AN IMPLEMENTATION OF A WIRELESS BODY AREA NETWORK FOR AMBULATORY HEALTH MONITORING

by

CHRIS OTTO

A THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering
in
The Department of Electrical and Computer Engineering
to
The School of Graduate Studies
of
The University of Alabama in Huntsville

HUNTSVILLE, ALABAMA

2006

THESIS APPROVAL FORM

Submitted by Chris Otto in partial fulfillment of the requirements for the degree of Master of Science in Engineering and accepted on behalf of the Faculty of the School of Graduate Studies by the thesis committee.

We, the undersigned members of the Graduate Faculty of The University of Alabama in Huntsville, certify that we have advised and/or supervised the candidate on the work described in this thesis. We further certify that we have reviewed the thesis manuscript and approve it in partial fulfillment of the requirements of the degree of Master of Science in Engineering.

____ Committee Chair
(Date)

____

____

____

____ Department Chair

____ College Dean

____ Graduate Dean

ABSTRACT

School of Graduate Studies
The University of Alabama in Huntsville

Degree	Master of Science in Engineering	College/Dept.	Engineering/Electrical and Computer Engineering

Name of Candidate Chris Otto

Title An Implementation Of A Wireless Body Area Network For Ambulatory Health Monitoring

Wireless Body Area Networks (WBANs) represent a promising trend in wearable health monitoring systems. WBANs promise to revolutionize health monitoring and increase a user's quality of life by offering continuous and ubiquitous ambulatory health monitoring at the least level of obtrusiveness. This thesis presents a WBAN implementation which consists of multiple intelligent physiological sensor nodes, a personal server, and a network coordinator. The sensor platforms and network coordinator are built from off-the-shelf wireless sensor platforms. The sensors feature custom-designed intelligent physiological sensor boards, implemented as daughter cards, for ECG monitoring and motion sensing. The nodes communicate wirelessly using standards-based IEEE 802.15.4 and a novel power-efficient TDMA scheme. The thesis explores practical implementation challenges and presents original solutions for time synchronization, event management, and for dynamic reallocation of on-chip resources for maximum efficiency.

Abstract Approval: Committee Chair ____________________

Department Chair ____________________

Graduate Dean ____________________

ACKNOWLEDGMENTS

The work described in this thesis would not have been possible without the assistance of a number of people who deserve special mention. First, I would like to thank Dr. Emil Jovanov for his suggestion of the research topic and for his guidance throughout all the stages of the work and collaboration on a number of journal and conference papers. Second, I would like to thank Dr. Aleksandar Milenković for his wisdom, moral support, and collaboration on a number of journal and conference papers. I am also thankful to Dr. Earl Wells who has been very helpful with comments and suggestions.

A number of students deserve recognition: Corey Sanders for his work on the personal server as an undergraduate senior design project, Reggie McMurtrey and John Gober for their hardware design of the ISPM board, Dennis Cox for his initial work on time synchronization protocol, and Brian Trotter and David Wachira for their help in collecting and analyzing accelerometer signals during organized tests of the WBAN system.

Last, but not least, I would like to thank Jennie, my wife, who first encouraged me to pursue a Master's degree and then endured all the long hours and late nights along side me. Without her love and encouragement, I would have never finished.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

A number of economic and demographic forces are challenging the long-term scalability of existing healthcare systems. The United States spent an estimated $1.8 trillion on healthcare in 2004 – approximately 15% of the GDP [NCHC]. This alone represents the single largest category of US spending, yet it does not include the estimated 30 million adults serving as informal caregivers – most of which work full-time. Lost wages and decreased quality of life to the caregiver are more difficult to quantify but are estimated at an additional $200 million in healthcare cost. Increased life expectancy and retiring baby boomers are compounding the problem – causing a dramatic shift in demographics in the United States and worldwide. In this century, it is expected that the elderly will outnumber children for the first time in history [UCB]. With insurance premiums increasing annually at rates above 10% and already 45 million Americans uninsured [NCHC], the future does not look promising.

Though the United States is the largest spender worldwide on healthcare, the value of that spending is arguably less than other nations. With an annual spending of $4178 per capita, the United States spends 50% more per capita than its nearest competitor (Switzerland), but continues to rank outside of the top ten in life expectancy [WHO2000]. In a 2004 American Medical Association study on the causes of death in the United States, it was determined that the second leading cause of death (400,000) was directly attributed to poor diet and physical inactivity [Mokdad00]. It has long been understood the relationship between wellness and physical activity [Steele03], yet this continues to be a significant health care problem. Early diagnosis is also well understood as a means to reduce overall treatment costs and increase life expectancy and quality of life – especially for cardiovascular disease. Yet, it is estimated that 81% of total health expenditures is spent on treatments, hospital stays, and rehabilitation measures while only 4% is spent on diagnostic measures. These facts underscore the shortcomings of existing healthcare systems. We need a dramatic shift from centralized reactive healthcare systems to distributed and proactive systems focused on managing wellness rather than illness.

Wearable systems for continuous health monitoring are a key technology in helping the transition to more proactive and affordable healthcare [Jovanov05a] [Jovanov05b]. They allow an individual to closely monitor changes in her or his vital signs and provide feedback to help maintain an optimal health status. If integrated into a telemedical system, these systems can even alert medical personnel when life-threatening changes occur. In addition, the wearable systems can be used for health monitoring of patients in ambulatory settings [Istepanian04]. For example, they can be used as a part of a diagnostic procedure, optimal maintenance of a chronic condition, a supervised recovery from an acute event or surgical procedure, to monitor adherence to treatment guidelines (e.g., regular cardiovascular exercise), or to monitor effects of drug therapy.

One of the most promising approaches in building wearable health monitoring systems utilizes emerging wireless body area networks (WBANs) [Jovanov05a]. A

WBAN consists of multiple sensor nodes, each capable of sampling, processing, and communicating one or more vital signs (heart rate, blood pressure, oxygen saturation, activity) or environmental parameters (location, temperature, humidity, light). Typically, these sensors are placed strategically on the human body as tiny patches or hidden in users' clothes allowing ubiquitous health monitoring in their native environment for extended periods of time. This offers the freedom of mobility and enhances the patient's quality of life [Martin00].

This thesis presents the prototype WBAN implementation and the specific design objectives. Specifically, we present the architecture and environment for our WBAN development, the power efficient wireless communication system, detail the embedded software, and describe our personal server application. The thesis is organized as follows: Chapter 2 details related work in the area of ambulatory health monitoring. Chapter 3 defines the overall system and component architecture. It begins by briefly describing the motivating system architecture and long-term vision, and then details the hardware architecture and the TinyOS environment used for software development. Chapter 4 describes the WBAN communication scheme and how its design is well suited for ultra-low power sensors such as the nodes in our WBAN. Chapter 5 describes the embedded software on the sensor nodes and the network coordinator. We discuss the details of our implementation for time synchronization, feature extraction, event management, and software management of constrained resources. Chapter 6 describes the personal server application and its role in the WBAN sensor network.

CHAPTER 2

AMBULATORY HEALTH MONITORING

Wearable systems for health monitoring come in a number of forms and serve a variety of applications [Raskovic04]. Available systems range from those serving personal fitness and wellness management to those serving to provide remote monitoring or diagnostic of cardiovascular problems. A number of research programs are pursuing development of health monitoring systems as well. This chapter provides a survey of both commercial systems and university or private research programs.

2.1 Commercial Devices

Holter monitors for ECG and EEG monitoring are among the first and most frequently used wearable sensors. Existing devices are limited, however, in that they are strictly data acquisition devices. Cardio Labs based in Franklin, TN was founded in 1995 and offers a slight variation on Holter monitors by introducing event-based monitoring. Their recorders allow extended ambulatory heart motoring and capture of cardiac arrhythmias and ischemic episodes [CardioLabs]. The devices are marketed to physicians with patients who report intermittent symptoms such as palpitations, chest pain, shortness of breath, etc., but have been unable to provide an in-office diagnosis. The patient wears the device for extended periods and when experiencing a symptomatic episode, presses the event button. The device will record up to 32 minutes of data surrounding the event. The data can then be extracted (at the physician's office) and analyzed by the physician to assist diagnosis.

CardioNet is perhaps the closest commercial product to our area of research. CardioNet provides systems for mobile cardiac outpatient telemetry (MCOT). The system includes a three lead ECG sensor and electrodes, a portable health monitoring device (in PDA form factor), and a service center for collecting data from various users. The electrodes are paced on the chest and the sensor (transmitter) is worn around the neck or on the user's belt. The monitor can be placed on a desk or table nearby or can be worn when the user is mobile. The system is unique in that the sensor communicates wirelessly to the monitoring device and then the monitoring device uploads data to the service center using existing cellular infrastructure. The user can enter occurring symptoms so that the service center can correlate the data; however, the monitoring device also performs some processing of the data and is capable of detecting certain abnormal heart rhythms. This represents a significant advantage over CardioLabs event-based heart monitors. In a clinical study, CardioNet demonstrated detection of serious arrhythmic events in patients that went undetected with standard event recorder and Holter monitor devices [CardioNet].

Polar Electro (or more commonly Polar), founded in 1977 has come to be one of the leaders in wireless heart rate monitors for personal health training. Polar has an extensive line of heart rate monitors primarily targeting fitness applications such as weight loss and digital personal trainers for athletes. The typical system includes a dry electrode ECG chest strap and a companion watch with wireless receiver; the watch provides simple heart rate measurement and averaging functions. Polar’s success and market share is evident with ongoing partnerships with a variety of wellness center-based gym equipment manufacturers. Many treadmill manufacturers integrate Polar heart monitor receivers into their equipment for display of real-time measurements or control of exercise while users are training. Polar has attracted a number of competitors in this space. Timex, Reebok and CardioSport offer similar products [Polar]. Polar also markets its heart rate monitors to cardiac rehabilitation patients; however, the systems only offer a simple heart rate in beats per minute (bpm). While this is certainly important feedback for a recovering heart attack victim, the system is limited in that it is not integrated into a telemedical system and it offers no way of sharing data with the physician nor does it provide detailed heart waveform analysis.

Finnish sports watch company, Suunto, has an impressive line of high end sports watches or “wrist top computers.” In particular, a line of watches designed for fitness training includes a heart rate monitor belt. Similar to the Polar heart rate monitors, Suunto differentiates itself by its recent release of the Suunto team pod and team pack pro. The system is designed to work with their t6 models and allows coach or trainer to monitor and collect heart rates from multiple athletes from up to 100 meters away [Suunto].

Still in the spirit of fitness and wellness-based training, Polar Electro, Timex, and Suunto all offer high end heart rate monitors that provide multimodal monitoring. These systems typically include external wireless foot pods utilizing either GPS or accelerometers for useful real-time feedback of speed and distance (through the watch or wrist-top computer) and also offer the ability for data upload for journal logging, and some analysis of training sessions.

Advances in MEMS technology and solid state accelerometers have made wearable motion sensors practical for a number of applications. Pedometers are perhaps the simplest variety of such systems. Most pedometers are fairly simple devices; however, Accusplit has developed an extensive line of intelligent pedometers with on-sensor processing. The sensor clips to the user's belt or waist line and uses MEMS accelerometers for motion sensing and on-sensor signal processing to detect steps. They report 99% step detection accuracy [Accusplit]. In addition, on-sensor processing is capable of providing useful metrics such as estimates of calorie consumption, distance traveled, and a speedometer.

Another variety serves the field of actigraphy – the study of human motor activity primarily for the purposes of analyzing wake-sleep patterns. Cambridge Neurotechnology offers the Actiwatch family of products. The Actiwatch has been used for a number of research studies for monitoring sleep and wake patterns, sleep disorders, and measure estimation of energy expenditure. The Actiwatch uses accelerometers to measure and record activity levels. The single axis accelerometer is sampled at a rate of 32Hz and the resulting signal is digitally integrated over a user-defined period (typically 1 minute). The measure of activity is reported in arbitrary "activity units" but provides a useful relative measurement scale for estimating activity. The Actiwatch can also be worn on the leg to monitor periodic leg movement syndrome (PLMS) or on infants to monitor infant wake and sleep patterns [CamN]. The Actiwatch offers a full line of models ranging in memory densities from 16KB to 64KB for activity monitoring between 11 and 44 days in length. In all cases, the watch-like sensor is only a data recorder. Using an electromagnetic interface, the user or researcher must download the stored data to a PC for post-session analysis. Cambridge Neurotechnology also offers the *Actiheart* device which is a direct extension of their Actiwatch family. The Actiheart includes an integrated accelerometer and ECG bioamplifier. Worn around the chest, the Actiheart is capable of recording simple heart rate and estimating activity-induced energy expenditure [CamN].

Perhaps the most advanced system of this kind is the system offered by Pittsburgh-based start-up Company, BodyMedia. BodyMedia offers wearable devices for monitoring of multimodal physiological data. They have a small selection of highly integrated health monitoring armbands such as the *bodybugg*, available on their website. The bodybugg includes a single axis accelerometer for recording body movement, two temperature sensors for detecting body temperature, and a galvanic-skin-response sensor to measure changes in skin conductivity due to the sweat gland dilation. The armband stores data locally and has capacity for up to seven days of continual monitoring. Data is then uploaded to BodyMedia's web server using a PC and USB interface. BodyMedia's proprietary algorithms utilize data from the four sensors to determine estimates of activity induced energy expenditure (AEE) and calorie consumption. Web portals allow users to access weight and calorie management data for two primary applications: clinical research programs and self weight management. BodyMedia appears well positioned to capitalize on this growing trend in wearable health monitoring systems. A reported $28 million in investor funding validates BodyMedia's business model and serves as a testament to the market size and emerging trends in this area [Body].

2.2 Research Projects

Several university and private research project projects, both past and ongoing are related to this work. The CodeBlue project, directed by the engineering and applied science department at Harvard University, is the closest of these to our area of research. They are developing technology to facilitate real-time triage in disaster relief situations. By outfitting patients with wireless sensors, they have demonstrated the system's ability to form ad-hoc sensor networks, collect vital statistics of each patient, and then use system software to identify those patients in most critical need of medical attention [Shnayder05]. Their system includes wireless pulse oximeter sensors, wireless two-lead ECG sensors, and triaxial accelerometer based motion sensors for health monitoring. They are using both custom boards and off-the-shelf sensor platforms using the TinyOS operating system. The real-time triage application and system user interface reside on a Personal Digital Assistant (PDA) [Lorincz04].

10Blade is a start-up research and development company founded in 2002. Working closely with the Harvard CodeBlue Project, they promote three products *iRevive, iIncise,* and *VitalDust*; however, these are simply components of the CodeBlue project. iRevive represents the mobile patient database and iIncise the PocketPC user application. They are working jointly with Harvard University, Boston University School of Management, and Boston Medical Center to test and develop the technology [10Blade].

MIThril is a wearable research platform developed by researchers at MIT's Media Lab. They are using wearable sensors to monitor a user's physiological state and the surrounding environment in order to discover new techniques for human-computer interaction. The MIThril system is centered about the "MIThril Real-Time Context Engine," which samples sensors worn on the body, extracts pertinent features from the raw data, and then uses this data to model the user's context [DeVaul03]. MIThril researchers are using both custom and off-the-shelf sensors in their network. The sensors communicate via wired interfaces and all reside in a vest. The MIThril includes ECG, skin temperature, galvanic skin response (GSR) sensors, tiny cameras, and microphones. In addition, team members demonstrated step and gait analysis using 3-axis accelerometers, rate gyros, and pressure sensors [Pentland04].

Microsoft researchers in the Adaptive Systems and Interaction (ASI) Group have developed HealthGear, a system of network of embedded sensors monitoring human physiological data. Their research is focused on using the system to recognize patterns of human behaviour when faced with external factors such as workload, stress, traffic situations, exercise, diet, sleep, etc. They have used the system in a study on sleep apnea. In particular, they used a combination of motion sensors and oximeter sensors to detect irregular sleep patterns [Oliver05].

The Advanced telemedical MONitor (AMON) project is the product of a collaboration of European industry and universities funded by the European Union (EU) Information Society Technology (IST). The system integrates a number of advanced bio-sensors on a Wrist-mounted Monitoring Device and includes sensors such as heart rate, heart rhythm, 2-lead ECG, blood pressure, blood-oxygen saturation, skin perspiration and body temperature. The system monitors vital signs by collecting data from each sensor, performing preliminary analysis to determine heart rate for example, and then, using

GSM/UMTS, transmits the data to a remote telemedicine system for further analysis and possible emergency care if needed [Anliker04].

CHAPTER 3

WBAN SYSTEM ARCHITECTURE

Our prototype implementation of the WBAN is best understood in the context of the motivating vision and proposed system architecture of a distributed ubiquitous health monitoring system. In the first subsection we describe this system architecture and the benefits it offers in light of the issues discussed in the introduction. In the following subsections we describe the hardware architecture of the WBAN prototype and the overview of the software architecture although primarily as a development environment.

3.1 System Overview

The proposed WBAN for ambulatory health monitoring is contained within a multi-tier telemedicine system as illustrated in Figure 3.1. The telemedicine system spans a network comprised of individual health monitoring systems that connect through the Internet to a medical server tier that resides at the top of this hierarchy. The system is not merely a distributed data logger, which in itself would provide great advantage over current systems, but provides distributed data processing and analysis functions. Each tier in the network is intelligent and provides some form of analysis; in some cases it may be possible for on-the-spot real-time diagnosis of conditions. The top tier, centered on a medical server, is optimized to service hundreds or thousands of individual users, and encompasses a complex network of interconnected services, medical personnel, and healthcare professionals. Each user wears a number of sensor nodes that are strategically placed on the body. The nodes are designed to unobtrusively sample vital signs and transfer the relevant data to a personal server through a wireless personal network implemented using ZigBee (802.15.4) or Bluetooth (802.15.1). The personal server, implemented on a home personal computer, handheld computer, smart phone, or residential gateway, controls the WBAN, performs sensor fusion, and preliminary analysis of physiological data. It provides graphical or audio interface to the user, and transfers captured health information to the medical server through the Internet or mobile telephone networks (e.g., GPRS, 3G).

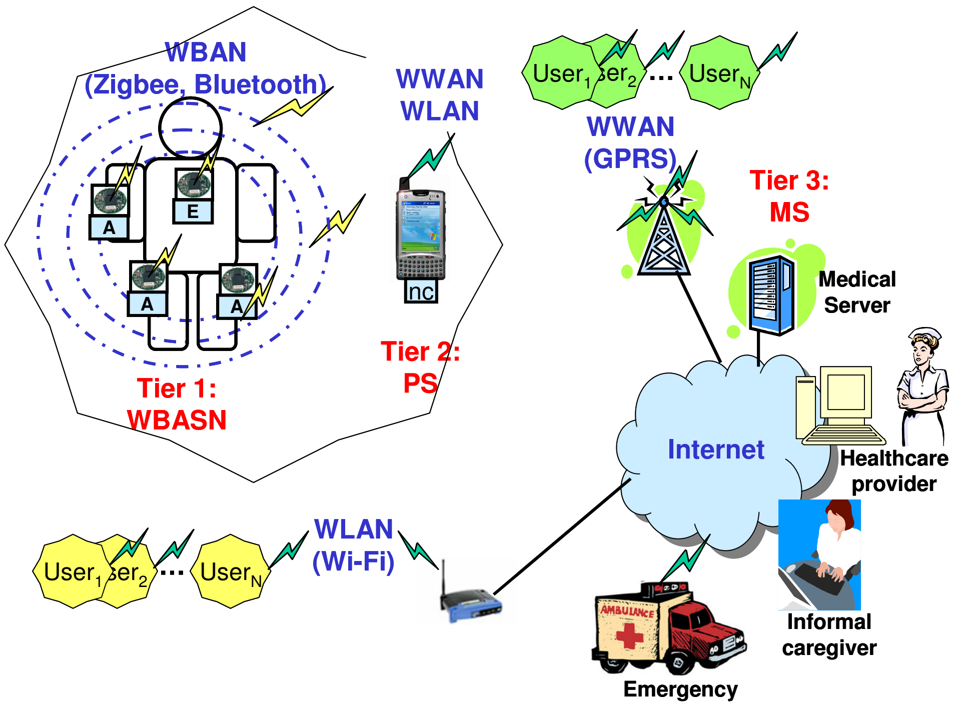


Figure 3.1 Health monitoring system network architecture

3.1.1 Medical Server

The medical server provides a variety of differing functions to WBAN users, medical personnel, and informal caregivers. The medical server stores electronic patient records in a database, provides a high availability daemon for authenticating registered WBAN users and accepting session uploads, summarizes physiological data and automatically analyzes the data to verify it is inside or outside acceptable health metrics (heart rate, blood pressure, activity) and identifies known patterns of health risks. It is the responsibility of the medical server to interface the electronic patient records and insert new session data, generate alerts to the physician and emergency health care professionals when abnormal conditions are detected, and provide physician and informal caregiver portals via the Internet for retrieving health summary reports remotely. This is especially powerful for the physician who can access the data at a convenient time to determine whether the patient is responding to a prescribed medication or exercise and make updates to those prescriptions and forward them electronically back to the patient where the user's personal server is responsible for delivering such changes to the user. The large amount of data collected through these services can also be utilized for knowledge discovery through data mining. Integration of the collected data into research databases along with quantitative analysis of conditions and patterns could prove invaluable to researchers trying to link symptoms and diagnoses with historical changes in health status, physiological data, or other parameters (e.g., gender, age, weight). In a similar way this infrastructure could significantly contribute to monitoring and studying of drug therapy effects.

3.1.2 Personal Server

The personal server, at the second tier, is responsible for interfacing with the medical server via the Internet, interfacing the WBAN sensors and fusing sensor data, and providing an intuitive graphical and/or audio interface to the end user. The personal server application can run on a variety of platforms with a variety of wide area network (WAN) access possibilities for Internet access. Platform selection is system specific and should be selected to minimize obtrusiveness for a given user. For in-home monitoring of elderly patients, a stationary residential gateway or personal computer might be the ideal platform, but for high mobility users, it may be necessary to use a smart phone or handheld computer with GPRS capabilities [Jovanov06] [Priddy06].

The personal server requires ZigBee or Bluetooth capability for communications within the WBAN; depending on the platform, this may be integrated in the device or provided as a separate plug-in network coordinator (NC). The NC is responsible for coordinating WBAN communications and managing aspects such as time synchronization, timeslot assignment, and channel sharing. In addition, the personal server is responsible for sensor configuration including node registration (type and number of sensors), initialization (e.g., specify sampling frequency and mode of operation), customization (e.g., run user-specific calibration or user-specific signal processing procedure upload), and setup of a secure communication (key exchange). Once the sensor nodes are configured, the personal server fuses sensor data into personalized session files. Based on synergy of information from the multiple medical sensors, the PS application should determine the user's state and his or her health status, providing user feedback through a friendly and intuitive graphical or audio user interface.

For interface to the medical server, the personal server requires some wireless wide area network (WWAN) or wireless local area network (WLAN) access such as GPRS or 802.11 respectively. In the case of a static residential gateway or home personal computer implementation, the personal server may be connected directly to a broadband Internet link. The personal server holds patient authentication information and is configured with IP address or domain name of the medical server so that it can access services over the Internet. The PS schedules upload of health monitoring session files at periodic intervals or defers transmission in the event an Internet connection is unavailable. In such cases, the personal server may be unable to propagate indicators of serious changes in health status. Because processing is performed on the personal server and on sensor nodes, the system should be capable of recognizing abnormalities and alerting the user to potential threatening physiological conditions.

3.1.3 Sensor Nodes

For every personal server, a network of intelligent sensor nodes captures various physiological signals of medical interest. Each node is capable of sensing, sampling, processing, and communicating physiological signals. For example, an ECG sensor can be used for monitoring heart activity, an EMG sensor for monitoring muscle activity, an EEG sensor for monitoring brain electrical activity, a blood pressure sensor for monitoring blood pressure, a tilt sensor for monitoring trunk position, a breathing sensor for monitoring respiration, while the motion sensors can be used to discriminate the user’s status and estimate her or his level of activity.

Each sensor node receives initialization commands and responds to queries from the personal server. WBAN nodes must satisfy requirements for minimal weight, miniature form-factor, low-power consumption to permit prolonged ubiquitous monitoring, seamless integration into a WBAN, standards based interface protocols, and patient-specific calibration, tuning, and customization. With further development of the technology, the wireless network nodes can be implemented as tiny patches or incorporated into the user's clothes. The network nodes continuously collect and process raw information, store them locally, and send processed event notifications to the personal server. The type and nature of a healthcare application will determine the frequency of relevant events (sampling, processing, storing, and communicating). Ideally, sensors process data on-sensor, minimizing the number of data transmissions, therefore significantly reducing power consumption and extending battery life. When local analysis of data is inconclusive or indicates an emergency situation, the node can transfer raw signals to the next tier of the network for further processing.

Patient privacy, an outstanding issue and a requirement by law, must be addressed at all tiers in the healthcare system. Data transfers between a user's personal server and the medical server require encryption of all sensitive information related to the personal health [Warren05]. Before possible integration of the data into research databases, all records must be stripped of all information that can tie it to a particular user. The limited range of wireless WBAN communications partially addresses security; in addition, the messages can and should be encrypted using either software or hardware techniques. Some wireless sensor platforms have already provided a low power hardware encryption solution for ZigBee communications [CC2420].

3.2 Hardware Architecture

We have designed and implemented a prototype WBAN for exploring issues and implementation details of the complete system proposed in the previous subsection. Figure 3.2 shows a photograph of two prototype sensors. The fully operational prototype system includes an integrated ECG and tilt sensor (*eActiS*), two activity sensors (*ActiS*), and a personal server with attached network coordinator (not shown). Each sensor node includes a custom application specific board and uses the *Tmote Sky* platform [Otto05] for processing and for ZigBee wireless communication. The personal server runs either on a laptop computer or a WLAN/WWAN-enabled handheld PocketPC. The network coordinator with wireless ZigBee interface is implemented on another *Tmote Sky* that connects to the personal server through a USB interface. Alternatively, a custom network coordinator that features the ZigBee wireless interface, an ARM processor, and a compact flash interface to the personal server is under development.

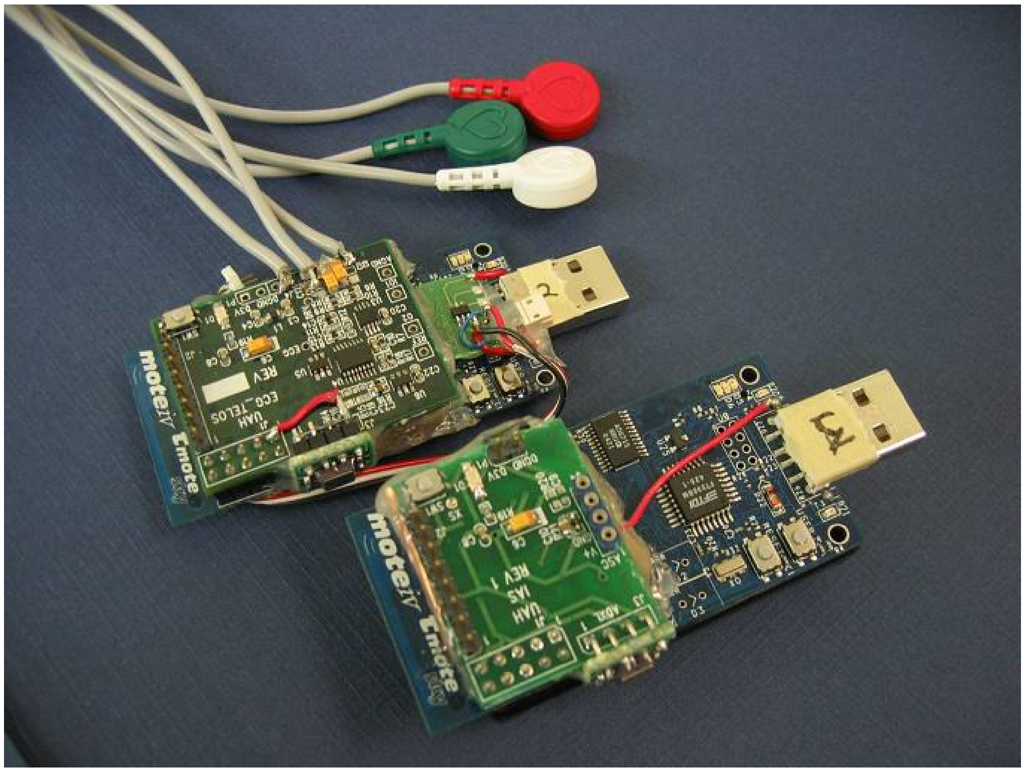


Figure 3.2 Prototype WBAN sensors. From left to right: *eActiS* with electrodes, *ActiS*

3.2.1 Wireless Sensor Platform - *Tmote Sky*

For the main processing board on the embedded sensor nodes, we used commercially available wireless sensor platforms from Moteiv [Moteiv]. During the course of development, we used Moteiv’s original *Telos rev A*, its successor *Telos rev B*, and finally the *Tmote Sk*y platform. Each platform is based on an MSP430 family microcontroller with integrated RAM and flash memory, a USB interface, and an integrated wireless ZigBee compliant radio with antenna. The *Telos rev A* utilizes the MSP430F149 microcontroller with 2KB RAM and 60KB flash memory, while the *Telos rev B* and *Tmote Sky* utilize the MSP430F1611 with 10KB of RAM and 48KB of flash memory, representing the largest capacity RAM offered in an MSP430 device. *Telos rev B* and *Tmote Sky* are 100% code compatible and can be used interchangeably

[Moteiv]. The *Tmote Sky* platform is an ideal fit for this application due to small footprint and out of the box TinyOS support. In addition, the *Tmote Sky* platform includes humidity, temperature, and light sensors that might be of interest for some applications.

The *Tmote Sky* platform features a 10-pin expansion header that allows one UART and I^2C interface, two general-purpose I/O lines, and three analog inputs to be connected to a custom daughter card. It is through this expansion header that we were able to integrate the *ActiS* and *eActiS* sensor nodes. The *Tmote Sky* from Moteiv serves as the main processing platform of the embedded sensor node as well as the network coordinator. Each *Tmote Sky* board utilizes an MSP430F1611 microcontroller and Chipcon’s CC2420 ZigBee radio interface. Figure 3.3 depicts a functional block diagram of the *Tmote Sky* platform and Figure 3.4 shows a photo of a *Tmote Sky* module.

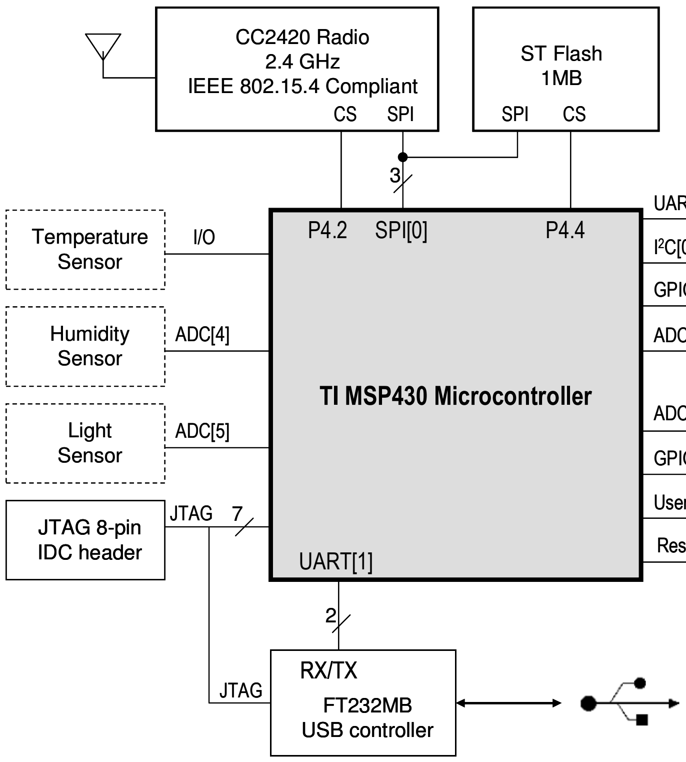


Figure 3.3 Functional block diagram of *Tmote Sky* platform

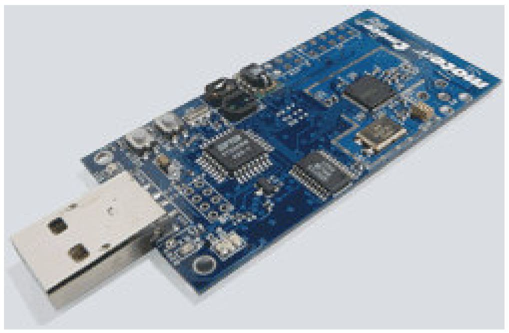


Figure 3.4 *Tmote Sky* wireless platform

Both the embedded *Tmote Sky* and our custom intelligent daughter cards utilize Texas Instrument’s MSP430 family microcontrollers. The MSP430 family is a set of mixed signal ultra-low power microcontrollers targeted for various embedded applications. Each MSP430 has a 16-bit RISC core, 16 general-purpose integer registers, Harvard architecture with internal program memory and internal SRAM for stack and data, a flexible clock subsystem, varying number of hardware timers, digital I/O, and depending on the specific device will have a variety of analog and digital peripherals. Possible on-chip peripherals include analog to digital converters (ADC), digital to analog converters (DAC), direct memory access (DMA) module, and universal synchronous / asynchronous receive transmit (USART) modules supporting SPI, I^2C, and standard UART modes of operation.

Figure 3.5 depicts the functional block diagram of the MSP430F1611 used on the *Tmote Sky* platform.

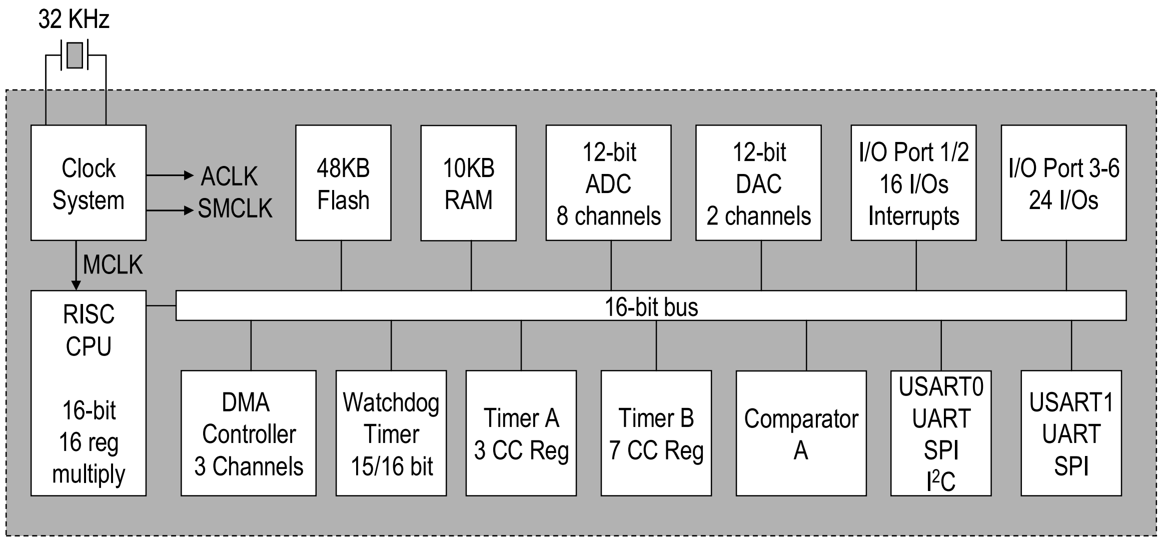


Figure 3.5 Functional block diagram of the MSP430F1611

The MSP430 family microcontroller is especially well-suited for ultra low power applications and features 250 µA/MIPS in active mode [MSP430x1xx]. Leveraging a versatile clock subsystem, the MSP430 supports five low-power operating modes.

Managing the clock subsystem is at the heart of achieving low power operation. The MSP430 has three internal clock buses: the CPU system clock (MCLK) and two peripheral clocks (SMCLK and ACLK). Each bus can be clocked from two basic clock sources – either an external crystal or an internal digitally controlled oscillator (DCO). A typical configuration utilizes a low cost / low power 32.768 KHz watch crystal to source ACLK while the DCO can be programmed up to a reasonable 4 MHz or 8 MHz. The high speed clock can then be used to source MCLK directly and a cascaded divider can be applied before sourcing SMCLK. This configuration allows a high performance MCLK to run the CPU when reacting to events, but still utilizing the deepest possible low power mode when idle. Ideally, as much time as possible would be spent operating in LPM3 mode where MCLK and SMCLK are both disabled, but ACLK continues to operate an on-chip timer that can be used for wake-up functions such as waking the CPU to perform a scheduled sensor reading. The MSP430 consumes less than 2 µA in LPM3 mode. An interrupt can wake the processor from any low power mode within a maximum of 6 µs [MSP430x1xx].

The MSP430 USART represents an important peripheral for this application. The USART facilitates off-chip peripheral communications such as SPI communications to the Chipcon CC2420 radio, serial EEPROM for mass data storage, or asynchronous operation for USB or standard EIA-232 interface. Typically, the USART module is clocked with the SMCLK bus so that higher data rates can be achieved above that offered by ACLK. One obstacle, however, arises when specific high data rate communications are required. If SMCLK is sourced by the internal DCO, its frequency is inherently imprecise. The DCO frequency can vary as a function of multiple environmental factors as well as from chip to chip. We address this issue by runtime tuning of the DCO as discussed in Section 5.3. Table 3.1 lists features of MSP430 microcontrollers used in the prototype WBAN system.

Table 3.1 MSP430 microcontrollers used in WBAN

	Flash	**SRAM**	**I/O**	**USART**	**Hardware Multiplier**
MSP430F1611	48 KB	10240	48	2	YES
MSP430F149	60 KB	2048	48	2	YES
MSP430F1232	8 KB	256	22	1	NO

The Chipcon CC2420 radio controller is used for wireless communication between our sensor boards and the network coordinator. The CC2420 is a low cost, single-chip, direct spread spectrum (DSSS) 2.4GHz RF transceiver. The CC2420 is IEEE 802.15.4 compliant, provides extensive hardware support for data encryption and authentication, and operates at a maximum data rate of 250 Kbps. It is well suited for low power, low data rate sensor networks. The CC2420 has programmable output power between -25dBm and 0dBm with typical transmit mode current consumption between 8.5mA and 17.4mA respectively. In receive mode, the CC2420 operates at a fixed 18.8mA.

The CC2420 provides a simple four-wire SPI interface for configuration and data transfer. In addition, it provides several digital output pins for augmenting the microcontroller interface. In particular it provides a start of frame delimiter (SFD) for hardware signalization of the start of 802.15.4 packet transmission. This is especially useful for employing wireless time synchronization protocols which we will discuss in Section 5.1.1.

3.2.2 Intelligent Signal Processing Modules

The activity sensor, *ActiS*, consists of the *Tmote Sky* platform and an Intelligent Signal Processing Module (ISPM), implemented as a daughter card. The ISPM utilizes an on-board MSP430F1232 microcontroller for pre-processing and filtering of sampled data. The ISPM is connected via the 10-pin *Tmote Sky* expansion header. A general purpose digital output is connected from the *Tmote Sky* to the ISPM interrupt request input (connected to the MSP430F1232 microcontroller). This allows the main platform to request samples periodically by interrupting the ISPM. Raw data or partially processed data can then be transmitted using the USART configured as a UART and simple serial communication protocol. Figure 3.6 shows a block diagram of the ISPM daughter card.

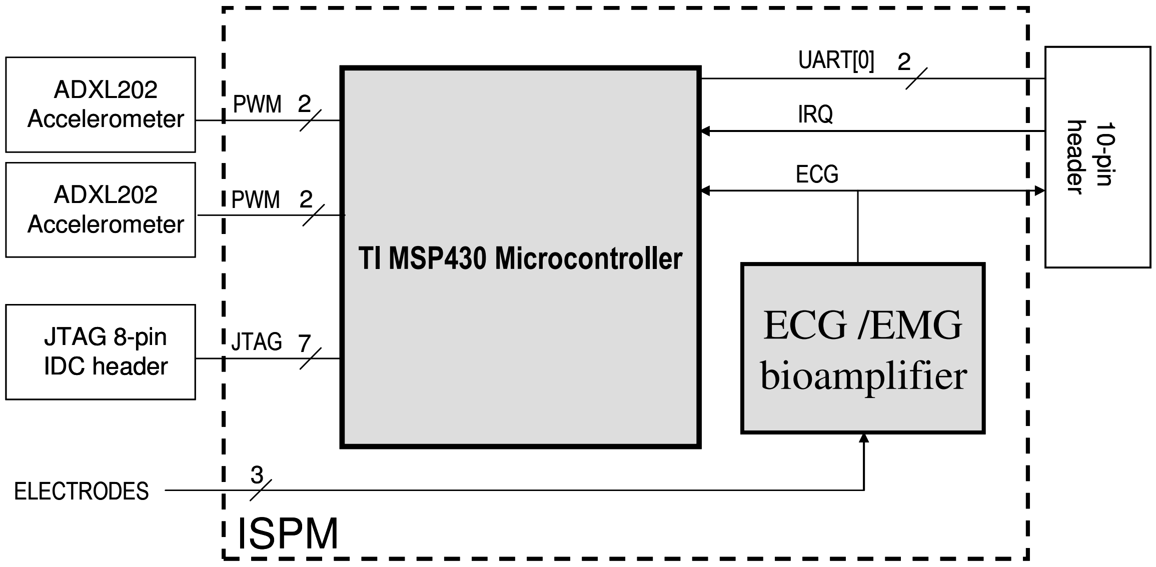


Figure 3.6 ISPM block diagram

The integrated ECG and tilt sensor (*eActiS*) consists of the *Tmote Sky* platform and an ISPM with a single-channel bio-amplifier for three-lead ECG/EMG. Electrodes are connected and placed on the chest for monitoring heart activity. The bioamplifier output (ECG signal) is connected directly to the expansion header for *Tmote Sky* processing as well as to the on-board MSP4301232 for optional pre-processing directly on the ISPM. When the ISPM is used as an ECG heart monitor and worn on the chest, the integrated accelerometers serve as an upper body tilt sensor.

The ISPM monitors motion using two dual-axis ADXL202 accelerometers from Analog Devices. The ADXL202 is a low cost, low power MEMS accelerometer capable of measuring both dynamic acceleration (e.g., vibration) and static acceleration (e.g., gravity) at magnitudes up to ±2g [ADXL202]. We measure motion in three axes by mounting the accelerometers orthogonally. One ADXL202 is mounted directly on the ISPM board and collects data for the X and Y axes as represented in Figure 3.7; the second ADXL202 is mounted vertically to measure motion in the Z axis. Figure 3.7 shows the ISPM used on an *ActiS* sensor node with the axes of motion sensing drawn.

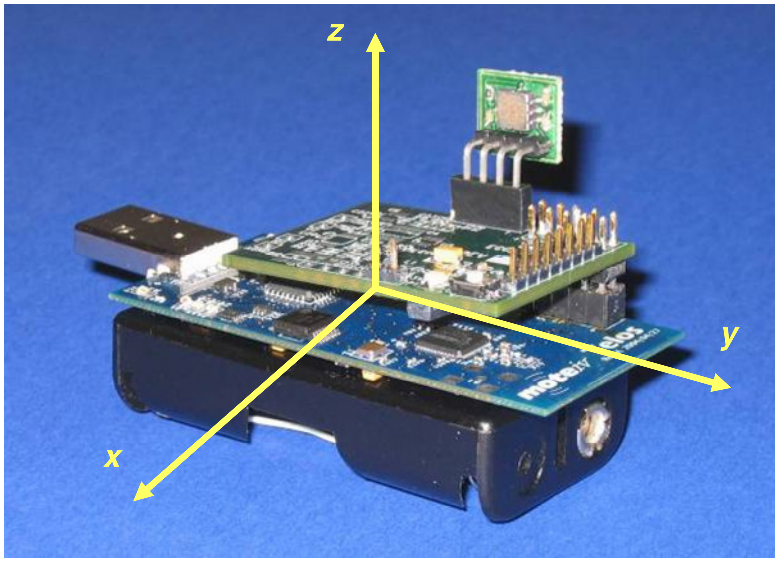


Figure 3.7 *ActiS* sensor node – *Tmote Sky* with ISPM motion sensing daughter card

The ADXL202 provides one digital PWM accelerometer output per axis [ADXL202] for a total of three axes presented to the ISPM microcontroller (MSP4301232). Each PWM output is connected to a MSP430 timer/capture input pin. This allows hardware time stamping of each edge transition and high precision duty cycle calculation without processor generated latency. The PWM waveforms and the relationship between duty cycle and acceleration (in g) are depicted in Figure 3.8. The period (T2) ranges from 0.5ms to 10ms and is set by an external resistor [ADXL202]. On the ISPM, this period is roughly 5ms yielding a 200Hz sampling rate for each axis. This higher sampling rate allows for on-board pre-processing. We employ a low pass filter to remove high frequency content and output a 40 Hz data stream to the wireless sensor platform. The 40Hz sampling rate was determined adequate based on earlier research indicating that the majority of frequency content for human motion is focused between 0.3 – 3.5Hz [Mathie04] and in the case of estimating activity induced energy expenditure (AEE) is limited to upper frequencies of 18Hz [Bouten97].

$$Acceleration\ (in\ \mathrm{g}) = \frac{(T1/T2) - 50\%}{12.5\%}$$

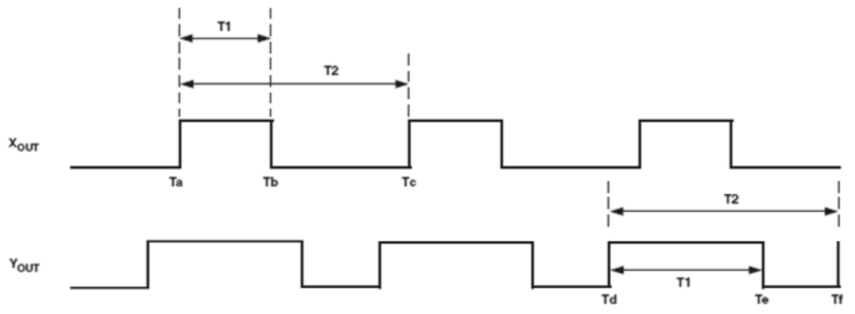


Figure 3.8 ADXL202 PWM description [ADXL202]

3.2.3 Network Coordinator

The network coordinator (NC) is an unmodified *Tmote Sky* platform. The USB interface allows connection to a laptop or desktop-based personal server, while the on-board ZigBee radio allows communication to other WBAN sensor nodes. The network coordinator is the only *Tmote Sky* platform in the network which uses the USB interface.

3.3 Software Architecture

This section provides an overview of the software tools, operating systems, and development environment employed for WBAN component development. Table 3.2

itemizes the operating system, language, and development tools employed for application development on each platform.

Table 3.2 WBAN system components and software environment.

Platform	**Operating System**	**Language**	**Compiler / IDE**
Sensor board (*Tmote Sky*)	TinyOS 1.1.15	nesC	gcc, cygwin
IAS board	-none-	C	IAR Embedded Workbench (3.30A)
Network Coordinator (*Tmote Sky*)	TinyOS 1.1.15	nesC	gcc, cygwin
Personal Server (PC)	Windows XP	Visual Basic	Visual Studio 2003 .NET
Personal Server (PockePC)	Windows CE	Visual Basic	Visual Studio 2003 .NET

3.3.1 TinyOS

TinyOS, the operating system used on each embedded sensor in the WBAN, is of special interest. TinyOS is a lightweight open source operating system specifically designed for networked embedded sensors [Gay03]. It preserves resources by using a modular component architecture. TinyOS applications are created by "wiring" together user defined components with existing TinyOS library components, resulting in a custom instantiation of operating system features and compilation of used functions only. This component framework approach minimizes application footprints and supports the spirit of small, low cost sensor development. The core TinyOS footprint is only 400 Bytes [TinyOS] and the *ActiS* application instantiation of TinyOS (including all radio drivers) is 2700 Bytes. TinyOS is implemented in the nesC programming language, a subset of C enriched with several significant language features facilitating the TinyOS programming model.

Through the use of nesC, developers realize native language support for the TinyOS component model. New keywords such as *components* and *interface*, as well as descriptive operator overloading (−> and =) allow developers to create applications using an object oriented / component model. Developers can create two types of components: *configurations* and *implementations*. By defining an implementation, a component writer is defining a module's functionality in terms of source code. Alternately, a *configuration* can be used to create a component from other components. An interface serves as a contract between components, defining the mechanisms for issuing commands and transferring data between components. Software components are then connected along matching interfaces in a fashion analogous to how ICs are connected in hardware design. Figure 3.9 demonstrates the TinyOS interface construct for two different timer interfaces and Figure 3.10 shows how this is used in TinyOS *TimerC* configuration.

```
interface Timer {
   command result_t start(char type, uint32_t interval);
   command result_t stop();

   event result_t fired();
}
```

```
interface TimerJiffy
{
  command result_t setPeriodic( int32_t jiffy );
  command result_t setOneShot( int32_t jiffy );
  command result_t stop();
  command bool isSet();
  command bool isPeriodic();
  command bool isOneShot();
  command int32_t getPeriod();

  event result_t fired();
}
```

Figure 3.9 TinyOS timer interfaces

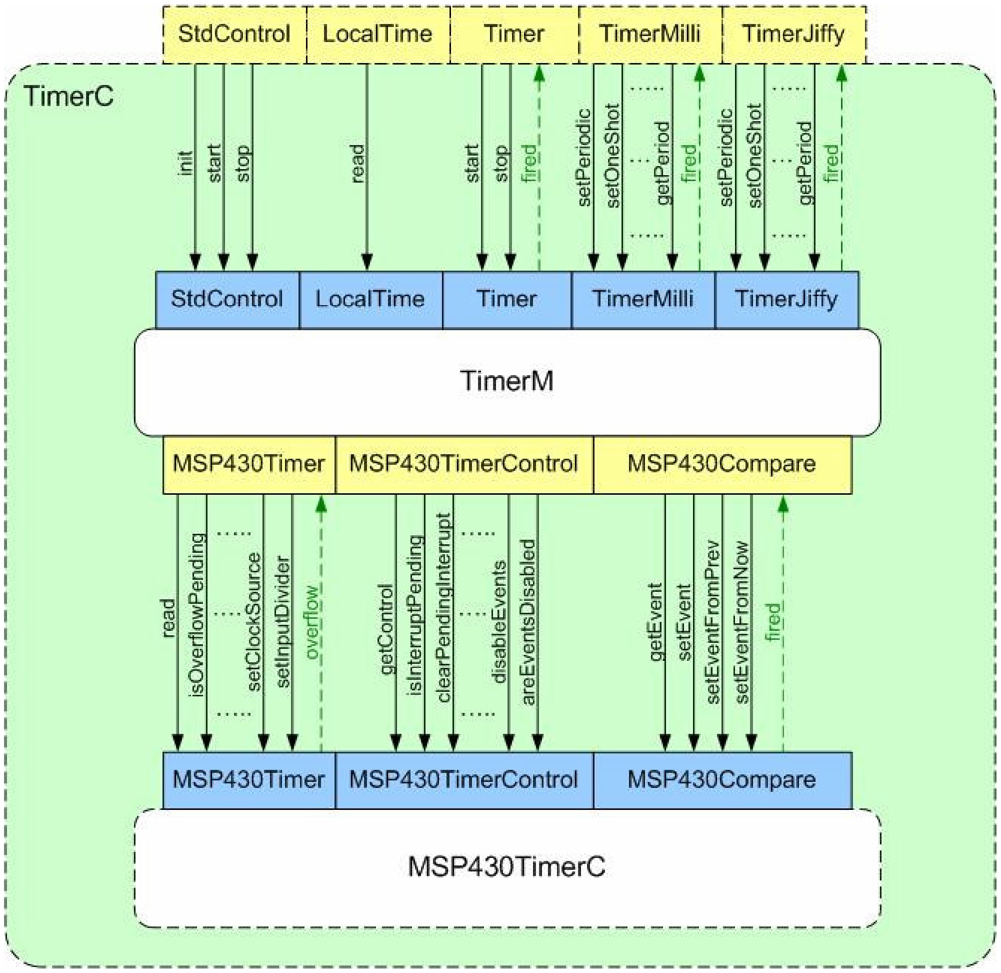


Figure 3.10 TinyOS TimerC configuration diagram

TinyOS supports two forms of concurrency: *tasks* and *events* – both with direct language support from nesC. The nesC keywords *task* and *post* allow users a deferred computation mechanism, while still keeping a system responsive to external events. Tasks are scheduled by calling *post* and supplying the task name as an argument. The post will return immediately and the task will run later as determined by the scheduler. For example, in response to completing a sensor sample, the application can *post* a task to process the data and prepare a radio message for transmission. TinyOS *tasks* are unique in both implementation and concept compared to *threads* in other operating systems.

Threads do not share context and have individual stacks which are memory intensive and not well suited for extremely resource constrained embedded sensors. Tasks, on the other hand, must run to completion, cannot preempt other tasks, and all run from the same context. TinyOS tasks offer an extremely lightweight method for implementing concurrency in user applications. Application designers are, however, encouraged to keep tasks short in execution time to maintain the reactive nature of the concurrency model.

Events offer another mechanism for concurrency in TinyOS. Events are really nothing more than a nesC supported callback mechanism, but come in two primary varieties: a) events representing *hardware interrupts* and b) *software events* invoked by *signalling*. *Hardware interrupt events* will run from the context of an interrupt service routine and are presented asynchronously to TinyOS applications. For example, the *timer.fired( )* event provides notification to the application that a scheduled timer has expired. *Software events* execute from the context they are signalled. If signalled from a hardware interrupt event, it too will execute in this context. However, it is not good practice to cascade event handlers in this fashion. Each cascaded event will contribute to the total interrupt service routine execution time and contribute to the maximum interrupt latency and reactiveness of the system. More typically, software events will be signalled from the context of a deferred task and run synchronously to offer some notification that an operation is complete. For example, the ECG processing task can signal the *SensorECG.ECGRpeak( )* event when the algorithms detect an R Peak event. This embodies the TinyOS split-phase operation model.

The split-phase model facilitates the small, short execution operations that are highly valued in the TinyOS environment. Components publish an interface to the module. Each routine in the interface must be of type *command* or *event*. Commands are typically requests to execute an operation such as send a message, acquire a sensor reading, or store data to flash. When a command has some measurable work to complete, it should post a task and return immediately. Such operations are *split-phase* and are non-blocking. A corresponding *event* handler is called when the operation is complete. When the task completes later, it should signal the corresponding event to notify the requested operation is complete. This design avoids complexity of the operating system and provides the majority of functionality needed by responsive, pervasive, embedded sensors.

The TinyOS concurrency model introduces a need for synchronization. Although tasks cannot preempt other tasks, events can preempt tasks (if executing from the context of an interrupt service routine) and events can preempt other events. TinyOS solves synchronization through the nesC *atomic* keyword. Developers declare code blocks *atomic* which guarantees the block will not be preempted. Implementing this support in the language allows optimal implementation due to compile time realization; however, this behavior is limited in functionality – typically implemented by disabling interrupts. For this reason, atomic sections should be kept short. Complex applications could benefit from a more sophisticated implementation so that completely independent events are not needlessly stalled. For most application specific embedded sensor implementation, however, this lightweight simple mechanism is ideal.

.

CHAPTER 4

WBAN WIRELESS COMMUNICATIONS

Long-life, persistent sensor nodes require efficient power management. With highly integrated electronics, the sensor size and weight becomes dominated by battery selection. An implementation must address conflicting requirements for small size and infrequent battery maintenance, striving for a balance that will maximize user compliance. It is our challenge as designers to minimize sensor power consumption and thus maximize battery life for a given size. In designing our prototype we have held low power consumption as a primary design goal in every component of the system – in processor and technology selection, in managing sensor data, in network organization, and in efficient communications.

Power consumption of the sensor node is dominated by the wireless radio. Nearly 85% can be attributed to CC2420 controller – even when not actively transmitting. The CC2420, although the lowest power of its kind, still draws 17.4mA when transmitting and 19.7mA when receiving. In contrast, the MSP430 utilizes 250µA/MIPS – typically just over 1mA when active. As an example, the MSP430 can execute 100,000 instructions for the same cost of transmitting a single 40 byte message. With that in mind, power savings can be realized by disabling the radio when not in use as well as reducing the total quantity of transmission – even if extensive computation is required.

Besides power efficiency, we were motivated to implement simple and scaleable communications, to use standards-based protocols, and to support multiple simultaneous WBANs within close proximity of one another. The resulting solution spans multiple layers, is IEEE 802.15.4 compliant and upholds the ZigBee star network topology. It leverages existing communication framework within TinyOS and addresses practical WBAN implementation issues.

4.1 IEEE 802.15.4 and ZigBee

Our prototype WBAN utilizes the IEEE 802.15.4 compliant CC2420 radio for wireless communications. The IEEE 802.15.4 standard defines communications for nodes in a low-rate wireless personal area network (LR-WPAN) and is well suited for our prototype WBAN. The standard specifies the physical (PHY) layer and data link / media access control (MAC) layer. At the physical layer, IEEE 802.15.4 defines three frequency bands, spread spectrum chip rate, and data encoding [IEEE802.15.4]. The CC2420 radio is fully compliant and hides these details from the system designer. The CC2420 operates at the highest frequency band – 2450 MHz (2.4 GHz). The standard specifies 16 channels in the 2.4 GHz ISM band. Channel selection is exposed to the application developer through the TinyOS-based CC2420 software driver. By exploiting different 802.15.4 channels, we have been able to operate multiple simultaneous WBANs in close proximity without interference. At present this feature is statically assigned, but satisfies proof of concept. IEEE 802.15.4 employs a carrier sense multiple access with collision avoidance (CSMA-CA) scheme for peer-to-peer communications. In the simplest form, communications are asynchronous and random access can occur. IEEE 802.1.5.4 includes specification for an optional super frame structure utilizing device timeslots which we exploit in the next section.

ZigBee and IEEE 802.15.4 are cooperating protocol stacks. ZigBee is tightly coupled to 802.15.4 in that the PHY and MAC layers are specified to be IEEE 802.15.4; however, the ZigBee specification details the upper protocol layers – network, application and application sublayer, and security. It specifies network topologies, routing mechanisms and dynamic discovery and registration of nodes as they enter and exit the network. ZigBee defines three network topologies: *star*, *tree*, and *mesh*. In the *star* topology a single node serves as the network coordinator; nodes communicate directly to the network coordinator, but not peer-to-peer. In a *tree* topology, nodes are arranged hierarchically so that each node communicates to a designated router node. Traffic propagates through the network by visiting router nodes. A *mesh* network topology allows full peer-to-peer communications [ZigBee]. Based on the human-centric WBAN model, we are able to exploit the simplified *star* topology.

Whenever possible we upheld the spirit of the ZigBee specification, but did not restrict ourselves to conform to the ZigBee specification. At the time of writing, TinyOS does not include a ZigBee protocol implementation nor did any open source ZigBee stack exist. With price tags over $5000 this did not seem like a viable option. Instead of developing this functionality, we chose to maintain the spirit of ZigBee, but allow for a simpler implementation. By doing so, our efforts could be focused on exploring more general challenges in WBAN implementation and maintaining flexibility for future implementations that may not use ZigBee – a Bluetooth WBAN for example.

Compared to Bluetooth which is primarily designed for wireless cable replacement for electronic devices, 802.15.4 / ZigBee offers lower data rates and lower power consumption. Bluetooth is limited to a relatively small number of network participants while 802.15.4 scales upward to 65,536 nodes. In addition, 802.15.4 implementations have smaller memory footprints [Bluetooth] [ZigBee].

4.2 Power Efficient TDMA

Exploiting the ZigBee star network topology [ZigBee] and an 802.15.4 like super frame, we employed a collision-free Time Division Multiple Access (TDMA) scheme. Figure 4.1 shows our communication super frame. All communications are between a sensor node and the network coordinator. Each communication super frame is divided into 50ms timeslots used for message transmissions. Each sensor uses its corresponding timeslot to transmit sensor data, command acknowledgements, and event messages. The first timeslot, however, belongs to the network coordinator and is used for transmitting configuration commands from the personal server and sending periodic beacon messages to synchronize other sensors and mark the start of super frames.

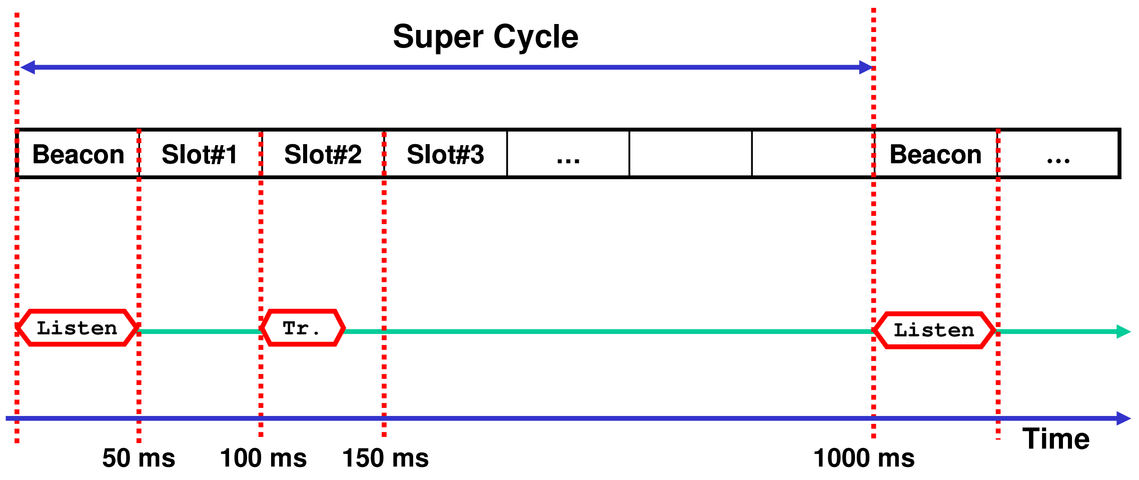


Figure 4.1 Communication super frame and an example of sensor with logical ID=2

This organization has several advantages. Rather than receiving and transmitting asynchronously, transmissions are scheduled for predetermined time instances, which make communication more deterministic. This approach serves as practical collision avoidance – making more efficient use of the available bandwidth when compared to using only the CC2420 Collision Sense Multiple Access (CSMA) scheme. Most importantly, this allows the sensor node radio to be disabled during inactive timeslots. Super frame period and timeslot size are application-specific and must be chosen to balance a tolerable event latency and sensor data bandwidth requirements with low power consumption. Based on a one second super frame and 50 ms timeslots, the radio on each sensor is active at most 10% of the time. By disabling the radio during inactive timeslots, an average current consumption of just 3.1mA can be realized, which achieves 7 times longer battery life compared to a radio that is never disabled [Otto06]. Figure 4.2 shows the recorded power profiles for a motion sensor using an environment for real time power monitoring [Milenkovic05]

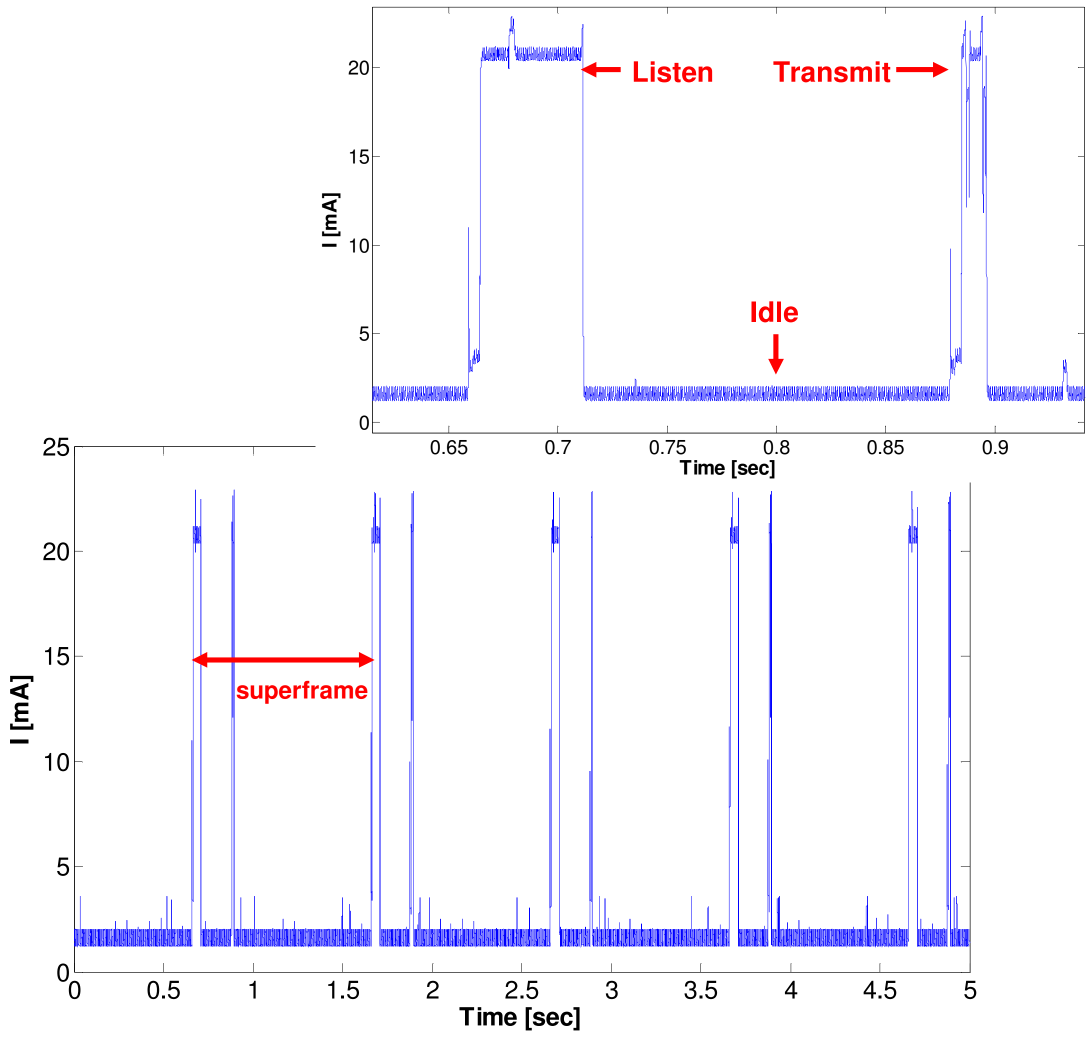


Figure 4.2 A wireless sensor power profile with TDMA WBAN communication; super frame cycle T_C = 1s

It should be noted that the super frame period and timeslot length directly affect power consumption and battery life. Extending the period between beacons allows the radio to stay in low power mode longer. This is not without cost, however. Extending the super frame period also increases the maximum latency between sensor communications and increases the event reporting latency. Every application will have a practical limit of the tolerable event detection latency. Detecting a heart attack may be of little use if the sensor delays notification for up to one hour. For an application specific maximum latency, system designers can make trade offs between battery capacity (weight) and battery life. Figure 4.3 shows how sensor battery life can now be estimated as a function of effective channel utilization and battery chemistry / size.

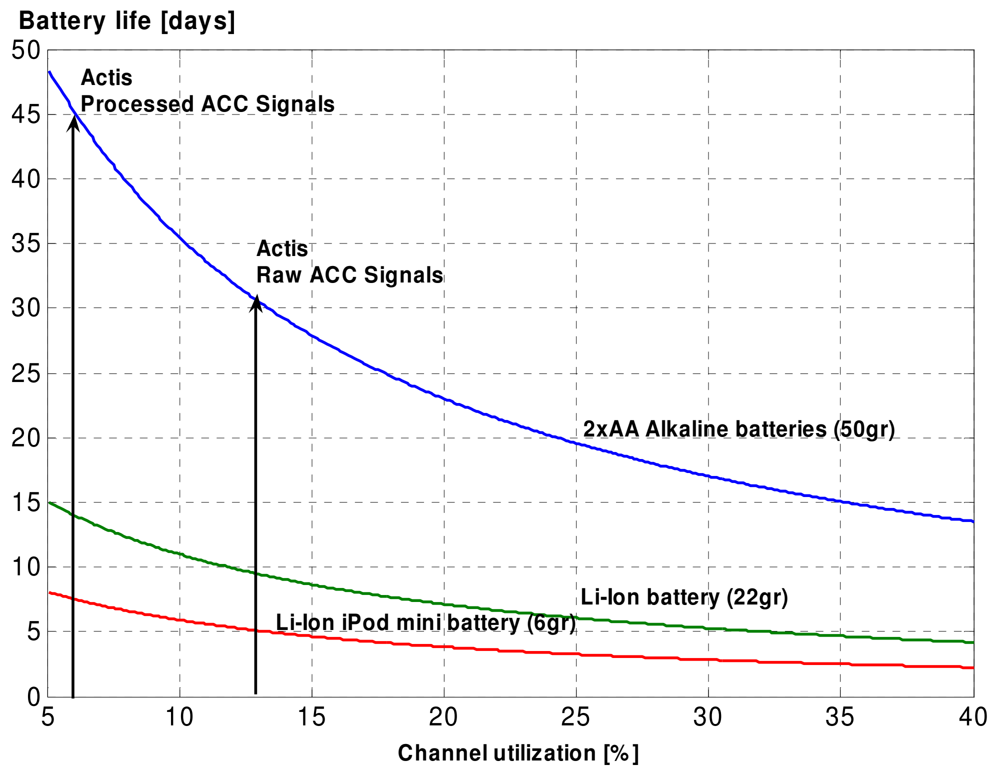


Figure 4.3 Battery life as a function of channel utilization

For our prototype WBAN, a one second super frame period and 50ms timeslots were selected, defining a one second maximum event latency and support for 20 nodes in the network (19 sensors and one network coordinator). Of course, these selections assume the maximum sensor bandwidth is within the constraints – which they are for our WBAN. Our chosen timeslot and super frame length will effectively create a 10% wireless channel utilization for individual sensors. Smart implementation, however, allows the radio to be disabled as soon as communications are complete – even if the entire timeslot has not expired. In this sense, the timeslot length defines the peak channel utilization for a sensor in the network. The width of the second peak in Figure 4.2 indicates the timeslot asymmetry due to early timeslot disabling.

It should be noted that for the dynamic control of the radio, some form of time synchronization is required. During time slots where the radio is disabled, it is not possible to listen to the incoming beacon frame which is used to delimit the super frame. With time synchronization, the radio can be disabled during idle time slots. Immediately before the beacon arrival time, a scheduled timer can wake the processor in order to enable the radio. The beacon is received and then the radio is returned to a disabled state. When a sensor's scheduled timeslot and events or data are queued waiting for transmission, the radio can once again be enabled for transmission.

4.3 Message Format

Figure 4.4 depicts a dissected WBAN message and the format of the TinyOS message. As the figure indicates, several fields of the TinyOS message header are directly shared with the 802.15.4 frame (length, frame control field, data link sequence number, and address). The 802.15.4 addressing supports 4 – 20 bytes in the address field; however, this is fixed to four bytes in the TinyOS implementation. It should be noted that the *mote ID* address refers to the destination address. Applications that require the source address of the node should include this in the TOSH_DATA field (application message).

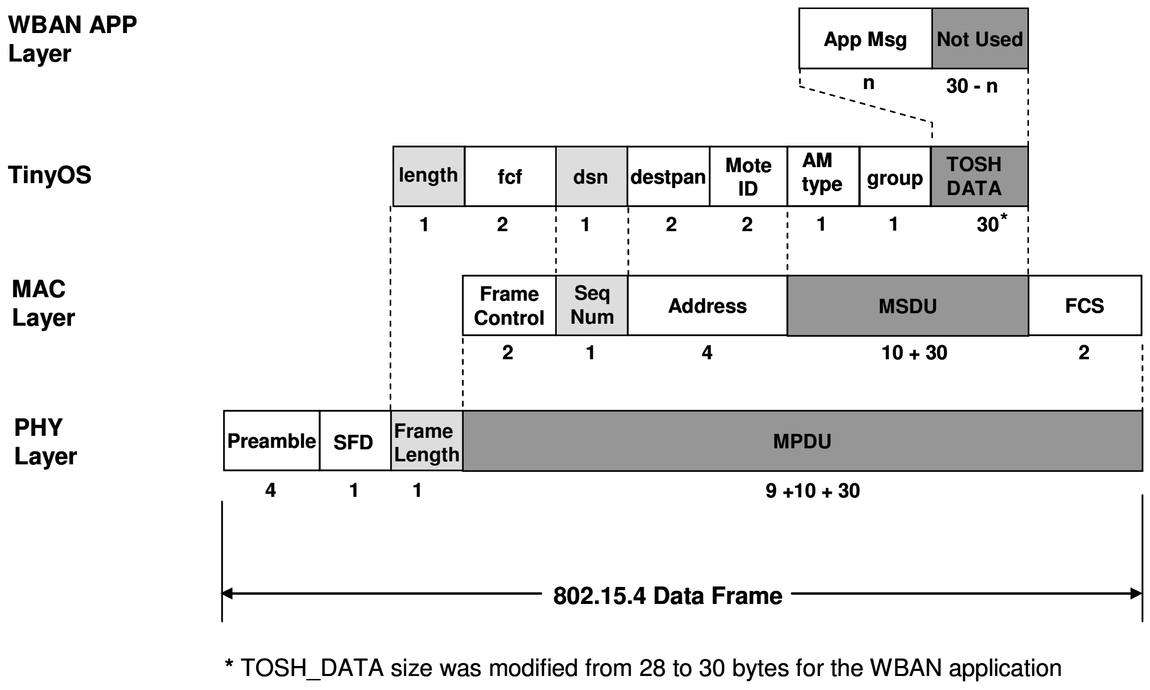


Figure 4.4 WBAN protocol stack

The default installation of TinyOS uses a fixed 28 byte message payload (TOSH_DATA) for all transmissions. During our WBAN implementation we extended this from 28 bytes to 30 bytes to accommodate a 32-bit timestamp on raw data messages and oscilloscope messages (see Appendix B). This payload is fixed in size so that unused portions are simply not used. Application developers benefit from a simple implementation and avoid the woes of dynamic memory allocation on extremely resource constrained systems. The Active Message Type (AM Type) field is analogous to a UDP port number. It facilitates transport functionality and multiplexing of multiple application sessions over the physical link. Application endpoints agree on the AM Type for a given application and then the representation of the data within the payload. The one byte group ID can be used for upper layer multiplexing techniques – partitioning wireless networks into multiple groups.

Using TinyOS native formats facilitated rapid development. It opened a suite of open source tools for development and testing. For example, the TinyOS *oscilloscope* and *Listen* utilities provide a mechanism for debugging, capturing session traffic, and graphically plotting sensor values. Before our personal server application reached maturity, or for occasional validation, this proved invaluable. In addition, this simplified software implementation and avoided a write-from-scratch protocol for these intermediate layers.

With 802.15.4 and TinyOS, we could quickly focus on design and implementation of the WBAN command and message protocol. This was hatched from the plan to facilitate common WBAN operations such as node discovery, sensor configuration, sensor calibration, the ability to receive events and sensor data in real-time, and the ability to display this data in real-time. Any control or feedback capability provided to the user interface must be implemented using the WBAN communication protocol. We strived for a simple set of WBAN commands that were flexible enough to still implement complex user interface functions. As an example, Figure 4.5 illustrates the use of WBAN messages to calibrate and configure an ECG senor which consequently sends notification of a detected heartbeat event.

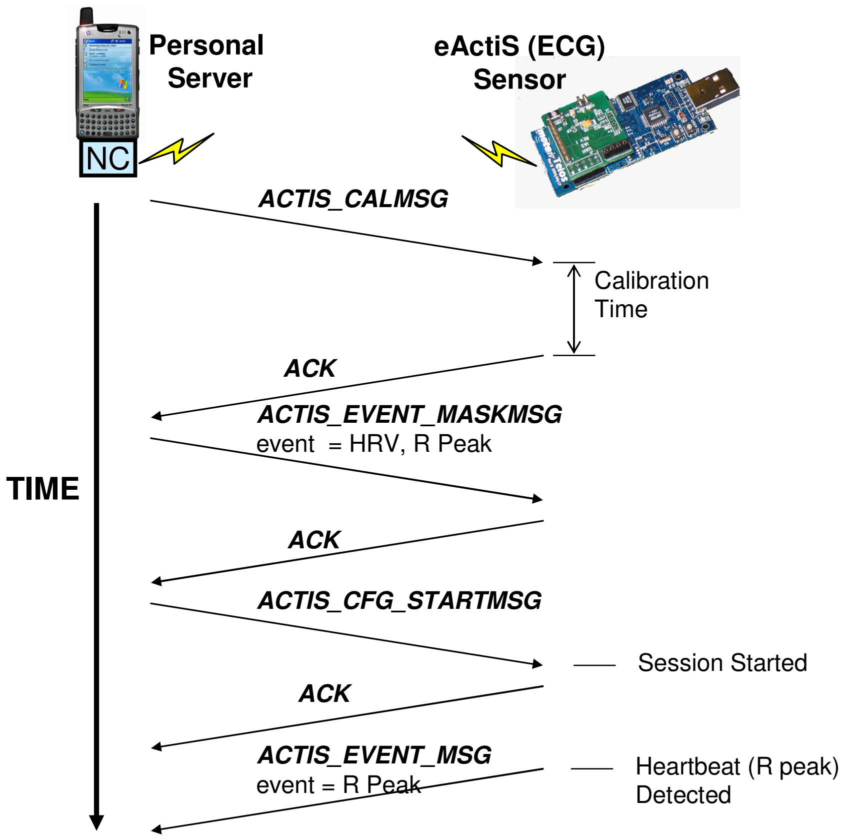


Figure 4.5 Calibration, configuration, and a detected event

Even in a deployed system where intelligent sensors process raw data and transmit application event messages, there may be cases where it is necessary to transmit raw physiological data samples. Such cases become apparent when considering a deployed ECG monitor. When embedded signal processing routines detect an arrhythmic event, the node should send an event message to the personal server which will then be relayed to the appropriate medical server. The medical server, in turn, will provide an alert to the emergency service if the user subscribes to this service. However, a missed heart beat can also be caused by electrode movement. Therefore, it would be useful to augment this event with actual recording of the fragment of unprocessed ECG sensor data. The recording can be used by a physician to evaluate the type and exact nature of the event or to dismiss it as a recording artifact. In this case, the embedded sensor will begin streaming the real-time data to the personal server during a predefined time period. Figure 4.6 illustrates how a detected event can trigger a real-time data stream. The complete working protocol and details of each message is outlined in entirety in Appendix B.

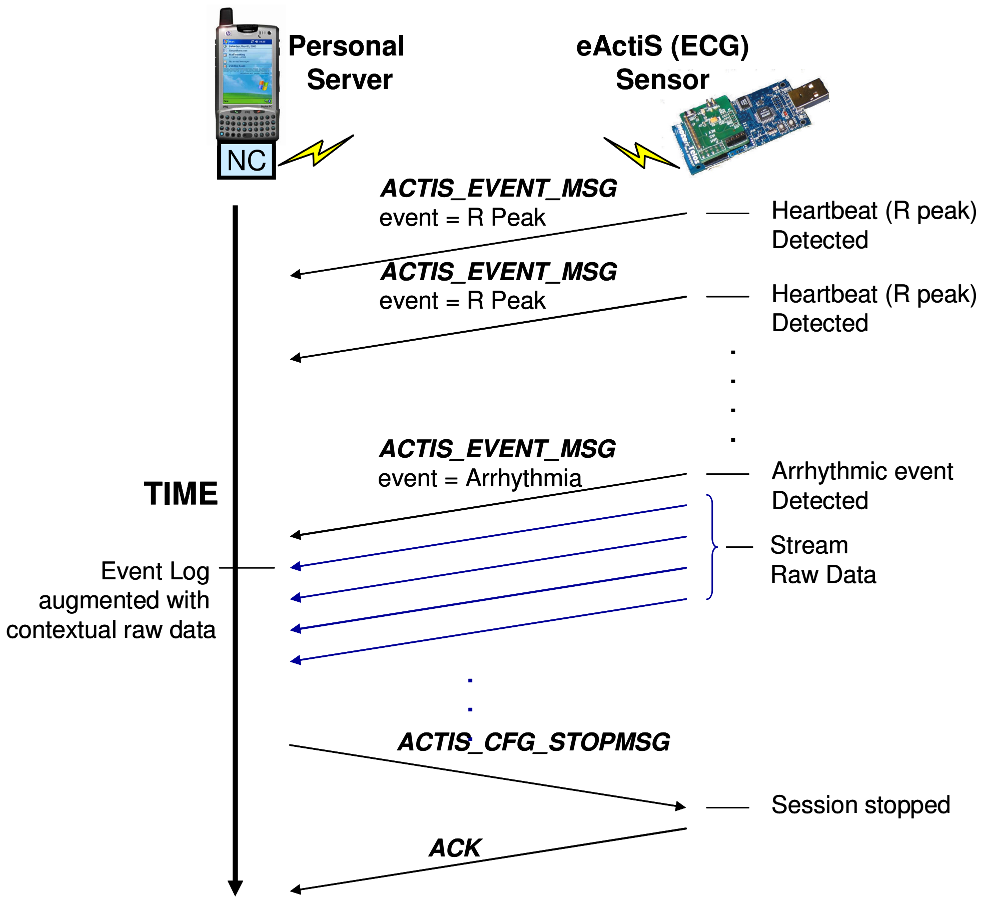


Figure 4.6 A detected event triggering a real-time data stream

CHAPTER 5

EMBEDDED SOFTWARE

The majority of the development, and of particular interest for this thesis, is the embedded software executing on the MSP430-based sensor boards. This software is developed either in the TinyOS development environment, as is the case of the *Tmote Sky* platforms, or without an operating system in the case of the intelligent signal processing modules. The software implementation must contend with extremely resource constrained microcontrollers and must make efficient use of available power. In this chapter we discuss the network coordinator, *ActiS* and *eActiS* sensor platforms, and the intelligent daughter cards. We present our techniques for time synchronization, feature extraction, event management, and USART multiplexing.

5.1 Network Coordinator

The network coordinator software runs within the TinyOS framework on a *Tmote Sky* platform and serves two major functions. First, it acts as a USB to radio gateway, and second, it provides a network time reference for synchronization of other sensors in the WBAN. In its gateway capacity, it handles traffic in each direction. As session management commands arrive from the Personal Server via the USB interface, the messages are appropriately formatted and transmitted from the radio interface. In the opposite direction, it must handle wireless reception of traffic from various sensor nodes in the network. Messages are received on the wireless interface and reformatted for USB transmission. In this role, the network coordinator does not terminate any of the active sessions – this is handled by the USB-connected personal server. The network coordinator also serves as the network time reference, managing physical access to the network. It transmits global time beacons once a second that serve to synchronize system clocks on all nodes, as well as delimit communication frames and provide a point of reference for a sensor node’s timeslot (offset) calculation. Figure 5.1 illustrates the software architecture of the network coordinator software (*ActisGway*).

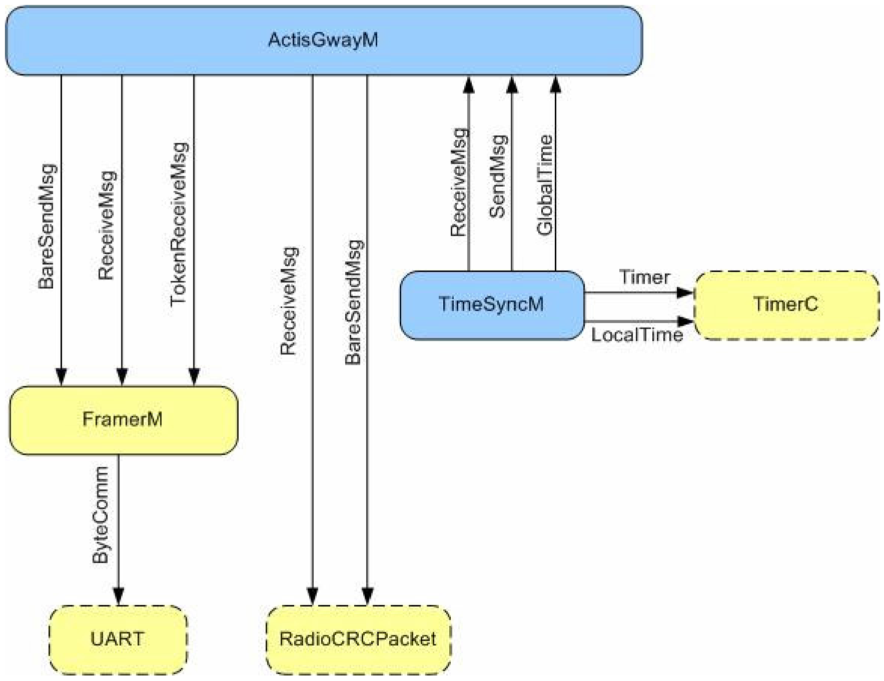


Figure 5.1 Network coordinator software architecture

The gateway functionality is based in part on *TOSBase,* a sample application included in the TinyOS installation. *TOSBase* is a straightforward application which receives packets on the USB and retransmits these on the wireless interface, and vice versa. Because it strictly provides transport only and does not terminate any of the application layer sessions, it relies on lower layer communication interfaces – the *ReceiveMsg* and *BareSendMsg* interfaces connected directly to the *FramerM* module (for USB communications) and *RadioCRCPacket* component for wireless communications.

For our network coordinator, the one exception is the transmission of time synchronization beacons which originate at the *TimeSyncM* module on the network coordinator. The beacons distribute the network coordinator's local time exposed through the TinyOS *TimerC* component and the *LocalTime* interface. As we will see, the network coordinator's local time becomes global time for the rest of the network. Client nodes (sensor nodes) can then include the TimeSyncM component and access this global time via the *GlobalTime* interface.

5.1.1 Time Synchronization

Time synchronization is crucial for providing TDMA timeslot recognition and efficiently sharing the communication channel; it is also critical for intra-WBAN event correlation. Although each sensor node has a local time reference, it is not sufficient due to clock *offset* and *skew* (drift). *Offset* refers to the instantaneous difference in clock times based on when the system clock was started. *Offset* can be partially addressed by synchronizing session start times. A start session message, for example, could cause every node to reset the system clock to zero. *Skew*, on the other hand, refers to small differences in crystal frequency which cause clocks to count at different rates and drift over time. This error is cumulative. Consider two 32.768 KHz crystals differing by a typical 10-20 ppm (parts per million). Over the course of a few hours, the sensor clocks can differ by more than several hundred milliseconds. For correlating physiological events among WBAN sensors, this is unacceptable. This could be partially improved by using more expensive crystals, but this conflicts with our requirements for low system cost and does not solve the problem completely.

A better solution is to employ a time synchronization protocol that gives attention to the issue of clock skew as well as offset. The Flooding Time Synchronization Protocol (FTSP) developed at Vanderbilt University addresses these issues [Maróti04]. In FTSP, nodes in the network agree on a master node. The master node's local time reference becomes the global time reference for the network. The master node transmits periodic beacons containing global time stamps. FTSP features MAC layer time stamping for increased precision and skew compensation with linear regression to account for clock drift. Our modified version exploits the WBAN's star network topology, integrated directly with our TDMA scheme by allowing the beacon messages to also delimit super frame boundaries. We also made changes to balance fast convergence and accuracy.

By using FTSP, we introduce a 32-bit WBAN "jiffy" timestamp based on the master node (network coordinator) 32.768 KHz clock. One "jiffy" is approximately 30.5 μs; this provides adequate resolution and up to 36 hours of health monitoring before rollover. Our sessions are not longer than 24 hours. We handle the rollover by periodically restarting health monitoring sessions from the personal server. It is worth noting that the global "jiffy" timestamp is not an absolute time; however, it is intra-WBAN synchronization that is most important for determining a node's timeslot and for correlating the occurrence of physiological events. We are also interested in time-of-day (wall clock) time stamping, but this can tolerate seconds of errors. For example, suppose an ECG sensor detects an abnormal heart rhythm at the same time a motion sensor detects a body tremor. Precise correlation of these events may be beneficial for a number of reasons; knowing the wall clock time is more informative than anything else. We achieve time of day time stamping through session start times assigned by the personal server assigned. When a new health monitoring session is started, a time of day is written to the session record. The wall clock time stamp of an event within the file is calculated by adding the jiffy time to the session start time.

FTSP works by assigning a timestamp to a globally detected event – the beacon transmission. When the network coordinator sends the message, it inserts the global timestamp. When a receiving node receives the message, it appends a local timestamp. The difference in timestamps represents offset. FTSP estimates skew by storing a number of previous offsets and using linear regression to estimate the change in offset over time. Figure 5.2 illustrates the effect of skew on beacon times. The difference in offset at beacon 4 compared to offset at beacon 2 is attributed to relative clock skew. By estimating skew locally, we benefit from an accurate global time – even if some time has elapsed since the last beacon received. This is not possible by using offset alone.

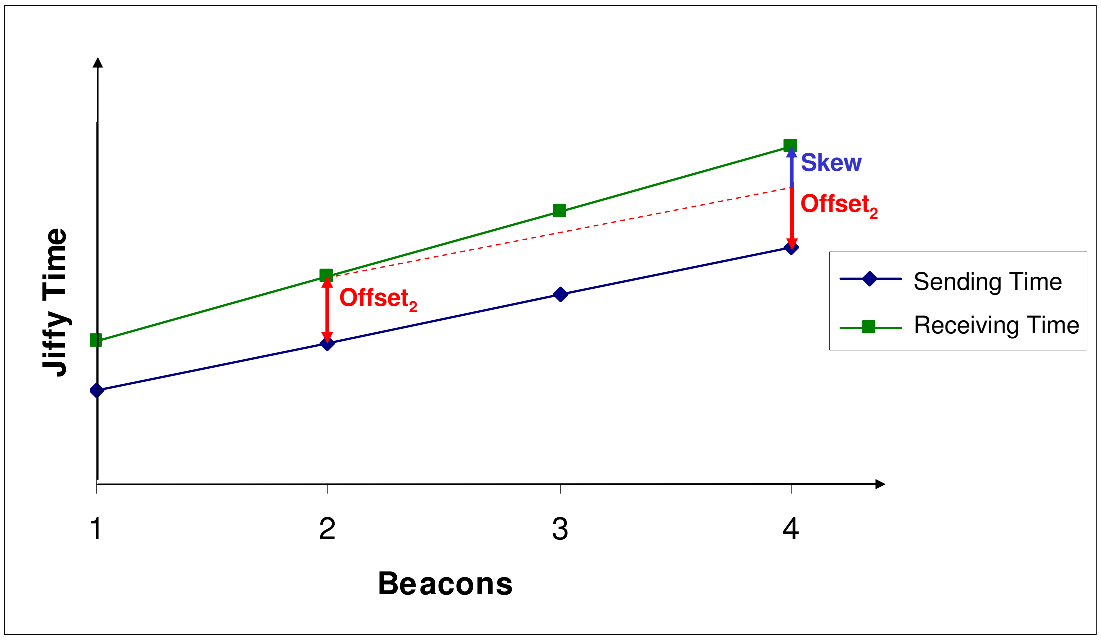


Figure 5.2 FTSP estimates of skew and offset

FTSP keeps a table of the last eight offsets and the corresponding local timestamps. At system start-up, the *TimeSyncM* module clears the table and places the module in a non-converged state. During this state, global time is unavailable to user applications. Once at least three valid beacons have been received and sufficient error tolerances are satisfied, the module enters a converged state and it is able to provide global time to TinyOS applications. The convergence latency is proportional to the beacon interval period; however, the linear regression skew estimates are better served by longer intervals. That is, the technique is more accurate when a larger time span is used in the linear regression. Consequently, we implemented a hybrid scheme where messages are sent more often. During convergence, nodes will process every beacon. After a coarse convergence is achieved, nodes will begin to process every 10th beacon, allowing for a more precise skew estimation.

The accuracy of assessing timestamps directly affects the success of this technique. Message preparation time, asynchronous event handling, and channel access time all introduce non-deterministic latencies in the transmit path. On the receiving node, interrupt latency and software processing overhead can introduce more errors in assessing the receive timestamp. FTSP overcomes this by assigning MAC-layer timestamps. The Chipcon radio provides a Start of Frame Delimiter (SFD) digital output that allows highly deterministic time stamping of the beacon message [Cox05]. Figure 5.3 demonstrates how this works. During beacon transmission, SFD is asserted following transmission of the 802.15.14 SFD field. At the receiving node SFD is asserted when the 802.15.4 SFD field is received (see Figure 5.3). On the *Tmote Sky* boards, this SFD signal is connected to an MSP430 timer capture input pin allowing precise hardware time stamping of the signal edge. Once the timestamp is captured, the network coordinator can insert the timestamp in the beacon after transmission has already begun, but before this field has been transmitted to the radio. On the receive side, the SFD timestamp can be compared to the sending time and used by the FTSP algorithm for offset and skew calculation. This MAC-layer time stamping eliminates all significant non-deterministic latency, leaving only the (highly deterministic) air propagation time.

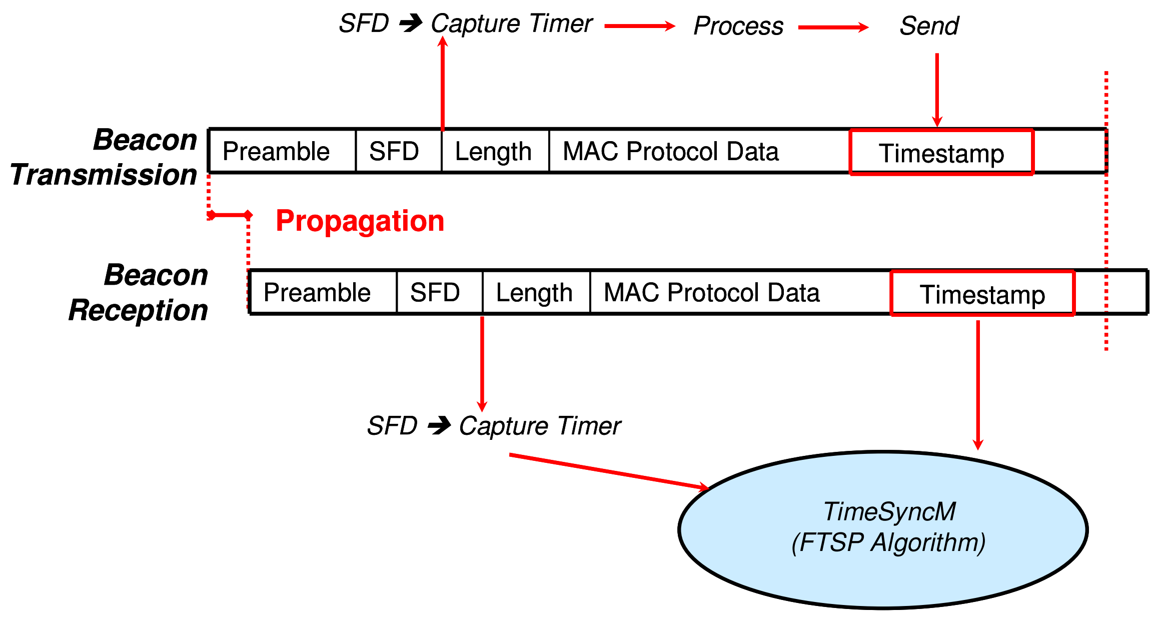


Figure 5.3 FTSP MAC-layer time stamping

The *TimeSyncM* module implements the FTSP algorithms, but due to the MAC-layer time stamping, some code is distributed in the CC2420 software driver (SFD capture). Through the addition of a The *GlobalTime* interface as shown in Figure 5.4, TinyOS applications at the client side (sensor node) can access global time for the purposes of assigning event time stamps and calculating time slot occurrence.

```
interface GlobalTime
{
  /* Gets the current global time */
  command result_t getGlobalTime(uint32_t *time);

  /* Converts given local time to global time */
  /* globalTime = local + offset + skew *(local-syncPoint) */
  command result_t local2Global(uint32_t *time);

  /* Determines how long (in local time) */
  /* until the given global time occurs. */
  command result_t localTilGlobal(uint32_t globalTime, int32_t *time);

  /* Returns Next Beacon time (in global time) */
  command uint32_t getGlobalNextBeacon();

  /* Returns Next Beacon time (in global time) */
  command uint32_t getGlobalLastBeacon();

  command int32_t getOffset();
  command float getSkew();
  command uint32_t getSyncPoint();
  command void resetGlobal();
}
```

Figure 5.4 GlobalTime interface

For testing of the time synchronization protocol, we developed a test bed where the network coordinator and WBAN sensor nodes are all connected to a common wired signal. Sensors detect the signal change and transmit a (locally estimated) global timestamp. WBAN event messages including this timestamp were generated and captured at the personal server. By comparing the timestamps we could determine the difference in the global time estimates. In most cases the node's error was within ±1 jiffy and the average error was approximately $0.1*T_{jiffy}$ or 3 μs [Milenkovic06] [Cox05].

5.2 Sensor Software

The sensor node software runs within the TinyOS framework on a *Tmote Sky* platform and is responsible for communicating with the ISPM daughter card to collect physiological data, processing the signals in real-time, and transmitting the results wirelessly to the personal server. Figure 5.5 depicts the *ActiS* software architecture inside the TinyOS component model. Those components with solid edges are component implementations and those with dashed edges are configuration components and contain one ore more sub-components; *GenericComm, CC2420RadioC, TimerC* and *BusArbitrationC* are in a lighter shade to denote a TinyOS library component. All other components are specific to the *ActiS* software.

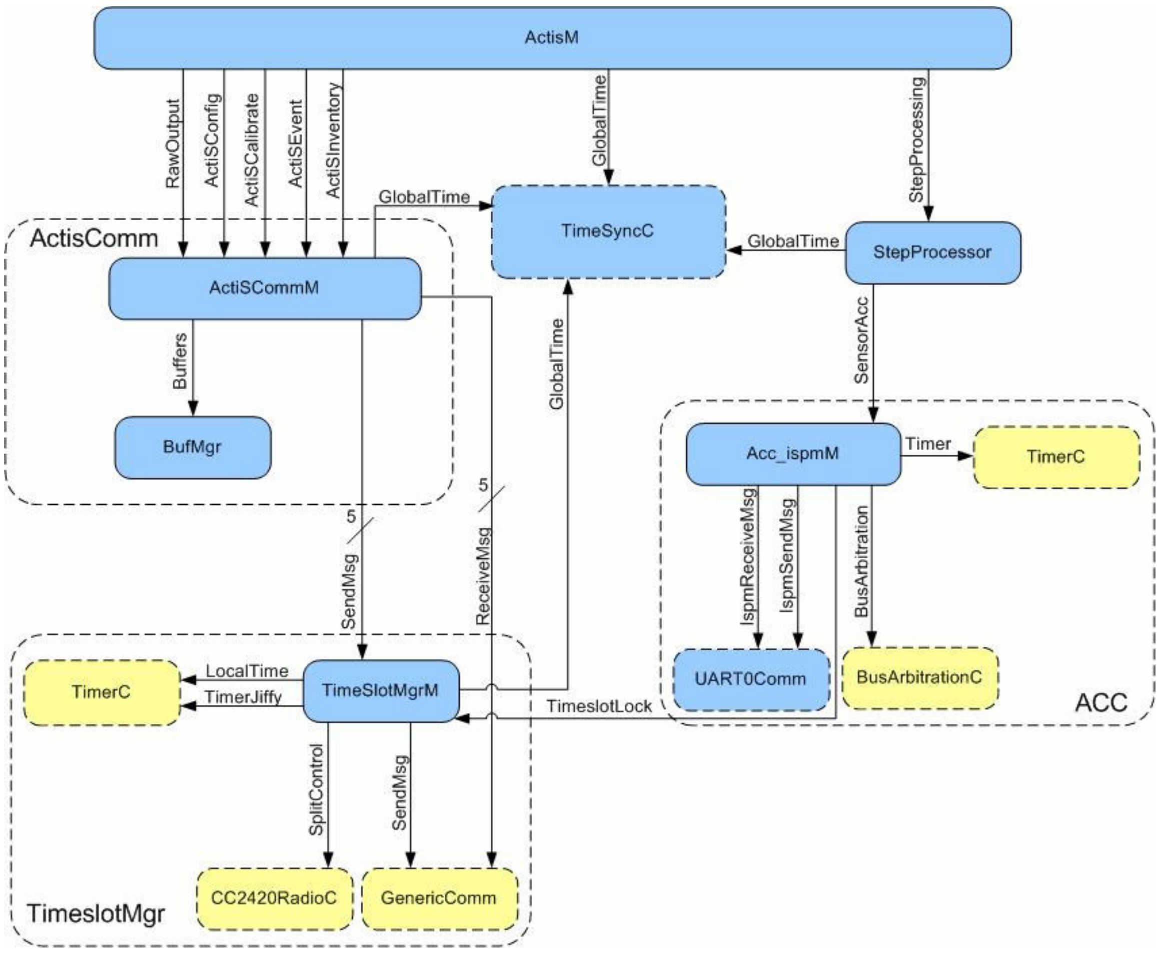


Figure 5.5 *ActiS* software architecture

5.2.1 Feature Extraction

The TDMA scheme presented in Section 4.2 addresses power savings by disabling the radio when it is not in use; however, timeslot length and the total bandwidth requirements are application driven. Additional power savings can be realized by performing on-sensor processing to reduce total transmission bandwidth. A small investment in computational power results in large power savings by minimizing radio communications. Given the high efficiency of MSP430 microcontroller family (250 µA/MIPS) and the relatively expensive 17.4mA / 250Kbps radio, we can perform over 100,000 instructions for the same energy cost of each 55 byte WBAN message. Consequently, we process raw data for the purposes of extracting pertinent features and sending descriptive event messages rather than sending all data over the wireless network. For example, a motion sensor analyzes forces to determine the occurrence of a step or a force exceeding a predefined force threshold; a heart rate sensor monitors ECG data to identify an R-Peak (heartbeat) event or the presence of an abnormal heartbeat. In addition, it may be necessary to extract other features such as estimates of activity induced energy expenditure (AEE) [Bouten97] which result in compilation and summary of data over some period of time, or to classify user activity (running, walking, sitting, etc.).

Consider a motion sensor with tri-axis accelerometer operating at a 200Hz sampling frequency per axis and 16-bit samples. A raw data transmission requires 9600 bps, not considering any compression techniques. Our raw data message format (see Appendix B) packs 10 samples per message. Including overhead, this yields an effective required bandwidth of 26.4 Kbps. Assuming we are interested in step detection, this can be performed on the embedded sensor node and avoid the transmission of raw data. Suppose in the worst case, we can expect to detect four steps per second. Each step results in a unique event message yielding an effective bandwidth of only 1760 bps – a 93% bandwidth savings which helps achieve low channel utilization. Of course this requires real-time signal processing and developers must contend with extremely constrained resources such as low MIPS rate, no hardware floating point support, and small memory capacity.

During development, we used raw data messages to remotely acquire raw data and partially processed data from the sensor nodes. Algorithms were developed off-line by examining session files for various users during walking, standing, sitting, and running. Once the algorithms were working in this controlled environment, we ported these to a nesC environment. Incremental testing of the algorithms was performed. For example, Figure 5.6 illustrates the signal from a mechanical footswitch alongside raw acceleration and calculated velocity from the sensor's Y axis, as well as calculated angle (alpha), representing the tilt of the activity sensor.

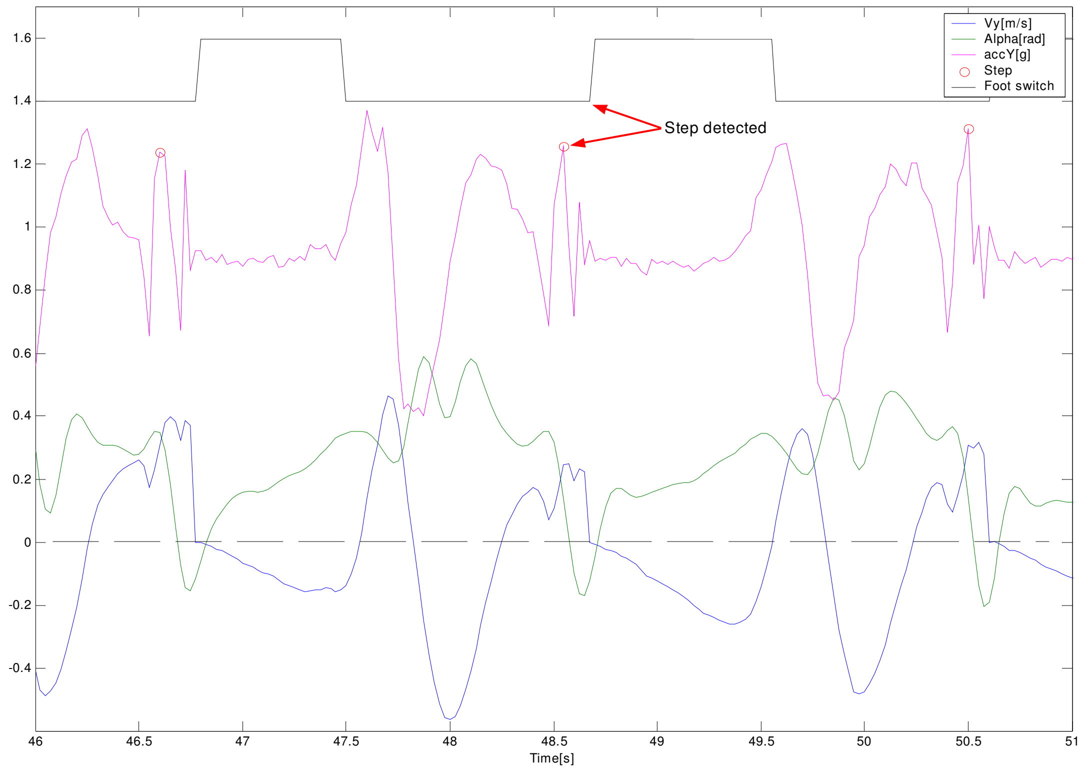


Figure 5.6 Accelerometer based step recognition

The current implementation of the prototype supports R-peak detection using a modified version of the algorithm presented by Pan, et al. [Pan85], user activity (AEE) events based on an algorithm proposed by Bouten, et al. [Bouten97], and a novel step detection algorithm. ST segment processing can be implemented using an approach as presented by Maglaveras [Maglaveras98] and Wheelock [Wheelock99]; however, when running the algorithms, experimental results indicate a reliability problem due to complete processor utilization. Table 5.1 shows the resource requirements of the average real-time processing per data sample for step recognition and R peak detection. For R peak detection, the minimum sampling rate is 100Hz (10ms) yielding 44% CPU utilization; however, communication and ADC sampling overhead will push utilization much higher.

Table 5.1 Cost of WBAN real-time algorithms

Algorithm	Execution Time	Code Size	Data Size
Step recognition	6.2 ms	2858 B	264 B
ECG R peak detection	4.4 ms	6092 B	642 B

Employing intelligent sensor nodes, capable of real-time feature extraction, provide the greatest benefit to WBAN users. Each sensor analyzes the physiological data and determines whether the sampled data constitutes an event of interest. On-sensor feature extraction requires sufficient computational power on each node to run the necessary algorithms in real time. This drives the CPU speed and memory requirements both of which affect power consumption; however, wireless communications remain a dominant factor in power consumption.

5.2.2 Event Message Management

In the WBAN prototype, a sensor node generates an event when a characteristic feature has been recognized. An event is described by event type, a global timestamp, and context-relevant data (see the event message format in Appendix B). For a health monitoring WBAN such as ours, it is imperative that event data can be correlated among the sensors in the distributed sensor network. Timestamps must be applied at the sending node, because transmission of the events can be delayed up to one second due to the TDMA scheme presented in Section 4.2.

Figure 5.7 shows how events are processed in the WBAN for one ECG sensor (timeslot 1) and one motion sensor (timeslot 2). Events occur asynchronously relative to a sensor’s designated timeslot and must be queued for transmission up to one super frame period (one second for our WBAN). At the precise time of event occurrence, an event message with global timestamp is constructed and the message is queued for transmission. When the sensor's scheduled timeslot arrives, all pending messages are transmitted. Currently, we have implemented a redundant data transmission scheme for R peak events where each event message describes both the current event and the previous event, making the system somewhat resilient to packet loss errors. For particularly critical events, however, an explicit acknowledgement is desired to ensure arrival at the Personal Server.

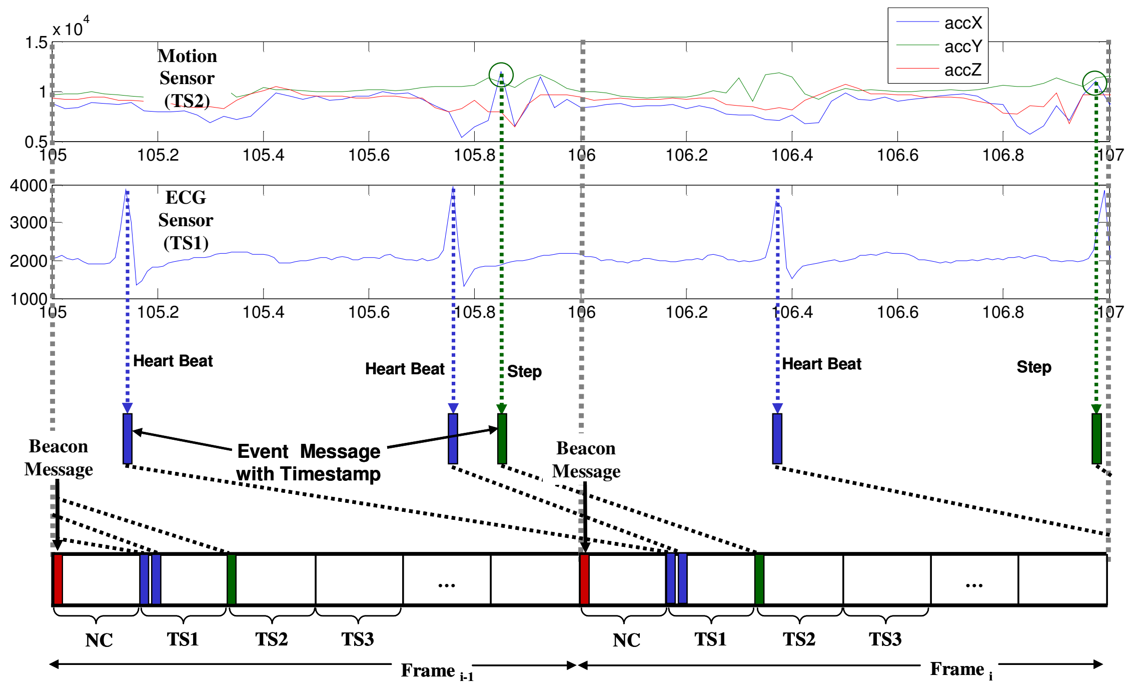


Figure 5.7 Event management in TDMA environment

5.2.3 Dynamic USART Multiplexing

One software challenge is realized by the *Tmote Sky* organization with respect to the two MSP430F1611 USARTs. Unfortunately, USART1 is dedicated to the USB interface – even though this is not used by the *ActiS* / *eActiS* embedded software. Consequently, USART0 is allocated both for CC2420 radio communications and daughter card communications via the 10-pin expansion header. We require both functions concurrently. On one hand, the SPI is needed for transferring WBAN messages between the microcontroller and the radio; and on the other hand, the UART is needed for acquiring data samples from the ISPM daughter card. This motivated our scheme for time multiplexed access and dynamic reassignment of USART0.

To facilitate deferred transmission of messages as well as to synchronize access to USART0, we created the *TimeslotMgr* (Timeslot Manager) component. The Timeslot Manager provides a parameterized *SendMsg* and *ReceiveMsg* interface so that existing components can naturally replace a connection to *GenericComm* with a connection to the *TimeslotMgr*. However, when an application attempts to send a message (by calling *SendMsg( )*), the message is not sent immediately as is the case with *GenericComm.* Instead, the message is queued for transmission pending occurrence of the sensor node's designated timeslot.

Leveraging the power of time synchronization, we can very accurately calculate scheduled beacon timeslots and scheduled transmission timeslots. The *GlobalTime* interface provides the *getGlobalNextBeacon( )* command which returns the global time of the next expected beacon occurrence. This future occurring global timestamp is passed to *localTilGlobal( )* which returns the number of skew-adjusted jiffy ticks from now until its occurrence. This number can be used to schedule a simple one shot timer event. Likewise the designated transmission timeslot can be scheduled by adding a fixed delay (50ms × *timeslot*) to the global timestamp of the last occurring beacon. Once again, this can be converted to the number of local jiffy clocks until its occurrence and scheduled using the local timer. For the benefit of reduced power consumption, the radio is disabled during these inactive periods and then enabled again once the timeslot arrives. In this fashion, the *TimeslotMgr* executes a continuous state machine: *receive beacon*, *disable radio*, *transmit* (until queue is empty or timeslot expires), and *disable radio*.

The implemented state machine is even more complex since USART0 must be shared between the SPI (radio) and UART (ISPM daughter card communications). The *Acc_ispmM* component must reconfigure the USART for UART before it can issue an accelerometer reading. TinyOS provides the *BusArbitrationC* component for this purpose, but it is insufficient for managing this access. This needs to be managed by the timeslot manager so that radio transmissions can be synchronized. For this reason, the *TimeslotMgr* exposes a *TimeslotLock* interface which is acquired by the *Acc_ispmM* component before attempting a reading. This is a split-phase operation so that the *Acc_ispmM* component must wait for the *TimeslotLock.acquired( )* event callback before continuing. Signaling of the event is immediate if the radio is disabled; otherwise, it must wait for a transmission to complete. The *TimeslotMgr* supports interleaving accesses when radio messages are waiting for transmission, but it will not interrupt an in-progress message. Likewise, the *TimeslotMgr transmit* state must check first that it has exclusive access of the resource and if the lock is acquired, it must wait until it is released before beginning a message transmission. Figure 5.8 illustrates this dynamic interleaving and sharing of the USART.

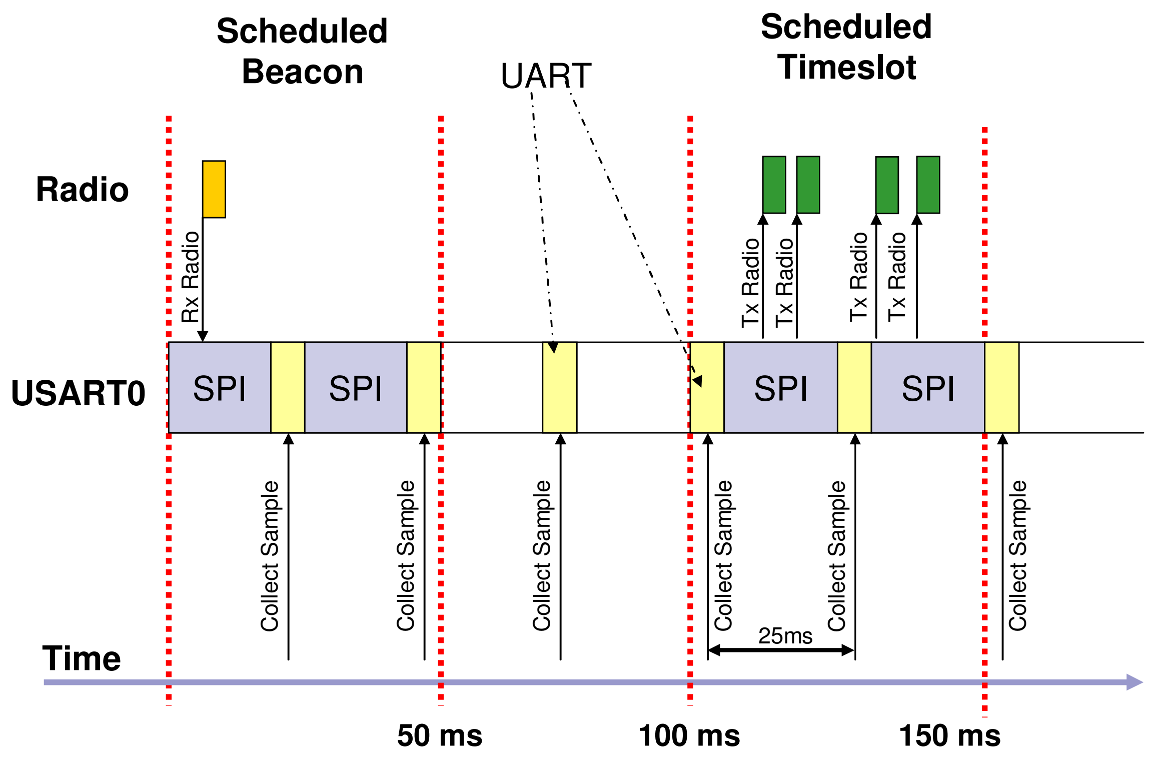


Figure 5.8 Dynamic USART multiplexing for *ActiS* application

5.2.3 Buffer Management

Most TinyOS applications execute a set of dedicated functions defined statically at design time; typically, the messages are transmitted as they occur, requiring a small number of TinyOS messages which can simply be statically allocated as data in the used component. The *ActiS* application is more complicated, as we support a number of runtime configurations that result in different transmission sequences. The *ActiS* node must transmit configuration acknowledgements at the beginning of a session followed by a sequence of event and data messages during a session, while the number and type of messages change during operation. In addition, our TDMA scheme requires message queuing and potentially numerous outstanding messages. Static allocation of the maximum number of message buffers for each message type is not scaleable. For this reason, we introduced the *BufMgr* component as a central buffer manager resource.

Both the *ActiS* and *eActiS* applications require about 25 KB of flash memory for TinyOS and program space and about 1.1 KB RAM for data and an additional 1.4 KB for message buffering (25 buffers with 56 bytes per message). It should be noted that changing the super frame period (discussed in Section 4.2) and increasing latency will also determine the required number of message buffers for an application. Current implementation achieves 52% flash utilization and 25% RAM utilization for a sensor node implemented on a *Tmote Sky* platform.

5.3 ISPM Software

The program running on the ISPM module is a simple application implemented on a resource constrained MSP430F1232. Because of the constraints and simplicity of the application, all ISPM software is implemented in C without the advantages of TinyOS and nesC environments. The module is slaved to the *Tmote Sky* module, responsible only for sampling the accelerometer inputs and providing reliable serial communications to the *Tmote Sky* board on request. The two dual-axis accelerometers are sampled at approximately 200Hz, the data is averaged over the most recent four samples, and the latest (x, y, z) results are passed to the *Tmote Sky* on interrupt request. To conserve power, the processor enters low power mode during idle time. All ISPM level services are interrupt-driven and will wake the processor from this sleep mode as necessary. An interrupt also signifies each UART character transmission, allowing the processor to sleep between characters and only wakes for FIFO management and preparation of the next character.

Mindful of the *Tmote Sky* platform's USART sharing constraints, ISPM to *Tmote Sky* serial communication protocols were designed with a minimalist approach. To conserve precious serial bandwidth, *Tmote Sky* requests are presented as discrete signals invoking interrupts. This also serves as inherent synchronization and delimits the ISPM responses. As a result, no framing protocol or start of message byte is required. The protocol employed only has one overhead byte which indicates the type of data transferred and to what level the data is processed. By minimizing the serial communications, the *Tmote Sky* can quickly acquire the sample and then reconfigure the USART for wireless radio operation.

A high baud rate of 115,200 bps was selected which allows the 7 byte message to be transferred in ~600μs. One challenge was guaranteeing reliable communications at this speed. At this speed, the serial clock must be derived from the higher speed (and unstable) SMCLK versus the lower speed (and relatively stable) ACLK. In both the ISPM and *Tmote Sky* systems, ACLK is directly generated from the stable on-board 32 kHz crystal oscillator. SMCLK, however, is derived from the internal runtime-programmable digitally controlled oscillator (DCO). The integrated DCO is a real estate and cost saving feature, however, its RC-type characteristics make its frequency vary with temperature, voltage, and from one device to the next making it a poor choice for baud rate generators [MSP430x1xx]. This problem, however, can be overcome by runtime tuning of the DCO. By using an internal timer capture and the on-board 32 kHz oscillator as an input, software control of the DCO can be achieved. By measuring the known 32 kHz reference clock with DCO clock ticks, it is possible to calculate the period of SMCLK. Recursive adjustments can be made until the period is satisfactory.

Assuming a 50ppm 32 kHz crystal, the DCO clock can be tuned to an accuracy of 0.5% [Muelhofer00].

The current implementation tunes the DCO once at system initialization and does not attempt to accommodate for temperature or voltage changes that may arise after system initialization. This is suitable for lab conditions and research purposes; however, more robust implementations would require a continuous real-time maintenance of the DCO clock. This would also require a dedicated timer capture. Because all of the MSP4301232 timer captures are utilized for accelerometer sensor sampling, our implementation only uses the timer capture for this purpose at start-up. Once the DCO is tuned, the timer capture is dedicated to accelerometer sampling.

CHAPTER 6

PERSONAL SERVER APPLICATION

We designed the prototype personal server to both address the needs of the research prototype WBAN as well as support for a deployed WBAN system, providing seamless control of the WBAN. The Personal Server application is responsible for node identification and node configuration (controlling the WBAN), sensor fusion, and provides the user interface. In a full multi-tier network as presented in Figure 3.1, the personal server would also be responsible for establishing secure communications with the medical server, upload session files, and download new caregiver instructions. The personal server application was developed in Visual Studio .NET 2003 for both the .NET compact framework (Pocket PC 2003) and full framework (Windows XP). WBAN access is achieved with the Network Coordinator implemented on a *Tmote Sky*.

Figure 6.1 shows the message flow during a typical WBAN health monitoring session. The Personal Server begins a health monitoring session by wirelessly configuring sensor parameters, such as location (left/right ankle, waist, and chest) selection of the type of physiological signal of interest, and specifying events of interest. For example, a motion sensor can be configured to send activity estimate events (AEE) when the level crosses a specified threshold, step recognition events, or raw data. The personal server performs sensor fusion on the multiple data streams, creates session files and archives the information in the patient database. Real-time feedback is provided through the user interface. The user can monitor his / her vital signs and be notified of any detected warnings or alerts.

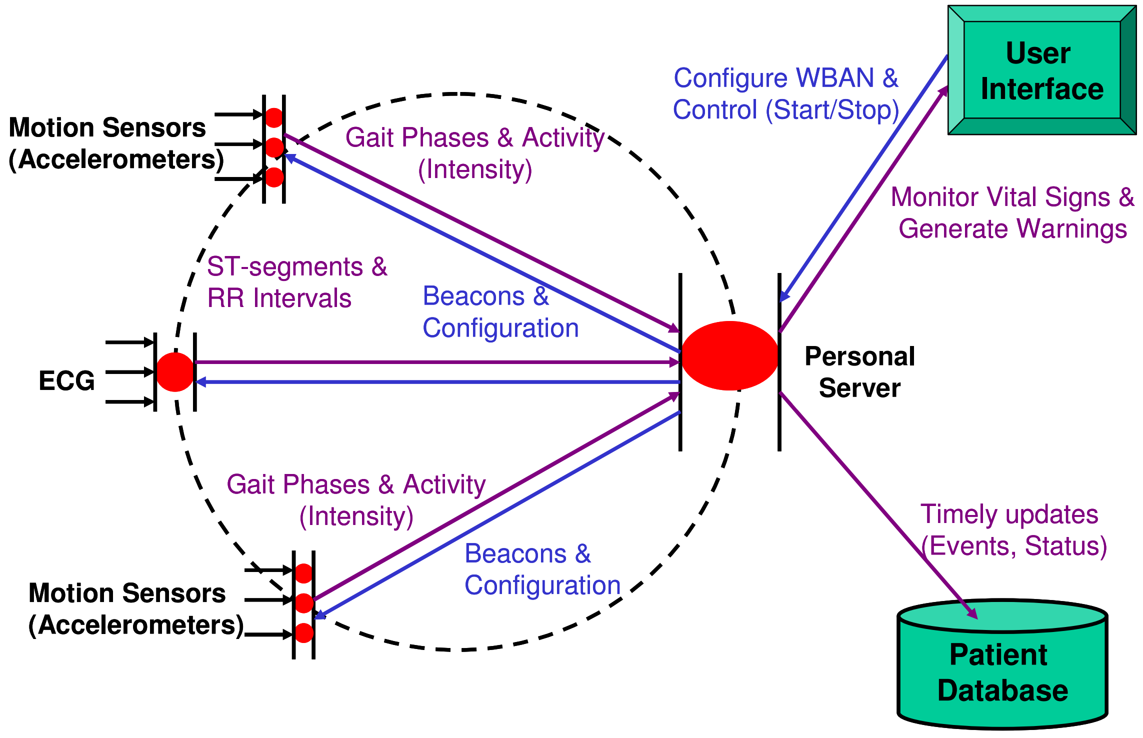
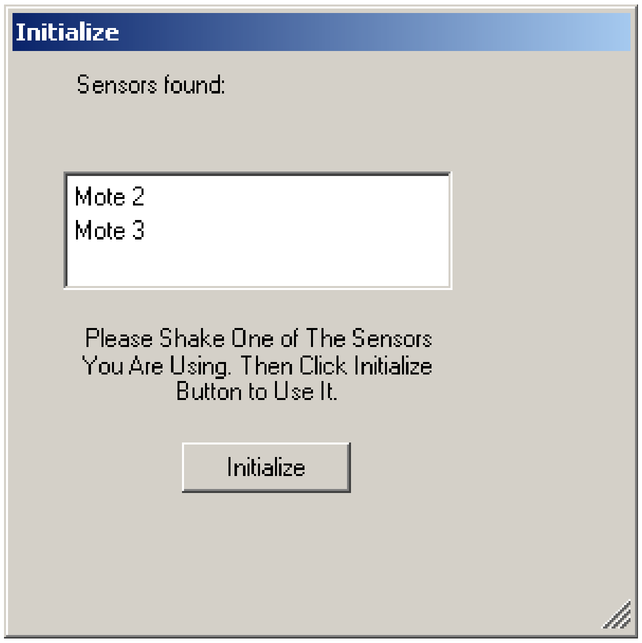


Figure 6.1 Data flows in prototype WBAN

6.1 Sensor Node Identification and Configuration

Sensor node identification requires a method for uniquely identifying a single sensor node to associate the node with a specific function during a health monitoring session. For example, a motion sensor placed on the arm performs an entirely different function than a motion sensor placed on the leg. Because two motion sensors are otherwise indistinguishable, it is necessary to identify which sensor should function as an arm motion sensor and which sensor should function as a leg motion sensor. In order to make node identification user friendly and intuitive, we developed a scheme taking advantage of the inherit motion sensing capabilities of each sensor. We let the user arbitrarily place a motion sensor on his arm or leg, and then we can identify and associate the sensor with the proper function through a series of easy to follow instructions. Our form instructs the user to "shake one of the sensors." While the user is moving the sensor, the PS broadcasts an *ACTIS_EVENT_MASKMSG* requesting all sensors to report activity level estimations. Based on the largest activity estimate returned, the PS can identify the sensor being moved. This interface is more intuitive from a user's perspective, but was easily implemented using WBAN protocol event messages already implemented for event processing. Figure 6.2 illustrates node identification.


Initialize

Sensors found:

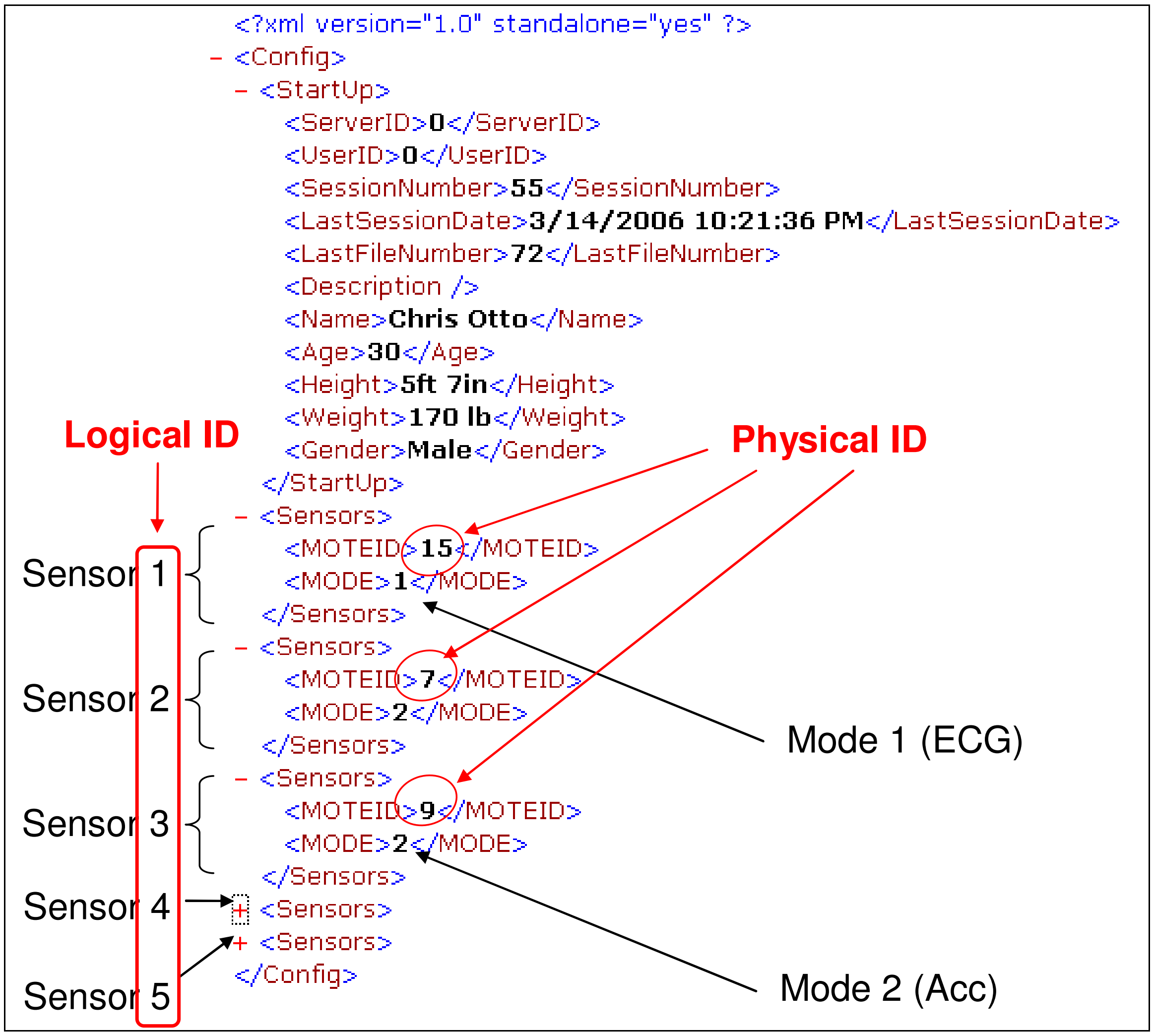
Mote 2
Mote 3

Please Shake One of The Sensors You Are Using. Then Click Initialize Button to Use It.

Figure 6.2 Node identification using personal server

For development, we also implemented a static node assignment scheme. The personal server adds another level of indirection between the mote address (mote ID) and the enumerated sensor number that is presented to the user. This is defined in config.xml.

Figure 6.3 illustrates the association between the personal server assigned logical ID and the compile-time assigned physical ID (WBAN Mote ID).



```
<?xml version="1.0" standalone="yes" ?>
- <Config>
  - <StartUp>
      <ServerID>0</ServerID>
      <UserID>0</UserID>
      <SessionNumber>55</SessionNumber>
      <LastSessionDate>3/14/2006 10:21:36 PM</LastSessionDate>
      <LastFileNumber>72</LastFileNumber>
      <Description />
      <Name>Chris Otto</Name>
      <Age>30</Age>
      <Height>5ft 7in</Height>
      <Weight>170 lb</Weight>
      <Gender>Male</Gender>
    </StartUp>
  - <Sensors>
      <MOTEID>15</MOTEID>
      <MODE>1</MODE>
    </Sensors>
  - <Sensors>
      <MOTEID>7</MOTEID>
      <MODE>2</MODE>
    </Sensors>
  - <Sensors>
      <MOTEID>9</MOTEID>
      <MODE>2</MODE>
    </Sensors>
  + <Sensors>
  + <Sensors>
  </Config>
```

Figure 6.3 Static ID assignments via config.xml

The Personal Server and *ActiS* nodes support two types of calibration. The first type is a sensor calibration; its purpose is to accommodate sensor-to-sensor variations and the exact nature of the calibration is sensor dependent. This is typically a one-time calibration and not expected to be a long-term function of the user interface, but certainly necessary for sensor preparation. The second type of calibration is a *session calibration*, required immediately prior to starting a new monitoring session to calibrate the sensor in the context of its current environment. For example, Activity sensors on the leg might need an initial calibration of default orientation on the body.

6.2 Sensor Fusion

The Personal Server is solely responsible for collecting data and events from the WBAN. Each sensor node in the network is sampling, collecting, and processing data. Depending on the type of sensor and the degree of processing specified at configuration, a variety of events will be reported to the Personal Server. User session files are created in real-time using a custom binary file format, and are then converted to a Microsoft Access database for off-line analysis. Table 6.1 shows the format of one record in the binary file; all data values are stored in little-endian format. This file is constructed by fusing event messages from all the sensors in the WBAN. The Personal Server must recognize events as they are received and make decisions based on the severity of the event. The following events are currently supported and recognized by the personal server (complete format is specified in Appendix B):

- STEP (includes timestamp, step length, maximum forces)
- RPEAK – detection of heart beats using recognition of the R phase; the system generates a precise timestamp and time interval between the current and a previous heart beat or R-R interval (RRINT)
- Sensor Error
- Force Threshold Exceeded (Excessive Force)
- User’s activity – AEE (one-second integration of the 3D motion vector)

- Triggered user’s activity – generates an event if the AEE exceeds a specified threshold.

Table 6.1 Personal Server Record Format

Octets	**Field**	**Description**
4	Timestamp	32-bit global jiffy time
2	numValues	Number of values used in *data* field
2	Not used	Reserved for future timing parameters
4	SessionSignalID	Unique identifier formed from concatenation of session number with signal type
20	data	Data dependent, for raw data this is 10 16-bit samples
32 bytes		

6.3 Graphical User Interface

The user interface on the personal server facilitates the user’s start / stopping of health monitoring sessions, configuration, and event management. This is all supported with the current personal server and WBAN nodes. Any control or feedback capability provided to the user interface must be implemented using the WBAN communication protocol. The protocol provides the tools enabling control of the WBAN and defines what can and cannot be accomplished. We strived to keep a simple set of WBAN message types, but still implement complex user interface functions and application flexibility. Figure 6.4 depicts the main personal server screen for controlling the WBAN.

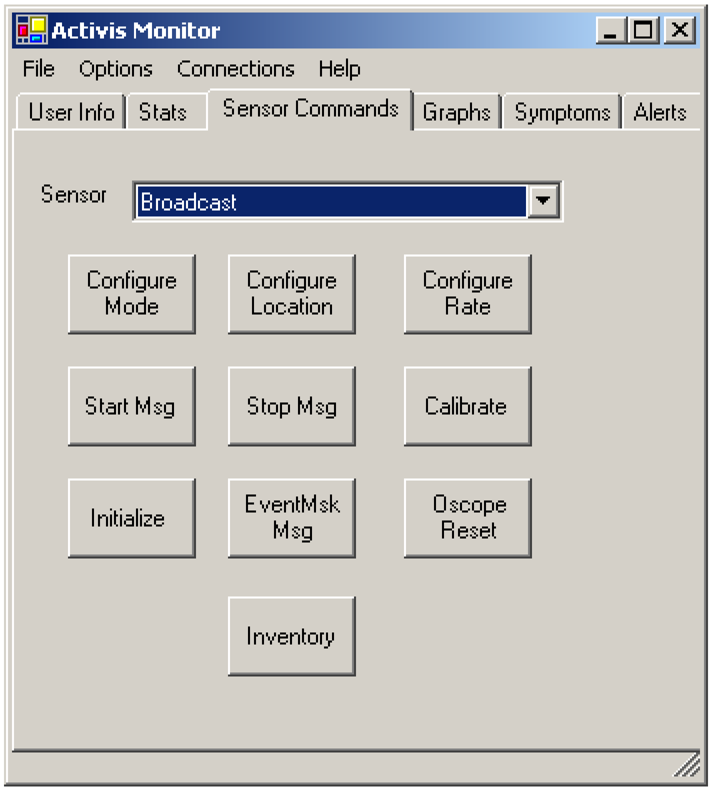


Figure 6.4 Personal Server Control of WBAN

Although all sensors in our system perform on-sensor processing and event detection, there are events where processed and summary events are not sufficient and real-time raw signal capture is necessary. During development, it was invaluable to be able to monitor sensor data in real-time. For heart rate sensors we implemented a single graphical ECG trace; for motion sensors we implemented three traces representing x, y, and z acceleration components on the same graph, as represented in Figure 6.5. This captured data is also stored to a file and can be analyzed off-line to improve step detection algorithms. In most cases, the algorithms were first developed on previously recorded sample data sets. When the algorithms worked well on the sample data sets, they were then implemented on the embedded sensors to run in real-time.

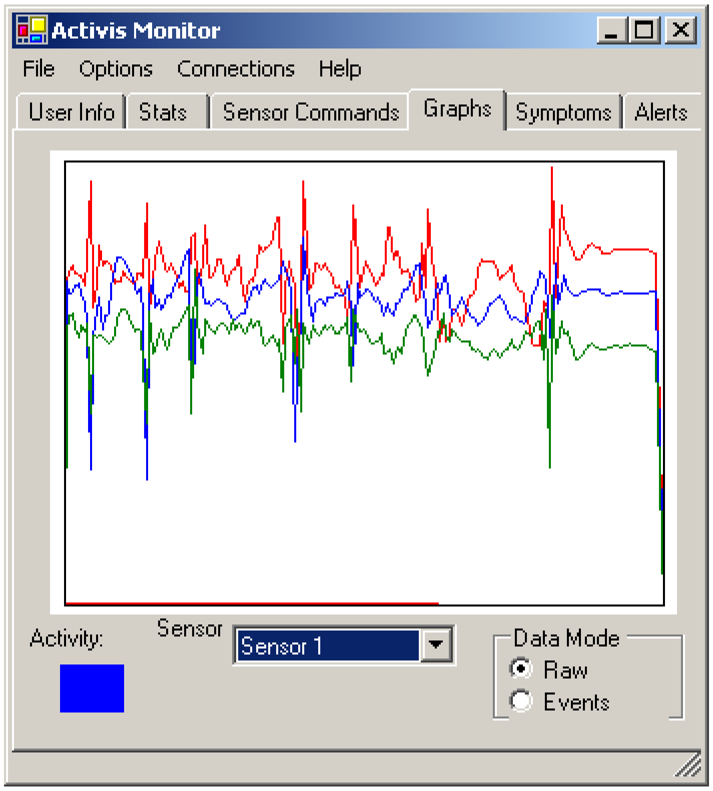


Figure 6.5 Real-time display of raw data

CHAPTER 7

CONCLUSIONS

Wireless Body Area Networks (WBANs), comprised of tiny intelligent physiological sensors, represent a promising addition to wearable systems for health monitoring. Following current trends in advances in size, low power, and dense integration (complete systems on a chip), it is expected that WBAN sensor nodes can be easily integrated into a user's clothing or worn as tiny patches on the skin. The absence of wires and small weight make them unobtrusive and allow ubiquitous, ambulatory health monitoring for extended periods. Integration of WBANs into a broader telemedicine system empowers patients and users with continuous ambulatory monitoring, a chance for remote rehabilitation at reduced cost while adding value, and the earliest possible detection of abnormal health indicators.

This thesis presents a WBAN implementation which consists of multiple sensor nodes, a personal server, and a network coordinator. The sensor boards and network coordinator were built from off-the-shelf wireless sensor platforms with custom-designed intelligent physiological sensor boards for ECG and activity monitoring. The nodes communicate wirelessly using standards-based IEEE 802.15.4 and a novel, health-monitoring specific, power-efficient TDMA scheme. In addition, we introduced novel techniques for time synchronization including an original hybrid convergence scheme and an event management scheme motivated by power efficiency. Most importantly, we demonstrated the capability to perform on-sensor processing – particularly with motion sensors – and outlined the power savings realized when features are extracted on-sensor, avoiding expensive wireless transmissions. We also developed a personal server application in order to facilitate system development and illustrate the WBAN's practical use in collecting data and events.

Perhaps the most lasting contribution is a working prototype for continued research in this area. Future work should consider enhancing algorithms for real-time signal processing (step detection, assessing user activity, ECG processing), using the WBAN to participate in an organized research trial, implementing encryption in the WBAN, and implementing a prototype medical server to illustrate the potential for research data mining. Other possibilities for future work are to implement compression to further decrease radio transmissions and enhance power saving capabilities, increase sensor sampling rates, and implement local flash storage to allow sensors to continue data collection in the absence of a personal server. In addition, runtime tuning of the DCO should be explored for achieving reliable daughter card communications under a number of adverse conditions.

APPENDICES

APPENDIX A

WBAN MESSAGE FORMAT

Figure A.1 depicts a dissected WBAN message and the format of the TinyOS message. As the figure indicates, several fields of the TinyOS message header are directly shared with the 802.15.4 frame (length, frame control field, data link sequence number, and address). The 802.15.4 addressing supports 4 – 20 bytes in the address field; however, this is fixed to four bytes in the TinyOS implementation. It should be noted that the *mote ID* address refers to the destination address. Applications that require the source address of the node should include this in the TOSH_DATA field (application message).

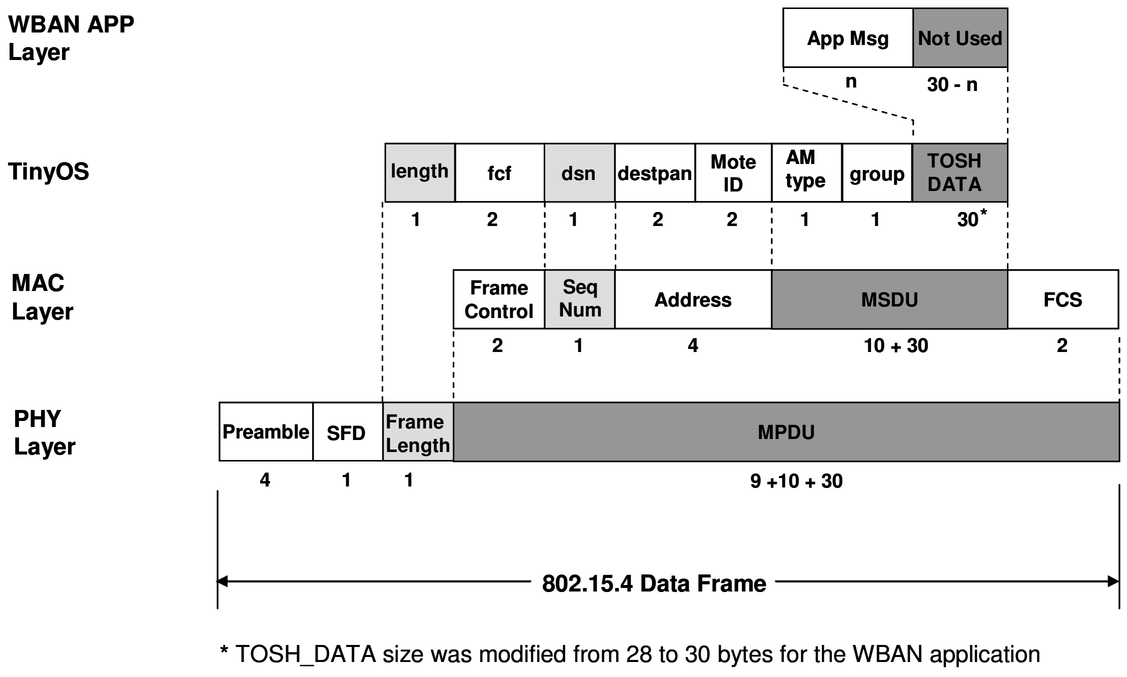


Figure A.1 WBAN protocol stack

The default installation of TinyOS uses a fixed 28 byte message payload (TOSH_DATA) for all transmissions. During our WBAN implementation we extended the message payload from 28 bytes to 30 bytes to accommodate a 32-bit timestamp on raw data messages and oscilloscope messages (see Appendix B). This payload is fixed in size so that unused portions are simply not used. Application developers benefit from a simple implementation and avoid the woes of dynamic memory allocation on extremely resource constrained systems. The Active Message Type (AM Type) field is analogous to a UDP port number. It facilitates transport functionality and multiplexing of multiple application sessions over the physical link. Application endpoints agree on the AM Type for a given application and then the representation of the data within the payload. The one byte group ID can be used for upper layer multiplexing techniques – partitioning wireless networks into multiple groups.

APPENDIX B

WBAN ACTIS APPLICATION MESSAGES

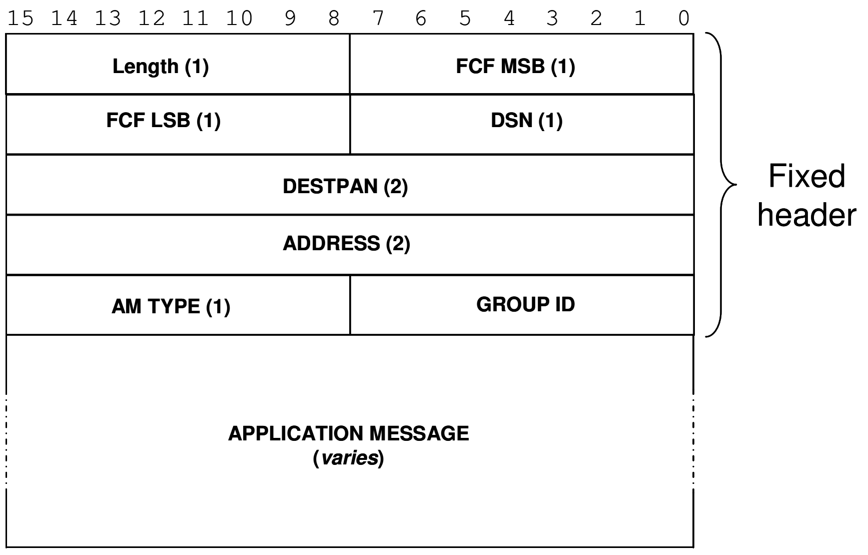


Figure B.1 General WBAN message format including TinyOS header

Table B.1 TinyOS header field descriptions

Field	Octets	Description
Length	1	Length of message
FCF	2	Frame Control Field (see 802.15.4)
DSN	1	Data link sequence number (maintained in radio driver)
DESTPAN	2	Destination PAN identifier (if inter-PAN addressing should be used)
Address	2	Destination Mote ID
AM Type	1	Active Message Type (see Table B.2)
Group ID	1	User defined Group Identifier

Table B.2 *ActiS* WBAN Active Message Types

AM Type	**Value**	**Description**
AM_OSCOPEMSG	10 (0x0A)	Raw Data / Oscilloscope application
AM_OSCOPERESETMSG	32 (0x20)	Reset the sample number for the next sample
AM_ACTIS_CFG_MODEMSG	71 (0x47)	configures sensor mode of operation
AM_ACTIS_CFG_LOCMSG	72 (0x48)	Configures sensor location
AM_ACTIS_CFG_RATEMSG	73 (0x49)	configures sensor sampling rate
AM_ACTIS_CFG_STARTMSG	74 (0x4A)	Begin operation
AM_ACTIS_CFG_STOPMSG	75 (0x4B)	Stop operation
AM_ACTIS_CALMSG	76 (0x4C)	Initiate sensor calibration
AM_ACTIS_EVENT_MASKMSG	77 (0x4D)	Configure event masking on sensor
AM_ACTIS_INVMSG	78 (0x4E)	Query sensor inventory
AM_ACTIS_CFG_ACKMSG	81 (0x51)	Return for any config message
AM_ACTIS_CAL_ACKMSG	82 (0x52)	Calibration complete
AM_ACTIS_EVENTMSG	83 (0x53)	Report Sensor Event
AM_ACTIS_INV_RESPMSG	88 (0x58)	Sensor Inventory Response
AM_TIME_SYNCMSG	170 (0xAA)	Beacon Message

B.1 Personal Server to Sensor Node Messages

B.1.1 OSCOPE RESET (AM_OSCOPERESETMSG = 0x20)

The *Oscope Reset* message is used by the personal server to reset the sequence number on any ongoing raw data message stream. This message is zero bytes in length (TOS length fields should be set to zero).

B.1.2 CONFIGURE OPERATING MODE (AM_CFG_MODEMSG = 0x47)

The *Configure Operating Mode* message is used by the personal server to configure the sensor node's mode of operation.

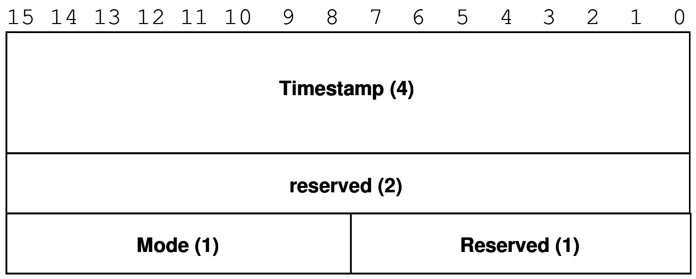


Figure B.2 *Configure Operating Mode* message format

Table B.3 *Configure Operating Mode* message fields

Octets	Field	Description
4	Timestamp	32-bit global timestamp
2	Reserved	Not implemented at this time
1	Mode	Mode of operation (see Table B.4)
1	Reserved	Not implemented at this time
8 bytes		

Table B.4 *ActiS* Operation Modes

Value	Operating mode
0	Disabled
1	ECG
2	Motion Sensor

B.1.3 CONFIGURE LOCATION (AM_CFG_LOCMSG = 0x48)

The *Configure Location* message is used by the personal server to configure the sensor node's location. This is especially important for motion sensors. For example, a motion sensor on the leg or foot would responsible for step detection, while a motion sensor on the waist is not. In addition, the algorithms for estimating energy expenditure might be different.

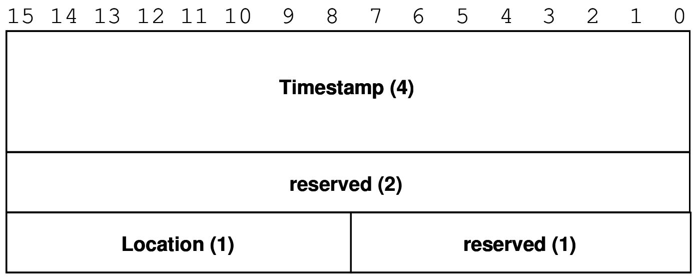


Figure B.3 *Configure Location* message format

Table B.5 *Configure Location* message fields

Octets	Field	Description
4	Timestamp	32-bit global timestamp
2	Reserved	Not implemented at this time
1	Location	Location (see Table B.6)
1	Reserved	Not implemented at this time
8 bytes		

Table B.6 *ActiS* Locations

Value	**Location**
1	Left Foot
2	Right Foot
3	Left ankle
4	Right ankle
5	Left knee
6	Right knee
7	Left thigh
8	Right thigh
9	Waist front
10	Waist back
11	Waist side
12	Center back
13	Left wrist
14	Right wrist
15	Left elbow
16	Right elbow
17	Left upper arm
18	Right upper arm
19	Forehead
20	Top of head
21	Back of head

B.1.4 CONFIGURE RATE (AM_ACTIS_CFG_RATEMSG = 0x49)

The *Configure Rate* message is used to configure the sensor node's sampling rate.

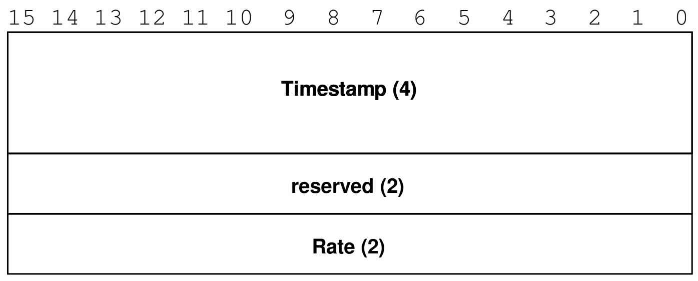


Figure B.4 *Configure Rate* message format

Table B.7 *Configure Rate* message fields

Octets	Field	Description
4	Timestamp	32-bit global timestamp
2	Reserved	Not implemented at this time
2	Rate	Sampling Rate (Hz)
8 bytes		

B.1.5 CONFIGURE START (AM_ACTIS_CFG_STARTMSG = 0x4A)

The *Configure Start* message is used to start a health monitoring session.

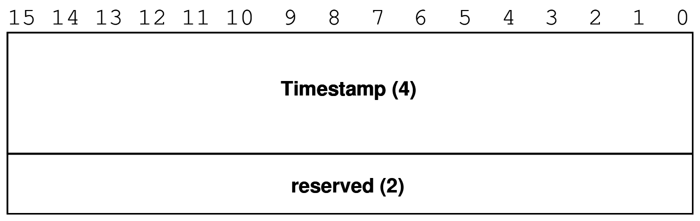


Figure B.5 *Configure Start* message format

Table B.8 *Configure Start* message fields

Octets	Field	Description
4	Timestamp	32-bit global timestamp
2	Reserved	Not implemented at this time
6 bytes		

B.1.6 CONFIGURE STOP (AM_CFG_STARTMSG = 0x4B)

The *Configure Stop* message is used to stop a health monitoring session.

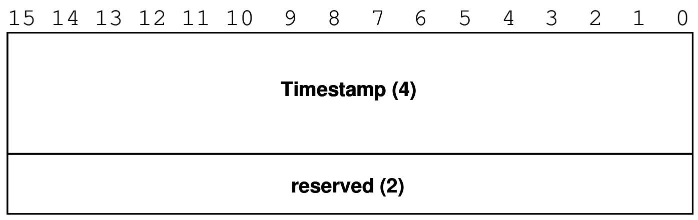


Figure B.6 *Configure Stop* message format

Table B.9 *Configure Stop* message fields

Octets	Field	Description
4	Timestamp	32-bit global timestamp
2	reserved	Not implemented at this time
6 bytes		

B.1.7 CALIBRATION (AM_CALMSG = 0x4C)

The *Calibration* message is designed primarily for motion sensors. The message is sent by the personal server to commence calibration. There are two primary types of calibration: Detailed one-time sensor calibration and session calibration.

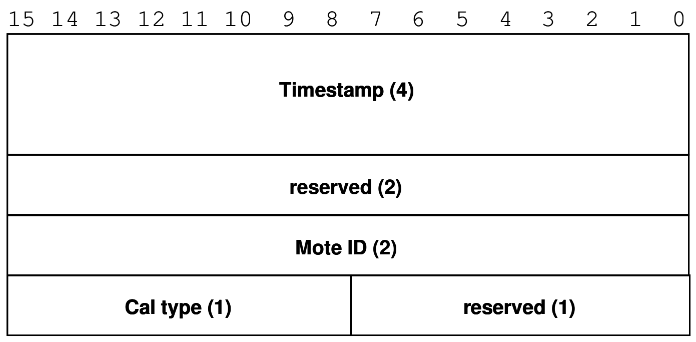


Figure B.7 *Calibration* message format

Table B.10 *Calibration* message fields

Octets	Field	Description
4	Timestamp	32-bit global timestamp
2	Reserved	Not implemented at this time
2	Mote ID	ID of sensor to calibrate
1	Cal type	Type of calibration (See Table B.11)
10 bytes		

Table B.11 *ActiS* Calibration Types

Value	Calibration Type
1	sensor calibration (x-y plane)
2	Sensor calibration (z plane)
3	Session calibration

B.1.8 EVENT MASK (AM_EVENT_MASKMSG = 0x4D)

The *Event Mask* message is used to sensitize a sensor to a specific event of interest.

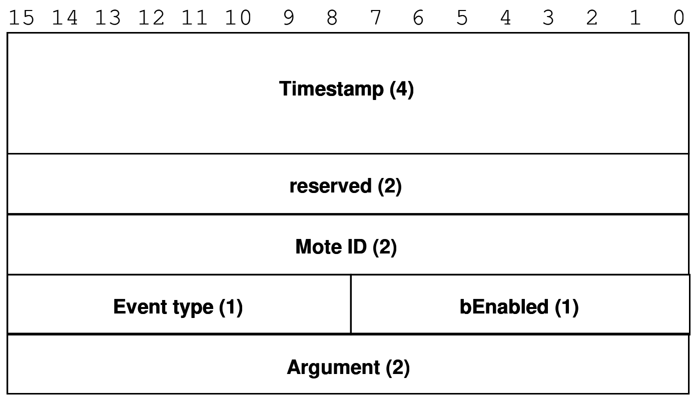


Figure B.8 *Event Mask* message format

Table B.12 *Event Mask* message fields

Octets	Field	Description
4	Timestamp	32-bit global timestamp
2	reserved	Not implemented at this time
2	Mote ID	ID of sensor
1	Event Type	Event of interest (See Table B.13)
1	bEnabled	0 = mask events, 1 = sensor should notify on event detection
2	Argument	Context sensitive (See Table B.13)
12 bytes		

Table B.13 *ActiS* Event Types and Argument in Event Mask message

Value	**Event Type**	**Argument Definition**
1	Reset (cannot be masked)	Not used
2	Sensor Error (cannot be masked)	Not used
3	Step Detected	Not used
4	R Peak Detected	Not used
5	Force Threshold	Minimum force for reporting
6	AEE (Activity induced Energy Estimate)	Minimum AEE for reporting
7	All (enable all events supported with default thresholds)	Not Used

B.1.9 INVENTORY (AM_INVENTORYMSG = 0x4E)

The *Inventory* message is sent by the personal server to query status and function of an individual sensor. When sent with a broadcast address, it can be used to search or locate sensors in a WBAN. The "full" field can be used to set the length of the reply (full status or brief status).

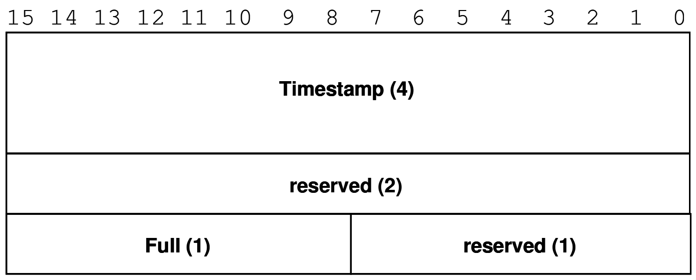


Figure B.9 *Inventory* message format

Table B.14 *Inventory* message fields

Octets	Field	Description
4	Timestamp	32-bit global timestamp
2	reserved	Not implemented at this time
1	Full	0 = short response, non-zero = full response
1	reserved	Not implemented at this time
8 bytes		

B.1.10 TIME SYNC BEACON (AM_TIME_SYNCMSG = 0xAA)

The *Time Sync Beacon* message is sent from the network coordinator to propagate time synchronization to sensor nodes in the WBAN. This message also delimits the super frame boundary.

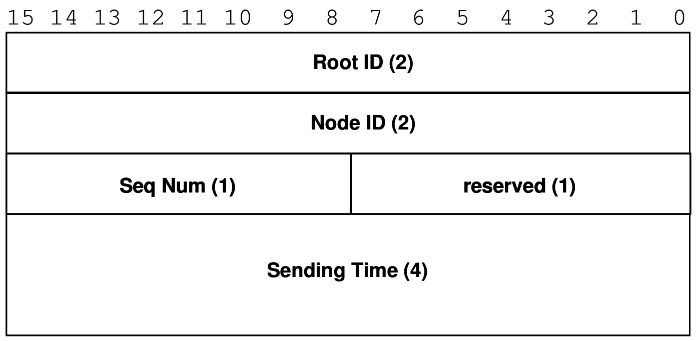


Figure B.10 *Time Sync Beacon* message format

Table B.15 *Time Sync Beacon* message fields

Octets	Field	Description
2	Root ID	Address of network coordinator
2	Node ID	Address of sending node
1	Seq Num	Beacon sequence number
1	Reserved	Not implemented at this time
4	Sending Time	NC assessed time stamp
10 bytes		

B.2 Sensor Node to Personal Server Messages

B.2.1 RAW DATA / OSCOPE (AM_OSCOPEMSG = 0x0A)

The *Raw Data / Oscope* message is sent by sensor nodes to transfer raw data in blocks of 10 discrete 16-bit samples at a time. This message is modified only slightly from the TinyOS Oscilloscope demonstration program included with the TinyOS distribution. The addition of the timestamp field is the only change, and it is carefully placed at the end of the message to maintain compatibility with the Java *oscilloscope* utility distributed with TinyOS.

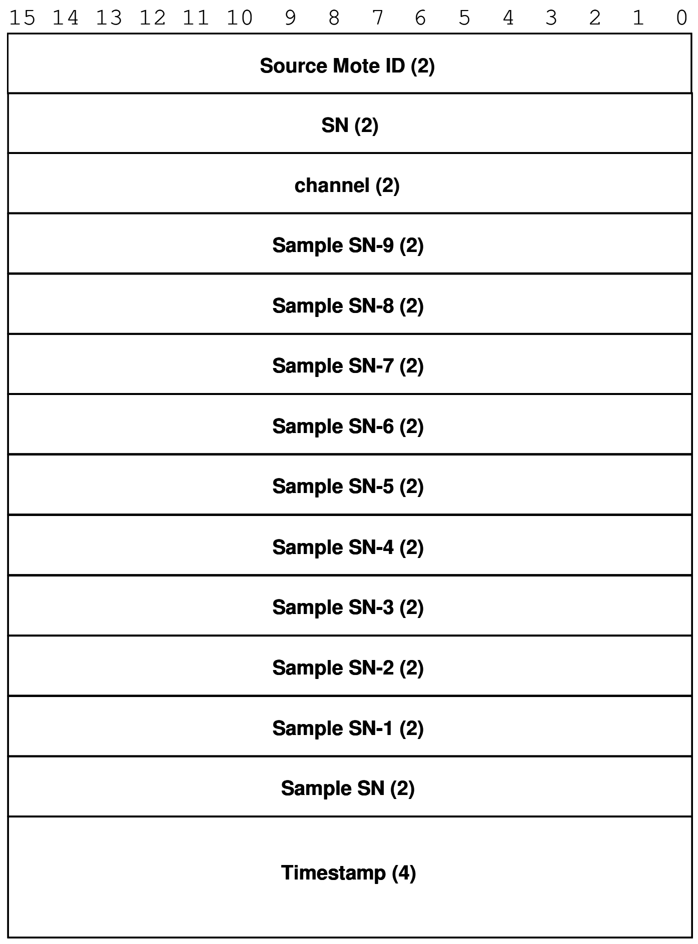


Figure B.11 *Raw Data / Oscope* message format

Table B.16 *Raw Data / Oscope* message fields

Octets	Field	Description
2	Source Mote ID	Address of sending node
2	SN	Sample number corresponding to the last sample
2	Channel	Signal ID (see Table B.17)
20	Sample SN-9..SN	10 16-bit data samples
4	Timestamp	32-bit jiffy timestamp
30 bytes		

Table B.17 WBAN defined signal IDs (channel field)

channel	Signal name	Description
0	ACCX	Accelerometer, x-axis
1	ACCY	Accelerometer, y-axis
2	ACCZ	Accelerometer, z-axis
3	ECG1	ECG, signal 1
4	ECG2	ECG, signal 2
5	AUX1	Auxiliary, user debug
6	AUX2	Auxiliary, user debug
7	AUX3	Auxiliary, user debug

B.2.2 CONFIGURE ACKNOWLEDGEMENT (AM_CFG_ACKMSG = 0x51)

The *Configure Acknowledgement* message is sent from the sensor node to the personal server in response to one of the configuration messages. The original configuration message is included for reference and a response code is supplied as indicated in Table B.18.

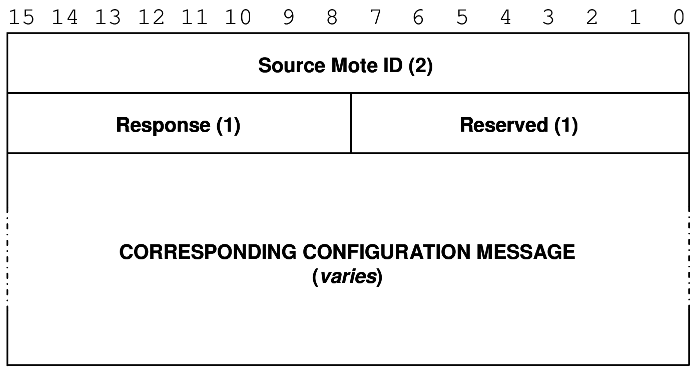


Figure B.12 *Configure Acknowledgement* message format

Table B.18 *Configure Acknowledgement* message fields

Octets	Field	Description
2	Source Mote ID	Address of sending node
1	Response	ACK = 0, NAK = 255
1	reserved	Not implemented at this time
Varies	Corresponding Configuration Message	The message to which this response applies
4 bytes + length of corresponding configuration message		

B.2.3 CALIBRATION ACKNOWLEDGEMENT (AM_CAL_ACKMSG = 0x52)

The *Calibration Acknowledgement* message is sent from the sensor node to the personal server in response to a calibration message.

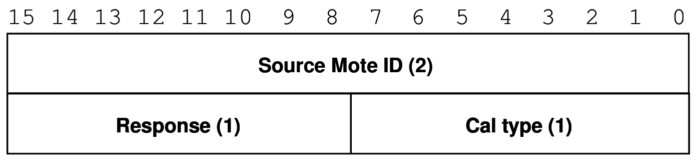


Figure B.13 *Calibration Acknowledgement* message format

Table B.19 *Calibration Acknowledgement* message fields

Octets	Field	Description
2	Source Mote ID	Address of sending node
1	Response	ACK = 0, NAK = 255
1	Cal type	Type of calibration that was performed (See Table B.11)
4 bytes		

B.2.4 EVENT (AM_EVENTMSG = 0x52)

The *Event* message is used to notify the personal server of an on-sensor event. Only non-maskable event types and event types that have been explicitly enabled by the personal server using an *event mask message* can generate event messages.

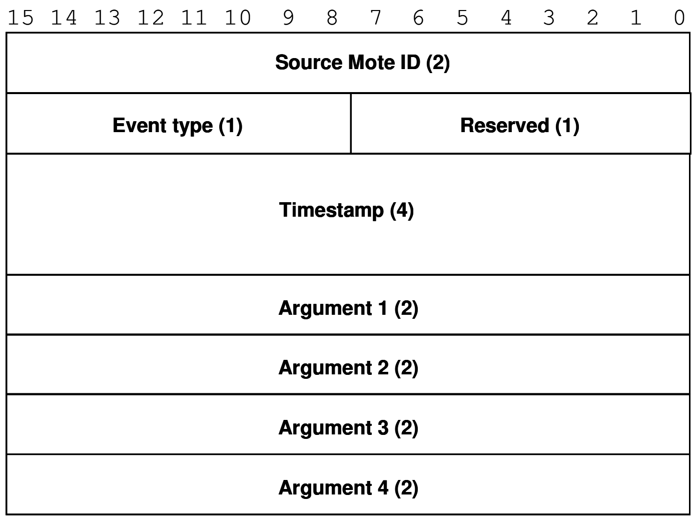


Figure B.14 *Event* message format

Table B.20 *Event* message fields

Octets	Field	Description
2	Source Mote ID	Address of sending node
1	Event Type	
1	Reserved	Not implemented at this time
4	Timestamp	32-bit global timestamp
8	Argument 1 – 4	See Table B.21
16 bytes		

Table B.21 *ActiS* Event Types and Arguments in Event Message

Value	**Event Type**	**Argument Use**
1	Reset (cannot be masked)	Not Used
2	Sensor Error (cannot be masked)	Not Used
3	Step Detected	Arg 1 - Vector force Arg 2 - time since last step Arg3,4 - not used
4	R Peak Detected	Arg 1 - RR interval Arg 2 - Baseline Arg 3 - Timestamp of last RPeak (15..0) Arg 4 - Timestamp of last RPeak (31..16)
5	Force Threshold	Arg 1 – measured force on x-axis Arg 2 – measured force on y-axis Arg 3 – measured force on z-axis Arg 4 – not used
6	AEE	Arg 1 – AEE for previous one second interval Arg 2,3,4 – Not Used

B.2.5 INVENTORY RESPONSE (AM_INVENTORY_RESPTMSG = 0x58)

The *Inventory Response* message is sent in response to the *INVENTORY MESSAGE*. The length of reply is dependent on the *full* field supplied by the personal server in the *INVENTORY MESSAGE*. The full response summarizes the sensors operating configuration and embedded software version. The short response can be used simply to verify a sensor's presence in the network.

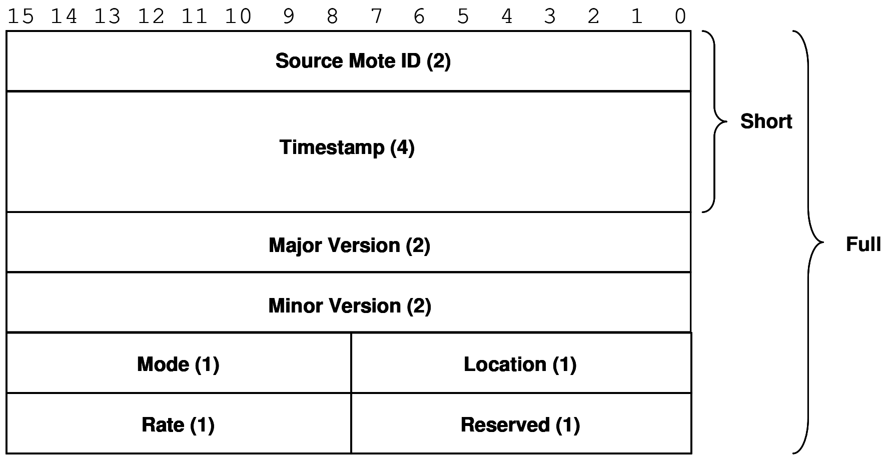


Figure B.15 *Inventory Response* message format

Table B.22 *Inventory Response* message fields

Octets	Field	Description
2	Source Mote ID	Address of sending node
4	Timestamp	32-bit global timestamp
2	Major Version	Most significant Version number
2	Minor Version	Least significant Version number
1	Mode	Operating Mode (See Table B.4)
1	Location	Location of sensor (See Table B.6)
1	Rate	Sampling Rate
1	Reserved	Not implemented at this time
14 bytes	**Full**	
6 bytes	**Short**	

APPENDIX C

TINYOS MODIFICATIONS

This appendix identifies the changes to the standard TinyOS 1.1.15 installation in order to build and run the ActiS applications. The changes are of two types: a) new *TimeSync* library component and b) modifications to existing files. For simplicity, we use *$ROOT* to refer to the system specific root of the TinyOS installation.

There are five files necessary for the TimeSync library component. Those are enumerated in Table C.1. The TimeSync folder does not exist in the standard TinyOS installation; this folder must be created and then simply copy these files from the WBAN project CD into the new folder.

Table C.1 TimeSync library files

File	**Description of File**
$ROOT\TOS\lib\TimeSyn\GlobalTime.h	TimeSync structure definitions
$ROOT\TOS\lib\TimeSyn\GlobalTime.nc	GlobalTime interface definition
$ROOT\TOS\lib\TimeSyn\TimeSyncC.nc	TimeSync configuration
$ROOT\TOS\lib\TimeSyn\TimeSyncM.nc	TimeSync implementation
$ROOT\TOS\lib\TimeSyn\TimeSyncMsg.h	TimeSync Beacon Message Format

The remaining seven files are existing files that must be modified to support the WBAN applications. The easiest way to modify these is to make a backup of the original (TinyOS) file and then copy the files directly from the WBAN project CD – overwriting the file in the process. For future versions of TinyOS (beyond 1.1.15), it becomes necessary to merge the files and understand the differences. Table C.2 enumerates the changes in each file in an effort to help with this process.

Table C.2 Changes to existing TinyOS files

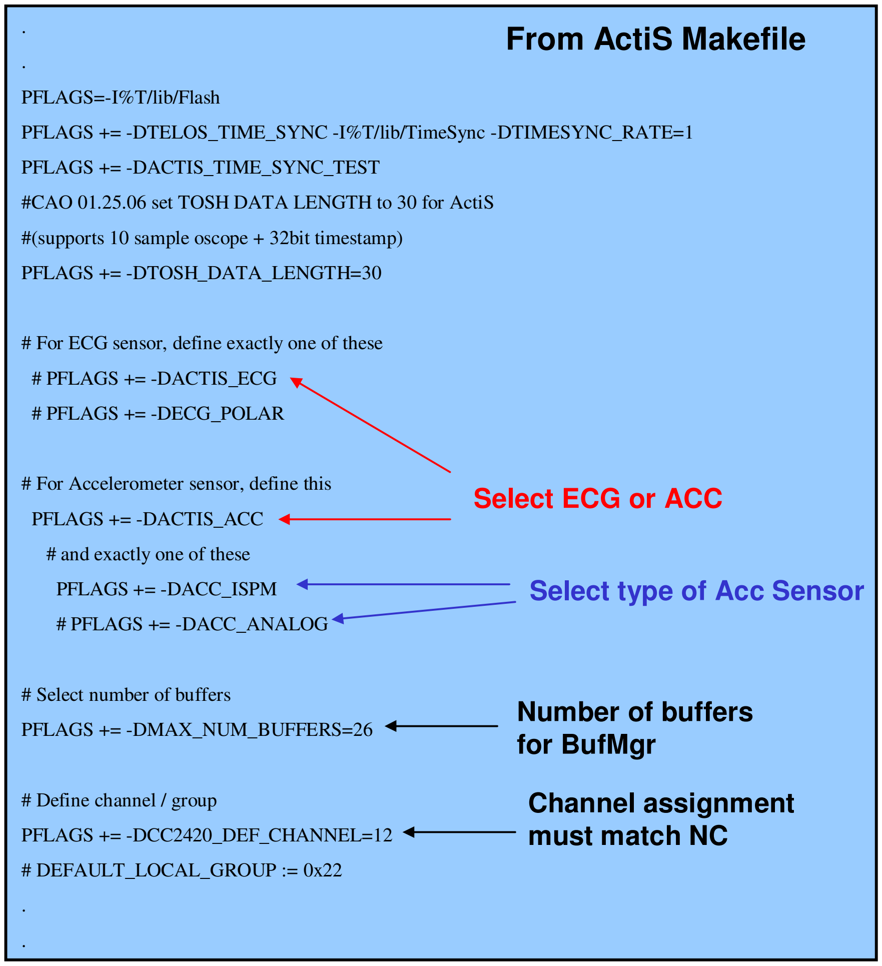
File	**Description of Change**
$ROOT\TOS\lib\CC2420\ CC2420RadioC.nc	Add *LocalTime* interface and *TimerM* component to the CC2420 Radio Configuration
$ROOT\TOS\lib\CC2420\ CC2420RadioM.nc	Add changes to CC2420 driver to support the MAC-layer time stamping of FTSP. All changes are inside #ifdef TELOS_TIME_SYNC
$ROOT\TOS\platform\MSP430\ HPLUSART0m.nc	Changes to *USARTControl.setClockSource( )* to overcome a software defect preventing dynamic selection of clock source (needed for USART multiplexing)
$ROOT\TOS\platform\telos\ AM.h	Added *localTime* to TinyOS packet definition. This is not transmitted over the radio, but is the field used for receive time stamping related to time synchronization protocol.
$ROOT\TOS\platform\telos\ BusArbitrationM.nc	Changes to support our *TimeslotMgr* scheme and USART multiplexing. Specifically, prevent calls to *StdControl.stop( )* from disabling bus arbitration (this is called from *RadioSplitControl stop( )*)
$ROOT\TOS\platform\telos\ hardware.h	Telos rev A GPIO assignments for testing purposes
$ROOT\TOS\platform\telosb\ hardware.h	Telos rev B / *Tmote Sky* GPIO assignments for testing purposes

APPENDIX D

BUILDING AND RUNNING APPLICATIONS IN TINYOS

This appendix is dedicated to providing quick start instructions for new project members. It identifies the common steps for building, installing, and testing WBAN projects.

ActiS and *eActiS* projects are built from the same project directory using makefile flags. Figure D.1 shows an excerpt from the ActiS makefile and a number of the configurable items for building the ActiS project.


From ActiS Makefile
Select ECG or ACC
Select type of Acc Sensor
Number of buffers for BufMgr
Channel assignment must match NC

```
.
.
PFLAGS=-I%T/lib/Flash
PFLAGS += -DTELOS_TIME_SYNC -I%T/lib/TimeSync -DTIMESYNC_RATE=1
PFLAGS += -DACTIS_TIME_SYNC_TEST
#CAO 01.25.06 set TOSH DATA LENGTH to 30 for ActiS
#(supports 10 sample oscope + 32bit timestamp)
PFLAGS += -DTOSH_DATA_LENGTH=30

# For ECG sensor, define exactly one of these
 # PFLAGS += -DACTIS_ECG
 # PFLAGS += -DECG_POLAR

# For Accelerometer sensor, define this
 PFLAGS += -DACTIS_ACC
   # and exactly one of these
    PFLAGS += -DACC_ISPM
    # PFLAGS += -DACC_ANALOG

# Select number of buffers
PFLAGS += -DMAX_NUM_BUFFERS=26

# Define channel / group
PFLAGS += -DCC2420_DEF_CHANNEL=12
# DEFAULT_LOCAL_GROUP := 0x22
.
.
```

Figure D.1 *ActiS* makefile configuration options

The application can be built and installed in the same step. If more than one *Tmote Sky* is connected, the COM port must be specified. This can be determined by using the *motelist* command:

```
Chris Otto@martini /opt/tinyos-1.x/Apps/ActiS
$ motelist
Reference  CommPort   Description
---------- ---------- ----------------------------------------
M4A0M2ET   COM6       tmote sky
```

Issue *make* <platform> [re]*install.*<addr> *bsl,<port>* where platform is either *telosa* or *telosb*. <Addr> is the desired moteID of the sensor and cannot be one; static timeslot assignments are based on moteID and the address should be programmed between 2 and 20. <port> is the zero-based COM port (i.e., COM6 is represented by a 5).

```
Chris Otto@martini /opt/tinyos-1.x/Apps/ActiS
$ make telosb install.3 bsl,5
.
.
.
24984 bytes programmed.
Reset device ...
rm -f build/telosb/main.exe.out-3 build/telosb/main.ihex.out-3
```

Use *reinstall* when the images have already been compiled and only the platform needs to be programmed. Similarly, the network coordinator (ActisGway) can be built from the ActiSGway project folder and an address of 1 should always be specified:

```
Chris Otto@martini /opt/tinyos-1.x/Apps/ActiSGway
$ make telosb install.1 bsl,4
.
.
.
14826 bytes programmed.
Reset device ...
rm -f build/telosb/main.exe.out-1 build/telosb/main.ihex.out-1

Chris Otto@martini /opt/tinyos-1.x/Apps/ActisGway
```

From time to time, it will be handy to use the TinyOS tools for debugging. In particular, the *Listen* tool provides a rudimentary packet sniffer and will display all packets forwarded to the network coordinator USB interface. The *Listen* tool uses the MOTECOM environment variable which selects the port and baudrate. This can be invoked from the *tools* directory:

```
Chris Otto@martini /opt/tinyos-1.x/tools
$ export MOTECOM=serial@COM5:telos

Chris Otto@martini /opt/tinyos-1.x/tools
$ java net.tinyos.tools.Listen
serial@COM5:57600: resynchronising
1E 01 08 EA FF FF FF FF 0A 7D 03 00 86 1A 00 00 99 21 88 21 89 21
8B 21 98 21 83 21 93 21 9F 21 A2 21 83 21 8F B0 1F 00
1E 01 08 EB FF FF FF FF 0A 7D 03 00 86 1A 01 00 15 21 16 21 2A 21
1A 21 0D 21 01 21 29 21 0E 21 0F 21 0E 21 8F B0 1F 00
```

Similarly, the Java scope can be invoked for the purposes of graphing raw data and various forms for outputs using the raw data / oscope message. This tool also requires assignment of the MOTECOM variable:

```
Chris Otto@martini /opt/tinyos-1.x/tools
$ export MOTECOM=serial@COM5:telos

Chris Otto@martini /opt/tinyos-1.x/tools
$ java net.tinyos.oscope.oscilloscope
serial@COM5:57600: resynchronising
```

REFERENCES

[10Blade] 10Blade, http://www.10blade.com

[Accusplit] Accusplit, http://www.accusplit.com

[ADXL202] Analog Devices, Low-Cost ±2 g Dual-Axis Accelerometer with Duty Cycle Output, http://www.analog.com/en/prod/0%2C2877%2CADXL202%2C00.html

[Anliker04] U. Anliker, J.A. Ward, P. Lukowicz, G. Tröster, F. Dolveck, M. Baer, F. Keita, E. Schenker, F. Catarsi, L. Coluccini, A. Belardinelli, D. Shklarski, M. Alon, E. Hirt, R. Schmid, and M. Vuskovic, "AMON: A Wearable Multiparameter Medical Monitoring and Alert System," *IEEE Transactions on Information Technology in Biomedicine, Vol.8, Issue 4,* December 2004, pp. 415-427.

[Body] BodyMedia, Inc. http://www.bodymedia.com

[Bluetooth] http://www.bluetooth.com.

[Bouten97] C.V.C. Bouten, K.T.M. Koekkoek, M. Verduin, R. Kodde, and J.D. Janssen, "A Triaxial Accelerometer and Portable Data Processing Unit for the Assessment of Daily Physical Activity," *IEEE Transactions on Biomedical Engineering*, 44 (3). pp. 136-147.

[CamN] Cambridge Neurotechnology Ltd., http://www.camntech.com/

[CardioLabs] CardioLabs, Inc. http://www.cardiolabs.com/

[CardioNet] CardioNet, http://www.cardionet.com

[CC2420] Chipcon CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver, http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf

[Cox05] D. Cox, E. Jovanov, and A. Milenkovic, “Time Synchronization for ZigBee Networks,” *Proceedings of the 37th SSST,* Tuskegee, Alabama, 2005.

[DeVaul03] R.W. DeVaul, M. Sung, J. Gips, and S. Pentland, “MIThril 2003: Applications and Architecture,” *Proceedings of ISWC 2003*, White Plains, U.S.A., 2003.

[Gay03] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, “The nesC Language: A Holistic Approach to Networked Embedded Systems,” *ACM SIGPLAN Conference on Programming Language Design and Implementation,* San Diego, U.S.A., 2003.

[IEEE802.15.4] IEEE std. 802.15.4 - 2003: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs), http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf, 2003.

[Istepanian04] R.S.H. Istepanian, E. Jovanov, and Y.T. Zhang, "Guest Editorial Introduction to the Special Section on M-Health: Beyond Seamless Mobility and Global Wireless Health-Care Connectivity," *IEEE Transactions on Information Technology in Biomedicine, Vol.8, Issue 4,* December 2004, pp. 405 - 414.

[Jovanov05a] E. Jovanov, A. Milenkovic, C. Otto, and P.C. de Groen, "A Wireless Body Area Network of Intelligent Motion Sensors for Computer Assisted Physical Rehabilitation," *Journal of NeuroEngineering and Rehabilitation*, March 2005, 2:6.

[Jovanov05b] E. Jovanov, A. Milenkovic, C. Otto, P. de Groen, B. Johnson, S. Warren, and G. Taibi, "A WBAN System for Ambulatory Monitoring of Physical Activity and Health Status: Applications and Challenges," *Proceedings of the 27th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Shanghai, China, September 2005.

[Jovanov06] E. Jovanov, and D. Raskovic, "Wireless Intelligent Sensors," in R.H. Istepanian, S. Laxminarayan, C.S. Pattichis, Eds, M-Health: Emerging Mobile Health Systems, Springer, 2006.

[Lorincz04] K. Lorincz, D.J. Malan, T.R.F Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, S. Moulton, and M. Welsh, "Sensor Networks for Emergency Response: Challenges and Opportunities," *IEEE Pervasive Computing, Special Issue on Pervasive Computing for First Response*, Oct/Dec 2004, pp. 16-23.

[Maglaveras98] N. Maglaveras, T. Stamkopoulos, C. Pappas, and M.G. Strintzis, "An Adaptive Backpropagation Neural Network for Realtime Ischemia Episodes Detection: Development and Performance Analysis Using the European ST-T Database," *IEEE Transactions on Biomedical Engineering, Vol. 45,* July 1998. pp. 805-813.

[Martin00] T. Martin, E. Jovanov, and D. Raskovic, “Issues in Wearable Computing for Medical Monitoring Applications: A Case Study of a Wearable ECG Monitoring Device,” *International Symposium on Wearable Computers ISWC 2000*, Atlanta, October 2000.

[Maróti04] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, “The Flooding Time Synchronization Protocol,” *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, Baltimore, U.S.A., 2004.

[Mathie04] M.J. Mathie, A.C.F. Coster, N.H. Lovell, and B.G. Celler “Accelerometry: Providing An Integrated, Practical Method For Long-Term, Ambulatory Monitoring Of Human Movement,” *Physiological Measurement Vol. 25*, February 2004, pp. R1-R20.

[Milenkovic05] A. Milenkovic, M. Milenkovic, E. Jovanov, and D. Hite, “An Environment for Runtime Power Monitoring of Wireless Sensor Network Platforms,” *Proceedings of the 37th SSST*, Tuskegee, Alabama, 2005.

[Milenkovic06] A. Milenkovic, C. Otto, and E. Jovanov, "Wireless Sensor Networks for Personal Health Monitoring: Issues and An Implementation," To appear in *Computer Communications (Special issue: Wireless Sensor Networks: Performance, Reliability, Security, and Beyond)*, Elsevier, 2006.

[Mokdad00] A.H. Mokdad, J.S. Marks, D.F. Stroup, and J.L. Gerberding, "Actual Causes of Death in the United States, 2000," *Journal of the American Medical Association Vol. 291, No. 10*, March 2004, pp. 1238-1241.

[Moteiv] Moteiv, http://www.moteiv.com

[MSP430x1xx] Texas Instruments, MSP430x1xx Family User’s Guide, http://focus.ti.com/lit/ug/slau049f/slau049f.pdf

[Muelhofer00] A. Muelhofer, “Controlling the DCO Frequency of the MSP430x1xx,” *Texas Instruments Application Report SLAA074*, April 2000, http://focus.ti.com/lit/an/slaa074/slaa074.pdf

[NCHC] National Coalition on Health Care, http://www.nchc.org/facts/cost.shtml

[Oliver05] N. Oliver, and F. Flores-Mangas, “HealthGear: A Real-time Wearable System for Monitoring and Analyzing Physiological Signals,” *MSR Technical Report*, May 2005.

[Otto05] C. Otto, J.P. Gober, R.W. McMurtrey, A. Milenkovic, and E. Jovanov, "An Implementation of Hierarchical Signal Processing on Wireless Sensor in TinyOS Environment," *43rd Annual ACM Southeast Conference ACMSE 2005, Vol. 2*, Kennesaw, Georgia, March 18-20, 2005, pp. 49-53.

[Otto06] C. Otto, A. Milenkovic, C. Sanders, and E. Jovanov, "System Architecture of a Wireless Body Area Sensor Network for Ubiquitous Health Monitoring," *Journal of Mobile Multimedia, Vol. 1*, *No. 4,* 2006, pp. 307-326.

[Pan85] J. Pan, and W.J. Tompkins, “A Real Time QRS Detection Algorithm,” *IEEE Transactions on Biomedical Engineering, Vol. 32*, March 1985. pp. 230-236.

[Pentland04] S. Pentland, “Healthwear: Medical Technology Becomes Wearable,” *IEEE Computer*, 37(5), 2004, pp. 34-41.

[Polar] Polar Electro, Inc. http://www.polarusa.com

[Priddy06] B. Priddy, and E. Jovanov, "Wireless LAN Technologies for Healthcare Applications," in R.H. Istepanian, S. Laxminarayan, C.S. Pattichis, Eds, M-Health: Emerging Mobile Health Systems, Springer, 2006.

[Raskovic04] D. Raskovic, T. Martin, and E. Jovanov, "Medical Monitoring Applications for Wearable Computing," The Computer Journal, July 2004, Vol. 47, Issue 4, pp. 495-504.

[Shnayder05] V. Shnayder, B. Chen, K. Lorincz, T.R.F. Fulford-Jones, and M. Welsch, "Sensor Networks for Medical Care," *Harvard University Technical Report TR-08-05*, 2005.

[Steele03] B.G. Steele, B. Belza, K. Cain, C. Warms, J. Coppersmith, and J. Howard, "Bodies In Motion: Monitoring Daily Activity And Exercise With Motion Sensors In People With Chronic Pulmonary Disease," Journal of Rehabilitation Research & Development, Sep/Oct 2003, Supplement 2, 40(5), pp. 45–58.

[Suunto] Suunto, http://www.suunto.com

[TinyOS] TinyOS, http://www.tinyos.net

[UCB] U.S. Census Bureau, U.S. Interim Projections by Age, Sex, Race, and Hispanic Origin, http://www.census.gov/ipc/www/usinterimproj

[Warren05] S. Warren, J. Lebak, J. Yao, J. Creekmore, A. Milenkovic, and E. Jovanov, "Interoperability and Security in Wireless Body Area Network Infrastructures," *Proceedings of the 27th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Shanghai, China, September 2005.

[Wheelock99] B. Wheelock, “Autonomous Real-Time Detection of Silent Ischemia,” M.S. thesis, University of Alabama in Huntsville, 1999.

[WHO2000] World Health Organization, “The World Health Report 2000 – Health Systems: Improving Performance,” 2000.

[ZigBee] *Zigbee Alliance: Zigbee Specification v1.0*, 2004, http://www.zigbee.org