature Sensor with Two-Wire Interface." December 2007. Temperature Sensors & Control ICs.

8. Appendices

Appendix A: Specifications

Power
External power in: 5-24VDC
Battery power: 2x AAA batteries
Radio on, transmitting current consumption: 30mA
Radio off, sleep current consumption: 500uA

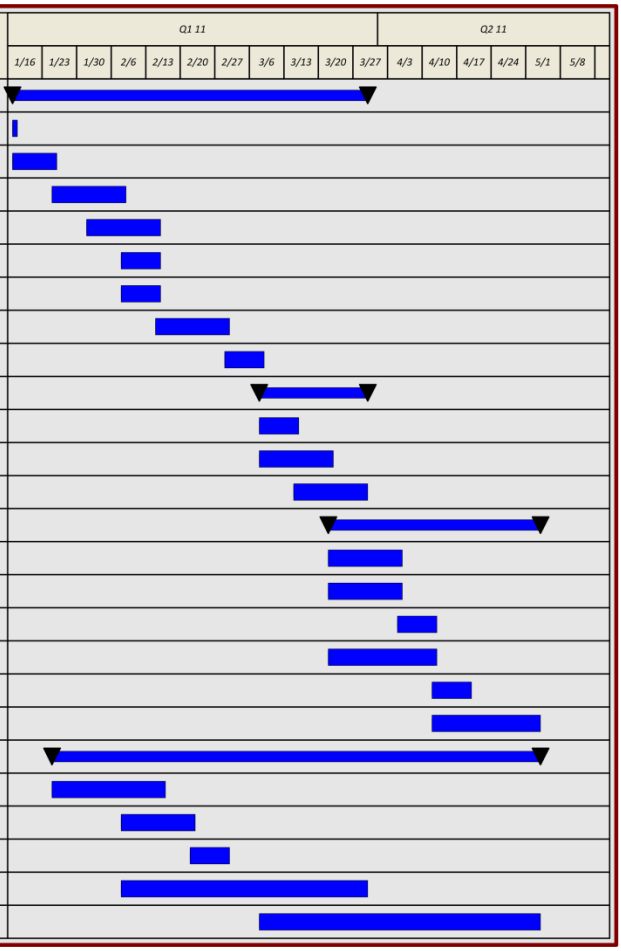
Terminal Block Pinout
1-2: External power in
3-4: Analog 1 (0-5VDC)/Current transformer (selectable during node construction)
5-6: Analog 2 (0-5VDC)
7-8: Contact Closure

User interface
Two LEDs: one red, one green; provide network and node status indication
Configuration DIP switches
Recessed slide power switch

Configuration DIP Switches
1: 1 = router, 0 = end device
2-3: 00 = 10 minute sleep, 01 = 5 minute sleep, 10 = 1 minute sleep, 11 = 5 second sleep
4-6: Reserved for future use

Appendix B: Project Schedule

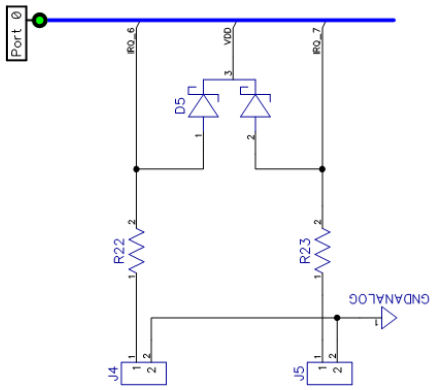
ID	Task Name	Start	Finish	Duration
1	**Sensor Node Software**	**1/17/2011**	**3/29/2011**	**52d**
2	Install/test AVR toolchain	1/17/2011	1/17/2011	1d
3	Build programming/expansion hardware for RCBs	1/17/2011	1/25/2011	7d
4	Reverse-engineer WSNDemo, run on RCBs	1/25/2011	2/8/2011	11d
5	Verify functionality of BSP and HAL	2/1/2011	2/15/2011	11d
6	Establish RS232 communication	2/8/2011	2/15/2011	6d
7	Develop base BitCloud application	2/8/2011	2/15/2011	6d
8	Two-node network formation	2/15/2011	3/1/2011	11d
9	Multi-node network formation	3/1/2011	3/8/2011	6d
10	**Develop sensor application**	**3/8/2011**	**3/29/2011**	**16d**
11	Application-layer packet protocol	3/8/2011	3/15/2011	6d
12	Coordinator/PC-access point software	3/8/2011	3/22/2011	11d
13	Finalize sensor node application	3/15/2011	3/29/2011	11d
14	**Sensor Node Hardware**	**3/22/2011**	**5/3/2011**	**31d**
15	Node DC-DC power supply	3/22/2011	4/5/2011	11d
16	Node RTC with standby power prototype and test	3/22/2011	4/5/2011	11d
17	BSP RTC driver development	4/5/2011	4/12/2011	6d
18	Input stage design	3/22/2011	4/12/2011	16d
19	Input stage prototype and test	4/12/2011	4/19/2011	6d
20	Layout/fabricate sensor node daughtercards	4/12/2011	5/3/2011	16d
21	**Server-Side Application**	**1/25/2011**	**5/3/2011**	**71d**
22	RS232 with .NET	1/25/2011	2/16/2011	17d
23	Databasing with .NET	2/8/2011	2/22/2011	11d
24	Database Structure development	2/22/2011	3/1/2011	6d
25	Node Configuration Utility	2/8/2011	3/29/2011	36d
26	Monitoring and Datalogging Utility	3/8/2011	5/3/2011	41d

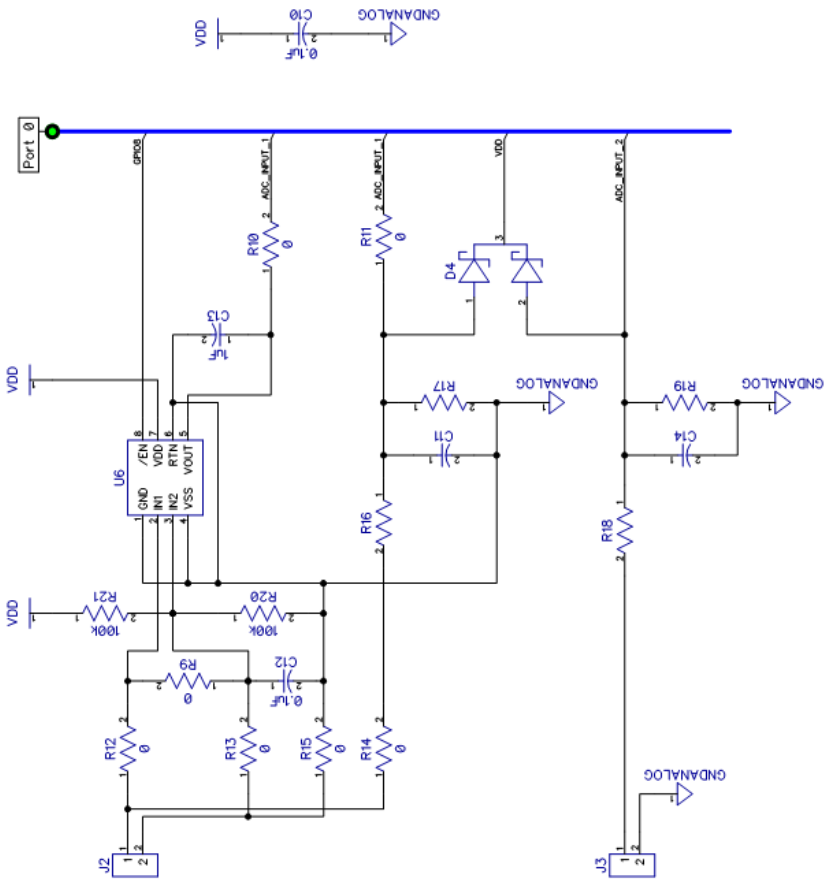


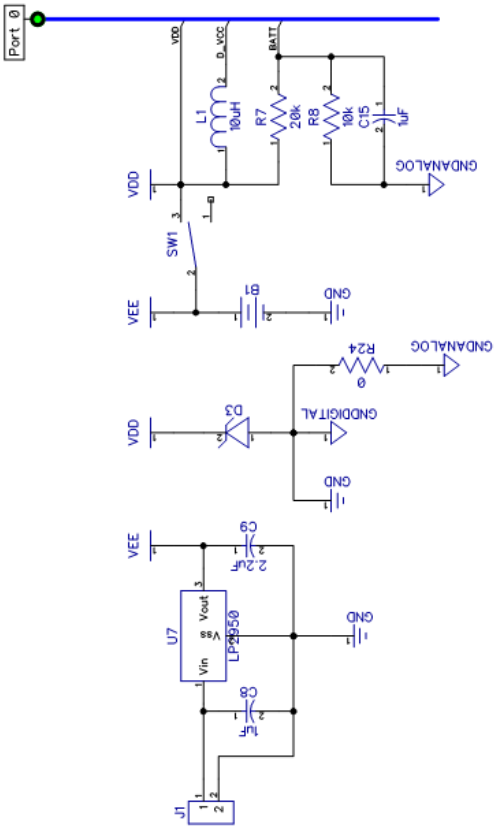
Appendix C. Sensor Node Parts List

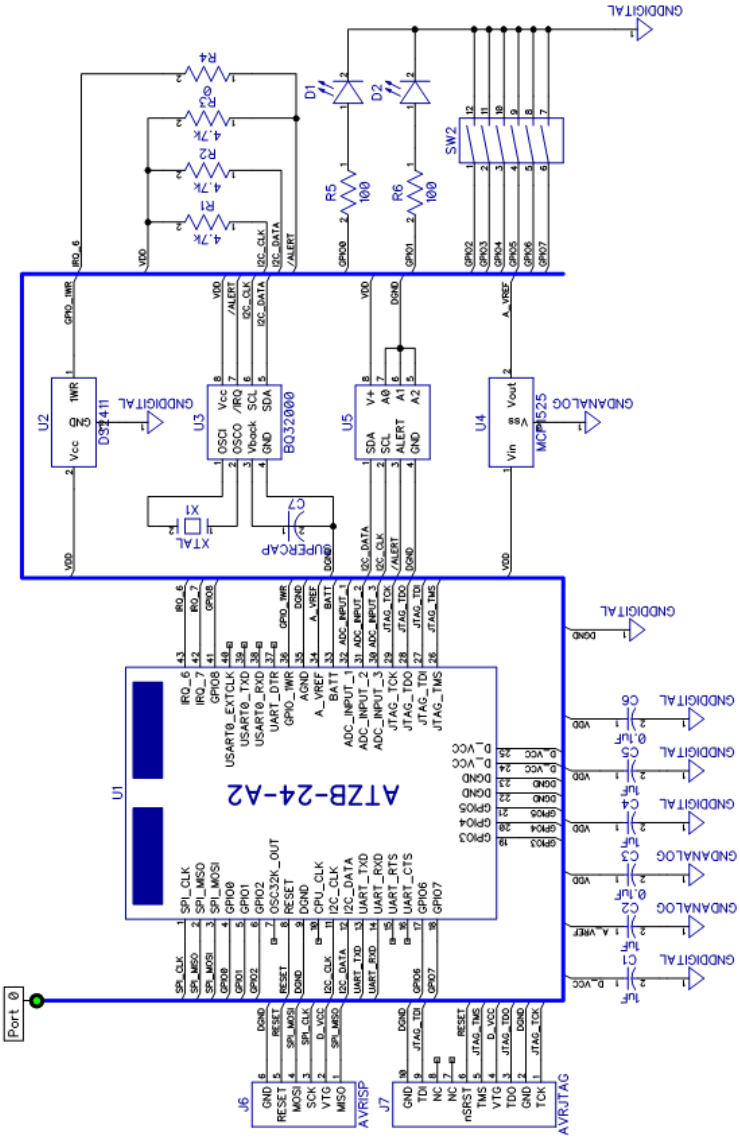
Digi-Key Part Number	Manufacturer	Manufacturer Part Number	Customer Reference	Quantity 1	Description
ATZB-24-A2RCT-ND	ATMEL (VA)	ATZB-24-A2R	U1	1	MOD 802.15.4/ZIGB 2.4GHZ CHIPANT
DS2411R+CT-ND	MAXIM INTEGRATED PRODUCTS (VA)	DS2411R+T&R	U2	1	IC SILICON SERIAL NUMBER SOT-23
296-24583-1-ND	TEXAS INSTRUMENTS (VA)	BQ32000DR	U3	1	IC SRL REAL TIME CLOCK 8SOIC
MCP1525T-I/TTCT-ND	MICROCHIP TECHNOLOGY (VA)	MCP1525T-I/TT	U4	1	IC VOLT REF 2.5V SOT23-3
296-17837-1-ND	TEXAS INSTRUMENTS (VA)	TMP175AIDR	U5	1	IC DIG TEMP SEN 2WIRE 8-SOIC
LTC1966CMS8#PBF-ND	LINEAR TECHNOLOGY	LTC1966CMS8#PBF	U6	1	IC PREC RMS/DC CONV MICRPWR 8MSOP
296-26120-ND	TEXAS INSTRUMENTS	LP2950-33LPE3	U7	1	IC VOLT REG MICRPWR 3.3V SD TO92
490-1582-1-ND	MURATA ELECTRONICS (VA)	GRM188F51C105ZA01D		10	CAP CER 1.0UF 16V Y5V 0603
490-1532-1-ND	MURATA ELECTRONICS (VA)	GRM188R71C104KA01D		10	CAP CER .1UF 16V 10% X7R 0603
493-2939-1-ND	NICHICON (VA)	F951C475MPAAQ2	C9	1	CAP TANT 16V 4.7UF SMD
604-1004-ND	ELNA AMERICA INC	DB-5R5D224T	C7	1	CAP DOUBLE LAYER .22F 5.5V RAD
475-2506-1-ND	OSRAM OPTO SEMICONDUCTORS INC(VA)	LS L29K-G1J2-1-0-2-R18-Z	D1	1	LED SMARTLED 630NM RED 0603 SMD
475-2709-1-ND	OSRAM OPTO SEMICONDUCTORS INC(VA)	LG L29K-G2J1-24-Z	D2	1	LED SMARTLED GREEN 570NM 0603
MMSZ5228BFSCT-ND	FAIRCHILD SEMICONDUCTOR (VA)	MMSZ5228B	D3	1	DIODE ZENER 3.9V 500MW SOD-123
BAT54CFSCT-ND	FAIRCHILD SEMICONDUCTOR (VA)	BAT54C		2	DIODE SCHOTTKY 30V 200MA SOT-23
277-1721-ND	PHOENIX CONTACT	1984617		5	CONN TERM BLOCK T/H 2POS 3.5MM
609-3218-ND	FCI	67996-406HLF	J6	1	CONN HEADER 6POS .100 STR TIN
609-3243-ND	FCI	67997-410HLF	J7	1	CONN HEADER 10POS .100 STR TIN
609-3464-ND	FCI	68000-203HLF	J8	1	BERGSTIK II .100" SR STRAIGHT
490-4029-1-ND	MURATA ELECTRONICS (VA)	LQM21FN100M70L	L1	1	INDUCTOR 10UH 100MA 0805
P4.7KGCT-ND	PANASONIC - ECG (VA)	ERJ-3GEYJ472V		3	RES 4.7K OHM 1/10W 5% 0603 SMD
P560GCT-ND	PANASONIC - ECG (VA)	ERJ-3GEYJ561V		2	RES 560 OHM 1/10W 5% 0603 SMD
P20KDBCT-ND	PANASONIC - ECG (VA)	ERA-3AEB203V	R7	1	RES 20K OHM 1/10W .1% 0603 SMD
P10KDBCT-ND	PANASONIC - ECG (VA)	ERA-3AEB103V	R8	1	RES 10K OHM 1/10W .1% 0603 SMD
P100KHCT-ND	PANASONIC - ECG (VA)	ERJ-3EKF1003V		6	RES 100K OHM 1/10W 1% 0603 SMD
P0.0GCT-ND	PANASONIC - ECG (VA)	ERJ-3GEY0R00V		12	RES 0.0 OHM 1/10W 0603 SMD
CKN9559-ND	C&K COMPONENTS	OS102011MA1QN1	SW1	1	SWITCH SLIDE SPDT 0.1A 12V
CT2196MST-ND	CTS ELECTROCOMPONENTS	219-6MST	SW2	1	SWITCH TAPE SEAL 6 POS SMD
300-8340-1-ND	CITIZEN FINETECH MIYOTA (VA)	CMR200T-32.768KDZF-UT	X1	1	CRYSTAL 32.768 KHZ 12.5PF SMD
HM1067-ND	HAMMOND MANUFACTURING	1551KFLBK		1	BOX ABS 3.15X1.58X0.79" BLACK
582-1004-ND	CR MAGNETICS INC	CR3110-3000		1	TRANSFORMER CURRENT SPLIT CORE

Appendix D. Sensor Node Hardware Schematics

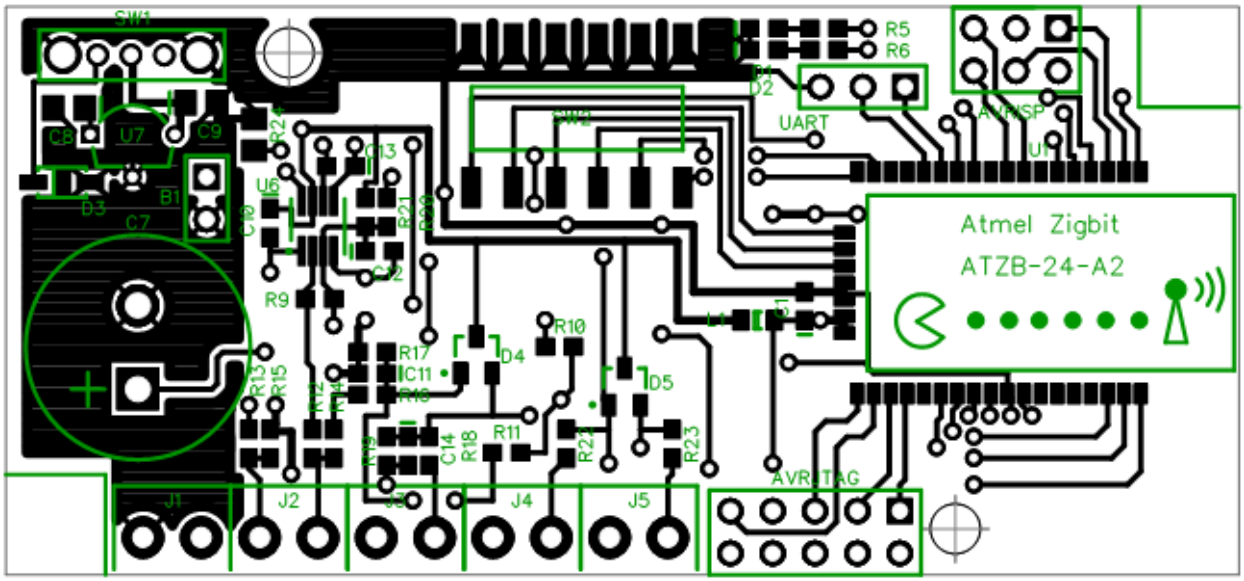



VDD
C10
0.1uF
GNDANALOG
Port 0
GPIO8
ADC_INPUT_1
ADC_INPUT_1
VDD
ADC_INPUT_2
R10
0
R11
0
D4
C13
1uF
VDD
R17
GNDANALOG
R19
GNDANALOG
C11
C14
U6
/EN
VDD
RTN
VOUT
GND
IN1
IN2
VSS
R16
R18
VDD
R21
100k
R20
100k
R9
0
C12
0.1uF
R12
0
R13
0
R15
0
R14
0
GNDANALOG
J2
J3

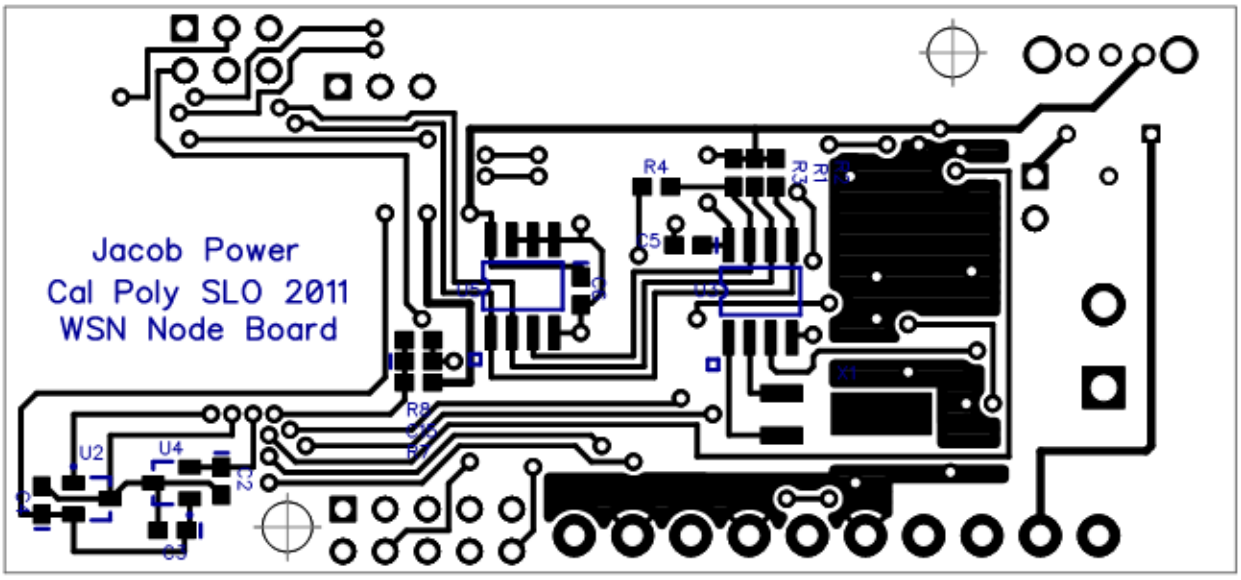

Port 0
VDD
D_VCC
BATT
L1
10uH
R7
20k
R8
10k
C15
1uF
GNDANALOG
VDD
SW1
VEE
B1
GND
R24
0
GNDANALOG
VDD
D3
GNDDIGITAL
GND
VEE
C9
2.2uF
U7
Vout
Vss
Vin
LP2950
GND
C8
1uF
J1
1
2


ATZB-24-A2
U1
U2
U3
U4
U5
SW2
D1
D2
R1
R2
R3
R4
R5
R6
C1
C2
C3
C4
C5
C6
C7
X1
J6
J7
AVRISP
AVRJTAG
Port 0
GNDDIGITAL
GNDANALOG

Appendix E. Sensor Node PCB Layout



Top Layer – Silkscreen and Copper



Bottom Layer – Silkscreen and Copper

Appendix F. Sensor Node Firmware Code

WSNNodeApp.c

```
/**************************************************************************//**
  \file WSNNodeApp.c
  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
  History:
    13/06/07 I. Kalganova - Modified
        10/6/11  Jacob Power - Modified
******************************************************************************/

#include <WSNNodeApp.h>
#include <WSNVisualizer.h>
#include <taskManager.h>
#include <zdo.h>
#include <configServer.h>
#include <aps.h>
#include <mac.h>


#define APP_TIMER_STARTING_NETWORK 500 // Period of blinking during starting network
#define TIMER_WAKEUP 50 // Wakeup period

/******************************************************************************
  Types
******************************************************************************/
typedef enum
{
  START_BLINKING,
  STOP_BLINKING
} BlinkingAction_t;


/******************************************************************************
  Prototypes
******************************************************************************/
void APS_DataIndCoord(APS_DataInd_t *indData);
void APS_DataIndRouter(APS_DataInd_t *indData);
void APS_DataIndDevice(APS_DataInd_t *indData);
void appPostSubTaskTask(void);
void appPostGlobalTask(void);

/******************************************************************************
  Global variables
 ******************************************************************************/
AppState_t     appState        = APP_INITING_STATE;//application state
DeviceState_t  appDeviceState  = INITIAL_DEVICE_STATE;

AppMessageRequest_t  appMessage;
//request parameters for APS_DataReq primitive
APS_DataReq_t messageParams;
HAL_AppTimer_t deviceTimer;
uint32_t sleepTime = 0;
uint8_t failedTransmission = 0;
//static bool globalTaskRequested = false;
```

```
/*****************************************************************************
  Local variables
 *****************************************************************************/
static DeviceType_t deviceType;
static ZDO_StartNetworkReq_t networkParams; //request params for ZDO_StartNetworkReq
//endpoint parameters
SimpleDescriptor_t simpleDescriptor;
static APS_RegisterEndpointReq_t endpointParams;
// Timer indicating starting network
static HAL_AppTimer_t startingNetworkTimer;
static ZDO_ZdpReq_t leaveReq;

/*****************************************************************************
  Local functions
 *****************************************************************************/
static void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t *confirmInfo);
static void startingNetworkTimerFired(void);
static void zdpLeaveResp(ZDO_ZdpResp_t *zdpResp);
static void initSubStateMachine(void);
static void appTaskHandler(AppEvent_t event, void *param);
static void manageBlinkingDuringRejoin(BlinkingAction_t action);

void initDIP(void);
void setupIRQ(void);

/*****************************************************************************
  Implementation
 *****************************************************************************/


/*****************************************************************************
  Application task.

  Parameters:
    none

  Return:
    none

 *****************************************************************************/
void APL_TaskHandler(void)
{
  appTaskHandler(APP_PROCESS, NULL);
}

static void appTaskHandler(AppEvent_t event, void *param)
{
  bool rxOnWhenIdleFlag = true;

  switch (appState)
  {
    //node is in network
    case APP_IN_NETWORK_STATE:
    {
      switch (event)
      {
        case APP_PROCESS:
        {
          if (deviceType == DEVICE_TYPE_ROUTER)
            appRouterTaskHandler(APP_PROCESS, NULL);
          if (deviceType == DEVICE_TYPE_END_DEVICE)
            appEndDeviceTaskHandler(APP_PROCESS, NULL);
          break;
        }

        case APP_NETWORK_STATE_UPDATED:
        {
          ZDO_MgmtNwkUpdateNotf_t *updateParam = param;
```

```
        switch (updateParam->status)
        {
          case ZDO_NETWORK_LOST_STATUS:
            APL_WRITE_LOG(0x03)
            appState = APP_STOP_STATE;
            break;

          case ZDO_NETWORK_LEFT_STATUS:
            APL_WRITE_LOG(0x04)
            appState = APP_STARTING_NETWORK_STATE;
            visualizeNwkLeft();
            failedTransmission = 0;
            appPostGlobalTask();
            break;

          case ZDO_NWK_UPDATE_STATUS:
          case ZDO_NETWORK_STARTED_STATUS:
            // Update parameters being sent to coordinator
            appMessage.data.shortAddr       = updateParam->nwkUpdateInf.shortAddr;
            appMessage.data.panID           = updateParam->nwkUpdateInf.panId;
            appMessage.data.parentShortAddr = updateParam->nwkUpdateInf.parentShortAddr;
            appMessage.data.workingChannel  = updateParam->nwkUpdateInf.currentChannel;
            break;

          default:
            break;
        }
      }

      default:
        break;
    }
    break;
  }

  case APP_INITING_STATE: //node has initial state
  {
    switch (event)
    {
      case APP_PROCESS:
      {
        visualizeAppStarting();
        SYS_InitLog();

        simpleDescriptor.endpoint = WSNDEMO_ENDPOINT;
        simpleDescriptor.AppProfileId = WSNDEMO_PROFILE_ID;
        simpleDescriptor.AppDeviceId = WSNDEMO_DEVICE_ID;
        simpleDescriptor.AppDeviceVersion = WSNDEMO_DEVICE_VERSION;

        endpointParams.simpleDescriptor = &simpleDescriptor;

       // Initalize GPIO 2-7 as input
       initDIP();

       // Set node type based on DIP switch setting
       if (GPIO_2_read() == 0)
       {
         appMessage.data.nodeType = DEVICE_TYPE_ROUTER;
         endpointParams.APS_DataInd = APS_DataIndRouter;

         //set ConfigServer CS_END_DEVICE_SLEEP_TIME to max for network
         sleepTime = 600000;
         CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID, &sleepTime);

         //set router transmit timer based on DIP switches
         if((GPIO_3_read() == 0) && (GPIO_4_read() == 0))
         {
             sleepTime = 5000;
```

```
        }
        else if((GPIO_3_read() == 0) && (GPIO_4_read() == 1))
        {
            sleepTime = 60000;
        }
        else if((GPIO_3_read() == 1) && (GPIO_4_read() == 0))
        {
            sleepTime = 300000;
        }
        else if((GPIO_3_read() == 1) && (GPIO_4_read() == 1))
        {
            sleepTime = 600000;
        }
      }
      else
      {
        appMessage.data.nodeType = DEVICE_TYPE_END_DEVICE;
        endpointParams.APS_DataInd = APS_DataIndDevice;

        //set end device sleep time in ConfigServer based on DIP switches
        if((GPIO_3_read() == 0) && (GPIO_4_read() == 0))
        {
            sleepTime = 5000;
            CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID, &sleepTime);
        }
        else if((GPIO_3_read() == 0) && (GPIO_4_read() == 1))
        {
            sleepTime = 60000;
            CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID, &sleepTime);
        }
        else if((GPIO_3_read() == 1) && (GPIO_4_read() == 0))
        {
            sleepTime = 300000;
            CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID, &sleepTime);
        }
        else if((GPIO_3_read() == 1) && (GPIO_4_read() == 1))
        {
            sleepTime = 600000;
            CS_WriteParameter(CS_END_DEVICE_SLEEP_PERIOD_ID, &sleepTime);
        }
      }
       // CS_UID_ID has been pre-filled based on the UID chip by the ZDO
       // Set the node's short address to the lower bits of the CS_UID
       {
         ExtAddr_t extAddr;

         CS_ReadParameter(CS_UID_ID, &extAddr);
         CS_WriteParameter(CS_NWK_ADDR_ID, &extAddr);
         appMessage.data.extAddr = extAddr;
       }
       APS_RegisterEndpointReq(&endpointParams);

       // Write device type to the ConfigServer
       // it is needed because devType is 2 bytes and nodeType is 1 byte
       deviceType = appMessage.data.nodeType;
       CS_WriteParameter(CS_DEVICE_TYPE_ID, &deviceType);

       if (DEVICE_TYPE_END_DEVICE == appMessage.data.nodeType)
       {
         rxOnWhenIdleFlag = false;
         CS_WriteParameter(CS_RX_ON_WHEN_IDLE_ID, &rxOnWhenIdleFlag);
       }
       else
       {
         rxOnWhenIdleFlag = true;
         CS_WriteParameter(CS_RX_ON_WHEN_IDLE_ID, &rxOnWhenIdleFlag);
       }

       APL_WRITE_LOG(0x00)
```

```
      appMessage.data.messageType     = APP_DATA_MESSAGE_TYPE;

      startingNetworkTimer.interval = APP_TIMER_STARTING_NETWORK;
      startingNetworkTimer.mode     = TIMER_REPEAT_MODE;
      startingNetworkTimer.callback = startingNetworkTimerFired;

      appState = APP_STARTING_NETWORK_STATE;
      appPostGlobalTask();
      break;
    }

    default:
      break;
  }
  break;
}

case APP_STARTING_NETWORK_STATE:
{
  switch (event)
  {
    case APP_PROCESS:
      manageBlinkingDuringRejoin(START_BLINKING);
      visualizeNwkStarting();
      failedTransmission = 0;

      APL_WRITE_LOG(0x01)
      // Start network
      networkParams.ZDO_StartNetworkConf = ZDO_StartNetworkConf;
      ZDO_StartNetworkReq(&networkParams);
      break;

    case APP_NETWORK_STARTING_DONE:
    {
      ZDO_StartNetworkConf_t *startInfo = param;
      manageBlinkingDuringRejoin(STOP_BLINKING);
      if (ZDO_SUCCESS_STATUS == startInfo->status)
      {
        APL_WRITE_LOG(0x02)
        appState = APP_IN_NETWORK_STATE;
        failedTransmission = 0;

        visualizeNwkStarted();
        // Network parameters, such as short address, should be saved
        appMessage.data.panID           = startInfo->PANId;
        appMessage.data.shortAddr       = startInfo->shortAddr;
        appMessage.data.parentShortAddr = startInfo->parentAddr;
        appMessage.data.workingChannel  = startInfo->activeChannel;

        //Enable and register interrupt handler for IRQ6 and 7
        setupIRQ();

        initSubStateMachine();
      }
      else
        appPostGlobalTask();
      break;
    }

    case APP_NETWORK_STATE_UPDATED:
    {
      ZDO_MgmtNwkUpdateNotf_t *updateParam = param;
      switch (updateParam->status)
      {
        case ZDO_NETWORK_STARTED_STATUS:
          APL_WRITE_LOG(0x0C)
          // Network parameters, such as short address, should be saved
          appMessage.data.panID           = updateParam->nwkUpdateInf.panId;
```

```
            appMessage.data.shortAddr       = updateParam->nwkUpdateInf.shortAddr;
            appMessage.data.parentShortAddr = updateParam->nwkUpdateInf.parentShortAddr;
            appMessage.data.workingChannel  = updateParam->nwkUpdateInf.currentChannel;
            manageBlinkingDuringRejoin(STOP_BLINKING);

            appState = APP_IN_NETWORK_STATE;
            failedTransmission = 0;
            visualizeNwkStarted();

            initSubStateMachine();
            break;
          default:
            break;
        }
      }
      default: break;
    }
    break;
  }
  case APP_LEAVING_NETWORK_STATE:
  {
    ZDO_MgmtLeaveReq_t *zdpLeaveReq = &leaveReq.req.reqPayload.mgmtLeaveReq;

    switch(event)
    {
      case APP_PROCESS:
        APL_WRITE_LOG(0x0A)
        visualizeNwkLeaving();
        leaveReq.ZDO_ZdpResp =  zdpLeaveResp;
        leaveReq.reqCluster = MGMT_LEAVE_CLID;
        leaveReq.dstAddrMode = EXT_ADDR_MODE;
        leaveReq.dstExtAddr = 0;
        zdpLeaveReq->deviceAddr = 0;
        zdpLeaveReq->rejoin = 0;
        zdpLeaveReq->removeChildren = 1;
        zdpLeaveReq->reserved = 0;
        ZDO_ZdpReq(&leaveReq);
        break;

      case APP_LEAVE_DONE:
        if (ZDO_SUCCESS_STATUS == ((ZDO_ZdpResp_t *)param)->respPayload.status)
        {
          APL_WRITE_LOG(0x0B)
          appState = APP_STOP_STATE;
        }
        appPostGlobalTask();
        break;

      case APP_NETWORK_STATE_UPDATED:
      {
        ZDO_MgmtNwkUpdateNotf_t *updateParam = param;

        switch (updateParam->status)
        {
          case ZDO_NETWORK_LOST_STATUS:
            APL_WRITE_LOG(0x03)
            appState = APP_STOP_STATE;
            break;
          case ZDO_NETWORK_LEFT_STATUS:
            APL_WRITE_LOG(0x04)
            appState = APP_STARTING_NETWORK_STATE;
            appPostGlobalTask();
            break;
          default:
            break;
        }
        break;
      }
```

```
        default:
          break;
      }
      break;
    }

    case APP_STOP_STATE:
    {
      switch(event)
      {
        case APP_NETWORK_STATE_UPDATED:
        {
          ZDO_MgmtNwkUpdateNotf_t *updateParam = param;

          switch (updateParam->status)
          {
            case ZDO_NETWORK_STARTED_STATUS:
              APL_WRITE_LOG(0x0C)
              appState = APP_IN_NETWORK_STATE;
              appMessage.data.shortAddr       = updateParam->nwkUpdateInf.shortAddr;
              appMessage.data.panID           = updateParam->nwkUpdateInf.panId;
              appMessage.data.parentShortAddr = updateParam->nwkUpdateInf.parentShortAddr;
              appMessage.data.workingChannel  = updateParam->nwkUpdateInf.currentChannel;
              appPostGlobalTask();
              break;

            case ZDO_NETWORK_LEFT_STATUS:
              APL_WRITE_LOG(0x04)
              appState = APP_STARTING_NETWORK_STATE;
              appPostGlobalTask();
              break;

            default:
              break;
          }
        }
        default:
          break;
      }
      break;
    }

    default:
      break;
  }
}

// The response means that the command has been received successfully but not precessed yet
static void zdpLeaveResp(ZDO_ZdpResp_t *zdpResp)
{
  appTaskHandler(APP_LEAVE_DONE, zdpResp);
}

/******************************************************************************
  ZDO_StartNetwork primitive confirmation was received.

  Parameters:
    confirmInfo - confirmation information

  Return:
    none

 ******************************************************************************/
static void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t *confirmInfo)
{
  appTaskHandler(APP_NETWORK_STARTING_DONE, confirmInfo);
}
```

```
void ZDO_MgmtNwkUpdateNotf(ZDO_MgmtNwkUpdateNotf_t *nwkParams)
{
   appTaskHandler(APP_NETWORK_STATE_UPDATED, nwkParams);
}

void ZDO_WakeUpInd(void)
{
  if (APP_IN_NETWORK_STATE == appState)
  {
      if (deviceType == DEVICE_TYPE_END_DEVICE)
      {
         appEndDeviceTaskHandler(APP_WOKEUP, NULL);
      }
      else
      {
         appRouterTaskHandler(APP_TIMER_FIRED, NULL);
      }
  }
}

// For blinking
static void startingNetworkTimerFired(void)
{
  visualizeNwkStarting();
}

void initSubStateMachine(void)
{
  if (deviceType == DEVICE_TYPE_ROUTER)
    appInitDeviceRouter();
  if (deviceType == DEVICE_TYPE_END_DEVICE)
    appInitDeviceEndDevice();
}

void appReadLqiRssi(void)
{
  ZDO_GetLqiRssi_t lqiRssi;

  lqiRssi.nodeAddr = appMessage.data.parentShortAddr;
  ZDO_GetLqiRssi(&lqiRssi);

  appMessage.data.lqi  = lqiRssi.lqi;
  appMessage.data.rssi = lqiRssi.rssi;
}

static void manageBlinkingDuringRejoin(BlinkingAction_t action)
{
  static bool run = false;

  if (START_BLINKING == action)
  {
    if (!run)
    {
      HAL_StartAppTimer(&startingNetworkTimer);
      run = true;
    }
  }

  if (STOP_BLINKING == action)
  {
    run = false;
    HAL_StopAppTimer(&startingNetworkTimer);
  }
}

void appPostGlobalTask(void)
{
  SYS_PostTask(APL_TASK_ID);
}
```

```
void appPostSubTaskTask(void)
{
  SYS_PostTask(APL_TASK_ID);
}

void appLeaveNetwork(void)
{
  if (APP_IN_NETWORK_STATE == appState)
  {
    appState = APP_LEAVING_NETWORK_STATE;
    appPostGlobalTask();
  }
}

void initDIP(void)
{
   GPIO_2_make_in();
   GPIO_3_make_in();
   GPIO_4_make_in();
   GPIO_5_make_in();
   GPIO_6_make_in();
   GPIO_7_make_in();

   GPIO_2_make_pullup();
   GPIO_3_make_pullup();
   GPIO_4_make_pullup();
   GPIO_5_make_pullup();
   GPIO_6_make_pullup();
   GPIO_7_make_pullup();

   GPIO_8_make_out();
   GPIO_8_clr();
}

void setupIRQ(void)
{
   HAL_RegisterIrq(IRQ_6, IRQ_ANY_EDGE, ZDO_WakeUpInd);
   HAL_RegisterIrq(IRQ_7, IRQ_ANY_EDGE, ZDO_WakeUpInd);

   HAL_EnableIrq(IRQ_6);
   HAL_EnableIrq(IRQ_7);
}

void disableIRQ(void)
{
   HAL_DisableIrq(IRQ_6);
   HAL_DisableIrq(IRQ_7);
}

void enableIRQ(void)
{
   HAL_EnableIrq(IRQ_6);
   HAL_EnableIrq(IRQ_7);
}

#ifdef _BINDING_
/******************************************************************************
  Stub for ZDO Binding Indication

  Parameters:
    bindInd - indication

  Return:
    none

 ******************************************************************************/
void ZDO_BindIndication(ZDO_BindInd_t *bindInd)
{
```

```
  (void)bindInd;
}

/**************************************************************************************
  Stub for ZDO Unbinding Indication

  Parameters:
    unbindInd - indication

  Return:
    none

 **************************************************************************************/
void ZDO_UnbindIndication(ZDO_UnbindInd_t *unbindInd)
{
  (void)unbindInd;
}
#endif //_BINDING_
//eof WSNNodeApp.c
```

WSNNodeApp.h

```
/**************************************************************************//**
  \file WSNNodeApp.h

  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
******************************************************************************/

#define _WSNNodeApp_H

/*****************************************************************************
                    Includes section
*****************************************************************************/
#include <macAddr.h>
#include <appFramework.h>
#include <configServer.h>
#include <appTimer.h>
#include <aps.h>
#include <uid.h>
#include <appTimer.h>
#include <zdo.h>
#include <dbg.h>
#include <gpio.h>
#include <irq.h>

/*****************************************************************************
                    Defines section
*****************************************************************************/

#define DEVICE1_EXT_ADDR      0x1ULL //0x000100001090C993LL
#define DEVICE2_EXT_ADDR      0x2ULL
#define COORDINATOR_EXT_ADDR  CS_APS_TRUST_CENTER_ADDRESS
#define LINK_KEY {0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
0xaa, 0xaa}


#define WSNDEMO_PROFILE_ID CPU_TO_LE16(1)
```

```
#define WSNDEMO_DEVICE_ID CPU_TO_LE16(1)
#define WSNDEMO_DEVICE_VERSION 1
#define WSNDEMO_ENDPOINT 1

#define MAX_USART_MESSAGE_QUEUE_COUNT 5
#define MAX_DEVICE_MESSAGE_LEN        10
#define APP_DATA_MESSAGE_TYPE         1
/* At worst each byte will be transformed to two bytes + size of crc
 * + size of 4 bytes: 0x10, 0x02, 0x10, 0x03. */
#define MAX_RAW_APP_MESSAGE_SIZE  (2 + 2 * sizeof(AppMessage_t) + 2 + 1)

typedef enum
{
  APP_PROCESS,
  APP_NETWORK_STARTING_DONE,
  APP_NETWORK_STATE_UPDATED,
  APP_LEAVE_DONE,
  APP_SENDING_DONE,
  APP_READING_DONE,
  APP_SLEEP_DONE,
  APP_WOKEUP,
  APP_TIMER_FIRED,
} AppEvent_t;


typedef enum
{
  APP_INITING_STATE,
  APP_STARTING_NETWORK_STATE,
  APP_IN_NETWORK_STATE,
  APP_LEAVING_NETWORK_STATE,
  APP_STOP_STATE
} AppState_t;

typedef enum
{
  INITIAL_DEVICE_STATE,
  SENDING_DEVICE_STATE,
  READING_SENSORS_STATE,
  SLEEPPING_DEVICE_STATE,
  STARTING_TIMER_STATE,
  WAITING_DEVICE_STATE,
  REINITIAL_DEVICE_STATE
} DeviceState_t;

BEGIN_PACK
typedef struct
{
   uint8_t     messageType;
   uint8_t     nodeType;
   ExtAddr_t   extAddr;
   ShortAddr_t shortAddr;
   PanId_t     panID;
   uint8_t     workingChannel;
   ShortAddr_t parentShortAddr;
   uint8_t     lqi;
   int8_t      rssi;

   uint16_t    battery;
   uint16_t    temp;
   uint32_t    time;
   uint32_t    date;
   uint16_t    analog1;
   uint16_t    analog2;
   uint8_t     cc1;
   uint8_t     cc2;
} PACK AppMessage_t;

typedef struct
```

```
{
  uint8_t        header[APS_ASDU_OFFSET];
  AppMessage_t   data;
  uint8_t        footer[APS_AFFIX_LENGTH - APS_ASDU_OFFSET];
} PACK AppMessageRequest_t;
END_PACK

extern uint32_t       sleepTime;
extern uint16_t       appPANId;
extern ShortAddr_t    appShortAddr;
extern uint8_t        appChannel;
extern DeviceState_t  appDeviceState;
extern DeviceType_t   appDeviceType;
extern ShortAddr_t    appParentAddr;
extern uint64_t       appUid;
extern SimpleDescriptor_t simpleDescriptor;
extern uint8_t        failedTransmission;

/******************************************************************************
                   Functions' prototypes section
******************************************************************************/
extern void appStartSensorManager(void);
extern void appStopSensorManager(void);
extern void appGetSensorData(void (*passedCB)(void));
extern void appRouterTaskHandler(AppEvent_t event, void *param);
extern void appEndDeviceTaskHandler(AppEvent_t event, void *param);
extern void appStopRouter(void);
extern void appPostSubTaskTask(void);
extern void SYS_InitLog(void);
extern void appReadLqiRssi(void);
extern void appInitDeviceRouter(void);
extern void appInitDeviceEndDevice(void);
extern void SYS_StartLog(void);
extern void SYS_StopLog(void);
extern void appSendMessageToUsart(AppMessage_t *newMessage);
extern void disableIRQ(void);
extern void enableIRQ(void);
```

WSNEndDevice.c

```
/*****************************************************************************//**
  \file WSNEndDeviceApp.c

  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
  History:
    13/06/07 I. Kalganova - Modified
        10/06/11 Jacob Power - Modified
******************************************************************************/

#include <WSNNodeApp.h>
#include <WSNVisualizer.h>

//request parameters for APS_DataReq primitive
extern APS_DataReq_t messageParams;
extern AppMessageRequest_t appMessage;
extern AppState_t appState;
extern bool subTaskRequested;
extern void appLeaveNetwork(void);
```

```
static ZDO_SleepReq_t sleepReq;

static void APS_DataConf(APS_DataConf_t *confInfo);
static void ZDO_SleepConf(ZDO_SleepConf_t *conf);
static void appSensorsReadDone(void);
void APS_DataIndDevice(APS_DataInd_t *indData);
void appInitDeviceEndDevice(void);


/************************************************************************************
  Process end device functionality.

  Parameters:
    none

  Return:
    none

 ************************************************************************************/
void appEndDeviceTaskHandler(AppEvent_t event, void *param)
{
  switch (appDeviceState)
  {
    case WAITING_DEVICE_STATE:
      switch (event)
      {
        case APP_SENDING_DONE:
          if (APS_SUCCESS_STATUS == ((APS_DataConf_t *)param)->status)
          {
            APL_WRITE_LOG(0x71)
            visualizeAirTxFInished();
            failedTransmission = 0;
            appDeviceState = SLEEPPING_DEVICE_STATE;
          }
          else
          {
            APL_WRITE_LOG(0x72)
            if (APP_THRESHOLD_FAILED_TRANSMISSION < ++failedTransmission)
            {
              appDeviceState = INITIAL_DEVICE_STATE;
              failedTransmission = 0;
              appLeaveNetwork();
              return;
            }
            else
              appDeviceState = SENDING_DEVICE_STATE;
          }
          appPostSubTaskTask();
          break;

        case APP_SLEEP_DONE:
          if (ZDO_SUCCESS_STATUS == ((ZDO_SleepConf_t *)param)->status)
          {
            appDeviceState = SLEEPPING_DEVICE_STATE;
//            visualizeSleep();
          }
          else
          {
            appDeviceState = SLEEPPING_DEVICE_STATE;
            APL_WRITE_LOG(0x7E)
            appPostSubTaskTask();
          }
          break;

        case APP_READING_DONE:
          appStopSensorManager();
          appDeviceState = SENDING_DEVICE_STATE;
          appPostSubTaskTask();
```

```
        break;

      default:
        break;
    }
    break;

  case READING_SENSORS_STATE:
    switch (event)
    {
      case APP_PROCESS:
          appStartSensorManager();
          appReadLqiRssi();
          appDeviceState = WAITING_DEVICE_STATE;
          appGetSensorData(appSensorsReadDone);
      break;

      default:
        break;
    }
    break;

  case SENDING_DEVICE_STATE:
    switch (event)
    {
      case APP_PROCESS:
        visualizeAirTxStarted();
        APL_WRITE_LOG(0x70)
        APS_DataReq(&messageParams);
        appDeviceState = WAITING_DEVICE_STATE;
        break;

      default:
        break;
    }
    break;

  case SLEEPPING_DEVICE_STATE:
    switch (event)
    {
      case APP_PROCESS:
        APL_WRITE_LOG(0x7C)
        visualizeSleep();
        appDeviceState = WAITING_DEVICE_STATE;
        enableIRQ();
        ZDO_SleepReq(&sleepReq);
        break;

      case APP_WOKEUP:
        visualizeWakeUp();

        APL_WRITE_LOG(0x7F)

        disableIRQ();

        appDeviceState = READING_SENSORS_STATE;
        appPostSubTaskTask();
        break;

      default:
        break;
    }
    break;

  case INITIAL_DEVICE_STATE:
    switch (event)
    {
      case APP_PROCESS:
        sleepReq.ZDO_SleepConf = ZDO_SleepConf;
```

```
          // Prefilling request parameters
          messageParams.profileId              = simpleDescriptor.AppProfileId;
          messageParams.dstAddrMode            = APS_SHORT_ADDRESS;
          messageParams.dstAddress.shortAddress = CPU_TO_LE16(0);
          messageParams.dstEndpoint            = 1;
          messageParams.clusterId              = CPU_TO_LE16(1);
          messageParams.srcEndpoint            = simpleDescriptor.endpoint;
          messageParams.asduLength             = sizeof(appMessage.data);
          messageParams.asdu                   = (uint8_t *)(&appMessage.data);
          messageParams.txOptions.acknowledgedTransmission = 1;
#ifdef _APS_FRAGMENTATION_
          messageParams.txOptions.fragmentationPermitted = 1;
#endif // _APS_FRAGMENTATION_
#ifdef _HIGH_SECURITY_
          messageParams.txOptions.securityEnabledTransmission = 1;
#endif
          messageParams.radius                 = 0x0;
          messageParams.APS_DataConf           = APS_DataConf;

          appDeviceState = READING_SENSORS_STATE;
          appPostSubTaskTask();
          break;

        default:
          break;
      }
      break;

    case REINITIAL_DEVICE_STATE:
      switch (event)
      {
        case APP_PROCESS:
          break;

        default:
          appDeviceState = INITIAL_DEVICE_STATE;
          appPostSubTaskTask();
          break;
      }

    default:
      break;
  }
}

// appEndDeviceTaskHandler must be called from main state machine after this function
void appInitDeviceEndDevice(void)
{
  if (WAITING_DEVICE_STATE == appDeviceState)
    appDeviceState = REINITIAL_DEVICE_STATE;
  else
  {
    appDeviceState = INITIAL_DEVICE_STATE;
    appPostSubTaskTask();
  }
}

static void appSensorsReadDone(void)
{
  appEndDeviceTaskHandler(APP_READING_DONE, NULL);
}

void ZDO_SleepConf(ZDO_SleepConf_t *conf)
{
  appEndDeviceTaskHandler(APP_SLEEP_DONE, conf);
}

static void APS_DataConf(APS_DataConf_t *confInfo)
{
```

```
  appEndDeviceTaskHandler(APP_SENDING_DONE, confInfo);
}

void APS_DataIndDevice(APS_DataInd_t *indData)
{
  indData = indData;
}

//eof WSNEndDevice.c
```

WSNRouter.c

```
/************************************************************************//**
  \file WSNRouterEndDeviceApp.c

  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
  History:
    13/06/07 I. Kalganova - Modified
        10/06/11 Jacob Power - Modified
*****************************************************************************/

#include <WSNNodeApp.h>
#include <WSNVisualizer.h>

extern AppMessageRequest_t appMessage;
extern AppState_t appState;
extern APS_DataReq_t messageParams;
extern HAL_AppTimer_t deviceTimer;
extern bool subTaskRequested;
extern void appLeaveNetwork(void);

static void deviceTimerFired(void);
static void APS_DataConf(APS_DataConf_t *confInfo);
void APS_DataIndRouter(APS_DataInd_t *indData);
static void appSensorsReadDone(void);
void appInitDeviceRouter(void);


/************************************************************************************
  Process router functionality.

  Parameters:
    none

  Return:
    none

 ************************************************************************************/
void appRouterTaskHandler(AppEvent_t event, void *param)
{
  switch (appDeviceState)
  {
    case WAITING_DEVICE_STATE:
      switch (event)
      {
        case APP_READING_DONE:
          appDeviceState = SENDING_DEVICE_STATE;
          appStopSensorManager();
```

```
        appPostSubTaskTask();
        break;

      case APP_SENDING_DONE:
        if (APS_SUCCESS_STATUS == ((APS_DataConf_t *)param)->status)
        {
          APL_WRITE_LOG(0x71)
          visualizeAirTxFInished();

          failedTransmission = 0;
          appMessage.data.parentShortAddr = NWK_GetNextHop(0x0000);
          appReadLqiRssi();
          appDeviceState = STARTING_TIMER_STATE;
        }
        else
        {
          appDeviceState = SENDING_DEVICE_STATE;
          APL_WRITE_LOG(0x72)

          if (APP_THRESHOLD_FAILED_TRANSMISSION < ++failedTransmission)
          {
            appDeviceState = INITIAL_DEVICE_STATE;
            failedTransmission = 0;
            appLeaveNetwork();
            return;
          }
        }
        appPostSubTaskTask();
        break;

      case APP_TIMER_FIRED:
        disableIRQ();
        appDeviceState = READING_SENSORS_STATE;
        appPostSubTaskTask();
        break;

      default:
        break;
    }
    break;

  case READING_SENSORS_STATE:
    switch (event)
    {
      case APP_PROCESS:
        appReadLqiRssi();
        appStartSensorManager();
       appDeviceState = WAITING_DEVICE_STATE; // need to put here in a case if run context isn't broken
        appGetSensorData(appSensorsReadDone);
        break;

      default:
        break;
    }
    break;

  case SENDING_DEVICE_STATE:
    switch (event)
    {
      case APP_PROCESS:
        visualizeAirTxStarted();
        APL_WRITE_LOG(0x70)
        APS_DataReq(&messageParams);
        appDeviceState = WAITING_DEVICE_STATE;
        break;

      default:
        break;
    }
```

```
      break;

    case STARTING_TIMER_STATE:
      switch (event)
      {
        case APP_PROCESS:
          enableIRQ();
          HAL_StartAppTimer(&deviceTimer);
          appDeviceState = WAITING_DEVICE_STATE;
          break;

        default:
          break;
      }
      break;

    case INITIAL_DEVICE_STATE:
      switch (event)
      {
        case APP_PROCESS:
          HAL_StopAppTimer(&deviceTimer); // Have to be stopped before start
          deviceTimer.interval = sleepTime;
          deviceTimer.mode     = TIMER_ONE_SHOT_MODE;
          deviceTimer.callback = deviceTimerFired;

          // Prefilling request parameters
          messageParams.profileId               = simpleDescriptor.AppProfileId;
          messageParams.dstAddrMode             = APS_SHORT_ADDRESS;
          messageParams.dstAddress.shortAddress = CPU_TO_LE16(0);
          messageParams.dstEndpoint             = 1;
          messageParams.clusterId               = CPU_TO_LE16(1);
          messageParams.srcEndpoint             = simpleDescriptor.endpoint;
          messageParams.asduLength              = sizeof(appMessage.data);
          messageParams.asdu                    = (uint8_t *)(&appMessage.data);
          messageParams.txOptions.acknowledgedTransmission = 1;
#ifdef _APS_FRAGMENTATION_
          messageParams.txOptions.fragmentationPermitted = 1;
#endif
#ifdef _HIGH_SECURITY_
          messageParams.txOptions.securityEnabledTransmission = 1;
#endif
          messageParams.radius                  = 0x0;
          messageParams.APS_DataConf            = APS_DataConf;

          appDeviceState = READING_SENSORS_STATE;
          appPostSubTaskTask();
          break;

        default:
          break;
      }
      break;

    case REINITIAL_DEVICE_STATE:
      switch (event)
      {
        case APP_PROCESS:
          break;

        default:
          appDeviceState = INITIAL_DEVICE_STATE;
          appPostSubTaskTask();
          break;
      }

    default:
      break;
  }
}
```

```
// appRouterTaskHandler must be called from main state machine after this function
void appInitDeviceRouter(void)
{
  if (WAITING_DEVICE_STATE == appDeviceState)
    appDeviceState = REINITIAL_DEVICE_STATE;
  else
  {
    appDeviceState = INITIAL_DEVICE_STATE;
    appPostSubTaskTask();
  }
}

void APS_DataIndRouter(APS_DataInd_t *indData)
{
  indData = indData;
}

static void appSensorsReadDone(void)
{
  appRouterTaskHandler(APP_READING_DONE, NULL);
}

/******************************************************************************
  Device timer event.

  Parameters:
    none

  Return:
    none

 ******************************************************************************/
static void deviceTimerFired(void)
{
  appRouterTaskHandler(APP_TIMER_FIRED, NULL);
}

/******************************************************************************
  Confirmation of message sent.

  Parameters:
    confInfo - confirmation information

  Return:
    none

 ******************************************************************************/
static void APS_DataConf(APS_DataConf_t *confInfo)
{
  appRouterTaskHandler(APP_SENDING_DONE, confInfo);
}
```

WSNSensorManager.c

```
/*****************************************************************************//**
  \file WSNSensorManager.c

  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
```

```
  History:
    13/06/07 I. Kalganova - Modified
   4/9/2011 J. Power
*****************************************************************************/

#include <adc.h>
#include <i2cpacket.h>
#include <types.h>
#include <taskManager.h>
#include <WSNNodeApp.h>
#include <WSNVisualizer.h>

/***************************
Defines
***************************/
#define TMP175_ADDR 0x48
#define TMP175_TMP_REG 0x00
#define TMP175_CONF_REG 0x01
#define TMP175_CONT_RUN_12BIT 0x60
#define TMP175_CONT_RUN_9BIT 0x00

//ADC stuff
HAL_AdcParams_t ADCParams;
uint16_t ADCdata;

//I2C stuff
HAL_I2cParams_t I2CParams;
HAL_i2cMode_t I2CMode;
uint8_t I2Cdata[5];

extern AppMessageRequest_t  appMessage;

void ADC0_Read(void);
void ADC1_Read(void);
void ADC2_Read(void);
void CC_Read(void);
void TMP175_Reg_Change(void);
void TMP175_Data_Read(void);
void TMP175_Data_Finish(void);

static void (*callback)(void);

////////////////////////////////////////////////////////////////////

void appStartSensorManager(void)
{
   I2CMode.clockrate = I2C_CLOCK_RATE_125;
   HAL_OpenI2cPacket(&I2CMode);
}

void appStopSensorManager()
{
   HAL_CloseI2cPacket();
}

void appGetSensorData(void (*passedCB)(void))
{
   callback = passedCB;

   GPIO_IRQ_6_make_in();
   GPIO_IRQ_6_make_pullup();
   GPIO_IRQ_7_make_in();
   GPIO_IRQ_7_make_pullup();
   //GPIO_8_make_out();
   //GPIO_8_clr();

   I2CParams.id = TMP175_ADDR;
   I2CParams.length = 2;
   I2CParams.data = I2Cdata;
```

```
   I2CParams.lengthAddr = HAL_NO_INTERNAL_ADDRESS;
   I2CParams.f = ADC0_Read;

   //set up initial configuration of TMP175 here
   I2Cdata[0] = TMP175_CONF_REG;
   I2Cdata[1] = TMP175_CONT_RUN_12BIT;
   HAL_WriteI2cPacket(&I2CParams);

/*
   //UNCOMMENT FOR DEBUG PURPOSES
   //OR FOR BOARDS WITH NO SENSORS (IE RCBs)
   appMessage.data.battery = (int32_t) 100;
   appMessage.data.temp = (int16_t) 0;
   appMessage.data.analog1 = (int16_t) 0;
   appMessage.data.analog2 = (int16_t) 0;
   appMessage.data.cc1 = (uint8_t) 0;
   appMessage.data.cc2 = (uint8_t) 0;
   callback();
   */
}

/**
* Call for a read on ADC channel 0 (battery)
*/
void ADC0_Read()
{
   ADCParams.resolution = RESOLUTION_10_BIT;
   ADCParams.sampleRate = ADC_4800SPS;
   ADCParams.voltageReference = AREF;
   ADCParams.bufferPointer = &ADCdata;
   ADCParams.selectionsAmount = 1;
   ADCParams.callback = ADC1_Read;
   HAL_OpenAdc(&ADCParams);
   HAL_ReadAdc(HAL_ADC_CHANNEL0);
}

/**
* Get the data from the ADC channel 0 read, call for read on
* ADC channel 1
*/
void ADC1_Read()
{
   HAL_CloseAdc();
   appMessage.data.battery = ADCdata;

   ADCParams.callback = ADC2_Read;
   HAL_OpenAdc(&ADCParams);
   HAL_ReadAdc(HAL_ADC_CHANNEL1);
}

/**
* Get the data from the ADC channel 1 read, call for read on
* ADC channel 2
*/
void ADC2_Read()
{
   HAL_CloseAdc();
   appMessage.data.analog1 = ADCdata;

   //GPIO_8_clr();
   //GPIO_8_make_in();

   ADCParams.callback = CC_Read;
   HAL_OpenAdc(&ADCParams);
   HAL_ReadAdc(HAL_ADC_CHANNEL2);
}

/**
* Get data from the ADC channel 2 read and read the states of
```

```
* CC0 and CC1.  Call for data read from TMP175.
*/
void CC_Read()
{
   HAL_CloseAdc();
   appMessage.data.analog2 = ADCdata;

   appMessage.data.cc1 = GPIO_IRQ_6_read();
   appMessage.data.cc2 = GPIO_IRQ_7_read();

   TMP175_Reg_Change();
}

void TMP175_Reg_Change(void)
{
   I2CParams.id = TMP175_ADDR;
   I2CParams.length = 1;
   I2CParams.data = I2Cdata;
   I2CParams.lengthAddr = HAL_NO_INTERNAL_ADDRESS;
   I2CParams.f = TMP175_Data_Read;

   I2Cdata[0] = TMP175_TMP_REG;

   HAL_WriteI2cPacket(&I2CParams);
}

void TMP175_Data_Read(void)
{
   I2CParams.id = TMP175_ADDR;
   I2CParams.length = 2;
   I2CParams.data = I2Cdata;
   I2CParams.lengthAddr = HAL_NO_INTERNAL_ADDRESS;
   I2CParams.f = TMP175_Data_Finish;

   HAL_ReadI2cPacket(&I2CParams);
}

void TMP175_Data_Finish(void)
{
   HAL_CloseI2cPacket();
   int32_t tempTemperature = 0;
   tempTemperature = (I2Cdata[0] << 4) | (I2Cdata[1] >> 4);
   appMessage.data.temp = tempTemperature;
   callback();
}
```

WSNVisualizer.c

```
/**************************************************************************//**
  \file WSNVisualizer.c

  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
  History:
    18/12/08 A. Luzhetsky - Modified
        10/06/11 Jacob Power - Modified
******************************************************************************/

/******************************************************************************
******************************************************************************/
#include <WSNDemoApp.h>
```

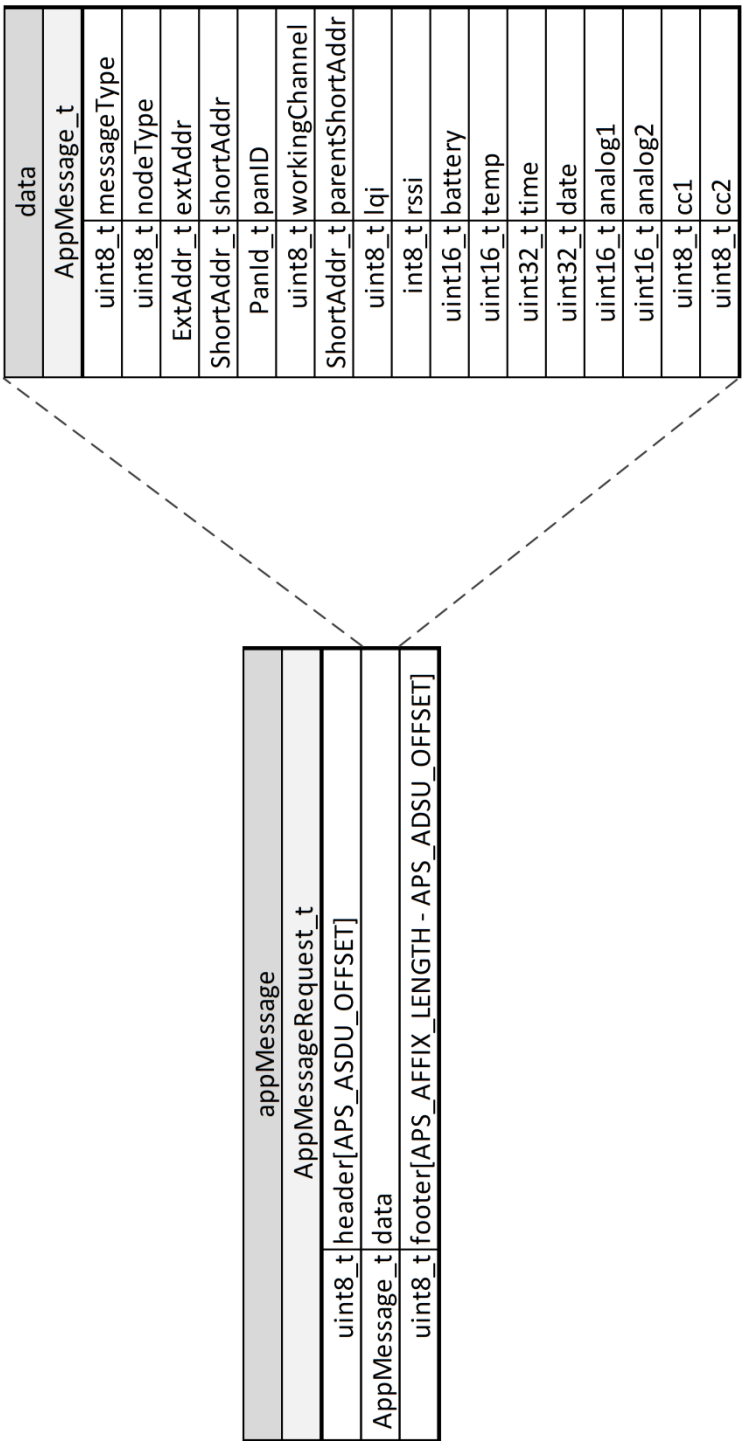
```
#define LED_RED 0
#define LED_GREEN 1

/************************************************************************
************************************************************************/
void visualizeAppStarting(void);
void visualizeNwkStarting(void);
void visualizeNwkStarted(void);
void visualizeNwkLeaving(void);
void visualizeNwkLeft(void);
void visualizeAirTxStarted(void);
void visualizeAirTxFInished(void);
void visualizeAirRxFinished(void);
void visualizeSerialTx(void);
void visualizeWakeUp(void);
void visualizeSleep(void);

void appOpenLeds(void);
void appCloseLeds(void);
void appOnLed(int led);
void appOffLed(int led);
void appToggleLed(int led);

void visualizeAppStarting(void)
{
  appOpenLeds();
}

void visualizeNwkStarting(void)
{
  appToggleLed(LED_RED);
}

void visualizeNwkStarted(void)
{
  appOnLed(LED_RED);
  appOffLed(LED_GREEN);
}

void visualizeNwkLeaving(void)
{
  appOffLed(LED_RED);
}

void visualizeNwkLeft(void)
{
  appOffLed(LED_RED);
}

void visualizeAirTxStarted(void)
{
  appOnLed(LED_RED);

}

void visualizeAirTxFInished(void)
{
  appOffLed(LED_RED);

}

void visualizeAirRxFinished(void)
{
  appToggleLed(LED_RED);

}

void visualizeSerialTx(void)
```

```
{
  appToggleLed(LED_RED);
}

void visualizeWakeUp(void)
{
  appOpenLeds();
  appOnLed(LED_RED);
}

void visualizeSleep(void)
{
  appCloseLeds();
}

void appOpenLeds(void)
{
   GPIO_0_make_out();
   GPIO_1_make_out();

   GPIO_0_set();
   GPIO_1_set();
}

void appCloseLeds(void)
{
   GPIO_0_clr();
   GPIO_1_clr();

   GPIO_0_make_in();
   GPIO_1_make_in();
}

void appOnLed(int led)
{
   if(led == 0)
   {
      GPIO_0_set();
   }
   else
   {
      GPIO_1_set();
   }
}

void appOffLed(int led)
{
   if(led == 0)
   {
      GPIO_0_clr();
   }
   else
   {
      GPIO_1_clr();
   }
}

void appToggleLed(int led)
{
   if(led == 0)
   {
      GPIO_0_toggle();
   }
   else
   {
      GPIO_1_toggle();
   }
}
```

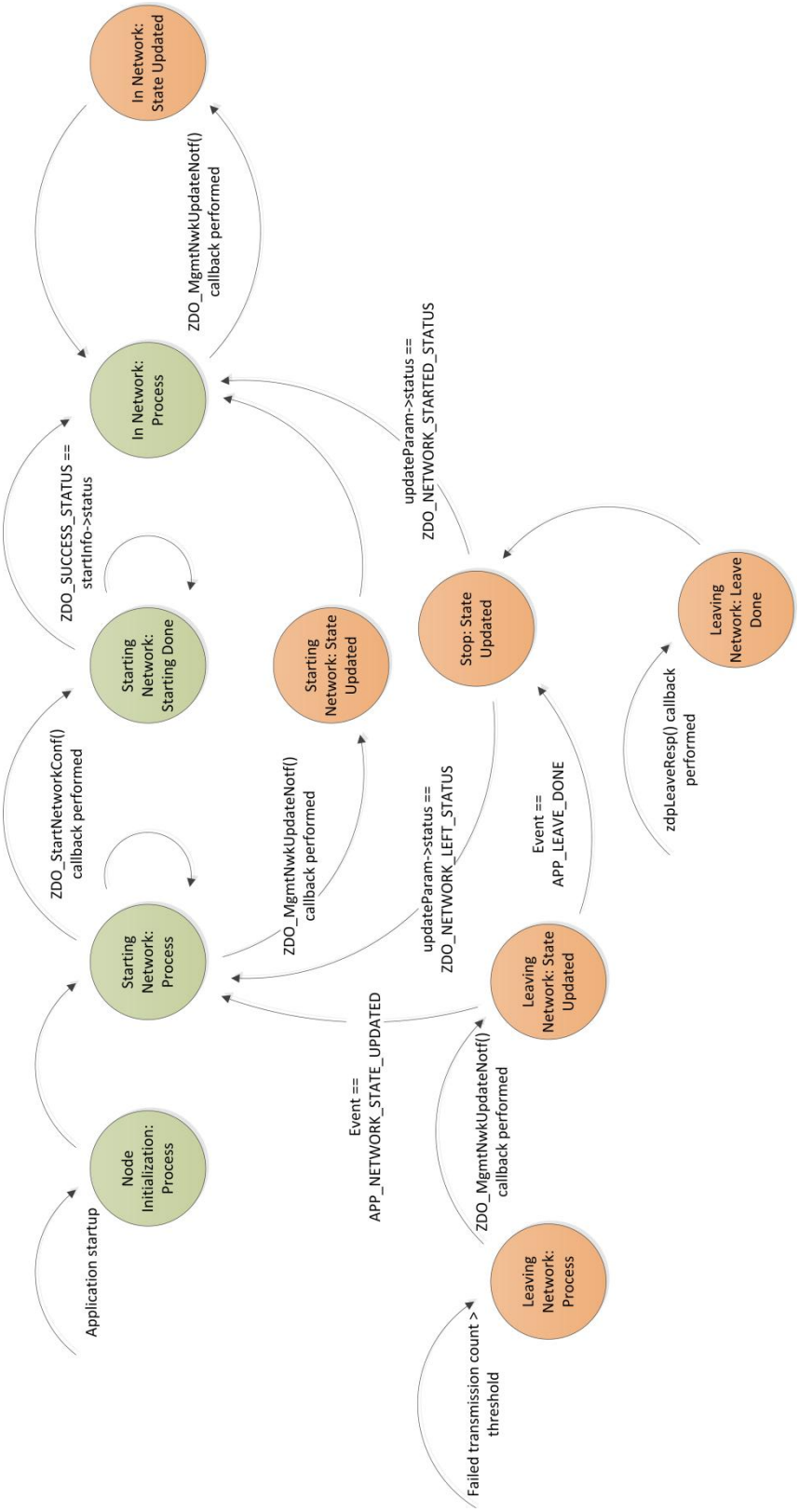
WSNVisualizer.h

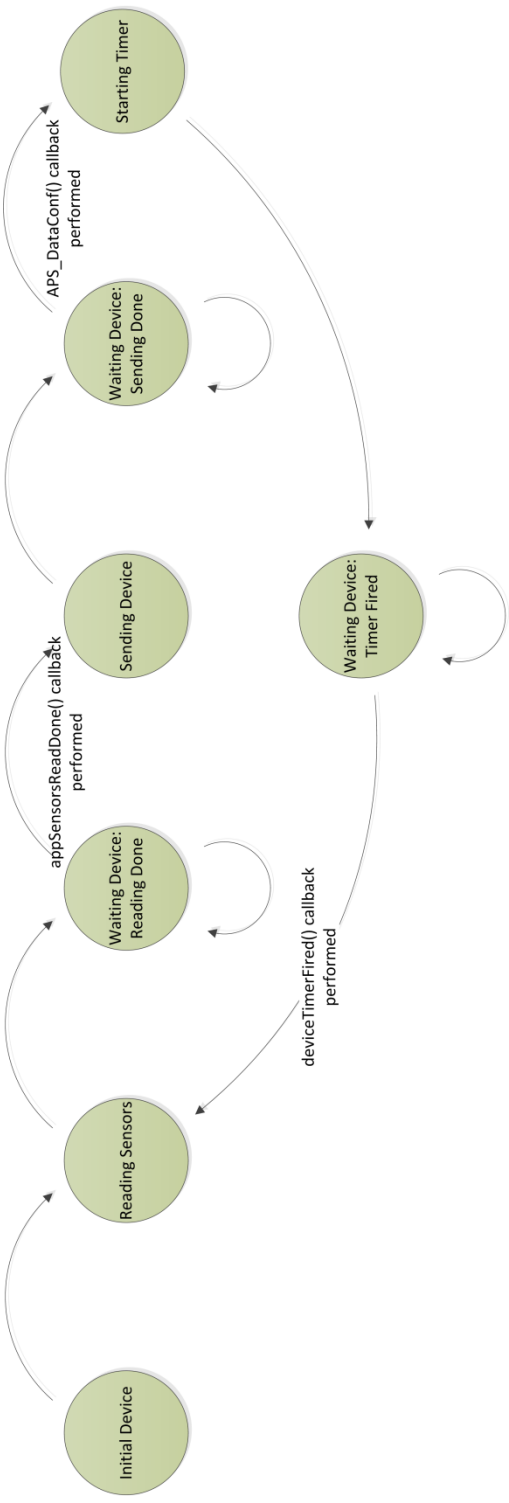
```
/**************************************************************************//**
  \file WSNVisualizer.h

  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
******************************************************************************/
#define _WSNVISUALIZER_H

void visualizeAppStarting(void);
void visualizeNwkStarting(void);
void visualizeNwkStarted(void);
void visualizeNwkLeaving(void);
void visualizeNwkLeft(void);
void visualizeAirTxStarted(void);
void visualizeAirTxFInished(void);
void visualizeAirRxFinished(void);
void visualizeSerialTx(void);
void visualizeWakeUp(void);
void visualizeSleep(void);
```

Appendix G. Sensor Node Packet Format

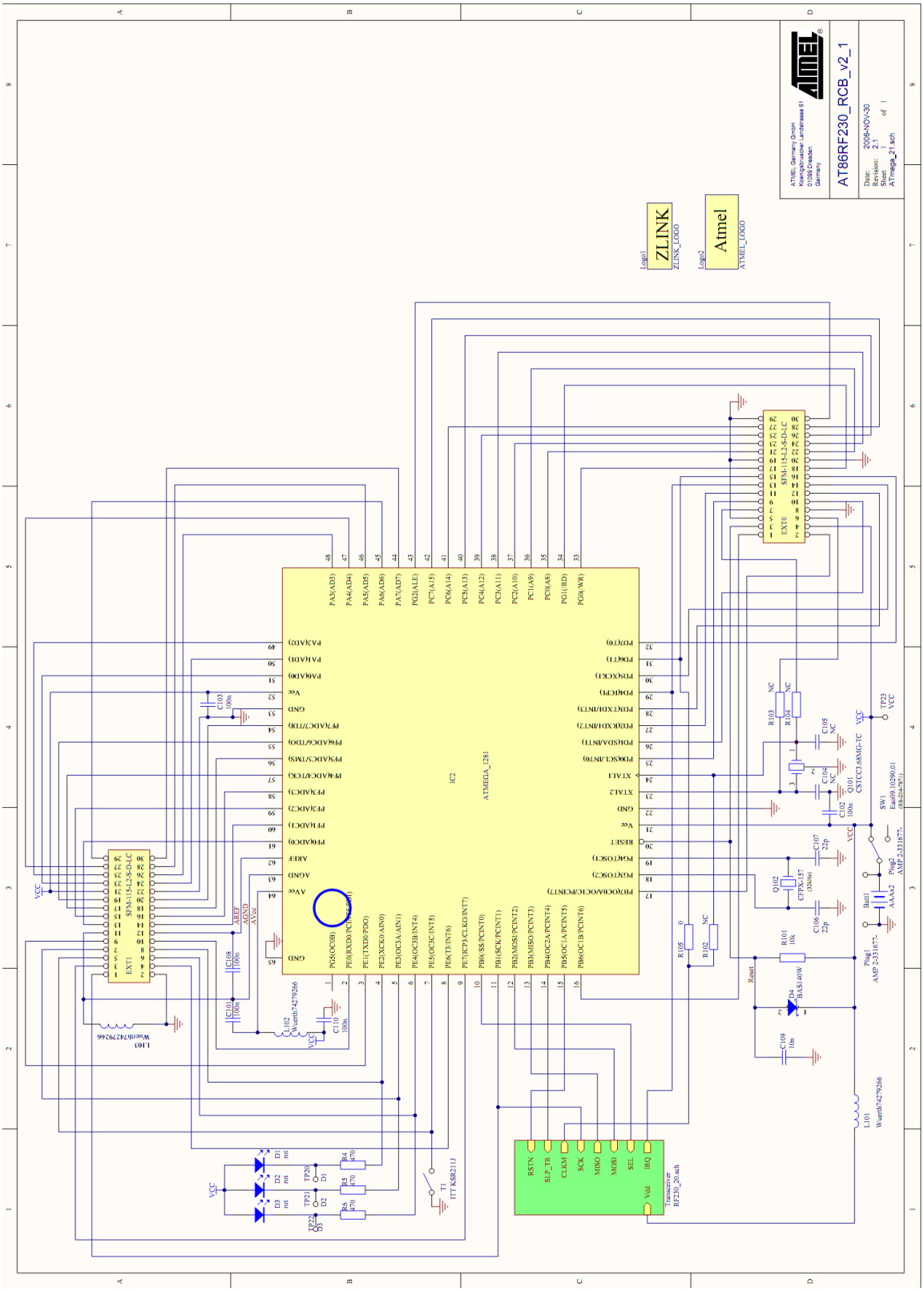


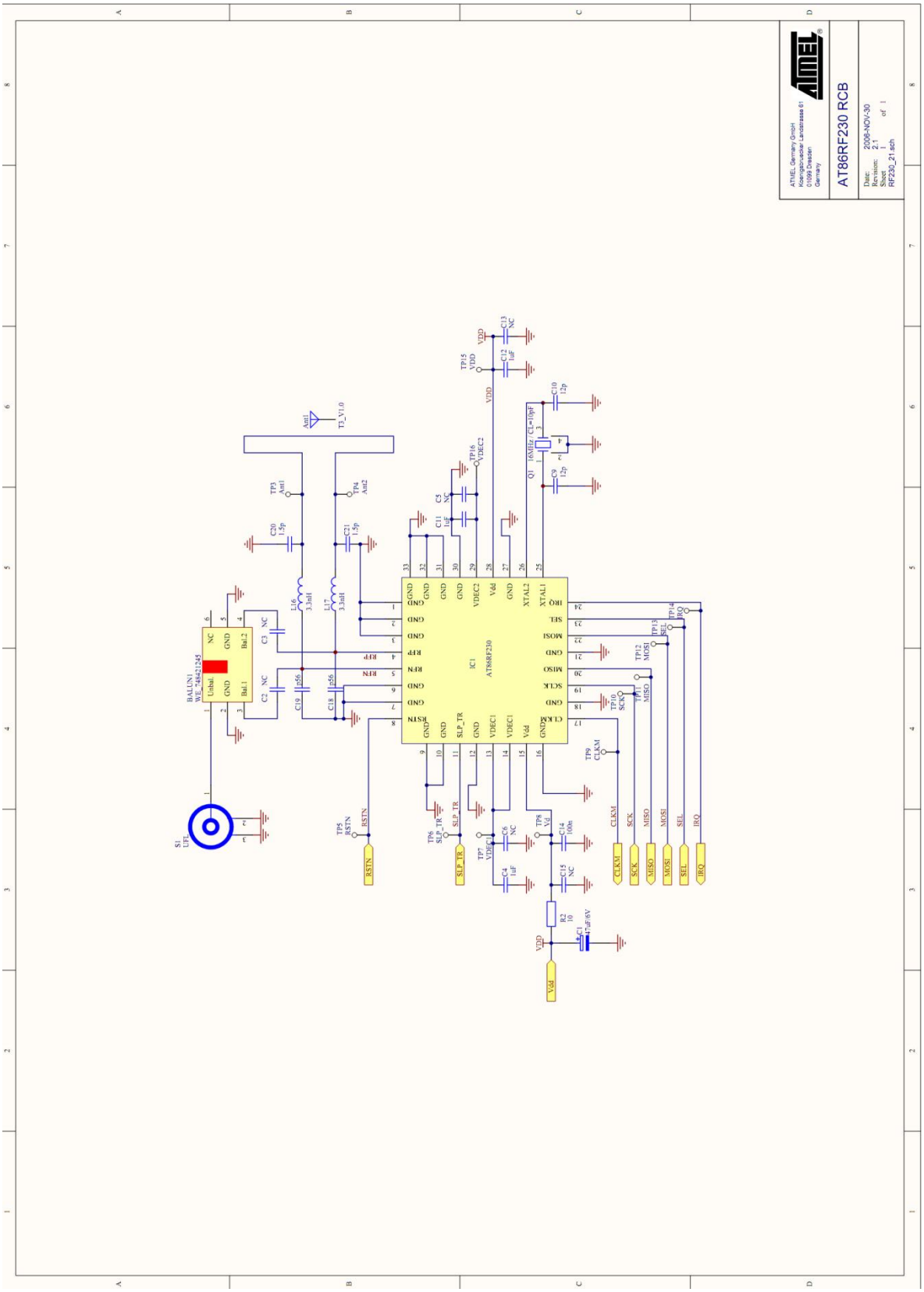
Appendix H. Sensor Node State Machines




Starting Timer
APS_DataConf() callback performed
Waiting Device: Sending Done
Sending Device
Waiting Device: Timer Fired
appSensorsReadDone() callback performed
Waiting Device: Reading Done
deviceTimerFired() callback performed
Reading Sensors
Initial Device

Appendix I. Coordinator Schematics




ATMEL Germany GmbH
Kaeslerstrasse 67
01069 Dresden
Germany
AT86RF230 RCB
Date: 2006-NOV-30
Revision: 2.1
Sheet 1 of 1
RF230_21.sch
IC1
AT86RF230
BALUN1
WE-748421245
S1
UFL
Ant1
T3_V1.0
Q1
16MHz CL=10pF
VDD
Vdd
RSTN
SLP_TR
CLKM
SCK
MISO
MOSI
SEL
IRQ

Appendix J. Coordinator Firmware Code

WSNCoordApp.c

```
/**************************************************************************//**
  \file WSNCoordApp.c
  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
  History:
    13/06/07 I. Kalganova - Modified
        10/06/11 Jacob Power - Modified
******************************************************************************/

#include <WSNCoordApp.h>
#include <WSNVisualizer.h>
#include <taskManager.h>
#include <zdo.h>
#include <configServer.h>
#include <aps.h>
#include <mac.h>

#define APP_TIMER_STARTING_NETWORK 500 // Period of blinking during starting network
#define TIMER_WAKEUP 50 // Wakeup period

/******************************************************************************
  Types
******************************************************************************/
typedef enum
{
  START_BLINKING,
  STOP_BLINKING
} BlinkingAction_t;


/******************************************************************************
  Prototypes
******************************************************************************/
void APS_DataIndCoord(APS_DataInd_t *indData);
void APS_DataIndRouter(APS_DataInd_t *indData);
void APS_DataIndDevice(APS_DataInd_t *indData);
void appPostSubTaskTask(void);
void appPostGlobalTask(void);
/******************************************************************************
  Global variables
 ******************************************************************************/
AppState_t     appState        = APP_INITING_STATE;//application state
DeviceState_t  appDeviceState  = INITIAL_DEVICE_STATE;

AppMessageRequest_t  appMessage;
//request parameters for APS_DataReq primitive
APS_DataReq_t messageParams;
HAL_AppTimer_t deviceTimer;
uint8_t failedTransmission = 0;
//static bool globalTaskRequested = false;

/******************************************************************************
  Local variables
```

```
 ******************************************************************************/
static DeviceType_t deviceType;
static ZDO_StartNetworkReq_t networkParams; //request params for ZDO_StartNetworkReq
//endpoint parameters
SimpleDescriptor_t simpleDescriptor;
static APS_RegisterEndpointReq_t endpointParams;
// Timer indicating starting network
static HAL_AppTimer_t startingNetworkTimer;
static ZDO_ZdpReq_t leaveReq;

/******************************************************************************
  Local functions
 ******************************************************************************/
static void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t *confirmInfo);
static void startingNetworkTimerFired(void);
static void zdpLeaveResp(ZDO_ZdpResp_t *zdpResp);
static void initSubStateMachine(void);
static void appTaskHandler(AppEvent_t event, void *param);
static void manageBlinkingDuringRejoin(BlinkingAction_t action);

/******************************************************************************
  Implementation
 ******************************************************************************/


/******************************************************************************
  Application task.

  Parameters:
    none

  Return:
    none

 ******************************************************************************/
void APL_TaskHandler(void)
{
  appTaskHandler(APP_PROCESS, NULL);
}

static void appTaskHandler(AppEvent_t event, void *param)
{
  bool rxOnWhenIdleFlag = true;

  switch (appState)
  {
    //node is in network
    case APP_IN_NETWORK_STATE:
    {
      switch (event)
      {
        case APP_PROCESS:
        {
            appCoordinatorTaskHandler(APP_PROCESS, NULL);
        }

        case APP_NETWORK_STATE_UPDATED:
        {
          ZDO_MgmtNwkUpdateNotf_t *updateParam = param;

          switch (updateParam->status)
          {
            case ZDO_NETWORK_LOST_STATUS:
              APL_WRITE_LOG(0x03)
              appState = APP_STOP_STATE;
              break;

            case ZDO_NETWORK_LEFT_STATUS:
              APL_WRITE_LOG(0x04)
```

```
            appState = APP_STARTING_NETWORK_STATE;
            visualizeNwkLeft();
            failedTransmission = 0;
            appPostGlobalTask();
            break;

          case ZDO_NWK_UPDATE_STATUS:
          case ZDO_NETWORK_STARTED_STATUS:
            // Update parameters being sent to coordinator
            appMessage.data.shortAddr       = updateParam->nwkUpdateInf.shortAddr;
            appMessage.data.panID           = updateParam->nwkUpdateInf.panId;
            appMessage.data.parentShortAddr = updateParam->nwkUpdateInf.parentShortAddr;
            appMessage.data.workingChannel  = updateParam->nwkUpdateInf.currentChannel;
            break;

          default:
            break;
        }
      }

      default:
        break;
    }
    break;
  }

  case APP_INITING_STATE: //node has initial state
  {
    switch (event)
    {
      case APP_PROCESS:
      {
        visualizeAppStarting();
          appStartUsartManager();  //init USART manager

        simpleDescriptor.endpoint = WSNDEMO_ENDPOINT;
        simpleDescriptor.AppProfileId = WSNDEMO_PROFILE_ID;
        simpleDescriptor.AppDeviceId = WSNDEMO_DEVICE_ID;
        simpleDescriptor.AppDeviceVersion = WSNDEMO_DEVICE_VERSION;

        endpointParams.simpleDescriptor = &simpleDescriptor;

          appMessage.data.nodeType = DEVICE_TYPE_COORDINATOR;
          endpointParams.APS_DataInd = APS_DataIndCoord;

        // For not coordinator CS_UID is set to equal UID chip by ZDO before
        { // To remove unalignment access warning for ARM.
          ExtAddr_t extAddr;

          CS_ReadParameter(CS_UID_ID, &extAddr);
          appMessage.data.extAddr = extAddr;
        }
        APS_RegisterEndpointReq(&endpointParams);

        // it is needed because devType is 2 bytes and nodeType is 1 byte
        deviceType = appMessage.data.nodeType;
        CS_WriteParameter(CS_DEVICE_TYPE_ID, &deviceType);

        if (DEVICE_TYPE_END_DEVICE == appMessage.data.nodeType)
        {
          rxOnWhenIdleFlag = false;
          CS_WriteParameter(CS_RX_ON_WHEN_IDLE_ID, &rxOnWhenIdleFlag);
        }
        else
        {
          rxOnWhenIdleFlag = true;
          CS_WriteParameter(CS_RX_ON_WHEN_IDLE_ID, &rxOnWhenIdleFlag);
        }
```

```
        APL_WRITE_LOG(0x00)

        appMessage.data.messageType     = APP_DATA_MESSAGE_TYPE;

        startingNetworkTimer.interval = APP_TIMER_STARTING_NETWORK;
        startingNetworkTimer.mode     = TIMER_REPEAT_MODE;
        startingNetworkTimer.callback = startingNetworkTimerFired;

        appState = APP_STARTING_NETWORK_STATE;
        appPostGlobalTask();
        break;
      }

      default:
        break;
    }
    break;
  }

  case APP_STARTING_NETWORK_STATE:
  {
    switch (event)
    {
      case APP_PROCESS:
        manageBlinkingDuringRejoin(START_BLINKING);
        visualizeNwkStarting();
        failedTransmission = 0;

        APL_WRITE_LOG(0x01)
        // Start network
        networkParams.ZDO_StartNetworkConf = ZDO_StartNetworkConf;
        ZDO_StartNetworkReq(&networkParams);
        break;

      case APP_NETWORK_STARTING_DONE:
      {
        ZDO_StartNetworkConf_t *startInfo = param;
        manageBlinkingDuringRejoin(STOP_BLINKING);
        if (ZDO_SUCCESS_STATUS == startInfo->status)
        {
          APL_WRITE_LOG(0x02)
          appState = APP_IN_NETWORK_STATE;
          failedTransmission = 0;

          visualizeNwkStarted();
          // Network parameters, such as short address, should be saved
          appMessage.data.panID           = startInfo->PANId;
          appMessage.data.shortAddr       = startInfo->shortAddr;
          appMessage.data.parentShortAddr = startInfo->parentAddr;
          appMessage.data.workingChannel  = startInfo->activeChannel;

          initSubStateMachine();
        }
        else
          appPostGlobalTask();
        break;
      }

      case APP_NETWORK_STATE_UPDATED:
      {
        ZDO_MgmtNwkUpdateNotf_t *updateParam = param;
        switch (updateParam->status)
        {
          case ZDO_NETWORK_STARTED_STATUS:
            APL_WRITE_LOG(0x0C)
            // Network parameters, such as short address, should be saved
            appMessage.data.panID           = updateParam->nwkUpdateInf.panId;
            appMessage.data.shortAddr       = updateParam->nwkUpdateInf.shortAddr;
            appMessage.data.parentShortAddr = updateParam->nwkUpdateInf.parentShortAddr;
```

```
            appMessage.data.workingChannel  = updateParam->nwkUpdateInf.currentChannel;
            manageBlinkingDuringRejoin(STOP_BLINKING);

            appState = APP_IN_NETWORK_STATE;
            failedTransmission = 0;
            visualizeNwkStarted();

            initSubStateMachine();
            break;
          default:
            break;
        }
      }
      default: break;
    }
    break;
  }
  case APP_LEAVING_NETWORK_STATE:
  {
    ZDO_MgmtLeaveReq_t *zdpLeaveReq = &leaveReq.req.reqPayload.mgmtLeaveReq;

    switch(event)
    {
      case APP_PROCESS:
        APL_WRITE_LOG(0x0A)
        visualizeNwkLeaving();
        leaveReq.ZDO_ZdpResp =  zdpLeaveResp;
        leaveReq.reqCluster = MGMT_LEAVE_CLID;
        leaveReq.dstAddrMode = EXT_ADDR_MODE;
        leaveReq.dstExtAddr = 0;
        zdpLeaveReq->deviceAddr = 0;
        zdpLeaveReq->rejoin = 0;
        zdpLeaveReq->removeChildren = 1;
        zdpLeaveReq->reserved = 0;
        ZDO_ZdpReq(&leaveReq);
        break;

      case APP_LEAVE_DONE:
        if (ZDO_SUCCESS_STATUS == ((ZDO_ZdpResp_t *)param)->respPayload.status)
        {
          APL_WRITE_LOG(0x0B)
          appState = APP_STOP_STATE;
        }
        appPostGlobalTask();
        break;

      case APP_NETWORK_STATE_UPDATED:
      {
        ZDO_MgmtNwkUpdateNotf_t *updateParam = param;

        switch (updateParam->status)
        {
          case ZDO_NETWORK_LOST_STATUS:
            APL_WRITE_LOG(0x03)
            appState = APP_STOP_STATE;
            break;
          case ZDO_NETWORK_LEFT_STATUS:
            APL_WRITE_LOG(0x04)
            appState = APP_STARTING_NETWORK_STATE;
            appPostGlobalTask();
            break;
          default:
            break;
        }
        break;
      }

      default:
        break;
```

```
      }
      break;
    }

    case APP_STOP_STATE:
    {
      switch(event)
      {
        case APP_NETWORK_STATE_UPDATED:
        {
          ZDO_MgmtNwkUpdateNotf_t *updateParam = param;

          switch (updateParam->status)
          {
            case ZDO_NETWORK_STARTED_STATUS:
              APL_WRITE_LOG(0x0C)
              appState = APP_IN_NETWORK_STATE;
              appMessage.data.shortAddr       = updateParam->nwkUpdateInf.shortAddr;
              appMessage.data.panID           = updateParam->nwkUpdateInf.panId;
              appMessage.data.parentShortAddr = updateParam->nwkUpdateInf.parentShortAddr;
              appMessage.data.workingChannel  = updateParam->nwkUpdateInf.currentChannel;
              appPostGlobalTask();
              break;

            case ZDO_NETWORK_LEFT_STATUS:
              APL_WRITE_LOG(0x04)
              appState = APP_STARTING_NETWORK_STATE;
              appPostGlobalTask();
              break;

            default:
              break;
          }
        }
        default:
          break;
      }
      break;
    }

    default:
      break;
  }
}

// The response means that the command has been received successfully but not precessed yet
static void zdpLeaveResp(ZDO_ZdpResp_t *zdpResp)
{
  appTaskHandler(APP_LEAVE_DONE, zdpResp);
}

/**************************************************************************************
  ZDO_StartNetwork primitive confirmation was received.

  Parameters:
    confirmInfo - confirmation information

  Return:
    none

 **************************************************************************************/
static void ZDO_StartNetworkConf(ZDO_StartNetworkConf_t *confirmInfo)
{
  appTaskHandler(APP_NETWORK_STARTING_DONE, confirmInfo);
}


void ZDO_MgmtNwkUpdateNotf(ZDO_MgmtNwkUpdateNotf_t *nwkParams)
{
```

```
    appTaskHandler(APP_NETWORK_STATE_UPDATED, nwkParams);
}

void ZDO_WakeUpInd(void)
{

}

// For blinking
static void startingNetworkTimerFired(void)
{
  visualizeNwkStarting();
}

void initSubStateMachine(void)
{
    appInitDeviceCoordinator();
}

void appReadLqiRssi(void)
{
  ZDO_GetLqiRssi_t lqiRssi;

  lqiRssi.nodeAddr = appMessage.data.parentShortAddr;
  ZDO_GetLqiRssi(&lqiRssi);

  appMessage.data.lqi  = lqiRssi.lqi;
  appMessage.data.rssi = lqiRssi.rssi;
}

static void manageBlinkingDuringRejoin(BlinkingAction_t action)
{
  static bool run = false;

  if (START_BLINKING == action)
  {
    if (!run)
    {
      HAL_StartAppTimer(&startingNetworkTimer);
      run = true;
    }
  }

  if (STOP_BLINKING == action)
  {
    run = false;
    HAL_StopAppTimer(&startingNetworkTimer);
  }
}

void appPostGlobalTask(void)
{
  SYS_PostTask(APL_TASK_ID);
}

void appPostSubTaskTask(void)
{
  SYS_PostTask(APL_TASK_ID);
}

void appLeaveNetwork(void)
{
  if (APP_IN_NETWORK_STATE == appState)
  {
    appState = APP_LEAVING_NETWORK_STATE;
    appPostGlobalTask();
  }
}
#ifdef _BINDING_
```

```
/******************************************************************************
  Stub for ZDO Binding Indication

  Parameters:
    bindInd - indication

  Return:
    none

 ******************************************************************************/
void ZDO_BindIndication(ZDO_BindInd_t *bindInd)
{
  (void)bindInd;
}

/******************************************************************************
  Stub for ZDO Unbinding Indication

  Parameters:
    unbindInd - indication

  Return:
    none

 ******************************************************************************/
void ZDO_UnbindIndication(ZDO_UnbindInd_t *unbindInd)
{
  (void)unbindInd;
}
#endif //_BINDING_
//eof WSNCoordApp.c
```

WSNCoordApp.h

```
/*****************************************************************************//**
  \file WSNCoordApp.h

  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
******************************************************************************/
#ifndef _WSNCoordApp_H
#define _WSNCoordApp_H

/******************************************************************************
                    Includes section
******************************************************************************/
#include <macAddr.h>
#include <appFramework.h>
#include <configServer.h>
#include <appTimer.h>
#include <aps.h>
#include <uid.h>
#include <appTimer.h>
#include <zdo.h>
#include <dbg.h>
#include "leds.h"

  #define appOpenLeds() BSP_OpenLeds()
```

```
  #define appCloseLeds() BSP_CloseLeds()
  #define appOnLed(id) BSP_OnLed(id)
  #define appOffLed(id) BSP_OffLed(id)
  #define appToggleLed(id) BSP_ToggleLed(id)

/************************************************************************
                    Defines section
************************************************************************/

#define DEVICE1_EXT_ADDR      0x1ULL //0x000100001090C993LL
#define DEVICE2_EXT_ADDR      0x2ULL
#define COORDINATOR_EXT_ADDR  CS_APS_TRUST_CENTER_ADDRESS
#define LINK_KEY {0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
0xaa, 0xaa}


#define WSNDEMO_PROFILE_ID CPU_TO_LE16(1)
#define WSNDEMO_DEVICE_ID CPU_TO_LE16(1)
#define WSNDEMO_DEVICE_VERSION 1
#define WSNDEMO_ENDPOINT 1

#define MAX_USART_MESSAGE_QUEUE_COUNT 5
#define MAX_DEVICE_MESSAGE_LEN        10
#define APP_DATA_MESSAGE_TYPE         1
/* At worst each byte will be transformed to two bytes + size of crc
 * + size of 4 bytes: 0x10, 0x02, 0x10, 0x03. */
#define MAX_RAW_APP_MESSAGE_SIZE  (2 + 2 * sizeof(AppMessage_t) + 2 + 1)

#ifndef APP_TIMER_SENDING_PERIOD
#define APP_TIMER_SENDING_PERIOD         1000
#endif

typedef enum
{
  APP_PROCESS,
  APP_NETWORK_STARTING_DONE,
  APP_NETWORK_STATE_UPDATED,
  APP_LEAVE_DONE,
  APP_SENDING_DONE,
  APP_READING_DONE,
  APP_SLEEP_DONE,
  APP_WOKEUP,
  APP_TIMER_FIRED,
} AppEvent_t;


typedef enum
{
  APP_INITING_STATE,
  APP_STARTING_NETWORK_STATE,
  APP_IN_NETWORK_STATE,
  APP_LEAVING_NETWORK_STATE,
  APP_STOP_STATE
} AppState_t;

typedef enum
{
  INITIAL_DEVICE_STATE,
  SENDING_DEVICE_STATE,
  READING_SENSORS_STATE,
  SLEEPPING_DEVICE_STATE,
  STARTING_TIMER_STATE,
  WAITING_DEVICE_STATE,
  REINITIAL_DEVICE_STATE
} DeviceState_t;

BEGIN_PACK
typedef struct
{
```

```
   uint8_t     messageType;
   uint8_t     nodeType;
   ExtAddr_t   extAddr;
   ShortAddr_t shortAddr;
   PanId_t     panID;
   uint8_t     workingChannel;
   ShortAddr_t parentShortAddr;
   uint8_t     lqi;
   int8_t      rssi;

   uint16_t    battery;
   uint16_t    temp;
   uint32_t    time;
   uint32_t    date;
   uint16_t    analog1;
   uint16_t    analog2;
   uint8_t     cc1;
   uint8_t     cc2;
} PACK AppMessage_t;

typedef struct
{
  uint8_t        header[APS_ASDU_OFFSET];
  AppMessage_t   data;
  uint8_t        footer[APS_AFFIX_LENGTH - APS_ASDU_OFFSET];
} PACK AppMessageRequest_t;
END_PACK

extern uint16_t       appPANId;
extern ShortAddr_t    appShortAddr;
extern uint8_t        appChannel;
extern DeviceState_t  appDeviceState;
extern DeviceType_t   appDeviceType;
extern ShortAddr_t    appParentAddr;
extern uint64_t       appUid;
extern SimpleDescriptor_t simpleDescriptor;
extern uint8_t        failedTransmission;

/*****************************************************************************
                    Functions' prototypes section
*****************************************************************************/
extern void appStartSensorManager(void);
extern void appStopSensorManager(void);
extern void appGetSensorData(void (*passedCB)(void));
extern void appCoordTaskHandler(void);
extern void appFillAppMessage(AppMessage_t *newMessagePoint);
extern void appStartUsartManager(void);
extern void appPostSubTaskTask(void);
extern void SYS_InitLog(void);
extern void appReadLqiRssi(void);
extern void appInitDeviceCoordinator(void);
extern void SYS_StartLog(void);
extern void SYS_StopLog(void);
extern void appSendMessageToUsart(AppMessage_t *newMessage);
extern void appCoordinatorTaskHandler(AppEvent_t event, void *param);

#endif /*_WSNCoordApp_H*/

// eof WSNCoordApp.h
```

WSNCoord.c

```
/*****************************************************************************//**
  \file WSNCoordinatorApp.c

  \brief
```

```
  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
  History:
    13/06/07 I. Kalganova - Modified
*****************************************************************************/
#ifdef _COORDINATOR_

#include <WSNCoordApp.h>
#include <WSNVisualizer.h>

extern AppMessageRequest_t  appMessage;
extern HAL_AppTimer_t deviceTimer;
extern bool subTaskRequested;

static void deviceTimerFired(void);
static void appSensorsReadDone(void);
void appInitDeviceCoordinator(void);

/*****************************************************************************
  Process coordinator functionality.

  Parameters:
    none

  Return:
    none

 *****************************************************************************/
void appCoordinatorTaskHandler(AppEvent_t event, void *param)
{
  param = param; // warning prevention

  switch (appDeviceState)
  {
    case READING_SENSORS_STATE:
      switch (event)
      {
        case APP_PROCESS:
          appReadLqiRssi();
          //appStartSensorManager();
          appGetSensorData(appSensorsReadDone);
          break;

        case APP_READING_DONE:
          appDeviceState = SENDING_DEVICE_STATE;
          //appStopSensorManager();
          appPostSubTaskTask();
          break;

        default:
          break;
      }
      break;

    case SENDING_DEVICE_STATE:
      switch (event)
      {
        case APP_PROCESS:
          visualizeSerialTx();
          appSendMessageToUsart(&appMessage.data);
          appDeviceState = STARTING_TIMER_STATE;
          appPostSubTaskTask();
          break;
```

```
        default:
          break;
      }
      break;

    case STARTING_TIMER_STATE:
      switch (event)
      {
        case APP_PROCESS:
          HAL_StartAppTimer(&deviceTimer);
          break;

        case APP_TIMER_FIRED:
          appDeviceState = READING_SENSORS_STATE;
          appPostSubTaskTask();
          break;

        default:
          break;
      }
      break;

    case WAITING_DEVICE_STATE:
      switch (event)
      {
        default:
          appDeviceState = INITIAL_DEVICE_STATE;
          appPostSubTaskTask();
          break;
      }
      break;

    case INITIAL_DEVICE_STATE:
      switch (event)
      {
        case APP_PROCESS:
          HAL_StopAppTimer(&deviceTimer);

          deviceTimer.interval = 300000;
          deviceTimer.mode     = TIMER_ONE_SHOT_MODE;
          deviceTimer.callback = deviceTimerFired;
          appDeviceState = READING_SENSORS_STATE;
          appPostSubTaskTask();
          break;

        default:
          break;
      }
      break;

    default:
      break;
  }
}


void appInitDeviceCoordinator(void)
{
  appDeviceState = INITIAL_DEVICE_STATE;
  appPostSubTaskTask();
}

static void appSensorsReadDone(void)
{
  appCoordinatorTaskHandler(APP_READING_DONE, NULL);
}

/*************************************************************************************
```

```
  Device timer event.

  Parameters:
    none

  Return:
    none

 ******************************************************************************/
static void deviceTimerFired(void)
{
  appCoordinatorTaskHandler(APP_TIMER_FIRED, NULL);
}


/******************************************************************************
  Radio data has been received. It should be resent into USART.

  Parameters:
    indData - received data parameters and payload

  Return:
    none

 ******************************************************************************/
void APS_DataIndCoord(APS_DataInd_t *indData)
{
  visualizeAirRxFinished();
  APL_WRITE_LOG(0x62)
  appSendMessageToUsart((AppMessage_t *)indData->asdu);
}

#endif

//eof WSNCoordinator.c
```

WSNSensorManager.c

```
/*****************************************************************************//**
  \file WSNSensorManager.c

  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
  History:
    13/06/07 I. Kalganova - Modified
   4/9/2011 J. Power
*****************************************************************************/

#include <types.h>
#include <taskManager.h>
#include <WSNCoordApp.h>
#include <WSNVisualizer.h>

extern AppMessageRequest_t  appMessage;

static void (*callback)(void);

//////////////////////////////////////////////////////////////////

void appStartSensorManager(void)
```

```
{

}

void appStopSensorManager()
{

}

void appGetSensorData(void (*passedCB)(void))
{
   callback = passedCB;
   appMessage.data.battery = (int32_t) 100;
   appMessage.data.temp = (int16_t) 0;
   appMessage.data.analog1 = (int16_t) 0;
   appMessage.data.analog2 = (int16_t) 0;
   appMessage.data.cc1 = (uint8_t) 0;
   appMessage.data.cc2 = (uint8_t) 0;
   callback();
}
```

WSNVisualizer.c

```
/*****************************************************************************//**
  \file WSNVisualizer.c

  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
  History:
    18/12/08 A. Luzhetsky - Modified
*****************************************************************************/

/******************************************************************************
******************************************************************************/
#include <WSNCoordApp.h>

/******************************************************************************
******************************************************************************/
void visualizeAppStarting(void);
void visualizeNwkStarting(void);
void visualizeNwkStarted(void);
void visualizeNwkLeaving(void);
void visualizeNwkLeft(void);
void visualizeAirTxStarted(void);
void visualizeAirTxFInished(void);
void visualizeAirRxFinished(void);
void visualizeSerialTx(void);
void visualizeWakeUp(void);
void visualizeSleep(void);

/******************************************************************************
******************************************************************************/
void visualizeAppStarting(void)
{
  appOpenLeds();

}

/******************************************************************************
```

```
*****************************************************************************/
void visualizeNwkStarting(void)
{
  appToggleLed(LED_RED);
  appOffLed(LED_YELLOW);
  appOffLed(LED_GREEN);
}

/*****************************************************************************
*****************************************************************************/
void visualizeNwkStarted(void)
{
  appOnLed(LED_RED);
  appOffLed(LED_YELLOW);
  appOffLed(LED_GREEN);
}

/*****************************************************************************
*****************************************************************************/
void visualizeNwkLeaving(void)
{
  appOffLed(LED_GREEN);
}

/*****************************************************************************
*****************************************************************************/
void visualizeNwkLeft(void)
{
  appOffLed(LED_GREEN);
}

/*****************************************************************************
*****************************************************************************/
void visualizeAirTxStarted(void)
{
  appOnLed(LED_GREEN);
}

/*****************************************************************************
*****************************************************************************/
void visualizeAirTxFInished(void)
{
  appOffLed(LED_GREEN);
}

/*****************************************************************************
*****************************************************************************/
void visualizeAirRxFinished(void)
{
  appToggleLed(LED_YELLOW);
}

/*****************************************************************************
*****************************************************************************/
void visualizeSerialTx(void)
{
  appToggleLed(LED_GREEN);
}

/*****************************************************************************
*****************************************************************************/
void visualizeWakeUp(void)
{
  appOpenLeds();
  appOnLed(LED_RED);
}

/*****************************************************************************
*****************************************************************************/
```

```
void visualizeSleep(void)
{
  appCloseLeds();

}


// eof WSNVisualizer.c
```

WSNVisualizer.h

```
/*************************************************************************//**
  \file WSNVisualizer.h

  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
*****************************************************************************/
#ifndef _WSNVISUALIZER_H
#define _WSNVISUALIZER_H

void visualizeAppStarting(void);
void visualizeNwkStarting(void);
void visualizeNwkStarted(void);
void visualizeNwkLeaving(void);
void visualizeNwkLeft(void);
void visualizeAirTxStarted(void);
void visualizeAirTxFInished(void);
void visualizeAirRxFinished(void);
void visualizeSerialTx(void);
void visualizeWakeUp(void);
void visualizeSleep(void);

#endif//  _WSNVISUALIZER_H

// eof WSNVisualizer.h
```

WSNUARTManager.c

```
/*************************************************************************//**
  \file USARTManager.c

  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
  History:
    13/06/07 I. Kalganova - Modified
        10/06/11 Jacob Power - Modified
*****************************************************************************/
#ifdef _COORDINATOR_
```

```
#include <types.h>
#include <usart.h>
#include <WSNCoordApp.h>
#include <WSNVisualizer.h>
#include <serialInterface.h>

typedef struct
{
  uint8_t payload[MAX_RAW_APP_MESSAGE_SIZE];
  uint8_t size;
} UsartMessage_t;


static struct
{
  UsartMessage_t usartMessageQueue[MAX_USART_MESSAGE_QUEUE_COUNT];
  bool isFreeUsart;
  uint8_t head,tail,size;
} wsn2usart;

static void readByteEvent(uint16_t readBytesLen);
static void writeConfirm(void);
static void sendNextMessage(void);
static HAL_UsartDescriptor_t usartDescriptor;
#if 0 != USART_RX_BUFFER_LENGTH
  static uint8_t rxBuffer[USART_RX_BUFFER_LENGTH];
#else
  static uint8_t *rxBuffer = NULL;
#endif

/*****************************************************************************
  Init USART, register USART callbacks.

  Parameters:
    none

  Return:
    none

 *****************************************************************************/
void appStartUsartManager(void)
{
  usartDescriptor.tty            = USART_CHANNEL_0;
  usartDescriptor.mode           = USART_MODE_ASYNC;
  usartDescriptor.flowControl    = USART_FLOW_CONTROL_NONE;
  usartDescriptor.baudrate       = USART_BAUDRATE_38400;
  usartDescriptor.dataLength     = USART_DATA8;
  usartDescriptor.parity         = USART_PARITY_NONE;
  usartDescriptor.stopbits       = USART_STOPBIT_1;
  usartDescriptor.rxBuffer       = rxBuffer;
  usartDescriptor.rxBufferLength = USART_RX_BUFFER_LENGTH;
  usartDescriptor.txBuffer       = NULL;
  usartDescriptor.txBufferLength = 0;
  usartDescriptor.rxCallback     = readByteEvent;
  usartDescriptor.txCallback     = writeConfirm;

  OPEN_USART(&usartDescriptor);

  memset(&wsn2usart, 0, sizeof(wsn2usart));
  wsn2usart.isFreeUsart = true;
}

void appStopUsartManager(void)
{
  CLOSE_USART(&usartDescriptor);
}

/*****************************************************************************
  New USART bytes were received.
```

```
  Parameters:
    readBytesLen - count of received bytes

  Return:
    none

 ******************************************************************************/
static void readByteEvent(uint16_t readBytesLen)
{
  readBytesLen = readBytesLen;
}

/******************************************************************************
  Send next message from queue.

  Parameters:
    none

  Return:
    none

 ******************************************************************************/
static void sendNextMessage(void)
{
  if (wsn2usart.size)
  {
    WRITE_USART(&usartDescriptor,
      wsn2usart.usartMessageQueue[wsn2usart.head].payload,
      wsn2usart.usartMessageQueue[wsn2usart.head].size
    );
  }
}

/******************************************************************************
  Writing confirmation has been received. New message can be sent.

  Parameters:
    none

  Return:
    none

 ******************************************************************************/
static void writeConfirm(void)
{
  if (wsn2usart.size)
  {
    wsn2usart.size--;
    if (++wsn2usart.head >= MAX_USART_MESSAGE_QUEUE_COUNT)
      wsn2usart.head -= MAX_USART_MESSAGE_QUEUE_COUNT;
  }
  wsn2usart.isFreeUsart = true;
  //send next message
  sendNextMessage();
}

/******************************************************************************
  New message being sent into USART has to be put into queue.

  Parameters:
    newMessage - new message fields.

  Return:
    none

 ******************************************************************************/
void appSendMessageToUsart(AppMessage_t *newMessage)
{
```

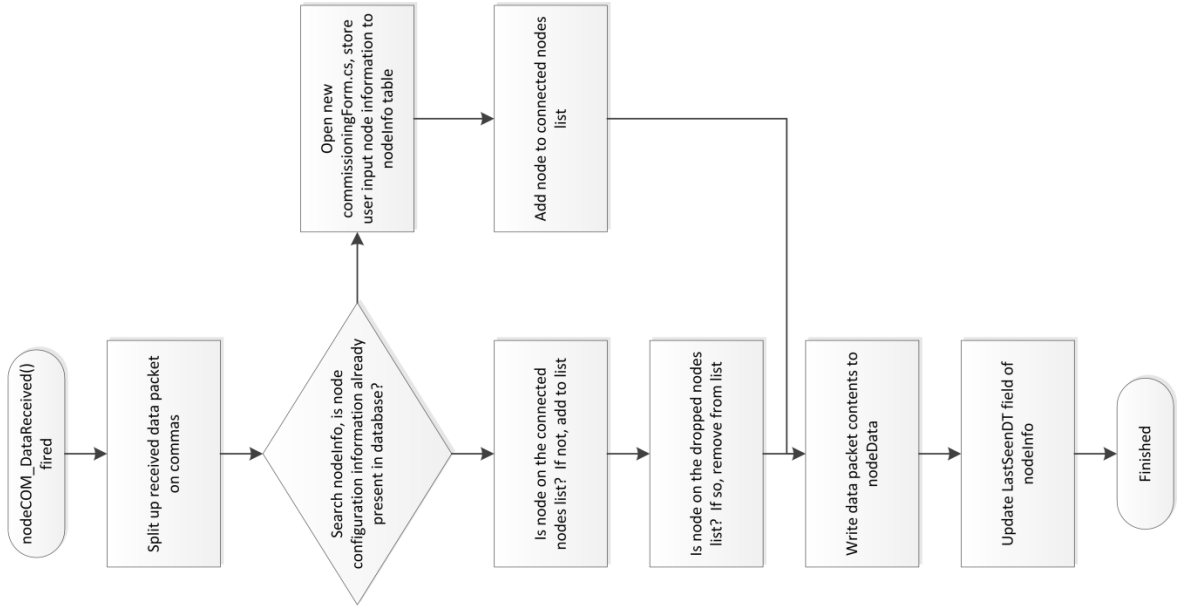
```
  UsartMessage_t *msg;

  if (wsn2usart.size < MAX_USART_MESSAGE_QUEUE_COUNT)
  {
    wsn2usart.size++;
    msg = &wsn2usart.usartMessageQueue[wsn2usart.tail];
    if (++wsn2usart.tail >= MAX_USART_MESSAGE_QUEUE_COUNT)
      wsn2usart.tail -= MAX_USART_MESSAGE_QUEUE_COUNT;

    sprintf(msg->payload, "%x,%x,0,%u,%u,%u,%u,%u,%u,%u,%u\n", newMessage->shortAddr,
      newMessage->parentShortAddr, newMessage->lqi, newMessage->rssi, newMessage->battery,
      newMessage->temp, newMessage->analog1, newMessage->analog2, newMessage->cc1,
      newMessage->cc2);

    msg->size = strlen(msg->payload);
  }

  //check sending state
  if (true == wsn2usart.isFreeUsart)
  {
    sendNextMessage();
  }
}

#endif
//eof UARTManager.c
```

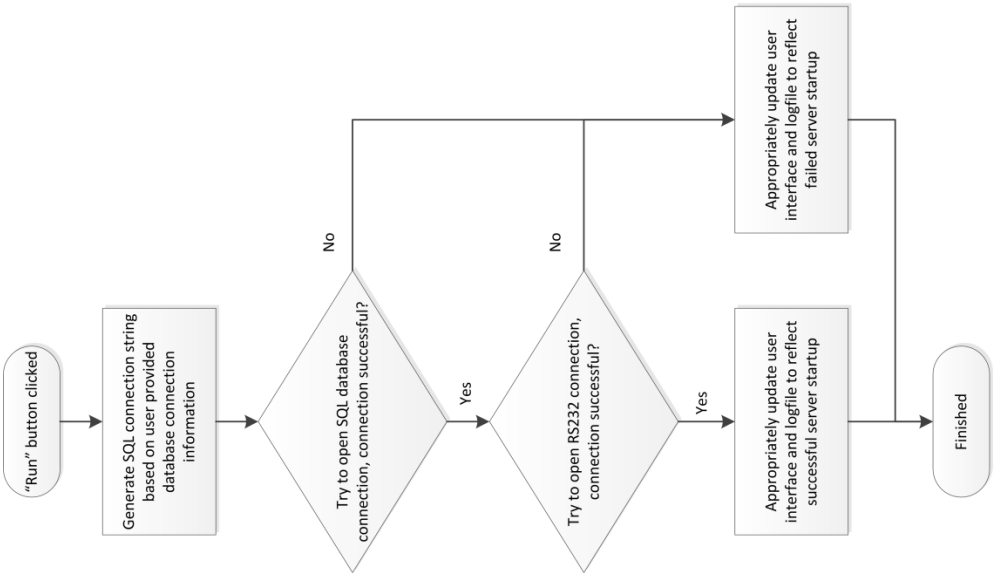
serialInterface.h

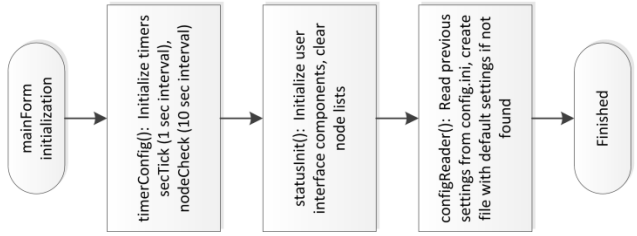
```
/**************************************************************************//**
  \file serialInterface.h

  \brief

  \author
    Atmel Corporation: http://www.atmel.com \n
    Support email: avr@atmel.com

  Copyright (c) 2008 , Atmel Corporation. All rights reserved.
  Licensed under Atmel's Limited License Agreement (BitCloudTM).

  \internal
******************************************************************************/
#ifndef _SERIALINTERFACE_H
#define _SERIALINTERFACE_H

/******************************************************************************
                    Includes section
******************************************************************************/
#include <stdio.h>
#include <string.h>
#include <usart.h>
#include <rs232Controller.h>

/******************************************************************************
                    Defines section
******************************************************************************/
#define APP_INTERFACE_USART 0x01
#define APP_INTERFACE_VCP 0x02

#if APP_INTERFACE == APP_INTERFACE_VCP
  #include <vcpVirtualUsart.h>
#endif // APP_INTERFACE_VCP

#ifndef APP_INTERFACE
  INLINE int OPEN_USART(HAL_UsartDescriptor_t *descriptor)\
  {\
```

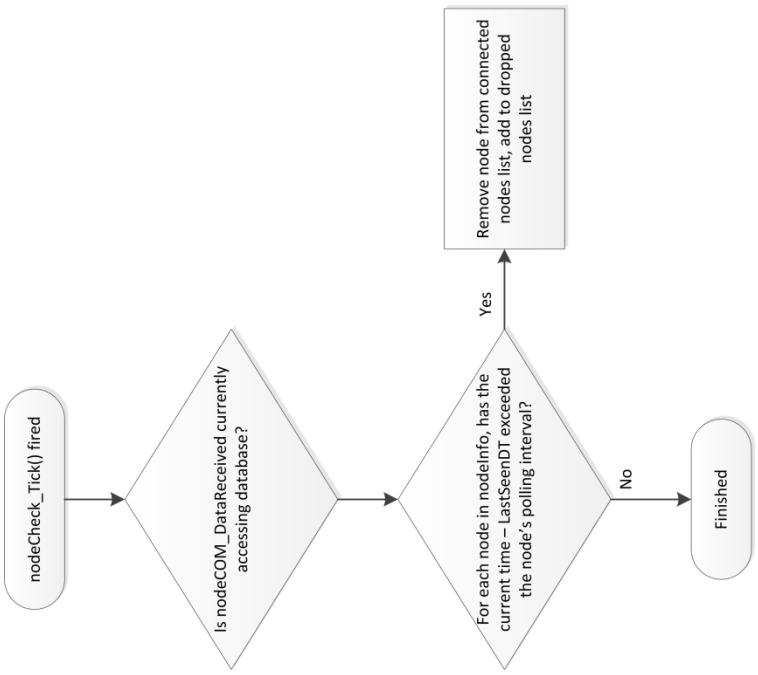
```
    (void)descriptor;\
    return 0;\
  }
  INLINE int CLOSE_USART(HAL_UsartDescriptor_t *descriptor)\
  {\
    (void)descriptor;\
    return 0;\
  }
  INLINE int WRITE_USART(HAL_UsartDescriptor_t *descriptor, uint8_t *buffer, uint8_t length)\
  {\
    (void)descriptor;\
    (void)buffer;\
    return length;\
  }
  INLINE int READ_USART(HAL_UsartDescriptor_t *descriptor, uint8_t *buffer, uint8_t length)\
  {\
    (void)descriptor;\
    (void)buffer;\
    (void)length;\
    return 0;\
  }
  #define USART_CHANNEL         APP_USART_CHANNEL
  #define USART_RX_BUFFER_LENGTH 0
#endif // APP_INTERFACE_USART

#if APP_INTERFACE == APP_INTERFACE_USART
  INLINE int OPEN_USART(HAL_UsartDescriptor_t *descriptor)\
  {\
    BSP_EnableRs232();
    return HAL_OpenUsart(descriptor);\
  }
  INLINE int CLOSE_USART(HAL_UsartDescriptor_t *descriptor)\
  {\
    BSP_DisableRs232();
    return HAL_CloseUsart(descriptor);\
  }
  #define WRITE_USART           HAL_WriteUsart
  #define READ_USART            HAL_ReadUsart
  #define USART_CHANNEL         APP_USART_CHANNEL
  #define USART_RX_BUFFER_LENGTH 0
#endif // APP_INTERFACE_USART

#if APP_INTERFACE == APP_INTERFACE_VCP
  #define OPEN_USART            VCP_OpenUsart
  #define CLOSE_USART           VCP_CloseUsart
  #define WRITE_USART           VCP_WriteUsart
  #define READ_USART            VCP_ReadUsart
  #define USART_CHANNEL         USART_CHANNEL_VCP
  #define USART_RX_BUFFER_LENGTH 64
#endif // APP_INTERFACE_VCP

#if APP_USART_SPEED == 9600
  #define USART_SPEED       USART_BAUDRATE_9600
#else
  #define USART_SPEED       USART_BAUDRATE_38400
#endif

#ifndef USART_CHANNEL
  #error USART interface is not defined.
#endif // OPEN_USART

#endif // _SERIALINTERFACE_H

// eof serialInterface.h
```

Appendix K. Monitoring and Logging Server Flowcharts








nodeCheck_Tick() fired
Is nodeCOM_DataReceived currently accessing database?
For each node in nodeInfo, has the current time – LastSeenDT exceeded the node's polling interval?
Yes
Remove node from connected nodes list, add to dropped nodes list
No
Finished

Appendix L. Monitoring and Logging Server Code

mainForm.cs

```
/**
 * mainForm.cs runs the main monitoring and data logging
 * logic for the WSN server.
 *
 * @author Jacob power
 */

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;
using System.IO.Ports;
using System.IO;
using System.Collections;
using System.Configuration;

namespace wsnManagementServer
{
    public partial class mainForm : Form
    {
        private String sqlServerLoc;
        private String sqlDBName;
        private String sqlUserID;
        private String sqlPwd;
        private String sqlConnType;
        private Boolean sqlIntSec;
        private String comPort;
        private Int32 comBaud;
        private Parity comParity;
        private Int32 comData;
        private StopBits comStop;
        private DateTime connDT;
        private Boolean busy = false;
        private List<String> connectedNodesList = new List<String>();
        private List<String> droppedNodesList = new List<String>();

        configForm ConfigWindow = new configForm();
        SqlConnection dbConnection;
        SerialPort nodeCOM;
        Timer secTick, nodeCheck;

        public mainForm()
        {
            InitializeComponent();
            timerConfig();
            statusInit();

            configReader();
            configWindowSync(false);
        }

        private void toolStripConfig_Click(object sender, EventArgs e)
        {
            if (DialogResult.OK == ConfigWindow.ShowDialog())
            {
                configWindowSync(true);
```

```
            configWriter();
        }
        else
        {
            configWindowSync(false);
        }
    }

    private void toolStripRun_Click(object sender, EventArgs e)
    {
        SqlConnectionStringBuilder dbConnString = new SqlConnectionStringBuilder();
        dbConnString.DataSource = sqlServerLoc;
        dbConnString.InitialCatalog = sqlDBName;
        dbConnString.UserID = sqlUserID;
        dbConnString.Password = sqlPwd;
        dbConnString.NetworkLibrary = sqlConnType;
        dbConnString.IntegratedSecurity = sqlIntSec;

        dbConnection = new SqlConnection(dbConnString.ConnectionString);
        dbConnection.StateChange += new StateChangeEventHandler(dbConnection_StateChange);

        try
        {
            dbConnection.Open();

        }
        catch (System.Data.SqlClient.SqlException f)
        {
            writeLog("SQL Connection failure: " + f.Message);
        }

        nodeCOM = new SerialPort(comPort, comBaud, comParity, comData, comStop);
        nodeCOM.DataReceived += new SerialDataReceivedEventHandler(nodeCOM_DataReceived);
        try
        {
            nodeCOM.Open();
            writeLog(comPort + " opened!");
        }
        catch (Exception g)
        {
            writeLog("Error: " + g.Message);
        }

        if ((dbConnection.State == ConnectionState.Open) & (nodeCOM.IsOpen))
        {
            statusRunning();
        }
        else
        {
            statusError();
        }
    }

    /**
     * nodeCOM_DataReceived handles data received events from the nodeCOM port,
     * and appropriately parses the incoming data string from the coordinator.
     */
    void nodeCOM_DataReceived(object sender, SerialDataReceivedEventArgs e)
    {
        String curShortAddr;
        Boolean newNode = true;
        //handle new data received
        //compare against nodeInfo - if node exists then continue, otherwise trash packet
        //push data up to DB
        String unparsedRXdata = "";
        SerialPort localPort = (SerialPort)sender;

        busy = true;
```

```
            unparsedRXdata = localPort.ReadLine();
            String[] splitRXdata = unparsedRXdata.Split(new char[] { ',' });
            SqlCommand dbCommand = dbConnection.CreateCommand();
            dbCommand.CommandText = @"SELECT ShortAddr FROM nodeInfo";
            SqlDataReader dbReader = dbCommand.ExecuteReader();
            while(dbReader.Read())
            {
                curShortAddr = dbReader.GetString(0).TrimEnd();
                if (splitRXdata[0].Equals(curShortAddr))
                {
                    if (!connectedNodesList.Contains(curShortAddr))
                    {
                        connectedNodesList.Add(curShortAddr);
                        if (droppedNodesList.Contains(curShortAddr))
                        {
                            writeLog("Node re-connected from drop: " + curShortAddr);
                            droppedNodesList.Remove(curShortAddr);
                        }
                        else
                        {
                            writeLog("Node connected: " + curShortAddr);
                        }
                    }
                    newNode = false;
                }
            }
            dbReader.Close();

            if (newNode)
            {
                writeLog("Unknown node seen on network: " + splitRXdata[0]);
                commissioningForm nodeSetup = new commissioningForm(splitRXdata[0]);
                if (DialogResult.OK == nodeSetup.ShowDialog())
                {
                    dbCommand.CommandText = @"INSERT INTO nodeInfo (ShortAddr,
                        ParentShortAddr, Name, Location, PollingInterval,
                        Analog1Conf, Analog2Conf, CC1Conf, CC2Conf,
                        FirstSeenDT, LastSeenDT) VALUES ('" +
                        splitRXdata[0] + "','" + splitRXdata[1] + "','" + nodeSetup.textBoxName.Text +
                        "','" + nodeSetup.textBoxLocation.Text + "','" + nodeSetup.comboBoxPolling.Text +
                        "','" + nodeSetup.textBoxAN0.Text + "','" + nodeSetup.textBoxAN1.Text +
                        "','" + nodeSetup.textBoxCC0.Text + "','" + nodeSetup.textBoxCC1.Text +
                        "','" + System.DateTime.Now + "','" + System.DateTime.Now + "')";
                    try
                    {
                        connectedNodesList.Add(splitRXdata[0]);
                        dbCommand.ExecuteNonQuery();
                        writeLog("New node added to database: " + splitRXdata[0]);
                    }
                    catch (System.Data.SqlClient.SqlException f)
                    {
                        writeLog("Error writing new node to database: " + f.Message);
                    }
                }
                else
                {
                    writeLog("Ignoring unknown node: " + splitRXdata[0]);
                }
            }

            dbCommand.CommandText = @"INSERT INTO nodeData (ShortAddr,LQI,RSSI,Batt,Temp," +
                "Analog1,Analog2,CC1,CC2,PacketDT) VALUES ('" + splitRXdata[0] + "'," +
                splitRXdata[3] + "," + splitRXdata[4] + "," + splitRXdata[5] + "," +
                splitRXdata[6] + "," + splitRXdata[7] + "," + splitRXdata[8] + "," +
                splitRXdata[9] + "," + splitRXdata[10] + ",'" + System.DateTime.Now + "')";
            try
            {
                dbCommand.ExecuteNonQuery();
            }
```

```
            catch (System.Data.SqlClient.SqlException f)
            {
                writeLog("Error writing packet to database: " + f.Message);
            }

            dbCommand.CommandText = @"UPDATE nodeInfo SET LastSeenDT = '" + System.DateTime.Now +
                "' WHERE ShortAddr = '" + splitRXdata[0] + "'";
            try
            {
                dbCommand.ExecuteNonQuery();
            }
            catch (System.Data.SqlClient.SqlException f)
            {
                writeLog("Error writing packet to database: " + f.Message);
            }

            busy = false;
        }

        /**
         * dbConnection_StateChange handles state change events on the
         * database connection, and appropriately writes to the event log
         */
        void dbConnection_StateChange(object sender, StateChangeEventArgs e)
        {
            writeLog("Database Status = " + dbConnection.State);
        }

        /**
         * toolStripStop_Click handles click events from the stop button
         * on the toolstrip.  The function closes the connections to the
         * SQL database and serial port.
         */
        private void toolStripStop_Click(object sender, EventArgs e)
        {
            toolStripRun.Enabled = true;
            toolStripStop.Enabled = false;

            dbConnection.Close();

            try
            {
                nodeCOM.Close();
                writeLog(comPort + " closed!");
            }
            catch (Exception g)
            {
                writeLog("Error: " + g.Message);
            }

            if ((dbConnection.State == ConnectionState.Closed) & (!nodeCOM.IsOpen))
            {
                statusStopped();
            }
            else
            {
                statusError();
            }
        }

        /**
         * configReader grabs the last-used program configuration from the
         * config.ini file located in the working directory.  These values
         * are placed into the instance variables for mainForm.cs
         */
        private void configReader()
        {
            String temp;
            Dictionary<String, String> configDict = new Dictionary<String, String>();
```

```
            using (StreamReader sr = new StreamReader("config.ini"))
            {
                String line;
                while ((line = sr.ReadLine()) != null)
                {
                    if (!line.StartsWith(";"))
                    {
                        String[] splitTemp = line.Split(new char[] { '=' });
                        configDict.Add(splitTemp[0].Trim(), splitTemp[1].Trim());
                    }
                }
            }

            configDict.TryGetValue("SQLDBLOC", out sqlServerLoc);
            configDict.TryGetValue("SQLDBNAME", out sqlDBName);
            configDict.TryGetValue("SQLDBNWK", out sqlConnType);
            if (configDict.TryGetValue("SQLDBINTSEC", out temp))
            {
                if (temp.Equals("1"))
                {
                    sqlIntSec = true;
                }
                else
                {
                    sqlIntSec = false;
                }
            }
            configDict.TryGetValue("SQLDBUSERID", out sqlUserID);
            configDict.TryGetValue("SQLDBPWD", out sqlPwd);
            configDict.TryGetValue("SERPORT", out comPort);
            if(configDict.TryGetValue("SERBAUD", out temp))
                comBaud = Convert.ToInt32(temp);
            if(configDict.TryGetValue("SERPAR", out temp))
                comParity = (Parity)Convert.ToInt16(temp);
            if(configDict.TryGetValue("SERDATA", out temp))
                comData = Convert.ToInt32(temp);
            if(configDict.TryGetValue("SERSTOP", out temp))
                comStop = (StopBits)Convert.ToInt16(temp);
        }


        /**
         * configWriter updates config.ini with the latest values
         * that are present in the local instance variables containing
         * program settings.
         */
        private void configWriter()
        {
            List<String> configContents = new List<String>();
            configContents.Add("; SQL Database Settings");
            configContents.Add("SQLDBLOC = " + sqlServerLoc);
            configContents.Add("SQLDBNAME = " + sqlDBName);
            configContents.Add("SQLDBNWK = " + sqlConnType);
            if (sqlIntSec)
            {
                configContents.Add("SQLDBINTSEC = 1");
            }
            else
            {
                configContents.Add("SQLDBINTSEC = 0");
            }
            configContents.Add("SQLDBUSERID = " + sqlUserID);
            configContents.Add("SQLDBPWD = " + sqlPwd);

            configContents.Add("; RS232 Settings");
            configContents.Add("SERPORT = " + comPort);
            configContents.Add("SERBAUD = " + comBaud);
            configContents.Add("SERPAR = " + (int)comParity);
            configContents.Add("SERDATA = " + comData);
```

```
            configContents.Add("SERSTOP = " + (int)comStop);

            try
            {
                File.WriteAllLines("config.ini", configContents);
            }
            catch (System.IO.IOException)
            {
                MessageBox.Show("Could not write config.ini :(", "Error!");
            }

        }

        /**
         * ConfigWindowSync synchronizes the configuration values present
         * in the ConfigWindow dialog box and the local instance variables
         * of mainForm.cs.
         *
         * @param dir Sets the direction to sync: 1 for ConfigWindow --> mainForm,
         *            0 for mainForm --> ConfigWindow
         */
        private void configWindowSync(Boolean dir)
        {
            if (!dir)
            {
                ConfigWindow.textBoxDBLoc.Text = sqlServerLoc;
                ConfigWindow.textBoxDBName.Text = sqlDBName;
                if (sqlConnType.Equals("dbmslpcn"))
                {
                    ConfigWindow.comboBoxConnType.SelectedIndex = 0;
                }
                else if (sqlConnType.Equals("dbmssocn"))
                {
                    ConfigWindow.comboBoxConnType.SelectedIndex = 1;
                }
                if (sqlIntSec)
                {
                    ConfigWindow.comboBoxIntSec.SelectedIndex = 1;
                }
                else
                {
                    ConfigWindow.comboBoxIntSec.SelectedIndex = 0;
                }
                ConfigWindow.textBoxDBUserID.Text = sqlUserID;
                ConfigWindow.textBoxDBPwd.Text = sqlPwd;
                ConfigWindow.comboBoxCOM.Text = comPort;
                ConfigWindow.comboBoxBaud.Text = comBaud.ToString();
                ConfigWindow.comboBoxParity.SelectedIndex = (int)comParity;
                ConfigWindow.comboBoxData.Text = comData.ToString();
                ConfigWindow.comboBoxStop.SelectedIndex = (int)comStop;
            }
            else
            {
                sqlServerLoc = ConfigWindow.textBoxDBLoc.Text;
                sqlDBName = ConfigWindow.textBoxDBName.Text;
                sqlUserID = ConfigWindow.textBoxDBUserID.Text;
                sqlPwd = ConfigWindow.textBoxDBPwd.Text;

                if (ConfigWindow.comboBoxConnType.SelectedIndex == 0)
                {
                    sqlConnType = "dbmslpcn";
                }
                else if (ConfigWindow.comboBoxConnType.SelectedIndex == 1)
                {
                    sqlConnType = "dbmssocn";
                }
                else
                {
                    MessageBox.Show("Bad Configuration, try again!");
```

```
            }

            if (ConfigWindow.comboBoxIntSec.SelectedIndex == 1)
            {
                sqlIntSec = true;
            }
            else
            {
                sqlIntSec = false;
            }
            sqlUserID = ConfigWindow.textBoxDBUserID.Text;
            sqlPwd = ConfigWindow.textBoxDBPwd.Text;
            comPort = ConfigWindow.comboBoxCOM.Text;
            comBaud = Convert.ToInt32(ConfigWindow.comboBoxBaud.Text);
            comParity = (Parity)ConfigWindow.comboBoxParity.SelectedIndex;
            comData = Convert.ToInt32(ConfigWindow.comboBoxData.Text);
            comStop = (StopBits)ConfigWindow.comboBoxStop.SelectedIndex;
        }
    }

    private void statusRunning()
    {
        toolStripConfig.Enabled = false;
        toolStripRun.Enabled = false;
        toolStripStop.Enabled = true;

        writeLog("Server running!");
        toolStripStatusLabel.Text = "Running...";
        toolStripStatusLabel.ForeColor = Color.Green;

        connDT = System.DateTime.Now;
        secTick.Start();
        nodeCheck.Start();
        progressBar1.Style = ProgressBarStyle.Marquee;
    }

    private void statusError()
    {
        toolStripConfig.Enabled = false;
        toolStripRun.Enabled = false;
        toolStripStop.Enabled = true;

        writeLog("Server Error!");
        toolStripStatusLabel.Text = "Error! :(";
        toolStripStatusLabel.ForeColor = Color.Yellow;

        secTick.Stop();
        nodeCheck.Stop();
        progressBar1.Style = ProgressBarStyle.Blocks;
    }

    private void statusStopped()
    {
        toolStripConfig.Enabled = true;
        toolStripRun.Enabled = true;
        toolStripStop.Enabled = false;

        TimeSpan duration = System.DateTime.Now - connDT;
        writeLog("Server stopped after running for " + duration.ToString(@"dd\.hh\:mm\:ss"));
        toolStripStatusLabel.Text = "Stopped";
        toolStripStatusLabel.ForeColor = Color.Red;

        connectedNodesList.Clear();
        droppedNodesList.Clear();

        secTick.Stop();
        nodeCheck.Stop();
        progressBar1.Style = ProgressBarStyle.Blocks;
    }
```

```
        private void statusInit()
        {
            toolStripConfig.Enabled = true;
            toolStripRun.Enabled = true;
            toolStripStop.Enabled = false;

            connectedNodesList.Clear();
            droppedNodesList.Clear();
        }

        private void writeLog(String msg)
        {
            /*
            if(listBoxStatus.Items.Count > 200)
            {
                listBoxStatus.Items.Clear();
            }
            listBoxStatus.Items.Add(System.DateTime.Now + " | " + msg);
            */

            using(StreamWriter sw = new StreamWriter("log.txt", true))
            {
                sw.WriteLine(System.DateTime.Now + " | " + msg);
            }
        }

        private void timerConfig()
        {
            secTick = new Timer();
            secTick.Interval = 1000;
            secTick.Tick += new EventHandler(secTick_Tick);

            nodeCheck = new Timer();
            nodeCheck.Interval = 10000;
            nodeCheck.Tick += new EventHandler(nodeCheck_Tick);
        }

        void nodeCheck_Tick(object sender, EventArgs e)
        {
            int NODE_DROP_TIMEOUT = 5;
            String curShortAddr;
            if (!busy)
            {
                SqlCommand checkDBCommand = dbConnection.CreateCommand();
                checkDBCommand.CommandText = @"SELECT ShortAddr,PollingInterval,LastSeenDT FROM nodeInfo";
                SqlDataReader checkDBReader = checkDBCommand.ExecuteReader();
                while (checkDBReader.Read())
                {
                    curShortAddr = checkDBReader.GetString(0);
                    TimeSpan delta = System.DateTime.Now - checkDBReader.GetDateTime(2);
                    TimeSpan interval = new TimeSpan(0, 0, checkDBReader.GetInt32(1) + NODE_DROP_TIMEOUT);
                    if ((delta.CompareTo(interval) > 0) & (connectedNodesList.Contains(curShortAddr)))
                    {
                        writeLog("Node dropped: " + curShortAddr);
                        droppedNodesList.Add(curShortAddr);
                        connectedNodesList.Remove(curShortAddr);
                    }
                }
                checkDBReader.Close();
            }
        }

        void secTick_Tick(object sender, EventArgs e)
        {
            TimeSpan duration = System.DateTime.Now - connDT;
            toolStripStatusLabel.Text = "Running... " + duration.ToString(@"dd\.hh\:mm\:ss");
            toolStripConnectedNodesLabel.Text = "Connected Nodes: " + connectedNodesList.Count;
            toolStripDroppedNodesLabel.Text = "Dropped Nodes: " + droppedNodesList.Count;
```

```
        }

        private void toolStripAbout_Click(object sender, EventArgs e)
        {
            aboutBox AboutBox = new aboutBox();
            AboutBox.ShowDialog();
        }
    }
}
```

configForm.cs

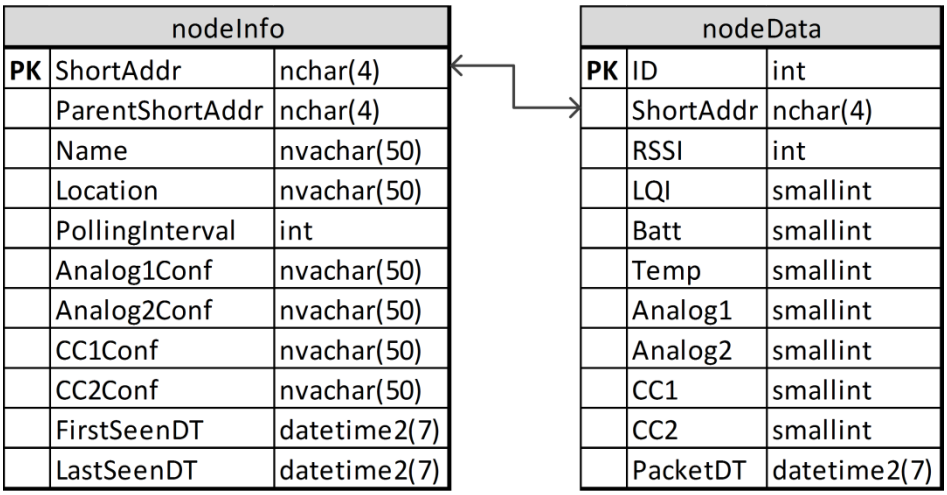
```
/**
 * configForm.cs provides a GUI form for the
 * end user to input their desired SQL database
 * and serial port connection settings
 *
 * @author Jacob Power
 */

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace wsnManagementServer
{
    public partial class configForm : Form
    {
        public configForm()
        {
            InitializeComponent();
            comboBoxCOM.Items.AddRange(SerialPort.GetPortNames());
        }

        private void buttonOkay_Click(object sender, EventArgs e)
        {

        }

    }
}
```

commissioningForm.cs

```
/**
 * commissioningForm.cs provides a GUI form for
 * the user to input configuration information for
 * a new node on the network.
 *
 * @author Jacob Power
 */

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace wsnManagementServer
{
    public partial class commissioningForm : Form
    {
        public commissioningForm(String shortAddr)
        {
            InitializeComponent();
            labelNodeAddr.Text = "Node Short Address: " + shortAddr;
        }

        private void commissioningForm_Load(object sender, EventArgs e)
        {

        }
    }
}
```

Appendix M. SQL Database Schema



nodeInfo		
PK	ShortAddr	nchar(4)
	ParentShortAddr	nchar(4)
	Name	nvachar(50)
	Location	nvachar(50)
	PollingInterval	int
	Analog1Conf	nvachar(50)
	Analog2Conf	nvachar(50)
	CC1Conf	nvachar(50)
	CC2Conf	nvachar(50)
	FirstSeenDT	datetime2(7)
	LastSeenDT	datetime2(7)

nodeData		
PK	ID	int
	ShortAddr	nchar(4)
	RSSI	int
	LQI	smallint
	Batt	smallint
	Temp	smallint
	Analog1	smallint
	Analog2	smallint
	CC1	smallint
	CC2	smallint
	PacketDT	datetime2(7)

Appendix N. Data Visualization Application Code

mapForm.cs

```
/**
 * mapForm.cs provides a map view of all nodes currently
 * in the database, and allows the user to select which
 * nodes to view data from.
 *
 * @author Jacob Power
 */

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace wsnVisualizer
{
    public partial class mapForm : Form
    {
        private String sqlServerLoc;
        private String sqlDBName;
        private String sqlUserID;
        private String sqlPwd;
        private String sqlConnType;
        private Boolean sqlIntSec;
        private SqlConnectionStringBuilder dbConnString;

        public mapForm()
        {
            InitializeComponent();

            sqlConfigForm configWindow = new sqlConfigForm();
            configWindow.ShowDialog();

            sqlServerLoc = configWindow.textBoxDBLoc.Text;
            sqlDBName = configWindow.textBoxDBName.Text;
            sqlUserID = configWindow.textBoxDBUserID.Text;
            sqlPwd = configWindow.textBoxDBPwd.Text;

            if (configWindow.comboBoxConnType.SelectedIndex == 0)
            {
                sqlConnType = "dbmslpcn";
            }
            else if (configWindow.comboBoxConnType.SelectedIndex == 1)
            {
                sqlConnType = "dbmssocn";
            }
            else
            {
                MessageBox.Show("Bad Configuration, try again!");
            }

            if (configWindow.comboBoxIntSec.SelectedIndex == 1)
            {
                sqlIntSec = true;
            }
            else
            {
```

```
                sqlIntSec = false;
            }
            sqlUserID = configWindow.textBoxDBUserID.Text;
            sqlPwd = configWindow.textBoxDBPwd.Text;

            dbConnString = new SqlConnectionStringBuilder();
            dbConnString.DataSource = sqlServerLoc;
            dbConnString.InitialCatalog = sqlDBName;
            dbConnString.UserID = sqlUserID;
            dbConnString.Password = sqlPwd;
            dbConnString.NetworkLibrary = sqlConnType;
            dbConnString.IntegratedSecurity = sqlIntSec;
        }

        private void pictureBox1_Click(object sender, EventArgs e)
        {
        }


        private void pictureBoxDesk_Click(object sender, EventArgs e)
        {
            visForm visualizer = new visForm(dbConnString.ConnectionString, "3ba4");
            visualizer.Show();
        }

        private void pictureBoxDoor_Click(object sender, EventArgs e)
        {
            visForm visualizer = new visForm(dbConnString.ConnectionString, "1a1");
            visualizer.Show();
        }

        private void pictureBoxCPE_Click(object sender, EventArgs e)
        {
            visForm visualizer = new visForm(dbConnString.ConnectionString, "7165");
            visualizer.Show();
        }

        private void pictureBoxPilkington_Click(object sender, EventArgs e)
        {
            visForm visualizer = new visForm(dbConnString.ConnectionString, "93f9");
            visualizer.Show();
        }
    }
}
```

visForm.cs

```
/**
 * visForm.cs provides graphing services for node
 * data from a defined node, using an existing
 * SQL connection string.
 *
 * @author Jacob Power
 */

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace wsnVisualizer
```

```
{
    public partial class visForm : Form
    {
        public String dbConnString;
        private String nodeAddr;
        private SqlConnection sqlDBConn;
        private SqlCommand sqlCommand;
        private Timer pollTimer;

        public visForm(String dbConnString, String nodeAddr)
        {
            InitializeComponent();
            this.nodeAddr = nodeAddr;
            this.dbConnString = dbConnString;
        }

        private void visForm_Load(object sender, EventArgs e)
        {
            sqlDBConn = new SqlConnection(dbConnString);

            try
            {
                sqlDBConn.Open();
            }
            catch (System.Data.SqlClient.SqlException f)
            {
                MessageBox.Show("SQL Connection failure: " + f.Message, "Error!");
            }

            sqlCommand = new SqlCommand(@"SELECT Name FROM nodeInfo WHERE ShortAddr = '" + nodeAddr + "'",
sqlDBConn);
            SqlDataReader rdr = sqlCommand.ExecuteReader();
            while (rdr.Read())
            {
                this.Text = this.Text + rdr.GetString(0);
            }
            rdr.Close();

            comboBoxPoints.SelectedIndex = 0;

            setupChartArea();

            updateForm();

            pollTimer = new Timer();
            pollTimer.Interval = 1000;
            pollTimer.Tick += new EventHandler(pollTimer_Tick);
            pollTimer.Start();
        }

        void pollTimer_Tick(object sender, EventArgs e)
        {
            updateForm();
        }

        private void updateForm()
        {
            Boolean first = true;
            for (int i = 0; i < chartVis.Series.Count; i++)
            {
                chartVis.Series[i].Points.Clear();
            }

            sqlCommand = new SqlCommand(@"SELECT TOP " + comboBoxPoints.Text +
                " PacketDT, LQI, Batt, Temp, Analog1, Analog2, CC1, CC2 FROM nodeData WHERE ShortAddr = '"
+
                nodeAddr + "' ORDER BY ID DESC", sqlDBConn);
            SqlDataReader rdr = sqlCommand.ExecuteReader();
            while (rdr.Read())
```

```
            {
                if (first)
                {
                    first = false;
                    labelAN1.Text = ((Convert.ToDouble(rdr.GetValue(4)) / 1024) * 5).ToString("F1") +"V";
                    labelAN2.Text = ((Convert.ToDouble(rdr.GetValue(5)) / 1024) * 5).ToString("F1") + "V";

                    if (rdr.GetValue(6).ToString().Equals("1"))
                    {
                        labelCC1.Text = "Open";
                    }
                    else
                    {
                        labelCC1.Text = "Closed";
                    }

                    if (rdr.GetValue(7).ToString().Equals("1"))
                    {
                        labelCC2.Text = "Open";
                    }
                    else
                    {
                        labelCC2.Text = "Closed";
                    }

                    labelTemp.Text = (Convert.ToDouble(rdr.GetValue(3)) / 16.0).ToString("F1") + "C";
                    labelBatt.Text = ((((Convert.ToDouble(rdr.GetValue(2)) / 1024) * 2.5 * 3) / 3.3) *
100).ToString("F1") + "%";
                    labelLQI.Text = ((Convert.ToInt32(rdr.GetValue(1)) / 255.0) * 100).ToString("F1") +
"%";
                }

                if (checkBoxAN1.Checked == true)
                {
                    chartVis.Series["Analog 1"].Points.AddXY(rdr.GetDateTime(0),
                        (Convert.ToDouble(rdr.GetValue(4)) / 1024) * 5);
                }

                if (checkBoxAN2.Checked == true)
                {
                    chartVis.Series["Analog 2"].Points.AddXY(rdr.GetDateTime(0),
                        (Convert.ToDouble(rdr.GetValue(5)) / 1024) * 5);
                }

                if (checkBoxCC1.Checked == true)
                {
                    chartVis.Series["CC 1"].Points.AddXY(rdr.GetDateTime(0),
                        Convert.ToInt32(rdr.GetValue(6)) * 100);
                }

                if (checkBoxCC2.Checked == true)
                {
                    chartVis.Series["CC 2"].Points.AddXY(rdr.GetDateTime(0),
                        Convert.ToInt32(rdr.GetValue(7)) * 100);
                }

                if (checkBoxTemp.Checked == true)
                {
                    chartVis.Series["Temp"].Points.AddXY(rdr.GetDateTime(0),
                        Convert.ToDouble(rdr.GetValue(3)) / 16.0);
                }

                if (checkBoxLQI.Checked == true)
                {
                    chartVis.Series["LQI"].Points.AddXY(rdr.GetValue(0),
                        (Convert.ToInt32(rdr.GetValue(1)) / 255.0) * 100);
                }

                if (checkBoxBatt.Checked == true)
```

```
                {
                    chartVis.Series["Batt"].Points.AddXY(rdr.GetDateTime(0),
                        (((Convert.ToDouble(rdr.GetValue(2)) / 1024) * 2.5 * 3) / 3.3) * 100);
                }
            }

            rdr.Close();
        }

        private void setupChartArea()
        {
            //any run-once per instance chart area setup stuff goes here
            chartVis.ChartAreas[0].AxisX.Title = "Date/Time";
            chartVis.ChartAreas[0].AxisX.LabelStyle.Format = "g";
            chartVis.ChartAreas[0].AxisX.LabelStyle.Angle = -25;

            chartVis.ChartAreas[0].AxisY.MajorGrid.LineColor = Color.Gray;
            chartVis.ChartAreas[0].AxisY.MajorGrid.LineDashStyle =
System.Windows.Forms.DataVisualization.Charting.ChartDashStyle.Dash;
            chartVis.ChartAreas[0].AxisY2.MajorGrid.LineColor = Color.Gray;
            chartVis.ChartAreas[0].AxisY2.MajorGrid.LineDashStyle =
System.Windows.Forms.DataVisualization.Charting.ChartDashStyle.Dash;
            chartVis.ChartAreas[0].AxisX.MajorGrid.LineColor = Color.Gray;
            chartVis.ChartAreas[0].AxisX.MajorGrid.LineDashStyle =
System.Windows.Forms.DataVisualization.Charting.ChartDashStyle.Dash;
        }

        private void checkBoxAN1_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBoxAN1.Checked == true)
            {
                chartVis.Series.Add("Analog 1");
                chartVis.Series["Analog 1"].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
                chartVis.Series["Analog 1"].XValueType =
System.Windows.Forms.DataVisualization.Charting.ChartValueType.DateTime;

                chartVis.ChartAreas[0].AxisY.Title = "Volts (V)";

                if (checkBoxTemp.Checked == true)
                {
                    checkBoxTemp.Checked = false;
                }
            }
            else
            {
                chartVis.Series.RemoveAt(chartVis.Series.IndexOf("Analog 1"));
            }
        }

        private void checkBoxAN2_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBoxAN2.Checked == true)
            {
                chartVis.Series.Add("Analog 2");
                chartVis.Series["Analog 2"].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
                chartVis.Series["Analog 2"].XValueType =
System.Windows.Forms.DataVisualization.Charting.ChartValueType.DateTime;

                chartVis.ChartAreas[0].AxisY.Title = "Volts (V)";

                if (checkBoxTemp.Checked == true)
                {
                    checkBoxTemp.Checked = false;
                }
            }
            else
            {
```

```
                chartVis.Series.RemoveAt(chartVis.Series.IndexOf("Analog 2"));
            }
        }

        private void checkBoxCC1_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBoxCC1.Checked == true)
            {
                chartVis.Series.Add("CC 1");
                chartVis.Series["CC 1"].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
                chartVis.Series["CC 1"].XValueType =
System.Windows.Forms.DataVisualization.Charting.ChartValueType.DateTime;
                chartVis.Series["CC 1"].YAxisType =
System.Windows.Forms.DataVisualization.Charting.AxisType.Secondary;

                chartVis.ChartAreas[0].AxisY2.Title = "%";
            }
            else
            {
                chartVis.Series.RemoveAt(chartVis.Series.IndexOf("CC 1"));
            }
        }

        private void checkBoxCC2_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBoxCC2.Checked == true)
            {
                chartVis.Series.Add("CC 2");
                chartVis.Series["CC 2"].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
                chartVis.Series["CC 2"].XValueType =
System.Windows.Forms.DataVisualization.Charting.ChartValueType.DateTime;
                chartVis.Series["CC 2"].YAxisType =
System.Windows.Forms.DataVisualization.Charting.AxisType.Secondary;

                chartVis.ChartAreas[0].AxisY2.Title = "%";
            }
            else
            {
                chartVis.Series.RemoveAt(chartVis.Series.IndexOf("CC 2"));
            }
        }

        private void checkBoxTemp_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBoxTemp.Checked == true)
            {
                chartVis.Series.Add("Temp");
                chartVis.Series["Temp"].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
                chartVis.Series["Temp"].XValueType =
System.Windows.Forms.DataVisualization.Charting.ChartValueType.DateTime;

                chartVis.ChartAreas[0].AxisY.Title = "Temperature (degC)";

                if (checkBoxAN1.Checked == true)
                {
                    checkBoxAN1.Checked = false;
                }

                if (checkBoxAN2.Checked == true)
                {
                    checkBoxAN2.Checked = false;
                }
            }
            else
            {
                chartVis.Series.RemoveAt(chartVis.Series.IndexOf("Temp"));
```

```
            }
        }

        private void checkBoxLQI_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBoxLQI.Checked == true)
            {
                chartVis.Series.Add("LQI");
                chartVis.Series["LQI"].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
                chartVis.Series["LQI"].XValueType =
System.Windows.Forms.DataVisualization.Charting.ChartValueType.DateTime;
                chartVis.Series["LQI"].YAxisType =
System.Windows.Forms.DataVisualization.Charting.AxisType.Secondary;

                chartVis.ChartAreas[0].AxisY2.Title = "%";
            }
            else
            {
                chartVis.Series.RemoveAt(chartVis.Series.IndexOf("LQI"));
            }
        }

        private void checkBoxBatt_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBoxBatt.Checked == true)
            {
                chartVis.Series.Add("Batt");
                chartVis.Series["Batt"].ChartType =
System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
                chartVis.Series["Batt"].XValueType =
System.Windows.Forms.DataVisualization.Charting.ChartValueType.DateTime;
                chartVis.Series["Batt"].YAxisType =
System.Windows.Forms.DataVisualization.Charting.AxisType.Secondary;

                chartVis.ChartAreas[0].AxisY2.Title = "%";
            }
            else
            {
                chartVis.Series.RemoveAt(chartVis.Series.IndexOf("Batt"));
            }
        }

        private void visForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            pollTimer.Stop();
            sqlDBConn.Close();
        }
    }
}
```