A Wireless Communication System for a Tactile Vest

by

Brett J. Lockyer

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 19, 2004 [illegible]

Author_____
Department of Electrical Engineering and Computer Science
May 19, 2004

Certified by_____
Dr. Lynette A. Jones
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

A Wireless Communication System for a Tactile Vest
by
Brett J. Lockyer

Submitted to the
Department of Electrical Engineering and Computer Science

May 19, 2004

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This research focuses on the development of wireless communication circuitry for a tactile display called the Tactile Vest. The display consists of a 4×4 array of vibrating motors worn around the user's torso in direct contact with the skin. It allows an operator with a notebook computer to issue navigational commands wirelessly to units in the vicinity and receive transmissions from them regarding their status. Each command is associated with a pattern of vibratory stimulation to which the user responds. The Tactile Vest system must be durable, lightweight, and consume very little power since it will be worn by mobile users for extended periods. The complete circuit design process is described, from initial prototype development to final layout and testing. A graphical user interface for the notebook computer, written in Visual Basic .NET, is also presented.

Thesis Supervisor: Lynette A. Jones

Title: Principal Research Scientist

Acknowledgements

The author would like to thank the students, researchers, and professors at the MIT BioInstrumentation Laboratory for all their support. Without the patience and guidance of Dr. Lynette Jones, this project would not have been successful. This research was supported by a grant from the Advanced Decision Architectures Collaborative Technology Alliance sponsored by the U.S. Army Research Laboratory under Cooperative Agreement DAAD19-01-2-0009.

Table of Contents

List of Figures

List of Tables

Introduction

The human body's means of perception are typically divided into five discrete modalities: vision, audition, touch, taste, and smell. Of these five senses, the first three are regularly employed to guide daily activities. They complement each other well, and in cases where one sense is impaired, the brain is often able to extract information from the remaining senses to compensate for the loss of sensory information. Methods for transmitting information to the human brain via the visual and auditory channels have been the focus of centuries of research, whereas the scientific community has only relatively recently studied information transmission via the tactile channel [1]. Since visual and auditory displays are simple to construct and operate, and can accommodate multiple users simultaneously, the associated senses are generally considered vastly superior to the tactile channel for conveying information. This is not necessarily the case, however. If the information is presented well, the processing capability of the tactile channel can approach that of the other two modalities [1].

A device that presents information to the user via the sense of touch is called a tactile display. It stimulates the nerve receptors in the skin using either electrical impulses from surface or subdermal electrodes (electrocutaneous stimulation), vibration signals from small actuators mounted on the skin (vibrotactile stimulation), or static deformation of the skin. Tactile displays are generally mounted on the fingertip, palm, or forehead where the skin is most sensitive to the type of stimulation presented. Unfortunately, this channel of communication remains highly underutilized in today's modern, information-driven world.

Approximately 2 m^2 of skin covers the average adult, about 90% of it hairy and 10% smooth or glabrous [2]. The skin on the torso constitutes about half of this surface area [3]. Hairy skin contains five main types of mechanoreceptors, each classified according to their rate of adaptation to a step change in applied skin pressure. These classifications were originally established to differentiate the mechanoreceptors in glabrous skin, but the same principle is applied to receptors in hairy skin. The groups are: fast adapting, small receptive field (FA I); fast adapting, large receptive field (FA II); slowly adapting, small receptive field (SA I); and slowly adapting, large receptive field (SA II). In hairy skin, the fast adapting mechanoreceptors include hair follicle receptors (FA I), field units, and Pacinian corpuscles (FA II), while the Merkel's cells (SA I) and Ruffini endings (SA II) are slowly adapting [2].

This speed of adaptation is crucial for determining which mechanoreceptors will respond best to vibrotactile stimulation within a specific frequency range. As the adaptation speed increases, so does the vibration frequency range that can elicit a response from the mechanoreceptor. Knowing which receptors will be stimulated by a particular frequency range is useful when deciding where to place a tactile display on the body for maximal information transfer.

The concept of vibrotactile stimulation is simple: use controllable vibrating electromechanical actuators in contact with the skin to stimulate the user's cutaneous receptors. The actuator (or tactor) technology varies with the application, from electromechanical speakers, to vibrating motors like those used in cell phones or pagers, to pneumatic actuators. Of all the mechanoreceptors that have been identified and characterized, only a few are sensitive to vibratory signals, namely the Meissner

corpuscle, the Pacinian corpuscle, Ruffini endings and hair follicle receptors. The Pacinian corpuscle (FA II), for example, is most sensitive to vibrations in the 200 to 300 Hz range. Meissner's corpuscles are only found in glabrous skin but respond well to mid-range frequencies (20 to 40 Hz). Ruffini endings, generally found around hair follicles in the dermis, are sensitive to low-frequency vibration (around 7 Hz) [2].

Research in a number of areas, from virtual reality to telerobotics to rehabilitation, has demonstrated the need for tactile displays that can augment or simply represent an environment (be it real or virtual). Braille is perhaps the most recognized example of a tactile display, providing a representation of text in a tactile format via deformation of the skin as users slide (scan) their fingertips across the display. The average reading rate for experienced Braille users is 125 words per minute, and rates as high as 200 words per minute are possible [2]. "With over 10,000 parallel channels, the tactile system is capable of processing a great deal of information *if it is properly presented,*" [2, p. 360]. One of the most difficult aspects of tactile display design is deciding upon the optimal method for presenting data to the tactile system.

Certain types of information may be more efficiently conveyed through a tactile display than with a visual or aural display. Examples include data on surface structures, textures in virtual environments, forces acting on objects or actuators in tele-manipulation, and orientation or motion information for pilots of remotely operated vehicles [1].

Tactile displays have also been used to substitute, in part, for the visual and auditory senses. A Tactile Vision Substitution (TVS) system developed for the blind in the 1970s allowed recognition of simple patterns and human faces. The system consisted

of a stimulator matrix mapped to the pixels of a television camera. The intensity of stimulation varied with the amplitude of light falling on the corresponding camera pixel (or group of pixels) [2]. Thus, an image in the camera was projected onto the tactile array. The system was limited with respect to the complexity of the patterns it could successfully convey, however. This can be attributed to the inherent low-pass filter behavior of the skin for spatial detail, which blurs highly detailed patterns [4].

Tactile substitution for the auditory sense has also produced some surprising results. An auditory prosthesis, called the Tacticon, which consists of 16 electrodes, actually improves the speech clarity of deaf children and the comprehension of auditory cues in older subjects [2]. Some of the Tacticon's success may be ascribed to its ingenious design, which is modeled after the human ear. Each electrode's intensity is varied based on the measured intensity of sound in a narrow passband of the audio spectrum, thus resembling the manner by which the ear decodes incoming sound into its various frequency components and amplitudes.

Despite the enhancements to daily life demonstrated by the aforementioned sensory substitution systems for disabled persons, only one system has become a commercial success. The Optacon (short for Optical to Tactile Converter, made by TeleSensory, Mountain View, CA) converts printed text characters directly to tactile stimulation patterns via a 24×6 vibrating (230 Hz) tactor array [2]. Unlike Braille, the Optacon uses a tiny, hand-held camera to capture images, and then it reproduces the same spatial pattern on the array. Experienced blind users can read text at up to 90 words per minute, but 28 words per minute is the average rate [2].

When compared with Braille, which uses a 2×3 array to represent most alphanumeric characters, the success of this system is surprising. Braille users can generally read three to five times faster than Optacon users. Clearly, Braille's method for representing characters is very intuitive, and differences between characters are easily recognized. The Optacon uses the visual pattern of each character for presentation to the tactile sense, which degrades performance since the eye is capable of quickly identifying fine spatial details that the tactile sense cannot distinguish on the same time scale.

Much of the research on vibrotactile displays has focused on augmenting the user's perception of his environment, especially in environments where there are discrepancies in the inputs to the sensory systems. These displays generally provide position or motion information to aid the users' other senses and to prevent them from becoming disoriented. The vibratory stimulus is usually between 125-250 Hz and is presented at fixed frequencies and amplitudes. It is sometimes modulated with a digital signal depending on the specific application.

The U.S. Navy has constructed a system called TSAS (Tactical Situation Awareness System) to help pilots avoid spatial disorientation, which is a common problem when performing complex maneuvers in jet fighters such as the F/A-18 [5]. It transfers azimuth and elevation data continually from the aircraft's flight instruments to a vest with embedded vibrotactile stimulators worn by the pilot. One prototype's stimulus, for example, is 10 pulses of a 150 Hz rectangular pulse train waveform with a 10% duty cycle, followed by a 450 ms break [6]. Small pager motors or electromechanical speakers provide the vibratory input [7].

The TSAS has been successfully tested in a T-34 aircraft and an H-60 helicopter. The test pilots were seated in a shrouded cockpit to eliminate all visual cues for movement or orientation. They were instructed to perform a series of maneuvers including level flight, climbing and descending turns, and simple acrobatics. Except for occasional problems with missed tactor cues due to intermittent contact with the skin, the pilots performed very well after less than 20 minutes of training [6]. The TSAS has been through several prototyping stages, each improving aspects of the previous design, but a final version for widespread use has not yet been developed. This system can increase the pilot's awareness in situations where misleading incoming sensory data would otherwise lead to spatial disorientation. By reducing the processing load required to fly the aircraft, the system allows the pilot to focus more on mission objectives, weapons systems, and communications [7].

One of the limitations of the TSAS system is its inefficient and cumbersome design. One of the latest prototypes uses pneumatic tactors that must be connected to a compressed air source, making it difficult to fit into the tiny cockpits of some aircraft. It also requires a separate computer (486 type processor) running custom software for control. A more compact and mobile device called the Tactor Interface Microcontroller System (TIMS) has been proposed. Its power-efficient design readily interfaces to the flight control system in most Naval fighter jets. The bulky computer is replaced with a Motorola MPC860 microcontroller and the pneumatic tactors are replaced with smaller electromechanical vibrating tactors. Once fully implemented, it will control 40 tactors or more with a palm-sized interface and use the MIL-STD-1553 aircraft bus for communication with the flight instruments [5].

Vibrotactile displays worn on the torso have also been tested for use in the space program. Rochlis and Newman have demonstrated the need for such a device to assist astronauts aboard the International Space Station (ISS) in Extravehicular Activity (EVA) [8]. During normal terrestrial activities, the brain receives information (some of which is redundant) from the visual, vestibular (inner ear), and somatosensory systems (skin, joint, and muscle sensors) [7]. This information is combined and processed in the brain to yield a determination of the body's movement and orientation with respect to some inertial reference frame. In space or other unusual gravitational environments, the accuracy of this sensory information is compromised by the lack of a clear gravitational vector, or any stable reference frame. Space Motion Sickness (SMS), a form of spatial disorientation, usually results from conflicting sensory inputs, and the astronauts' ability to execute their missions becomes impaired [8].

Since the brain generally relies on the visual modality during activity in weightless environments, astronauts may become distracted from their primary objectives during the EVA simply because their spatial awareness has decreased. The Tactor Locator System (TLS) was designed to help astronauts maintain a high level of spatial orientation during EVA missions [8]. It uses a vibrotactile display that provides a redundant (and intuitive) sensory cue to aid the visual system in unusual environments. In the prototype phase, the system consists of an array of six vibrating tactors, driven at 250 Hz with a 6 V peak-to-peak waveform. Four tactors are spaced evenly around the midsection of the torso, and the other two are located at the base of the neck and the buttocks [8].

During preliminary testing, the user was given a task to execute on a PC-based space simulator. Results indicated an improvement in the reaction time to an unfamiliar situation, as well as an improvement in the user's ability to maneuver in the environment. Tactile cues reduced the subjects' maneuvering times (time to acquire a target) by 92 s on average and decreased their reaction time to new situations by an average of 4.5 s [8].

A further study of the usefulness of vibrotactile cues in maintaining spatial orientation will be conducted by a Dutch astronaut named Andre Kuipers on the International Space Station in April 2004. This project is not related to the aforementioned research of Rochlis and Newman [8], but its goals are similar. The suit consists of a matrix of vibrating tactors that will be actuated based on signals from a two-axis gyroscope. The system is battery powered and interactive via an arm-mounted control panel. The experiment is broken down into two phases, one in which the astronaut is instructed to execute various orientation tasks while wearing the suit, and the other will instruct the astronaut to carry out his daily activities while wearing the suit. The second phase will evaluate the suit's effectiveness in an operational setting. Performance data will be monitored and recorded on removable flash memory cards [9].

Wall and Weinberg have built another type of tactile display that provides vibratory stimulation to the skin to assist in postural stability in people with balance impairments [10]. This may include people recovering from inner-ear surgery or elderly persons who have impaired balance and are prone to falling. The system incorporates a vibrotactile array that displays the body's tilt with respect to the vertical, thus enabling vestibulopathic subjects to reduce their body sway and prevent falls [10].

The tactile display used in Wall and Weinberg's prototype wearable prosthesis is a 3×16 array of vibrating tactors operating at 250 Hz [10]. They are spaced evenly around the lower torso (horizontally) and arranged into three vertical rows. The subject receives body tilt direction information via actuation of the tactor on the ring corresponding to the angular direction of the measured tilt. The number of tactors actuated in that column then depends on the magnitude of the tilt. The 802.11b (Wi-Fi) protocol is used to communicate with the wearable prosthesis via a notebook computer [10]. The computer, running a custom LabView interface, records the prototype's measurements for analysis and updates the software running on the device's various processors.

Tactile displays can also be used as active devices, actually instructing the user to perform various actions instead of passively representing the environment or the user's orientation within that environment. This application is generally reserved for visual or auditory displays because they are considered more mature technologies and are easy to construct. Unfortunately, there is a limit to the amount of information those two senses can process without causing confusion. A tactile display can take advantage of the sense of touch, thereby reducing the demands placed on the other two modalities and potentially increasing the information throughput to the user.

This thesis details the design and implementation of a wireless communication circuit for a tactile display called the Tactile Vest. The complete system consists of a 4×4 array of vibrating motors worn around the torso, along with the associated embedded control electronics that receive commands wirelessly from a notebook computer and translate them into sequences of vibration. The user will sense this vibrotactile

stimulation, associate it with a command, and take the appropriate action. Since the vest is worn around the torso, it will not interfere with the user's arms or legs, allowing a full range of motion while wearing the system. Its operation is virtually silent, which is desirable in many military environments.

Circuit Design

The Tactile Vest circuit has two main components: a wireless transceiver module for communication with a notebook computer and an embedded processor, or microcontroller, to receive commands from the wireless module and translate them into sequences of motor actuation. Each component will be discussed in detail in this section together with the rationale for selecting the technology.

The wireless communication system represents the most challenging aspect of the design process. The features desired in a wireless system for this application are not necessarily realistic given the current state of wireless technology. Desired specifications for the Tactile Vest's wireless system include a 100-meter range both indoors and outdoors, the ability to operate for at least 48 hours on a single charge of a reasonably sized battery, a fast connection time, a high data communication rate with encryption, the ability to communicate with multiple units at the same time and to handle a constantly changing spatial distribution of units. The wireless module for the vest circuit must be small and inexpensive, and on the notebook computer side, it should have a Universal Serial Bus (USB) interface.

Two different wireless modules satisfy most of the above criteria: 9XStream radio modules from MaxStream, Inc. (Orem, UT) and BR-C11 Bluetooth modules from BlueRadios, Inc (Englewood, CO). Unfortunately, neither system possesses all the specifications listed above, but they are the best options available.

The MaxStream modules use spread-spectrum 900 MHz wireless modem technology. This is the same frequency band used by cordless phones for short-range communication within a house or business. One module connects to the RS-232 (serial)

port of the notebook computer and the other interfaces to the microcontroller on the tactile vest via a Universal Asynchronous Receiver Transmitter (UART). The wireless system can accommodate up to 7 networks, each with 65,536 individually addressable modules. Communication speed is rather slow, however, at only 19.2 kbps (over air). The modules themselves are 84.3 mm long, 40.64 mm wide, and 16.89 mm thick, making them rather large, especially with the half-wave antenna attached, which is about 167 mm long [11].

The BlueRadios modules are spread-spectrum, 2.4 GHz Bluetooth Class 1 devices that are easily integrated with Microsoft Windows (see [12] for more information about the Bluetooth standard and Special Interest Group). They allow for high-speed communication of up to 721 kbps (over air), transmission security via 128-bit encryption and a ten-digit PIN, and have a relatively long range of 100 m (line-of-sight, outdoors) [13]. Unfortunately, this technology can only handle "pico-nets" of seven users or less, which could present a problem when trying to communicate with large groups of people.

The Bluetooth module contains an integrated ceramic chip antenna, but it requires an external RF "backplane" to function properly. This "backplane" must consist of non-magnetizable metal residing in very close proximity (about 0.4 mm) to the metal component shield on the device (measured in the direction perpendicular to the plane of the module). Electrical contact between the metal shield and the "backplane" is not necessary. It must also be mounted at least 6.1 mm away from the chip antenna in the lateral direction (along the plane of the module). Details of the antenna's transmission profile, directionality, and physical design could not be obtained from the manufacturer, and so the reason for this "backplane" is not precisely known. According to the

manufacturer, the radio frequency output is amplified by eddy currents induced in the metal "backplane," therefore leading to a stronger, more favorable antenna transmission profile.

Low power consumption is critical to this application due to the constraints on battery size and weight, which limit the available power to the circuit during operation. Ideally, the battery would weigh no more than 500 grams so it can be easily carried for two days without encumbering the user. To meet this objective, the circuit's power consumption must be minimized.

The MaxStream modules allow for several low-power modes of operation: pin sleep and cyclic sleep. Pin sleep is the lowest power mode, consuming only 0.3 mW, and it is initiated by toggling one of the input pins (SLEEP, pin 2) on the module. The MaxStream unit then remains in pin sleep mode until the pin is brought low again. The transition back to idle mode takes about 40 ms [11]. The other low-power mode implemented on this module is cyclic sleep. In this mode the unit transitions from idle to sleep after a user-defined period of inactivity. The module then transitions back to idle periodically (based on a user-defined period from 0.5 s to 16 s) to check for incoming data packets over the RF link or the UART [11]. If it detects any valid packets during the 100 ms window, it receives or transmits them and then enters the sleep state again [11]. With no sleep modes initialized, the MaxStream device consumes 750 mW while transmitting, 250 mW while receiving, and 15mW while idle [11]. Cyclic sleep mode can significantly reduce the average power consumption in this case. When the module enters into 1 s cyclic sleep mode, it draws only 19.3 mW on average, and when it enters 16 s cyclic sleep mode, average power consumption drops to 1.38 mW.

The Bluetooth module can also enter a type of cyclic sleep mode. The scan interval and window can be set independently, thereby specifying the time between searches (interval) for incoming data or connection requests and the duration of each search session (window). The default setting gives a 1.024 s interval and a 0.512 s window. Under these conditions, the module operates at a 50% duty cycle, nearly cutting its average power consumption in half. Between searches, the module draws about 7 mW [14]. Without sleep mode implemented, the Bluetooth module draws 396 mW while transmitting, and 132 mW while receiving data [13].

Reduced power consumption leads inevitably to an increase in latency. Measured as the time between command transmission and initiation of a vibration pattern on the array, the latency must be minimized in this application. Otherwise, commands may not be relevant by the time the user has received them. If sensors on the vest detect a critical situation, that information must be relayed back to the host and a command must be sent informing the user to take immediate action. These critical data must reach their destination as quickly as possible to increase the operator's safety.

Since the host cannot issue commands to the vest while the wireless module is in low-power mode, the command must be sent repeatedly until the vest module receives and acknowledges it. This process could take up to 16 s for the MaxStream module (in the lowest power cyclic sleep mode) and up to 0.5 s for the Bluetooth module (in factory default cyclic sleep mode), depending on the module's duty cycle, or the length of time it sleeps versus the time it checks for incoming data. The overall system latency would be the sum of this latency, the time required for the vibration pattern to be executed, and the

time a person takes to react to the vibrating stimulus (about 200 ms, [2]). This delay may be unacceptable in some applications.

Tests performed in the laboratory have shown that the MaxStream modules have a better range both indoors and outdoors (up to 200 m indoors, over 200 m outdoors) than the BlueRadios modules. The latter have a range of only 170 m outdoors and around 23 m indoors in the presence of obstacles such as doors or walls. However, the BlueRadios modules are much smaller, both on the embedded side and on the notebook computer end (which uses a USB interface), and can communicate at faster rates than the MaxStream modules. MaxStream's proprietary technology can limit its usefulness in a particular application, whereas Bluetooth technology has the potential built-in to expand effortlessly into other applications such as voice transmission. Figure 1 shows the two modules.

One reason for the MaxStream module's superior indoor range is its operating frequency. The wavelength of a 900 MHz wave is 0.33 m, whereas the wavelength of a 2.4 GHz signal is 0.125 m. The longer the wavelength, the greater the diffraction around obstacles and the stronger the received signal remains. When an electromagnetic wave encounters an opaque obstacle or a gap between obstacles, the wave "bends" and diffraction occurs. The resulting diffraction pattern is caused by interference between different parts of the wave as it travels through space. If a wave is "bent" such that its path length, measured from source to observer, is changed, then its phase will also be modified. Two waves with different phases that overlap in space can interfere either constructively or destructively, thereby producing wave amplitudes either greater than or less than the original amplitudes, respectively [15].

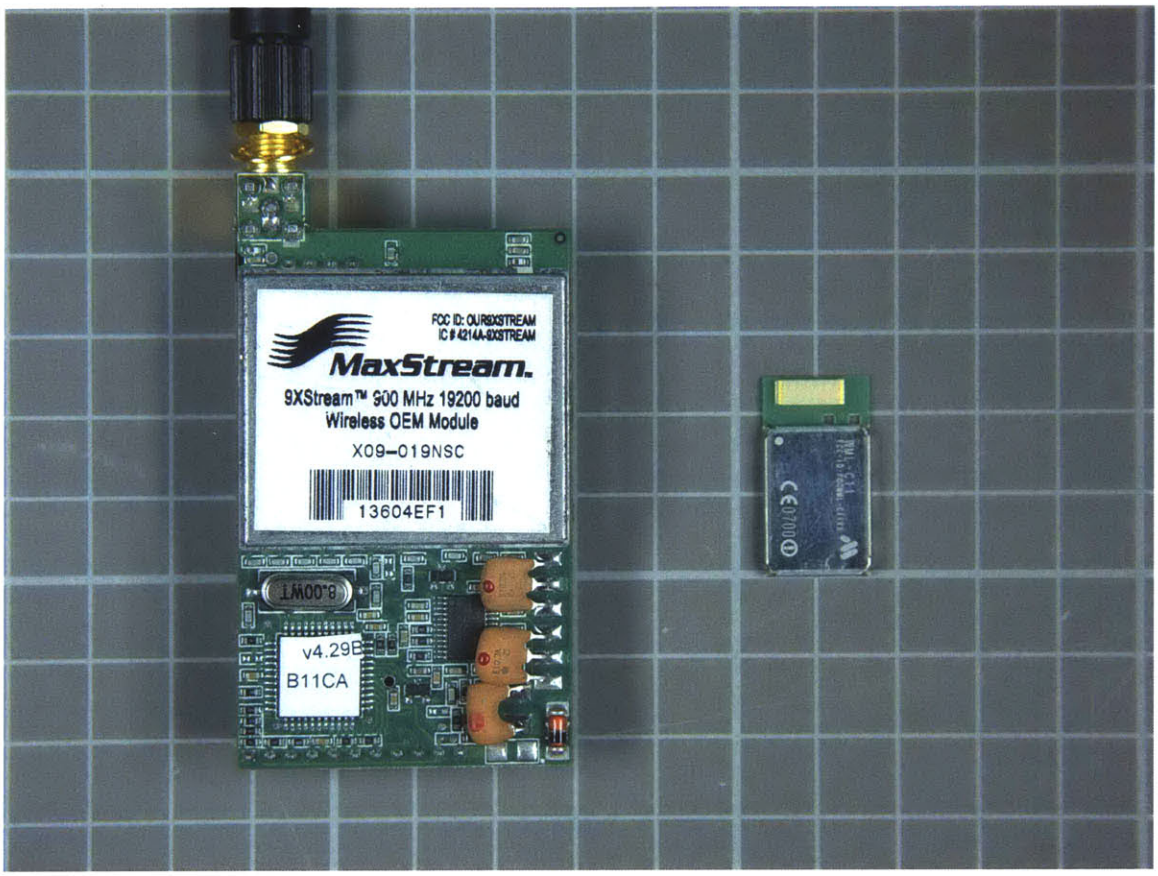

Figure 1: MaxStream (left) and Bluetooth (right) modules. The grid is 1 cm^2.

The extent and spatial location of this interference depends on the size of the obstacle or gap relative to the wavelength of the incoming radiation. In general, if the wavelength is smaller than the dimensions of the obstacle, distinct shadows will be cast behind the obstacle. These shadows are the result of destructive interference between sections of the original wave after encountering the obstacle and undergoing phase shifts. If the wavelength is larger than the dimensions of the obstacle, the wave will tend to spread out spatially after encountering the obstacle [15]. No distinct shadows are cast, and the likelihood of the observer (or receiver, in this case) being positioned within a destructive interference zone decreases. The effect is similar to that of a sound wave (with a frequency of 500 Hz and a wavelength of about 68 cm) "bending" around

obstacles such as trees and telephone poles, while the same obstacles cast distinct shadows when illuminated by light (with a wavelength of 500 nm or so) [15].

Another reason for the MaxStream module's enhanced range is the high receive sensitivity it exhibits. The transmission power output of both the MaxStream and the Bluetooth modules is 100 mW (20 dBm), but the Bluetooth module's typical receive sensitivity is 10^{-8} mW (–80 dBm) compared with the MaxStream module's 2×10^{-11} mW (–107 dBm) sensitivity [5, 6]. Range differences can be ascertained using the Friis transmission formula for matched antennas:

$$P_R = \frac{P_T G_T G_R \lambda^2}{(4\pi r)^2} \tag{1}$$

where P_R is received power (mW), P_T is transmitted power (mW), G_T is linear gain on transmitting antenna, G_R is linear gain of receiving antenna, λ is wavelength (m), and r is distance between transmitter and receiver (m) [16]. By equalizing transmitting and receiving antenna gains between Bluetooth and MaxStream systems, and assuming line-of-sight conditions, the only parameters that affect the received power, and hence range, are the wavelength and the distance between transmitter and receiver. Given their operating wavelengths and sensitivities, the MaxStream module can communicate at a distance of about 60 times the maximum range of the Bluetooth module. This simplified, first-order approximation demonstrates the fundamental operating differences between the two systems and suggests why one easily outperforms the other.

Although the MaxStream modules have a greater range when compared with the Bluetooth modules, neither technology satisfies all the original design requirements. Therefore, the system has been designed around the Bluetooth wireless modules simply because they are easy to use and represent a standardized solution instead of a proprietary

one. Eventually, the Tactile Vest may be used with proprietary wireless modules, and so the system is being designed to be as independent of the wireless protocol as is possible in such an embedded application.

The embedded processor in the vest receives commands from the host and translates these into patterns of motor actuation. Each pattern will be distinct and associated with only one command. The system is configured to accommodate any number of commands, most of which are represented by sequences of vibrotactile stimulation that are essentially self-explanatory. For example, a directional command instructing the operator to turn left can be associated with an actuation pattern that begins by vibrating the rightmost column of motors on the vest and then shifts through the columns to the leftmost column. This is a very unambiguous and easily interpreted pattern.

The microcontroller must meet a number of criteria including having a UART interface, a reasonably fast clock speed, an analog-to-digital (A/D) converter, low-power sleep modes, a watchdog timer, large flash memory space, 16-bit timers for long delays and timing events, and a number of I/O pins for motor control and sensor interfacing. The Atmel AT90LS8535, a member of the AVR family of microcontrollers, incorporates all these features and is readily available from electronics parts suppliers such as DigiKey. It is manufactured in space-saving 44-lead TQFP and PLCC surface mount packages with very low profiles, making it ideal for this application [17]. The microcontroller and its development environment are very low-cost, compared with similar microcontrollers from manufacturers such as Motorola.

The AT90LS8535 has one feature that is especially useful in this application. It supports in-circuit serial programming (ISP), which allows the microcontroller to be reprogrammed or the contents of its memory verified while embedded in its target application. The microcontroller's on-board flash memory can be read, written, or erased with purely electronic signals, making ISP compatibility possible. All this takes place via a six-pin connector on the circuit board. Potential benefits of this feature include an in-the-field programming capability that allows users to modify the microcontroller's list of recognized commands and vibratory patterns quickly and easily.

A low quiescent current, open-drain motor driver integrated circuit is also required for the Tactile Vest because the microcontroller's outputs cannot drive high-current inductive loads such as motors or solenoids. The motor drivers, each of which can control eight motors, will serve as the interface between the microcontroller and the 16-motor array. The open-drain specification is important because the design must accommodate motors running at 3.3 V instead of the 5.0 V used by the microcontroller and motor controller logic. The low quiescent current specification is vital as the motor controller will be idle most of the time.

The best candidate for the motor drivers is Allegro's A6B259KLW, an 8-bit addressable DMOS power driver with open-drain outputs, available in a 20-lead surface mount package that conserves space. The device has eight outputs per package so two packages are needed for this application. Since they function as decoders, only nine control lines are necessary to control sixteen outputs. Its quiescent current consumption (with the outputs off) is only 20 μA, but it is capable of driving up to 150 mA per output

pin [18]. This is more than enough current to power the small vibrating motors on the vest, which draw about 50 mA at most.

The circuit must be designed to make the most efficient use of the energy available to it. Two National Semiconductor LM2595 switching regulators are used to convert the battery pack's voltage, nominally 7 to 10 V, into 3.3 V and 5.0 V for the electronics [19]. The Bluetooth module and motor array are driven from the 3.3 V supply, while the microcontroller and the motor controller logic are driven from the 5.0 V supply. With efficiencies around 80%, the switching regulators represent the optimal solution to the power conversion problem. They are also simple to implement, with only four external components required per regulator.

Prototype Construction

The Tactile Vest prototype circuit was constructed on a "breadboard" for evaluation of the circuit design. This was a temporary version of the circuit that could be modified easily to accommodate the addition of new functionality or the adjustment of component values. The dual in-line package (DIP) versions of the microcontroller and motor controllers were used to facilitate insertion into the breadboard. While these packages take up more space than their surface-mount counterparts, they respond similarly and have nearly identical pin layouts, which allows for the complete circuit to be constructed and tested on the breadboard. In addition, light-emitting diodes (LEDs) were connected to the microcontroller's output pins to be used as status indicators. The LEDs were not included in the final design. If necessary, a voltage level measurement of the first four pins of Port B on the microcontroller will provide status information in the final version. Figure 2 is a picture of the prototype setup. The Bluetooth module may be seen protruding from the prototype board on the right side of the picture. The microcontroller is the largest integrated circuit on the board, seen near the top of the photograph. The two power converters are near the bottom, with the motor controllers in the middle. The cable extending to the 9 motors used for testing (in a 3×3 matrix) can be seen on the left-hand side of the picture.

The microcontroller is programmed while installed in the target application via its in-circuit programming (ISP) interface. The Atmel STK500 Development System provides the necessary software and hardware to develop the code and program the microcontroller. The system is designed to interface with and program any microcontroller in Atmel's AVR family. Sockets of several different sizes are mounted

on the development board, which allows the user to program the microcontrollers directly. There is also a 6-pin header to allow connection to an external microcontroller via ISP mode. The development board interfaces with a notebook computer over a standard serial (RS-232) cable.

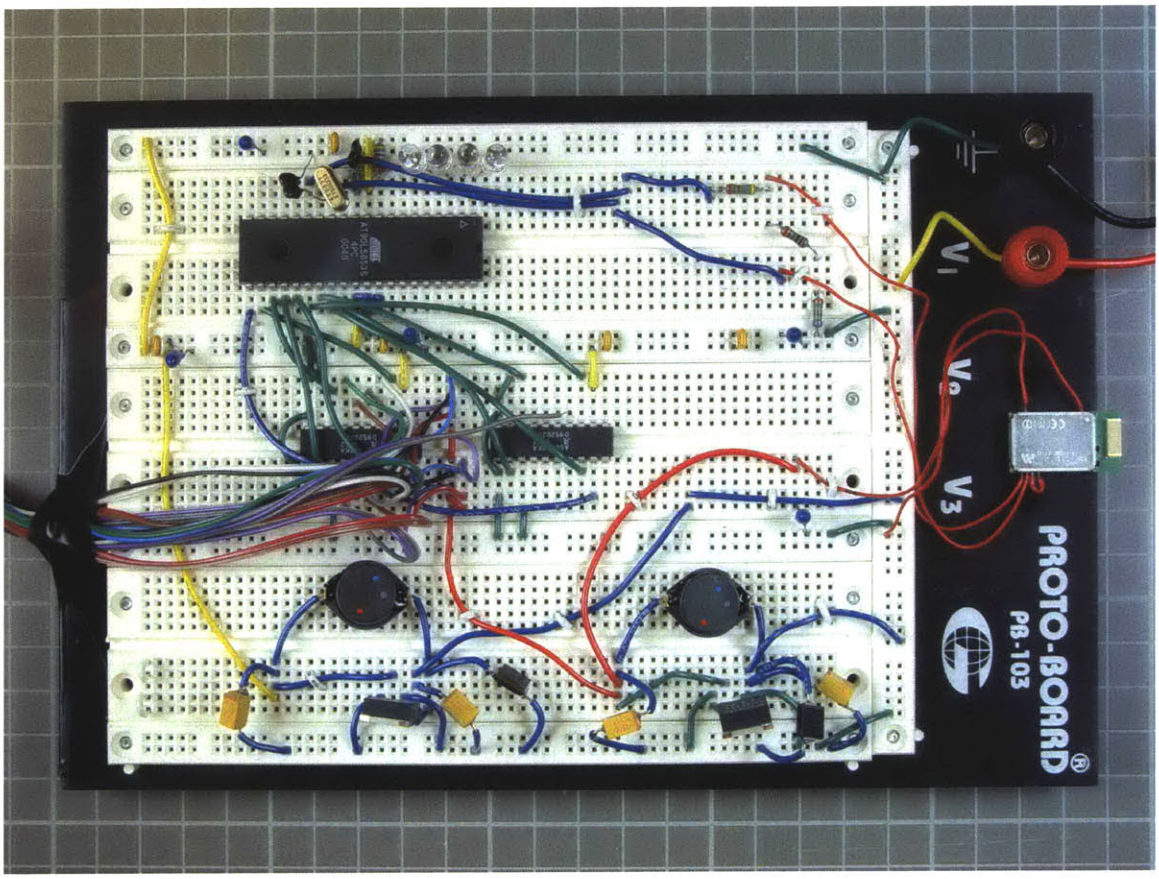

Figure 2: The prototype circuit.

The assembly software is written using Atmel's AVR Studio 4.0. This application provides an environment that facilitates code development and communication with the STK500. The software also contains a debugger and simulator for a variety of AVR microcontrollers, as well as an integrated assembler to convert code in assembly language into machine language read by the microcontroller. Extra features

such as automatic color-coding of typed code and tab stops allow quick debugging and make the code generated neat and easily readable.

The prototype Tactile Vest circuit performed well, exhibiting all the features desired in the final version, including efficient use of power and low latency (less than one second). The Bluetooth module in the circuit receives commands from the USB Bluetooth module connected to a notebook computer. Because the circuit expects to receive commands in the form of plain ASCII characters, Microsoft HyperTerminal is used to communicate with the vest prototype. Any terminal program will work for this application, but HyperTerminal comes with the Windows operating system, so it is generally the most convenient program to use. The use of this program, instead of a custom graphical user interface (GUI), eliminates the possibility of errors in the GUI code affecting the prototype testing. Since HyperTerminal is known to work well, any problems exhibited by the system must originate in the prototype circuit.

The prototype's embedded processor recognizes eight commands, seven of which are linked to vibration patterns on the motor array. A 3×3 motor array was used with the prototype because it is sufficient to demonstrate proof-of-concept, and the system can be easily scaled up to accommodate a 4×4 array in the final version. Each command translates into an unambiguous "sweeping" pattern on the vest. For example, the "up" command begins by actuating the lowest row of tactors in the array simultaneously. It then shifts to the middle row, and then to the top row of tactors. The wearers of the vest feel an upward-moving stimulation with respect to their body, which they can immediately associate with the command "up."

Layout and Fabrication

Following the successful testing of the prototype circuit and 3×3 motor array, the final design was entered into OrCAD's Schematic Capture CIS and Layout Plus programs. The Schematic Capture program is the first step towards creating a functional printed circuit board (PCB) from the prototype circuit's schematic. Data entry began with the selection of parts from a library of predefined packages or the creation of custom parts to match those in the prototype circuit. Since the Schematic Capture program is not a simulator, it does not require information about the part's dynamics during operation, just how many pins each part has and a reference name. Standard parts are represented on the screen by their schematic symbols, and custom parts are usually rectangles with short, numbered line segments protruding from each side to represent the pins.

Each part must also be linked to a footprint file from either a standard or a custom library. The footprint file contains information about the physical part, including package and pin dimensions and the necessary clearance around the part on the printed circuit board. The footprint will eventually be etched out of a copper-clad board and the part will be soldered onto it, so everything must be specified exactly if a good fit is desired. Figure 3 shows the completed Tactile Vest schematic, taken from the OrCAD Schematic Capture program.

Once each part is chosen and its symbol placed in the workspace on the screen, connectivity information may be entered into the program via "wires" added to the diagram. These wires are just line segments that tell the program which pins are connected together in the circuit. Once this is completed, all the part, footprint, and connectivity information is packed into a single file called a netlist.

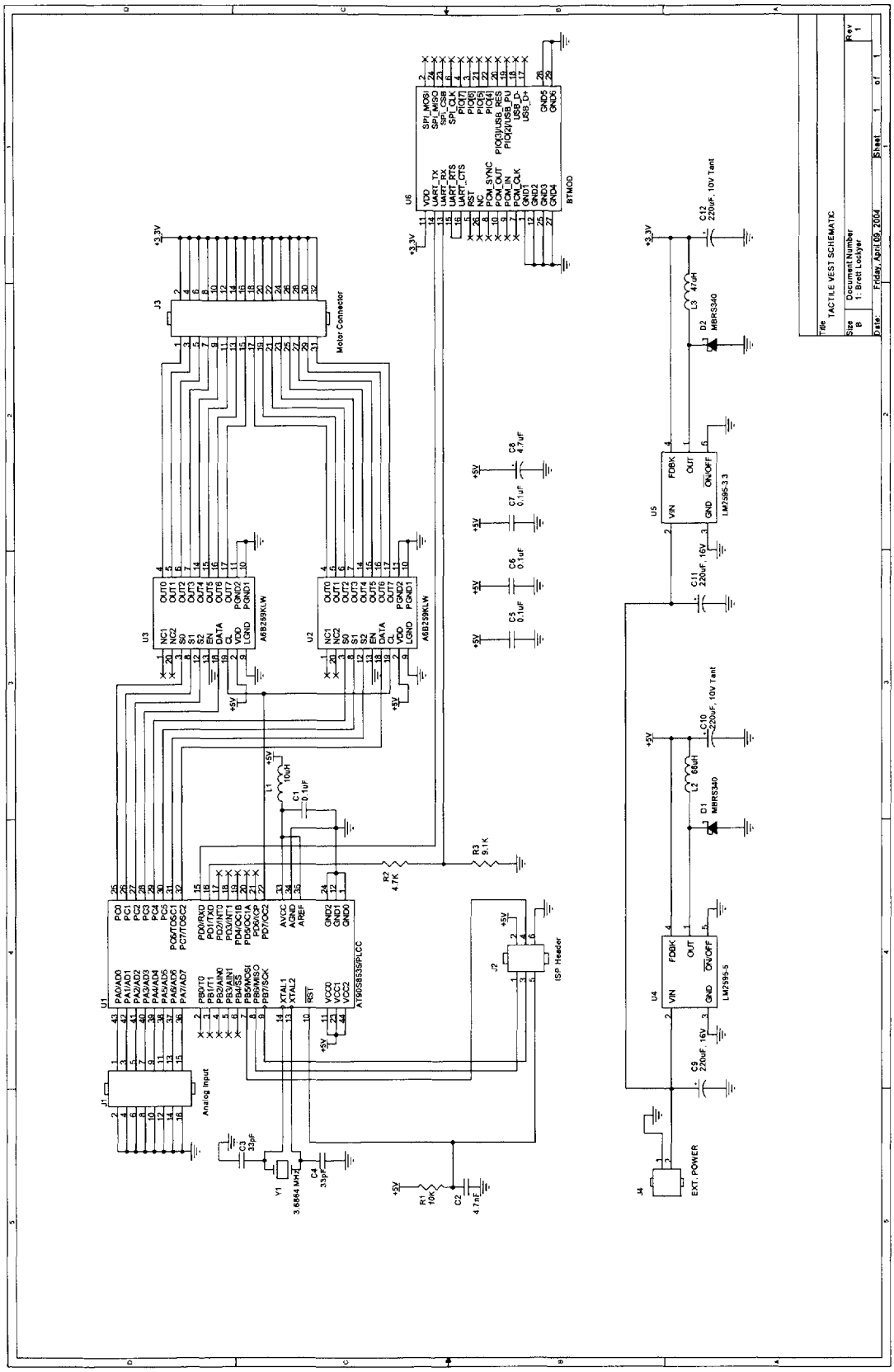

Figure 3: Tactile Vest schematic (from OrCAD Capture CIS).

The OrCAD Layout program imports this netlist and generates a "ratsnest," which it displays on the screen. This is a massive conglomeration of all the parts' footprints in the circuit, with yellow lines connecting the appropriate pins together. Once the PCB's dimensions have been determined, the "ratsnest" must be organized and manipulated such that everything fits satisfactorily inside the desired board outline. This is generally the most daunting aspect of printed circuit board development.

When all the footprints have been placed within the board outline, the yellow lines representing the circuit's connectivity information must be converted to copper traces. This can be done manually or with an automatic software router that attempts to find a routing solution meeting the design constraints specified. It is often possible to "autoroute" an entire board, but usually not desirable. It can lead to inefficient use of space and unnecessarily long traces, leaving the circuit vulnerable to external noise sources. This process is especially difficult when there are stringent layout constraints. These can include requirements that certain parts are placed directly above a ground plane to minimize noise, or that other parts are routed with wide traces that will carry high currents during operation. Too many constraints will usually cause the autorouter to fail to converge on a satisfactory routing solution.

The manual routing option was chosen for this circuit. This gave the designer full control over placement and trace width, it maximized the use of space on the board and ensured that the final design was not extremely vulnerable to noise. Manually routing a board takes much longer than autorouting does, but the result is worth the extra effort since debugging time is greatly reduced. Most digital signal lines were routed using 10 mil (0.010 in wide) traces. The smallest trace width that can be used on boards

manufactured with standard PCB fabrication machines is usually 7 mils. Higher current lines such as those for the two voltage regulators were routed with up to 40 mil traces. At least 10 mils of space separate every object on the board, placing the board design well within the requirements of standard PCB manufacturing equipment.

Since the vast majority of parts in the circuit could be purchased in surface-mount packages, the board was designed with only two layers: top and bottom. All the components were mounted on the top layer, along with some traces, and the bottom layer was used only for traces and the ground plane. The only thru-hole parts in the design are the connectors, which are placed side-by-side on one edge of the board.

Traces on the top and bottom layers are connected with "vias" when necessary. Vias are simply holes drilled through the board that are plated with metal around their circumferences. Traces that are to be connected on each layer terminate directly on top of one another, and vias are drilled at the exact spots needed to connect them electrically. This is essential when two traces on the same layer must overlap, which often occurs. One of the traces then remains on the top layer, and the other one connects to a trace on the bottom layer that passes under the first trace.

The final layout design satisfied all the constraints on trace width and component placement, including a large ground plane on the bottom layer to minimize noise. The trace pattern for the top layer can be seen in Figure 4 and the pattern for the bottom layer is shown in Figure 5.

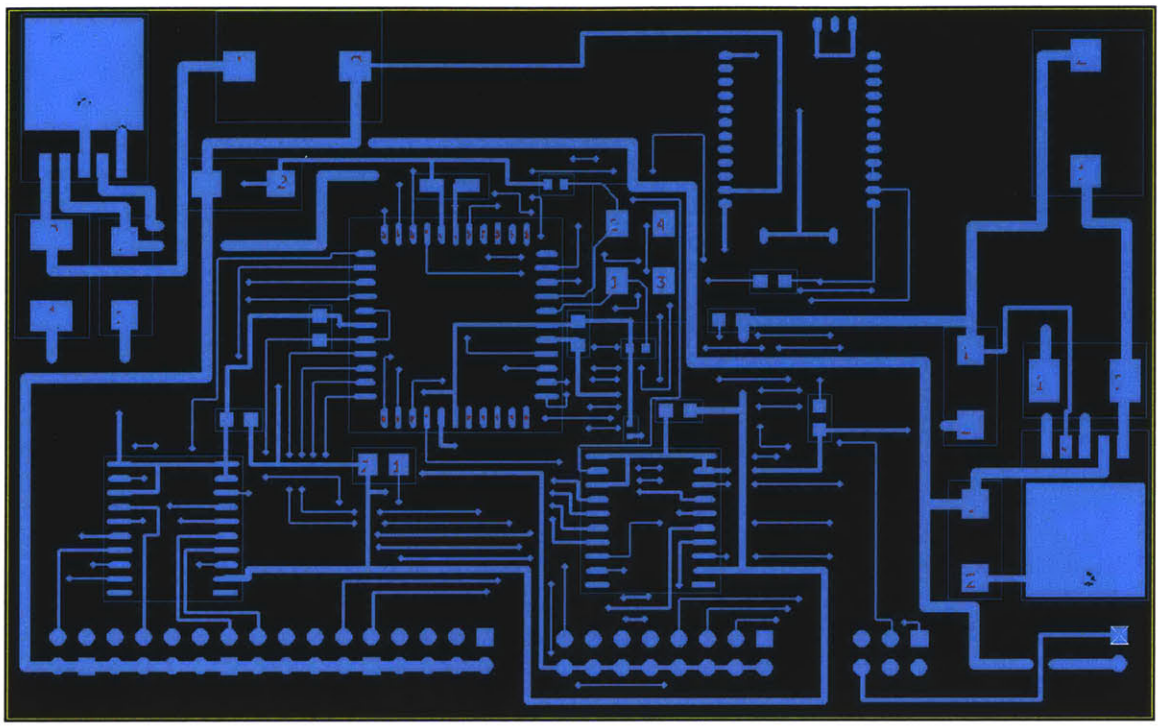

Figure 4: Top layer of PCB layout (from OrCAD Layout Plus).

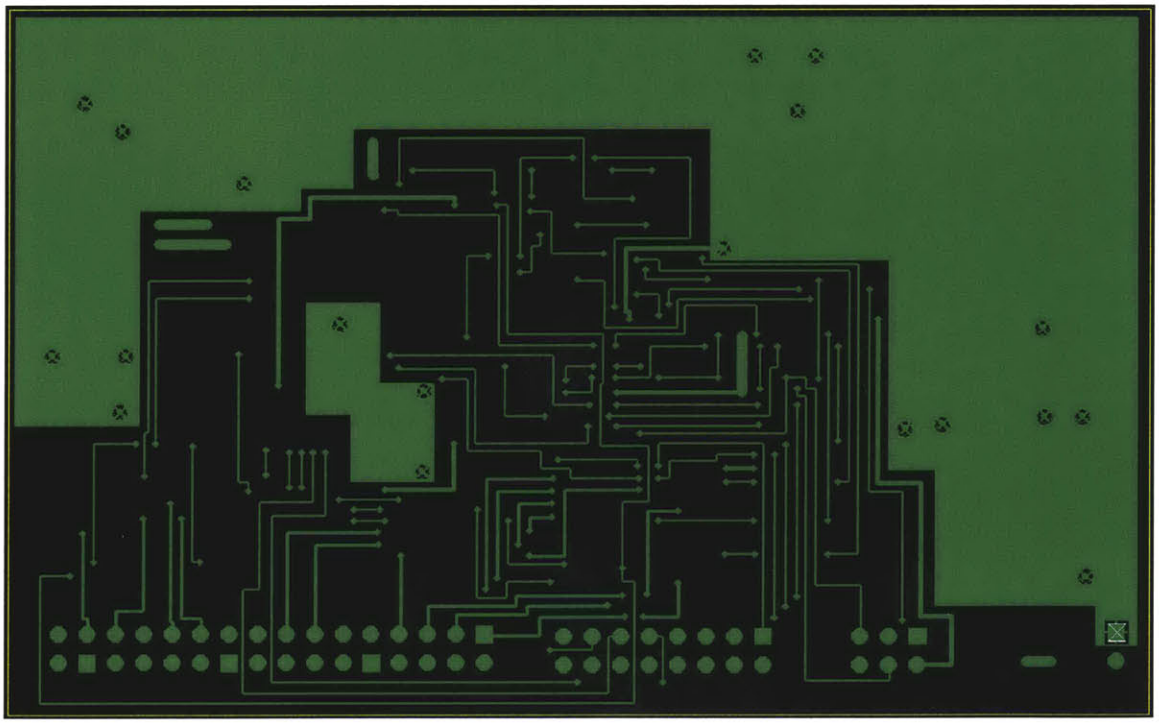

Figure 5: Bottom layer of PCB layout (from OrCAD Layout Plus).

The finalized layout design for each layer was then sent to a PCB fabrication company (E-teknet Inc., Gilbert, AZ). They accepted the output files generated by OrCAD and used them to etch the correct patterns into a copper board. The board was then coated with green insulation (except where the parts must be soldered) and printed with text using a silkscreen process. The printed circuit board can be seen in Figures 6 (top layer) and 7 (bottom layer).

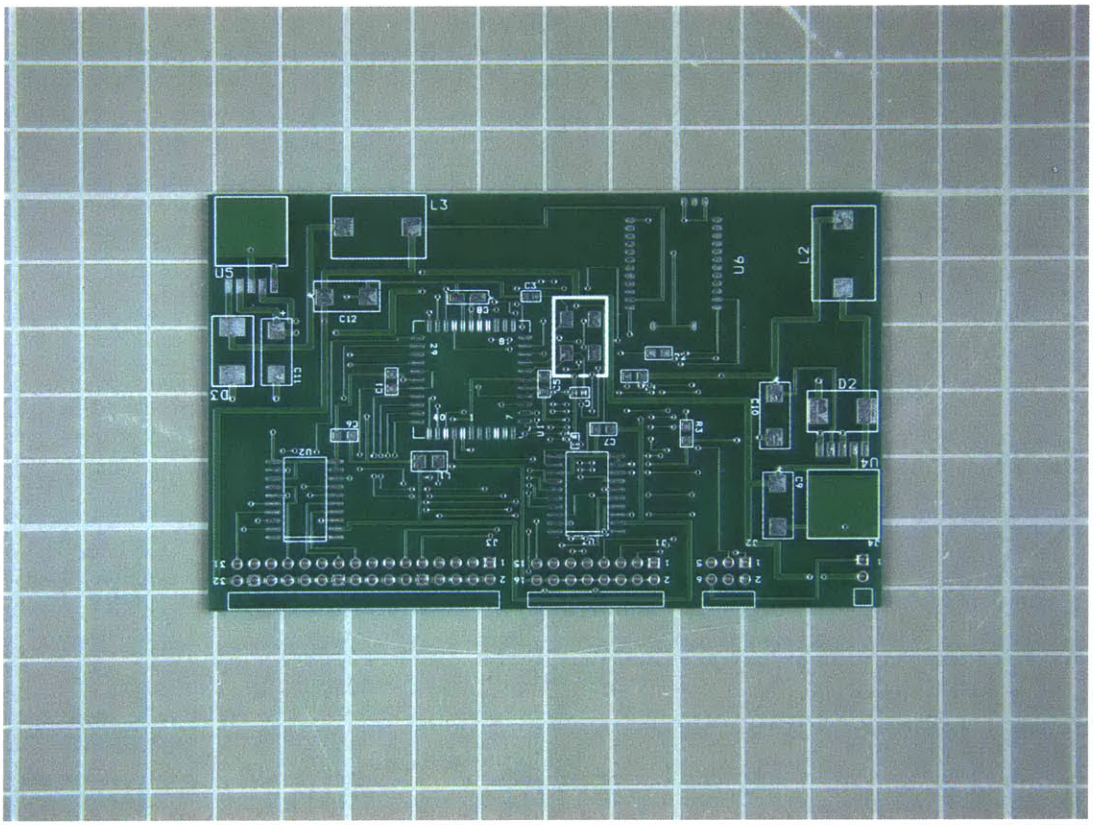

Figure 6: Top layer of unpopulated PCB.

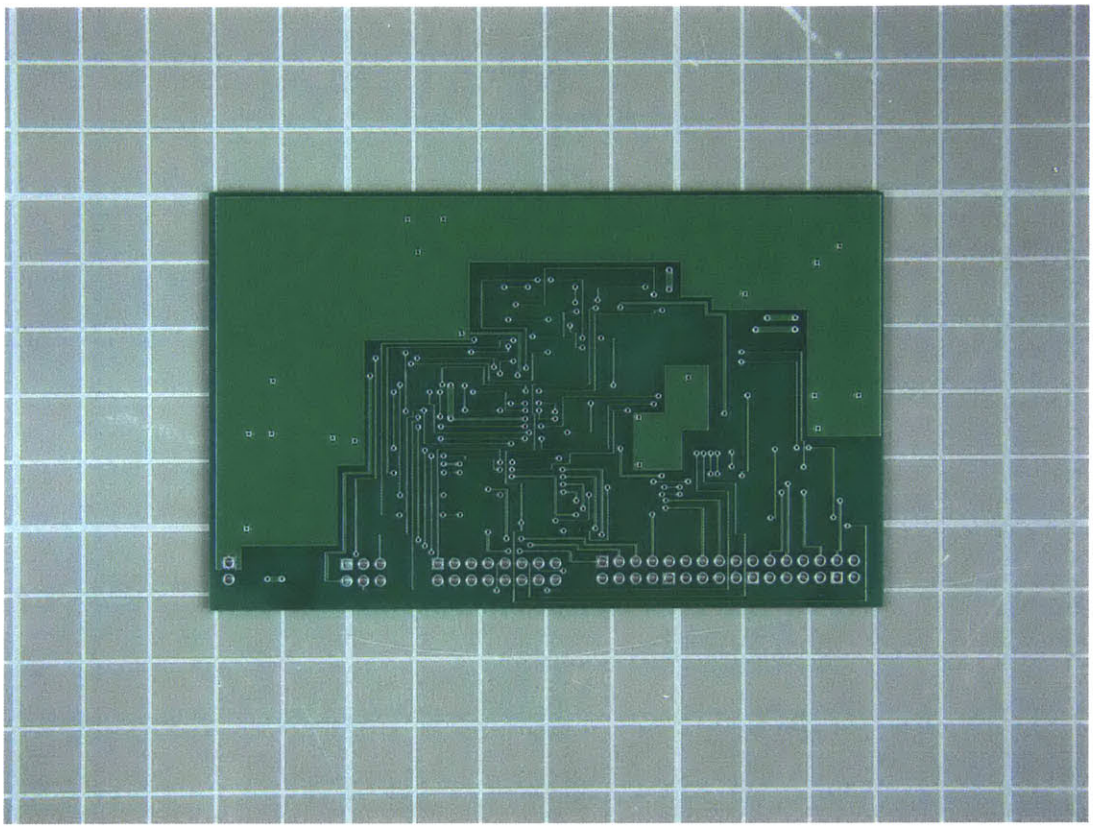

Figure 7: Bottom layer of unpopulated PCB.

Population and Testing

Once the layout process was completed, the board was then "populated" with components and tested. The population process was demanding since many of the miniscule surface-mount parts were difficult to solder by hand. Several tools were useful in this endeavor, including a normal soldering iron with a tiny, pointed tip and a hot air gun whose tip did not make direct contact with the component's pin to heat it. The air gun made it easier to hold the part in place while applying heat and solder. A Zeiss Stemi SV8 magnifying station is essential for successful board population since it provides all the lighting, mounting, and magnification options one requires when working with miniscule components on a densely populated board. Inspection of the final version for solder bridges or gaps was also crucial, and the magnifying station vastly simplified this process.

The final PCB is shown in Figure 8. All the connectors are mounted on the top of the board for convenience. From left to right, the connectors are for the battery (7 to 10 V), the ISP cable to the STK500, the A/D inputs, and the motor array. The Bluetooth device can be seen protruding from the bottom (without the small RF "backplane" mounted). The power regulators are on each side, and the microcontroller is the square chip in the middle. The motor controllers can be seen to the upper left and right of the microcontroller.

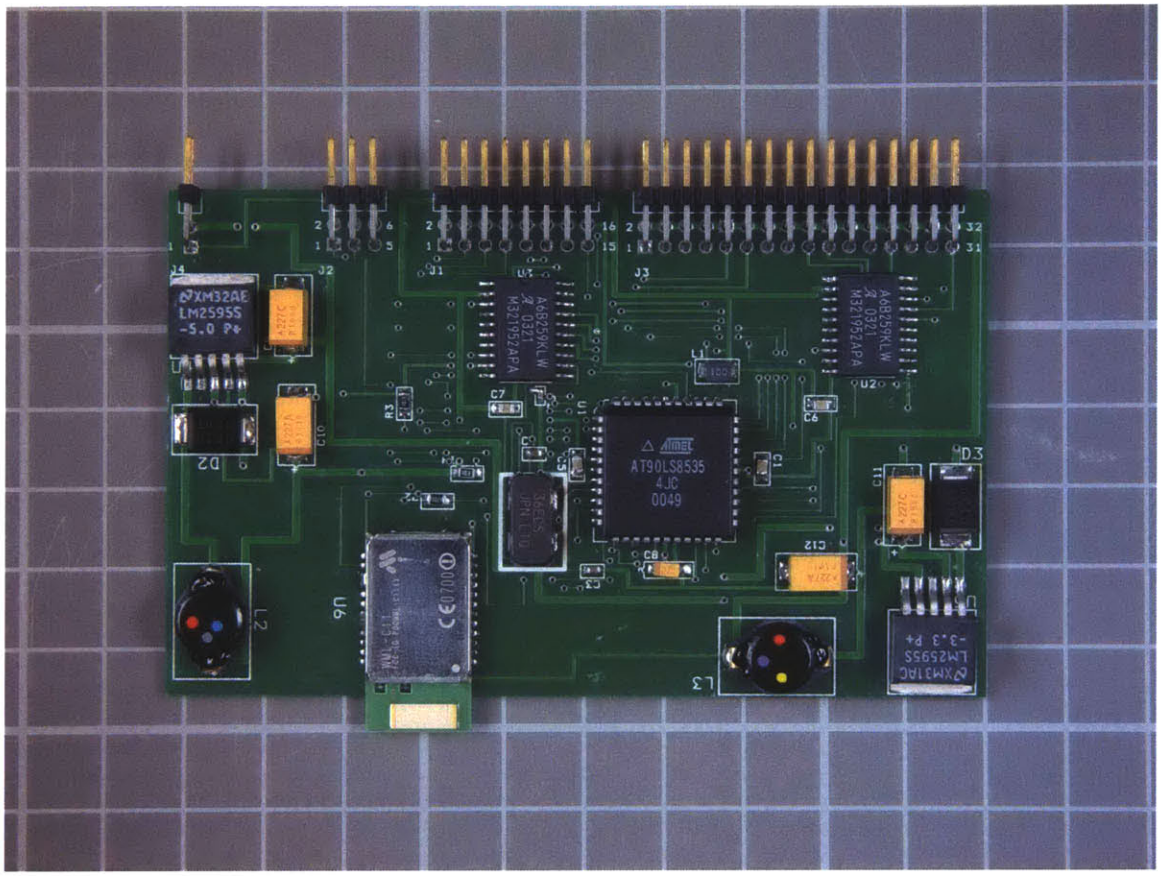

Figure 8: The final wireless communication circuit.

The final power consumption measurements for the board are listed in Table 1. Since the consumption varied depending on the Bluetooth module's connection status, measurements for both connected and disconnected states are given. The results would suggest a minimum in the operating power curve near 8.0 V, probably due to variations in regulator efficiency and duty cycle across changes in input voltage, although the exact cause is unclear.

Operating Voltage (V)	Current (A)	Power (W)	Connected (C) or Disconnected (D)
7.00	0.052	0.364	D
7.00	0.038	0.266	C
8.00	0.045	0.360	D
8.00	0.034	0.272	C
9.00	0.041	0.369	D
9.00	0.031	0.279	C
10.0	0.038	0.380	D
10.0	0.029	0.290	C

Table 1: Power consumption at various operating voltages (Note: motors not active during measurements).

The prototype vest is actually a stretch-fabric sleeve that fits over the user's forearm. A 3×3 array of motors is attached with double-sided tape to the outside of the sleeve, as shown in Figure 9. The DC motors vibrate at 110 Hz when supplied with 3.3 V from the board. The sleeve demonstrated that a number of tactile commands could be accurately identified by the user — paving the way for the construction of a 4×4 torso-mounted array.

The analog-to-digital (A/D) converter is routed to the 16-pin connector on the board, but it was only tested for basic functionality in this phase of the design. The assembly code running on the microcontroller can initiate a query of only the first channel in the 8-channel A/D. It then transmits both bytes (10-bit resolution), starting with the least significant byte, to the host after completing the conversion. Furthermore, only four directional commands (up, down, left, and right) are implemented on this prototype. More commands can easily be added as needed, along with additional code to read the other seven A/D channels upon request from the host.

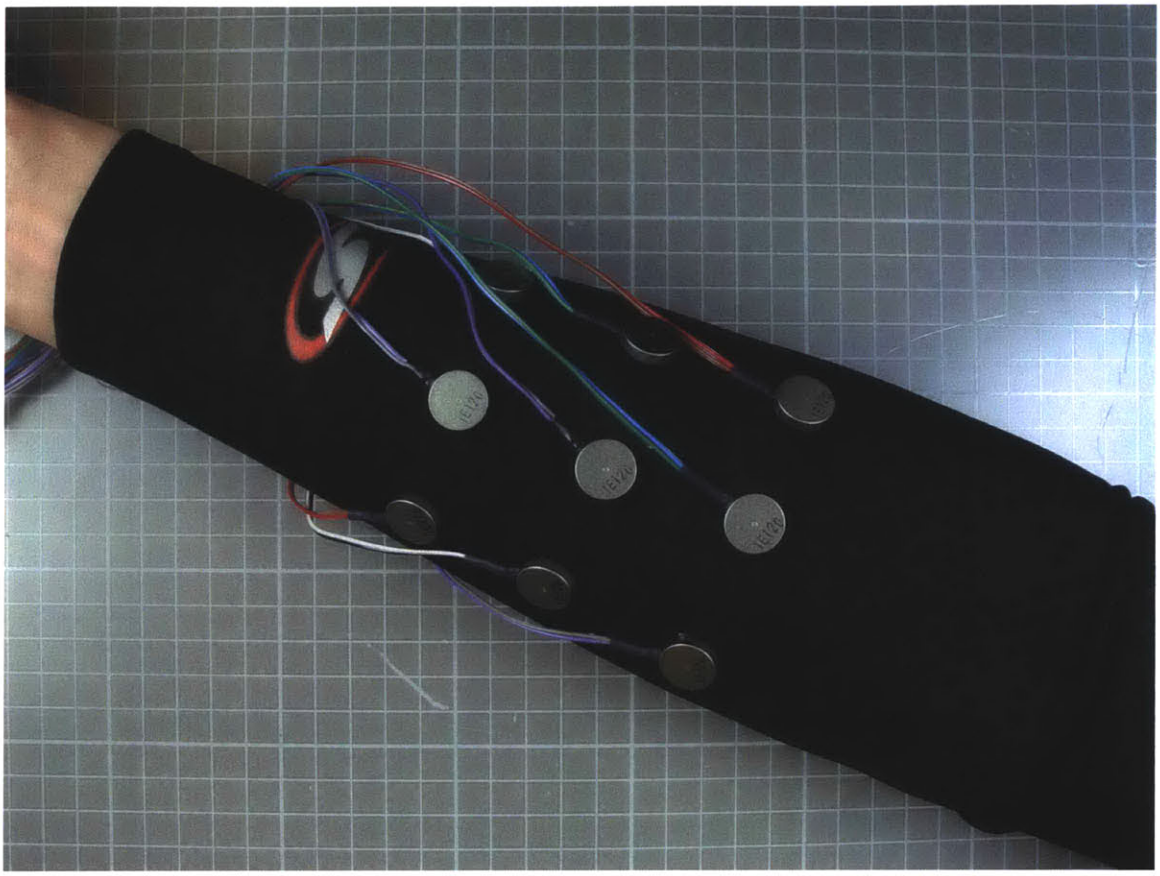

Figure 9: The prototype 3×3 motor array.

Software

The software required for the Tactile Vest includes a graphical user interface for the notebook computer as well as an assembly program for the microcontroller. The software for the notebook computer will reside on the computer's hard drive, to be run during operation of the vest, while the assembly program will be stored in the microcontroller's flash memory, running automatically each time the vest circuit is powered on. Both programs may be easily modified to accommodate additional features, commands, or vibratory patterns.

A graphical user interface (GUI) has been written in Microsoft Visual Basic .NET (see [20] for information about Visual Basic .NET) to run on the notebook computer using the Windows XP operating system. It provides the user with a list of commands to transmit to the vest and displays the data returning from the vest's sensors. The software interfaces with the computer's COM (serial) port, which communicates using the RS-232 protocol, and manages the transmission and reception of characters and data. The Bluetooth module plugs into the computer's USB port and creates a virtual serial port in software that allows communication as if it were actually connected to the serial port. The GUI software was initially written by an undergraduate student working in the lab and has been updated and customized to work with the Tactile Vest system.

The GUI has eight buttons, each one sending a separate command that can be linked to a vibratory pattern via the embedded processor's software. Six directional commands are implemented here as an example. These are: up, down, left, right, forward, and back. A warning command is also incorporated to inform the operator of imminent danger in the vicinity. The final command is a request for the vest to send data

from its sensors. There is currently no vibratory pattern linked to this eighth command. Figure 10 shows a screenshot of the GUI running on Windows XP.

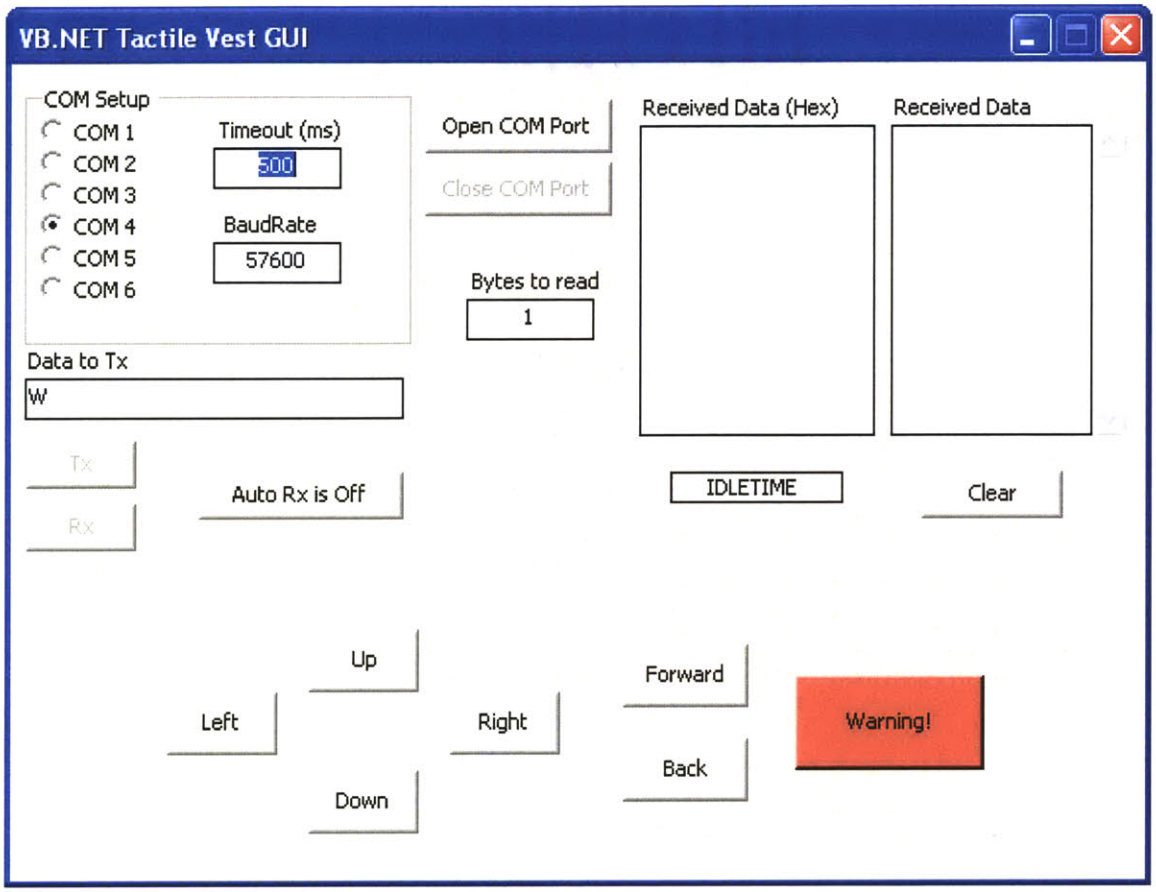

Figure 10: A screenshot of the Visual Basic .NET GUI.

The microcontroller's software is written in assembly language. Unlike higher-level languages, such as C++ or Visual Basic, which have a predefined set of instructions and will run on virtually any computer hardware platform, assembly language is based entirely on the hardware platform on which it will be executed. The set of instructions therefore varies with the embedded processor model and manufacturer. The Tactile Vest's assembly program is written for the Atmel AVR series microcontroller, and it uses only the commands implemented on the AT90LS8535 processor.

The microcontroller recognizes 118 separate commands [17] which grant the programmer full control over data storage and usage, leading to highly efficient code that is executed very quickly. After the program is written, an assembler program translates the assembly code into machine code — the most basic representation possible of the code. This machine code is simply a string of bytes that is stored in the processor's eight kilobytes of flash memory. Flash memory is convenient because it is electrically erasable and writable and does not require ultraviolet light exposure to erase its contents. The bytes stored in the flash memory represent different commands or arguments of commands depending on their location, or address, in the memory space.

There are C language compilers available for this processor, but these take the high-level commands from C and translate them into equivalent assembly language commands, which are then translated into machine language to be stored on and read by the microcontroller. This programming method is convenient because C contains hardware-independent commands that can be used to program any microcontroller with a C compiler (most of them have one available), but it can lead to inefficient code that is not optimized for the specific application since the translation from C to machine code is not direct. Assembly language translates directly to machine code, so programming in assembly leads to the most efficient code possible. This minimizes the latency between command transmission and vibratory pattern execution.

Latency of response is not the only design consideration for this system. Ultra-low power consumption is a priority as well. The microcontroller features several low-power modes of operation, which facilitate minimizing power consumption. Idle mode is the only one used in this application because it stops the central processing unit (CPU)

but allows the UART, A/D converter, and timers to continue functioning. No instructions can be executed without the CPU, but during periods of inactivity such as when waiting for a command to be sent, idle mode can reduce the microcontroller's power consumption by about 70% [17].

The assembly code instructs the processor to enter idle mode whenever it is feasible, even if only for a few microseconds, to minimize the average power consumption. In fact, during a typical operating session with the vest, the microcontroller "sleeps" in idle mode nearly all of the time. Idle mode is entered while waiting for a command to be sent, which is the system's state for the vast majority of its operating time. The microcontroller is woken up (switched to active mode) automatically upon reception of a character, and the requested motor control pattern is executed. Idle mode is entered once again while timing this pattern. Since the motor controllers will maintain their states until the microcontroller modifies them, the microcontroller is free to sleep between state changes for a given pattern, which could take one to two seconds. The timer is used to awaken the microcontroller for the next state change because it remains functional during idle mode.

The intrinsic tradeoff between latency and power consumption does not apply in this situation. The microcontroller transitions almost instantaneously from idle to active mode, thereby reducing power consumption without an appreciable increase in the latency. The wireless transceivers remain the most significant contributors to the overall system latency.

Further Research

There are numerous possibilities for further development of the Tactile Vest system, including modifications to both hardware and software. Scenarios in which multiple vests are in use simultaneously within range of each other were not the focus of this research project, but algorithms to handle these situations effectively would be invaluable to the system's effective operation in the field. New wireless technologies currently under development may simplify these algorithms drastically, allowing for the creation of much more complex and dynamic transceiver networks than the Bluetooth system can support.

The multiple-vest situation introduces a number of new design considerations and necessitates a change in the software's structure both on the embedded and computer sides. One of the most important issues is that of data collision. If the vests are programmed to send data back to the host periodically and at random time intervals, the potential for data collision increases as the number of vests within communication range of the host grows. Data transmitted to the host simultaneously from separate vests would collide, causing the loss or corruption of both data sets. Potential advantages of this communication method, however, include immediate transmission of vital information detected by the vest's sensors, allowing for a fast user response and increasing the user's safety and awareness.

Another strategy for dealing with this potential problem might be to initiate data transmission on the host side and allocate a separate time interval for communication with each module to avoid data collisions. This leads to more complex host software and increases the communication delay with any given module. For example, if a vest's

sensors detect a chemical hazard in the operator's vicinity, the information may not be queried by the host for some time. This delay would be unacceptable in some circumstances, so other strategies should be explored.

An additional problem is that each module's position is not always fixed. The vests will be attached to mobile operators, so the network's structure will be constantly changing. This essentially rules out Bluetooth for use with an expanded system. Although the Bluetooth modules may be configured to automatically connect to a specific host when within range, the connection process takes too long (2-30 sec), and the technology can only handle simultaneous communications with 7 devices per network [14]. Therefore, another wireless technology is required for optimal functionality of this system in the field.

The Zigbee protocol currently under development will address many of the shortcomings of Bluetooth in the field of low-power, low-data-rate communication within expansive, complex, dynamically restructuring networks. The Tactile Vest system would benefit substantially from the integration of Zigbee technology, allowing it to fulfill nearly all the desired specifications outlined earlier and providing much greater flexibility and functionality than is possible with Bluetooth.

The IEEE is responsible for detailing the physical (PHY) and medium access control (MAC) layers of the protocol. This standard, IEEE 802.15.4, supports data rates of 20 kbps, 40 kbps, and 250 kbps [21]. Working across multiple RF bands (868 MHz, 915 MHz, and 2.4 GHz), this wireless technology can occupy unlicensed frequency bands throughout the world, leading to simple, globally functional solutions [21].

The ZigBee Alliance is responsible for outlining the boundaries and design specifications of the network and security layers, as well as the application framework layers of the protocol [21, 22]. These layers are "built" upon the PHY and MAC layers. Preliminary specifications include low power consumption, long battery life in target applications, low cost, and support for up to 255 devices per network [22]. User applications and profiles represent the uppermost layer of the protocol, which, of course, vary with the specific implementation.

Zigbee provides ample bandwidth for the Tactile Vest system since only a few characters are exchanged with the host each time a command is sent. One character is used for the command and another for the reply. Sensor data from the vest consists of two bytes of information per reading, due to the 10-bit A/D converter used to digitize each sensor's output. Even if all eight sensors are queried sequentially, only 16 bytes of information will be transmitted in a single communication session. Since the sessions are sporadic and take less than a second each on average, the system's duty cycle is very low. Zigbee technology is perfectly suited to an application like this whereby a device wakes up from a low-power mode, connects to the network, receives or transmits data, then resumes low-power operation.

This low duty cycle scenario also requires a fast connection time because most wireless modules are incapable of maintaining a connection with the host while in low-power mode. Zigbee's connection and authentication process takes only 30 ms, while a single module takes about 15 ms to transition from low-power mode to active mode [21]. This is two orders of magnitude faster than Bluetooth, and it allows for the creation of seamlessly self-forming, self-healing networks.

The types of networks supported in the Zigbee protocol include star, mesh, and cluster tree [21]. Star networks usually have a central node that communicates via separate channels to outlying nodes in the vicinity. The topology resembles an old wagon wheel with the central node at the hub and each spoke representing the communication link between the hub and the outlying nodes around the wheel's circumference. The outlying nodes do not communicate with each other. The mesh network topology is more free-formed in that there is no central hub and communication links may be established from any node to any other node (also called peer-to-peer). Cluster tree networks incorporate hybrid star/mesh topology to provide multiple communication pathways between modules to improve reliability while retaining some localized structure.

With multiple communication pathways between nodes, the communication range of a given module can be greatly enhanced, providing a significant advantage over Bluetooth and other wireless technologies that lack inherent support for this feature. In harsh RF environments such as dense urban locales, where the Tactile Vest system will often be operated, the range of each module can be significantly reduced due to interference from nearby obstacles. By working together to relay information to and from the host, a Zigbee system should exhibit very high reliability.

Reliability in demanding RF environments is further enhanced by features such as packet freshness data, collision avoidance algorithms, guaranteed time slots for critical data transmission, and acknowledgement of packet reception [21]. Because these features are already integrated into the Zigbee stack, the user does not need to develop

new algorithms for handling these situations, thereby reducing the development time for new systems and allowing for rapid integration into existing systems.

Stack size is another important concern in embedded applications, especially when memory space is limited. The Zigbee stack, containing all the necessary software to manage network connections, low power modes, and communications, is around one-tenth the size of the Bluetooth stack (28 kilobytes for Zigbee versus 250 kilobytes for Bluetooth) [21]. This leaves more space for the user's application software and data storage.

Zigbee is not the only wireless technology on the market that can be incorporated into the Tactile Vest project. Table 2 provides an overview of the specifications of most wireless technologies available today. Note that the UHF and Near Field Magnetic communication protocols are only standardized for very specific applications such as Citizens Band (CB) radio. In other words, one UHF or Near Field Magnetic device is not guaranteed to communicate with another from a different manufacturer outside of specific applications. Other, more standardized protocols such as Wireless USB (WUSB) and Wi-Fi (802.11b) will be considered for completeness.

Wireless USB (WUSB) is a high-speed, point-to-point wireless protocol designed to replace cable USB connections in the personal computer peripheral market. The radio specifications, as detailed by the newly formed WUSB Promoter Group, will be based on ultra wideband radio technology efforts by the MultiBand OFDM Alliance (MBOA) and the WiMedia Alliance [23] (see [24] for additional information on WUSB). It will initially support the same bandwidth as USB 2.0 (480 Mbps), with a projected bandwidth

exceeding 1 Gbps as ultra wideband radio technology evolves [25]. Since it is designed as cable replacement technology, the range is limited to less than 10 m [25].

Wireless Communication Systems

Property	802.11b/g (Wi-Fi)	Bluetooth	ZigBee	UWB	UHF	Wireless USB	IR Wireless	Near Field Magnetic
Operating Frequency	2.4 GHz	2.4 GHz	868 MHz (Europe), 902 - 928 MHz (Americas), 2.4 GHz (Worldwide)	3.1 - 10.6 GHz	260 - 470 MHz, 902 - 928 MHz	2.4 GHz	Infrared (800 - 900 nm)	Magnetic Coupling
Data Rate	802.11b: 11Mbps, 802.11g: 54 Mbps	1 Mbps	20 kbps, 40 kbps, 250 kbps	100 - 500 Mbps	10 - 100 kbps	62.5 kbps	20 - 40 kbps, 115 kbps, 4 & 16 Mbps	64 - 384 kbps
Range	50 - 100 m	10 m	10 - 100 m	< 10 m	10 m - 16 km	10 m	1 - 9 m (line-of-sight)	1 - 3 m
Networking	Point-to-multipoint	Ad hoc piconets	Ad hoc, star, peer-to-peer, mesh	Point-to-point	Point-to-point	Point-to-point	Point-to-point	Point-to-point
Complexity	High	High	Low	Medium	Lowest	Low	Low	Low
Power Consumption	High	Medium	Very Low	Low	Low	Low	Low	Low
Applications	WLAN Hotspots.	Wireless Headsets, PC-PDA-Laptop connections.	Industrial monitoring and control, home automation and control, sensor networks, toys, games, automotive, medical.	Home entertain-ment networks, streaming video.	Coded remote control, remote keyless entry, garage doors.	PC peripherals.	Remote control, PC-PDA-Laptop connections.	Wireless headsets, automotive.

Table 2: Comparison of various wireless technologies (adapted from [11]).

The primary supported network topology is the point-to-point, hub-and-spoke network, with a single host communicating with as many as 127 devices simultaneously [25]. Each slave device, nominally embedded in a computer peripheral such as a printer or scanner, can only communicate with the host and not with other slave devices. In certain circumstances, however, a slave module may exhibit limited host capabilities. Multiple clusters may be created in this fashion, but details as to how many clusters can coexist within radio range of each other are still being considered [25].

The power consumption target for the initial phase of WUSB devices is less than 300 mW, and projected consumption for future revisions is 100 mW [25]. This is less than the MaxStream modules, but comparable to the Bluetooth devices, which are still considered relatively high-power solutions for the Tactile Vest application.

In short, Wireless USB technology is designed for low-power, high-bandwidth, short-range, point-to-point communication. The Tactile Vest requires low-bandwidth communication, high efficiency, low-power operation, point-to-multipoint networking capability, and long-range transceivers. Wireless USB cannot enhance the current system's functionality or reliability enough to justify its incorporation into the Tactile Vest's circuitry.

The final wireless protocol under consideration for overall system augmentation is Wi-Fi (short for Wireless Fidelity), or IEEE 802.11. It is designed mostly for wireless networking in personal computer applications. Therefore, it supports point-to-multipoint network topologies, providing connections between computers or from a computer to a wireless hub or access point. It also offers very high bandwidth (up to 54 Mbps for IEEE 802.11g, [26]) for streaming video and audio, file sharing, and Internet access.

Unfortunately, power consumption is less of an issue in this market, but emerging handheld computer technologies are making this a consideration. In certain mobile applications, power consumption for Wi-Fi systems has been significantly reduced (to around 1 W, [27]), but it remains the most power-hungry of all the wireless technologies [21]. Its significantly increased bandwidth generally justifies the additional power consumption, but in the application considered here, bandwidth is the least important consideration.

Besides power consumption, another disadvantage of Wi-Fi technology is its sheer complexity, in both software and hardware. On the software side, an implementation of Wi-Fi in an embedded system generally requires a TCP/IP stack, a real-time operating system (RTOS), and application-specific software, not to mention an optional web server for remote monitoring over the Internet using standard web browser software [27]. Surprisingly, many hand-held computing devices, such as the Compaq IPAQ or the Palm Pilot, are able to support this extensive software requirement, but their hardware is very complex, while the Tactile Vest's hardware is simple and efficient.

Wi-Fi module manufacturers realize that many of their customers do not possess the necessary resources (or board and memory space) to implement complex software and hardware in their embedded applications, and so the manufacturers usually incorporate everything into the modules themselves [27, 28]. This provides the consumer with fully functional, "drop-in" Wi-Fi solutions, thereby drastically reducing development and product integration time, as well as system complexity (not including the module).

One considerable advantage of Wi-Fi technology over most other standardized wireless systems is its increased operating range. With transmission distances of over 100 m between unobstructed modules [21, 28], the range is slightly superior to that of Class 1 Bluetooth modules. Of course, the range varies with the transmission power of the module, which varies with the manufacturer. Some companies, namely SyChip (Plano, TX) advertise outdoor ranges of around 122 m and indoor ranges of around 37 m while transmitting at only a quarter of the output power of the Bluetooth device used in

the Tactile Vest circuit [28]. This may be attributed to increased sensitivity in the SyChip module's receiver circuitry.

The Tactile Vest could easily be modified to work with Wi-Fi wireless technology, but this would not be the most rational course of action. The overall system complexity would increase significantly to provide a less significant increase in its functionality. Wi-Fi's extraordinarily high data rate is countered by its prohibitive power consumption, leading one to believe that a more suitable solution to the Tactile Vest application must exist. Integration of the Zigbee wireless technology into the Tactile Vest system would provide such a significant enhancement to its overall functionality that it would constitute the next logical step in the further development of the vest.

References

[1] J.B.F. van Erp and J.J. van den Dobbelsteen. "On the Design of Tactile Displays." TNO Human Factors report, TM-98-B012. TNO, The Netherlands, 1998.

[2] K.A. Kaczmarek and P. Bach-Y-Rita. "Tactile Displays." In *Virtual Environments and Advanced Interface Design*, W. Barfield and T.A. Furness, III (Eds.). New York, NY: Oxford University Press, 1995, pp. 349-414.

[3] L.A. Jones, M. Nakamura, B. Lockyer. "Development of a Tactile Vest." Proceedings of the IEEE *12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*. Chicago, IL: IEEE, March 27-8, 2004, pp. 82-89.

[4] R.W. Cholewiak and A.A. Collins. "The Generation of Vibrotactile Patterns on a Linear Array: Influences of Body Site, Time, and Presentation Mode," *Perception & Psychophysics*, Vol. 62, No. 6, 2000, pp. 1220-1235.

[5] B. Luke. "Design of a Microelectronic Controller with a MIL-STD-1553 Bus Interface for the Tactile Situation Awareness System." Engineer's Thesis, Naval Postgraduate School, Monterey, CA, September 1998.

[6] A.H. Rupert. "An Instrumentation Solution for Reducing Spatial Disorientation Mishaps." *IEEE Engineering in Medicine and Biology*, March/April 2000, pp. 71-80.

[7] A.H. Rupert, F.E. Guedry, and M.F. Reschke, "The Use of a Tactile Interface to Convey Position and Motion Perceptions," presented at an *AGARD Meeting on "Virtual Interfaces: Research and Applications,"* October 1993, pp. 20-1 – 20-7.

[8] J.L. Rochlis and D.J. Newman. "A Tactile Display For International Space Station (ISS) Extravehicular Activity (EVA)," *Aviation, Space, and Environmental Medicine*, Vol. 71, No. 6, June 2000, pp. 571-578.

[9] S.P. Korolev. "DSM Delta Project: SUIT Experiment," Science Research on ISS Russian Segment, RSC Energia. Internet: http://www.energia.ru/english/energia/iss/researches/medic-53.html, obtained May 2004.

[10] C. Wall, III, and M.S. Weinberg. "Balance Prostheses for Postural Control: Preventing Falls in the Balance Impaired by Displaying Body-Tilt Information to the Subject via an Array of Tactile Vibrators." *IEEE Engineering in Medicine and Biology Magazine*, March/April 2003, pp. 84-90.

[11] MaxStream, Inc., "XStream 900MHz & 2.4GHz Wireless OEM Modules: OEM Manual," Data Sheet, June 2002.

[12] Anonymous, "The Official Bluetooth Membership Site," Bluetooth Special Interest Group (SIG) information page, Internet: http://www.bluetooth.org, Bluetooth Special Interest Group, 2004.

[13] BlueRadios, Inc., "Bluetooth Module BR-C11 Class 1," Data Sheet.

[14] BlueRadios, Inc., "User Guide for BlueRadios Bluetooth Serial Module AT Command Set." Data Sheet, Rev. 1.1.9, April 2004.

[15] E. Hecht. *Optics*, 4th ed. San Francisco, CA: Addison Wesley, 2002, pp. 443-485.

[16] D.H. Staelin, A.W. Morgenthaler, J.A. Kong. *Electromagnetic Waves*. Upper Saddle River, NJ: Prentice Hall, 1998, pp. 406-489.

[17] Atmel, "8-bit AVR Microcontroller with 8K Bytes In-System Programmable Flash: AT90S8535, AT90LS8535," Data Sheet Rev. 1041H, November 2001.
[18] Allegro Microsystems, Inc., "6B259," Data Sheet No. 26186.122, January 2000.
[19] National Semiconductor, "LM2595 SIMPLE SWITCHER Power Converter 150kHz 1A Step-Down Voltage Regulator," Data Sheet No. DS012565, May 1999.
[20] Anonymous, "Microsoft Visual Basic Developer Center," Microsoft Developer Network (MSDN) Visual Basic .NET information page, Internet: http://msdn.microsoft.com/vbasic/, Microsoft Corporation, 2004.
[21] Frenzel. "The ZigBee Buzz is Growing: New Low-Power Wireless Standard Opens Powerful Possibilities," *Electronic Design*, January 12, 2004.
[22] Bahl. "Zigbee Overview," Marketing and information brief. Internet: http://www.zigbee.com; ZigBee Alliance, September 2002.
[23] Anonymous. "Industry Leaders Developing First High-Speed Personal Wireless Interconnect." *Intel Press Release.* Internet: http://www.intel.com/pressroom/archive/releases/20040218corp_c.htm, February 18, 2004.
[24] Anonymous, "Wireless Universal Serial Bus," Intel WUSB information page, Internet: http://www.intel.com/labs/wusb/, Intel Corporation 2004.
[25] R. Kolic. "Wireless USB Brings Greater Convenience and Mobility to Devices." *Technology @ Intel Magazine.* Internet: http://www.intel.com/update/contents/wi02041.htm, Feb./Mar. 2004.
[26] Reaff. "IEEE Std. 802.11g – 2003," Amendment to "IEEE Std. 802.11, 1999 Edition." New York, NY: IEEE, June 25, 2003.
[27] DPAC Technologies, "Airborne 802.11b Wireless LAN Node Module: WLNB-AN-DP101," Data Sheet 30B211-01, Rev. E, January 2004.
[28] SyChip, "SyChip WLAN6060EB Embedded Module Product Brief," Product Brief, WLAN6060EBC1, Ver. 2.1, obtained May 2004.

Appendix A

This appendix contains the code that executes on the microcontroller. It is programmed in assembly language for the Atmel AT90LS8535 processor. The code assumes the Bluetooth module is already configured to communicate at 57,600 baud and respond to commands in short form with no authorization enabled. See [14] for details on configuring the Bluetooth module.

```
;Tactile Vest
;AT90LS8535 Microcontroller code
;Version 5 (Updated Bluetooth)
;5/5/04

.include "8535def.inc"

.org 0x00
start:   rjmp setup

.org 0x008
         rjmp tisr              ;timer 1 has overflowed

.org 0x00B
         rjmp rxisr             ;character received

.org 0x100
setup:   ldi r16, 0x9F
         out SPL, r16           ;initialize stack pointer to 0x9F
         clr r16
         out SPH, r16
         out DDRA, r16          ;configure port A for inputs
         out PORTA, r16         ;clear port A
         out ADMUX, r16         ;ADC0 is input (pin A0)
         ser r16
         out DDRB, r16          ;configure Port B as outputs only
         out DDRC, r16          ;configure Port C as outputs only
         out DDRD, r16          ;configure Port D as outputs only
         ldi r16, 0x0F
         out PORTB, r16         ;port B = 0x0F
         clr r16
         out PORTC, r16         ;port C = 0x00
         out PORTD, r16         ;port D = 0x00
         ldi r16, 0x03
         out UBRR, r16          ;set UBRR to 57600 baud, 3.6864 MHz xtal
         ldi r16, 0x98
         out UCR, r16           ;set UCR to enable interrupt on RX done
         ldi r16, 0x85
         out ADCSR, r16         ;enable ADC, no interrupts, CLK/32 freq,
                                ;  single conversion mode
         ldi r16, 0x04
         out TIMSK, r16         ;enable interrupt on timer 1 overflow
         ldi r16, 0x40
         out MCUCR, r16         ;enable idle sleep mode
         rcall waitlong         ;wait for module to start up
         rcall btcmd
```

```
        rcall waitmore
        rcall btconn        ;make connection to USB Bluetooth module
        rjmp rxchar

;*****************************************************
;BLUETOOTH CONFIGURATION AND CONNECTION ROUTINES

btconn: ldi r18, 0x41
        rcall txchar        ;send A
        ldi r18, 0x54
        rcall txchar        ;send T
        ldi r18, 0x44
        rcall txchar        ;send D
        ldi r18, 0x53
        rcall txchar        ;send S
        ldi r18, 0x0D
        rcall txchar        ;send cr
        ldi r18, 0x0A
        rcall txchar        ;send lf
        clr r7
        rcall rxresp        ;looking for 0 (OK) which is 5 chars
        rcall respok
        ldi r16, 0x80
        out SREG, r16       ;enable interrupts (global)
        sleep               ;waiting for receive char (connect= 1)
        nop
        nop
        nop
        clr r16
        out SREG, r16       ;disable interrupts (global)
        clr r7
        inc r7              ;looking for 18-1=17 chars now since 1st
                            ;  was received during sleep
        rcall rxrespl
        rcall respconnl     ;looks for the last digit in the USB
                            ;communicator's 12-digit address
                            ;to validate the connection
        ldi r16, 0x80
        out SREG, r16       ;enable interrupts (global)
        ldi r16, 0x02
        out PORTB, r16      ;set port pin PB1 high if connection made
        ret                 ;the module should be connected by the
                            ;  time this is executed

btcmd:  ldi r18, 0x41
        rcall txchar        ;send A
        ldi r18, 0x54
        rcall txchar        ;send T
        ldi r18, 0x4D
        rcall txchar        ;send M
        ldi r18, 0x43
        rcall txchar        ;send C
        ldi r18, 0x0D
        rcall txchar        ;send cr
        ldi r18, 0x0A
        rcall txchar        ;send lf
        clr r7
```

```
        rcall rxresp        ;look for 6 chars
        rcall respok        ;is response 0=OK ?
        ldi r16, 0x01
        out PORTB, r16      ;set port pin PB0 high if module
                            ;  communication successful
        ret

;************************************************
;BLUETOOTH MODULE RESPONSE RECEPTION ROUTINES

rxresp: in r19, USR         ;MUST BE CALLED WITH THE PROPER VALUE
                            ;  LOADED INTO R7!
        sbrs r19, 7
        rjmp rxresp         ;loop back if char not received yet
        rcall rxchek        ;read char to r17 after reception
                            ;  (if no data errors)
        inc r7              ;r7 used as an indicator of # chars to
                            ;  look for (call w/ r7=x; 5-x=#chars)
        mov r16, r7
        cpi r16, 0x05
        breq rxend
        cpi r17, 0x0D       ;is char cr?
        breq rxresp
        cpi r17, 0x0A       ;is char lf?
        breq rxresp
        mov r0, r17         ;***store response in r0 (will not store
                            ;  CR or LF chars)***
        rjmp rxresp
rxend:  clr r7
        ret                 ;r17 contains 0x0A here (usually)

rxchek: in r20, USR
        sbrs r20, 4         ;check USR's FE bit for framing error
        rjmp chekor
        rjmp ferr
chekor: sbrs r20, 3         ;check USR's OR bit for overrun error
        rjmp rxok
        rjmp orerr
ferr:   rjmp end
orerr:  rjmp end
rxok:   in r17, UDR         ;received char --> r17
        ret


rxrespl: in r19, USR        ;MUST BE CALLED WITH THE PROPER VALUE
                            ;  LOADED INTO R7!
        sbrs r19, 7
        rjmp rxrespl        ;loop back if char not received yet
        rcall rxchekl       ;read char to r17 after reception
                            ;  (if no data errors)
        inc r7              ;r7 used as an indicator of # chars to
                            ;  look for (call w/ r7=x, 18-x=#chars)
        mov r16, r7
        cpi r16, 0x12       ;looking for 18 chars
        breq rxendl
        cpi r17, 0x0D       ;is char cr?
        breq rxrespl
```

```
        cpi r17, 0x0A       ;is char lf?
        breq rxrespl
        mov r0, r17         ;***store response in r0 (will not store
                            ;  CR or LF chars)***
        rjmp rxrespl
rxendl: clr r7
        ret                 ;r17 contains 0x0A here (usually)
rxchekl: in r20, USR
        sbrs r20, 4         ;check USR's FE bit for framing error
        rjmp chekorl
        rjmp ferrl
chekorl: sbrs r20, 3        ;check USR's OR bit for overrun error
        rjmp rxokl
        rjmp orerrl
ferrl:  rjmp end
orerrl: rjmp end
rxokl:  in r17, UDR         ;received char --> r17
        ret

;***************************************
;BLUETOOTH MODULE RESPONSE HANDLERS
;The response is expected in r0 for all routines.

respok: ldi r16, 0x30
        cpse r16, r0
        rjmp end
        ret                 ;returns if response = OK = 0x30 = 0

respokl: ldi r16, 0x4B
        cpse r16, r0        ;does r0 contain K=4B? (from response
                            ;  cr,lf,O,K,cr,lf = 6 chars)
        rjmp end            ;skip this if response = OK (long)
        ret

respconn:ldi r16, 0x31
        cpse r16, r0
        rjmp end
        ret                 ;returns if response = CONNECT = 0x31 = 1

respconnl:ldi r16, 0x41     ;this number must be changed to the last
                            ;digit (hex) in the 12-digit USB master's
                            ;address (***NOTE: Will only recognize
                            ;certain masters).
        cpse r16, r0
        rjmp end
        ret

;************************************
;CHARACTER TRANSMISSION ROUTINES

txchar: rcall txloop
        out UDR, r18        ;***transmits char in r18***
        rcall txloop
        ret

txloop: in r20, USR
        sbrs r20, 5         ;check UDRE to see if ready for new char
```

```
        rjmp txloop
        ret                     ;return when ready to transmit

;*******************************************
;COMMAND RECEPTION/READY STATUS ROUTINES

rxchar: rcall ready             ;transmit ready (# LF)
        sleep
        nop                     ;nops in case more instructions are
                                ;  executed before processing interrupt
        nop
        nop
        rcall motor             ;received char in r17
        rjmp rxchar

ready:  ldi r18, 0x23           ;send #
        rcall txchar
        ldi r18, 0x0A           ;send LF (new line)
        rcall txchar
        ret

;*****************************
;MOTOR CONTROL ROUTINES
;Want to sleep during timing and wake up on timer overflow (idle mode)
;Each increment of the 16-bit timer register equals 256 microseconds at
; CK/1024 (i.e. 2 seconds should be about 0x1E84 increments,
; so load into timer register 0xE17B because it counts UP until
; overflow)
; i.e. 0xF85E for 0.5 second
;On motor controllers, EN1 and EN2 grounded; DATA1 on PC3, DATA2 on
; PC7; CLR1 and CLR2 on PD7.  Control lines for 1 on PC0,1,2 and for 2
; on PC4,5,6.
;*MOTOR CONTROLLER ADDRESS LINES MUST NOT CHANGE AT THE SAME TIME AS
; DATA LINES*

motor:  cpi r17, 0x55           ;must translate command char in r17
                                ;  (U,D,L,R,F,B,W) to actuations
        breq brup               ;char=U
        cpi r17, 0x44
        breq brdown             ;char=D
        cpi r17, 0x4C
        breq brleft             ;char=L
        cpi r17, 0x52
        breq brright            ;char=R
        cpi r17, 0x46
        breq brforw             ;char=F
        cpi r17, 0x42
        breq brback             ;char=B
        cpi r17, 0x57
        breq brwarn             ;char=W
        cpi r17, 0x5A
        breq brana              ;char=Z
        ldi r18, 0x3F
        rcall txchar            ;send ? for unrecognized char
        ret

brup:   rcall up
```

```
        ret
brdown: rcall down
        ret
brleft: rcall left
        ret
brright: rcall right
        ret
brforw: rcall forward
        ret
brback: rcall back
        ret
brwarn: rcall warning
        ret
brana:  rcall analog
        ret


up:     ldi r16, 0x01
        out PORTB, r16
        sbi PORTD, 7          ;set CLR high
        clr r16
        out PORTC, r16
        sbi PORTC, 3
        ldi r16, 0x09
        out PORTC, r16
        ldi r16, 0x0A
        out PORTC, r16
        ldi r31, 0xF8         ;high byte of timer 1 count
        ldi r30, 0x5E         ;low byte of timer 1 count
        rcall delay
        cbi PORTC, 3
        ldi r16, 0x01
        out PORTC, r16
        clr r16
        out PORTC, r16
        ldi r16, 0x03
        out PORTC, r16
        sbi PORTC, 3
        ldi r16, 0x0C
        out PORTC, r16
        ldi r16, 0x0D
        out PORTC, r16
        ldi r31, 0xF8         ;high byte of timer 1 count
        ldi r30, 0x5E         ;low byte of timer 1 count
        rcall delay
        cbi PORTC, 3
        ldi r16, 0x04
        out PORTC, r16
        ldi r16, 0x03
        out PORTC, r16
        ldi r16, 0x06
        out PORTC, r16
        sbi PORTC, 3
        ldi r16, 0x0F
        out PORTC, r16
        ldi r16, 0x8F
        out PORTC, r16
```

```
        ldi r31, 0xF8       ;high byte of timer 1 count
        ldi r30, 0x5E       ;low byte of timer 1 count
        rcall delay
        cbi PORTC, 7
        ldi r16, 0x07
        out PORTC, r16
        ldi r16, 0x06
        out PORTC, r16
        clr r16
        out PORTC, r16
        cbi PORTD, 7        ;set CLR low
        ret

down:   ldi r16, 0x02
        out PORTB, r16
        sbi PORTD, 7        ;set CLR high
        clr r16
        out PORTC, r16
        ldi r16, 0x07
        out PORTC, r16
        ldi r16, 0x8F
        out PORTC, r16
        ldi r16, 0x8E
        out PORTC, r16
        ldi r31, 0xF8       ;high byte of timer 1 count
        ldi r30, 0x5E       ;low byte of timer 1 count
        rcall delay
        ldi r16, 0x06
        out PORTC, r16
        ldi r16, 0x07
        out PORTC, r16
        ldi r16, 0x05
        out PORTC, r16
        ldi r16, 0x0D
        out PORTC, r16
        ldi r16, 0x0C
        out PORTC, r16
        ldi r16, 0x0B
        out PORTC, r16
        ldi r31, 0xF8       ;high byte of timer 1 count
        ldi r30, 0x5E       ;low byte of timer 1 count
        rcall delay
        ldi r16, 0x03
        out PORTC, r16
        ldi r16, 0x04
        out PORTC, r16
        ldi r16, 0x05
        out PORTC, r16
        clr r16
        out PORTC, r16
        ldi r16, 0x08
        out PORTC, r16
        ldi r16, 0x09
        out PORTC, r16
        ldi r16, 0x0A
        out PORTC, r16
        ldi r31, 0xF8       ;high byte of timer 1 count
```

```
        ldi r30, 0x5E       ;low byte of timer 1 count
        rcall delay
        ldi r16, 0x02
        out PORTC, r16
        ldi r16, 0x01
        out PORTC, r16
        clr r16
        out PORTC, r16
        cbi PORTD, 7        ;set CLR low
        ret

left:   ldi r16, 0x04
        out PORTB, r16
        sbi PORTD, 7        ;set CLR high
        clr r16
        out PORTC, r16
        ldi r16, 0x85
        out PORTC, r16
        ldi r16, 0x8D
        out PORTC, r16
        ldi r16, 0x8A
        out PORTC, r16
        ldi r31, 0xF8       ;high byte of timer 1 count
        ldi r30, 0x5E       ;low byte of timer 1 count
        rcall delay
        ldi r16, 0x02
        out PORTC, r16
        ldi r16, 0x05
        out PORTC, r16
        ldi r16, 0x07
        out PORTC, r16
        ldi r16, 0x0F
        out PORTC, r16
        ldi r16, 0x0C
        out PORTC, r16
        ldi r16, 0x09
        out PORTC, r16
        ldi r31, 0xF8       ;high byte of timer 1 count
        ldi r30, 0x5E       ;low byte of timer 1 count
        rcall delay
        ldi r16, 0x01
        out PORTC, r16
        ldi r16, 0x04
        out PORTC, r16
        ldi r16, 0x07
        out PORTC, r16
        ldi r16, 0x06
        out PORTC, r16
        ldi r16, 0x0E
        out PORTC, r16
        ldi r16, 0x0B
        out PORTC, r16
        ldi r16, 0x08
        out PORTC, r16
        ldi r31, 0xF8       ;high byte of timer 1 count
        ldi r30, 0x5E       ;low byte of timer 1 count
        rcall delay
```

```
        clr r16
        out PORTC, r16
        ldi r16, 0x03
        out PORTC, r16
        ldi r16, 0x06
        out PORTC, r16
        clr r16
        out PORTC, r16
        cbi PORTD, 7          ;set CLR low
        ret

right:  ldi r16, 0x08
        out PORTB, r16
        sbi PORTD, 7          ;set CLR high
        clr r16
        out PORTC, r16
        ldi r16, 0x06
        out PORTC, r16
        ldi r16, 0x0E
        out PORTC, r16
        ldi r16, 0x0B
        out PORTC, r16
        ldi r16, 0x08
        out PORTC, r16
        ldi r31, 0xF8         ;high byte of timer 1 count
        ldi r30, 0x5E         ;low byte of timer 1 count
        rcall delay
        clr r16
        out PORTC, r16
        ldi r16, 0x03
        out PORTC, r16
        ldi r16, 0x06
        out PORTC, r16
        ldi r16, 0x07
        out PORTC, r16
        ldi r16, 0x0F
        out PORTC, r16
        ldi r16, 0x0C
        out PORTC, r16
        ldi r16, 0x09
        out PORTC, r16
        ldi r31, 0xF8         ;high byte of timer 1 count
        ldi r30, 0x5E         ;low byte of timer 1 count
        rcall delay
        ldi r16, 0x01
        out PORTC, r16
        ldi r16, 0x04
        out PORTC, r16
        ldi r16, 0x07
        out PORTC, r16
        ldi r16, 0x85
        out PORTC, r16
        ldi r16, 0x8D
        out PORTC, r16
        ldi r16, 0x8A
        out PORTC, r16
        ldi r31, 0xF8         ;high byte of timer 1 count
```

```
          ldi r30, 0x5E        ;low byte of timer 1 count
          rcall delay
          ldi r16, 0x02
          out PORTC, r16
          ldi r16, 0x05
          out PORTC, r16
          clr r16
          out PORTC, r16
          cbi PORTD, 7         ;set CLR low
          ret

forward:  ldi r16, 0x10
          out PORTB, r16
          ret

back:     ldi r16, 0x20
          out PORTB, r16
          ret

warning:  ldi r16, 0x40
          out PORTB, r16
          ret

analog:   ldi r16, 0x80
          out PORTB, r16
          ldi r16, 0xD5        ;11010101 = D5 to clear ADIF first
          out ADCSR, r16
          clt                  ;clear T flag
aloop:    in r16, ADCSR
          bst r16, 4           ;store bit 4 of ADCSR (ADIF) to T
                               ;  flag
          brts axmit           ;branch to axmit if T flag is set
                               ; (conversion done, regs updated)
          rjmp aloop
axmit:    ldi r16, 0x95
          out ADCSR, r16       ;95=10010101 to clear ADIF flag
          clt                  ;clear T flag
          in r18, ADCL         ;conversion done, read low byte and xmit
          rcall txchar
          in r18, ADCH         ;read high byte and transmit
          rcall txchar
          ret


delay:    out TCNT1H, r31      ;always write HIGH byte first (assumes
                               ; this is in R31 when called)
          out TCNT1L, r30      ;writing low byte (assumes this is in R30
                               ; when called)
          ldi r16, 0x05
          out TCCR1B, r16      ;turn on timer 1 w/ CK/1024 prescale val.
          sleep
          ret
```

```
;********************
;DELAY ROUTINES
; Delay times given for a 4 MHz crystal

wait:     ser r27               ;this subroutine just delays
waitlp:   dec r27               ;1 cycle per dec, 1 cycle per false breq,
                                ; 2 cycles per rjmp = 1*255+1*254+2+2*254
                                ;= 1019 cycles
          breq goback           ;plus 1 for ser, 4 for ret = 5 cycles
          rjmp waitlp           ;total clock cycles = 1024 = 256 us
goback:   ret

waitmore:ser r28                ;this delays for 256*256us = 65.536ms
waitmlp: rcall wait
          dec r28
          breq dun
          rjmp waitmlp
dun:      ret

waitlong:ldi r29, 0x19          ;this delays for 25*65.536ms = 1.6384s
waitllp: rcall waitmore
          dec r29
          breq dunl
          rjmp waitllp
dunl:     ret

;*********************************************
;BLUETOOTH MODULE SOFTWARE RESET ROUTINE
; This subroutine issues a reset command to the Bluetooth module from
; software (in command mode).

btreset: rcall btcmd
          ldi r18, 0x41
          rcall txchar          ;send A
          ldi r18, 0x54
          rcall txchar          ;send T
          ldi r18, 0x55
          rcall txchar          ;send U
          ldi r18, 0x52
          rcall txchar          ;send R
          ldi r18, 0x53
          rcall txchar          ;send S
          ldi r18, 0x54
          rcall txchar          ;send T
          ldi r18, 0x0D
          rcall txchar          ;send cr
          ldi r18, 0x0A
          rcall txchar          ;send lf
          ret

;****************************
;ERROR HANDLING ROUTINE

end:      clr r16
          out SREG, r16         ;disable interrupts
          ldi r16, 0x0AA
          out PORTB, r16        ;Port B set high/low alternating
```

```
endlp:   rjmp endlp

;*********************************
;INTERRUPT SERVICE ROUTINES

rxisr:   in r20, USR
         sbrs r20, 4          ;check USR's FE bit for framing error
         rjmp checkor
         rjmp end
checkor: sbrs r20, 3          ;check USR's OR bit for overrun error
         rjmp rxokay
         rjmp end
rxokay:  in r17, UDR          ;received char --> r17
         reti


tisr:    clr r16
         out TCCR1B, r16      ;turn off timer 1
         reti
```

Appendix B

The following section of code is a customized class for Microsoft Visual Basic .NET (VB.NET) that provides support for simple serial (RS-232) communications. It uses native VB.NET and application programming interface (API) features and was written by Corrado Cavalli (http://www.corradocavalli.cjb.net, corrado@mvps.org). The code is freely redistributable.

```
Imports System.Runtime.InteropServices
Imports System.Text
Imports System.Threading

#Region "RS232"
Public Class Rs232 : Implements IDisposable
    '=================================================
    '
    '  Module            :     Rs232
    '  Description       :     Class for handling RS232 comunication
    '                          with VB.Net
    '  Created           :     10/08/2001 - 8:45:25
    '  Author            :     Corrado Cavalli
    '                          (corrado@mvps.org)
    '  WebSite           :     www.corradocavalli.cjb.net
    '
    '  Notes             :
    '=================================================
    '// Class Members
    Private mhRS As Int32 = -1   '// Handle to Com Port

    Private miPort As Integer = 1   '//  Default is COM1
    Private miTimeout As Int32 = 70   '// Timeout in ms
    Private miBaudRate As Int32 = 9600
    Private meParity As DataParity = 0
    Private meStopBit As DataStopBit = 0
    Private miDataBit As Int32 = 8
    Private miBufferSize As Int32 = 512   '// Buffers size default to
                                          ' 512 bytes
    Private mabtRxBuf As Byte()   '//  Receive buffer
    Private meMode As Mode  '//  Class working mode
    Private mbWaitOnRead As Boolean
    Private mbWaitOnWrite As Boolean
    Private mbWriteErr As Boolean
    Private muOverlapped As OVERLAPPED
    Private muOverlappedW As OVERLAPPED
    Private muOverlappedE As OVERLAPPED
    Private mabtTmpTxBuf As Byte()  '// Temporary buffer for Async Tx
    Private moThreadTx As Thread
    Private moThreadRx As Thread
    Private miTmpBytes2Read As Int32
    Private meMask As EventMasks
    Private mbDisposed As Boolean
    '-----------------------------------------------------------------
```

```
#Region "Enums"
   '// Parity Data
   Public Enum DataParity
      Parity_None = 0
      Pariti_Odd
      Parity_Even
      Parity_Mark
   End Enum
   '// StopBit Data
   Public Enum DataStopBit
      StopBit_1 = 1
      StopBit_2
   End Enum
   Private Enum PurgeBuffers
      RXAbort = &H2
      RXClear = &H8
      TxAbort = &H1
      TxClear = &H4
   End Enum
   Private Enum Lines
      SetRts = 3
      ClearRts = 4
      SetDtr = 5
      ClearDtr = 6
      ResetDev = 7    '  // Reset device if possible
      SetBreak = 8    '  // Set the device break line.
      ClearBreak = 9   '      // Clear the device break line.
   End Enum
   '// Modem Status
   <Flags()> Public Enum ModemStatusBits
      ClearToSendOn = &H10
      DataSetReadyOn = &H20
      RingIndicatorOn = &H40
      CarrierDetect = &H80
   End Enum
   '// Working mode
   Public Enum Mode
      NonOverlapped
      Overlapped
   End Enum
   '// Comm Masks
   <Flags()> Public Enum EventMasks
      RxChar = &H1
      RXFlag = &H2
      TxBufferEmpty = &H4
      ClearToSend = &H8
      DataSetReady = &H10
      ReceiveLine = &H20
      Break = &H40
      StatusError = &H80
      Ring = &H100
   End Enum
#End Region
#Region "Structures"
   <StructLayout(LayoutKind.Sequential, Pack:=1)> Private Structure DCB
      Public DCBlength As Int32
      Public BaudRate As Int32
```

```
        Public Bits1 As Int32
        Public wReserved As Int16
        Public XonLim As Int16
        Public XoffLim As Int16
        Public ByteSize As Byte
        Public Parity As Byte
        Public StopBits As Byte
        Public XonChar As Char
        Public XoffChar As Char
        Public ErrorChar As Char
        Public EofChar As Char
        Public EvtChar As Char
        Public wReserved2 As Int16
    End Structure


    <StructLayout(LayoutKind.Sequential, Pack:=1)> Private Structure
COMMTIMEOUTS
        Public ReadIntervalTimeout As Int32
        Public ReadTotalTimeoutMultiplier As Int32
        Public ReadTotalTimeoutConstant As Int32
        Public WriteTotalTimeoutMultiplier As Int32
        Public WriteTotalTimeoutConstant As Int32
    End Structure
    <StructLayout(LayoutKind.Sequential, Pack:=1)> Private Structure
COMMCONFIG
        Public dwSize As Int32
        Public wVersion As Int16
        Public wReserved As Int16
        Public dcbx As DCB
        Public dwProviderSubType As Int32
        Public dwProviderOffset As Int32
        Public dwProviderSize As Int32
        Public wcProviderData As Byte
    End Structure
    <StructLayout(LayoutKind.Sequential, Pack:=1)> Public Structure
OVERLAPPED
        Public Internal As Int32
        Public InternalHigh As Int32
        Public Offset As Int32
        Public OffsetHigh As Int32
        Public hEvent As Int32
    End Structure
#End Region
#Region "Constants"
    Private Const PURGE_RXABORT As Integer = &H2
    Private Const PURGE_RXCLEAR As Integer = &H8
    Private Const PURGE_TXABORT As Integer = &H1
    Private Const PURGE_TXCLEAR As Integer = &H4
    Private Const GENERIC_READ As Integer = &H80000000
    Private Const GENERIC_WRITE As Integer = &H40000000
    Private Const OPEN_EXISTING As Integer = 3
    Private Const INVALID_HANDLE_VALUE As Integer = -1
    Private Const IO_BUFFER_SIZE As Integer = 1024
    Private Const FILE_FLAG_OVERLAPPED As Int32 = &H40000000
    Private Const ERROR_IO_PENDING As Int32 = 997
    Private Const WAIT_OBJECT_0 As Int32 = 0
```

```
    Private Const ERROR_IO_INCOMPLETE As Int32 = 996
    Private Const WAIT_TIMEOUT As Int32 = &H102&
    Private Const INFINITE As Int32 = &HFFFFFFFF


#End Region

#Region "Win32API"
    '// Win32 API
    <DllImport("kernel32.dll")> Private Shared Function
SetCommState(ByVal hCommDev As Int32, ByRef lpDCB As DCB) As Int32
    End Function
    <DllImport("kernel32.dll")> Private Shared Function
GetCommState(ByVal hCommDev As Int32, ByRef lpDCB As DCB) As Int32
    End Function
    <DllImport("kernel32.dll")> Private Shared Function
BuildCommDCB(ByVal lpDef As String, ByRef lpDCB As DCB) As Int32
    End Function
    <DllImport("kernel32.dll")> Private Shared Function SetupComm(ByVal
hFile As Int32, ByVal dwInQueue As Int32, ByVal dwOutQueue As Int32) As
Int32
    End Function
    <DllImport("kernel32.dll")> Private Shared Function
SetCommTimeouts(ByVal hFile As Int32, ByRef lpCommTimeouts As
COMMTIMEOUTS) As Int32
    End Function
    <DllImport("kernel32.dll")> Private Shared Function
GetCommTimeouts(ByVal hFile As Int32, ByRef lpCommTimeouts As
COMMTIMEOUTS) As Int32
    End Function
    <DllImport("kernel32.dll")> Private Shared Function
ClearCommError(ByVal hFile As Int32, ByVal lpErrors As Int32, ByVal l
As Int32) As Int32
    End Function
    <DllImport("kernel32.dll")> Private Shared Function PurgeComm(ByVal
hFile As Int32, ByVal dwFlags As Int32) As Int32
    End Function
    <DllImport("kernel32.dll")> Private Shared Function
EscapeCommFunction(ByVal hFile As Integer, ByVal ifunc As Long) As
Boolean
    End Function
    <DllImport("kernel32.dll")> Private Shared Function
WaitCommEvent(ByVal hFile As Integer, ByRef Mask As EventMasks, ByRef
lpOverlap As OVERLAPPED) As Int32
    End Function
    <DllImport("kernel32.dll")> Private Shared Function WriteFile(ByVal
hFile As Integer, ByVal Buffer As Byte(), ByVal nNumberOfBytesToWrite
As Integer, ByRef lpNumberOfBytesWritten As Integer, ByRef lpOverlapped
As OVERLAPPED) As Integer
    End Function
    <DllImport("kernel32.dll")> Private Shared Function ReadFile(ByVal
hFile As Integer, ByVal Buffer As Byte(), ByVal nNumberOfBytesToRead As
Integer, ByRef lpNumberOfBytesRead As Integer, ByRef lpOverlapped As
OVERLAPPED) As Integer
    End Function
    <DllImport("kernel32.dll")> Private Shared Function
CreateFile(<MarshalAs(UnmanagedType.LPStr)> ByVal lpFileName As String,
```

```
ByVal dwDesiredAccess As Integer, ByVal dwShareMode As Integer, ByVal
lpSecurityAttributes As Integer, ByVal dwCreationDisposition As
Integer, ByVal dwFlagsAndAttributes As Integer, ByVal hTemplateFile As
Integer) As Integer
   End Function
   <DllImport("kernel32.dll")> Private Shared Function
CloseHandle(ByVal hObject As Integer) As Integer
   End Function
   <DllImport("kernel32.dll")> Private Shared Function
FormatMessage(ByVal dwFlags As Integer, ByVal lpSource As Integer,
ByVal dwMessageId As Integer, ByVal dwLanguageId As Integer,
<MarshalAs(UnmanagedType.LPStr)> ByVal lpBuffer As String, ByVal nSize
As Integer, ByVal Arguments As Integer) As Integer
   End Function
   <DllImport("kernel32.dll")> Public Shared Function
GetCommModemStatus(ByVal hFile As Int32, ByRef lpModemStatus As Int32)
As Boolean
   End Function
   <DllImport("kernel32.dll")> Private Shared Function
CreateEvent(ByVal lpEventAttributes As Int32, ByVal bManualReset As
Int32, ByVal bInitialState As Int32, <MarshalAs(UnmanagedType.LPStr)>
ByVal lpName As String) As Int32

   End Function
   <DllImport("kernel32.dll")> Private Shared Function GetLastError()
As Int32
   End Function
   <DllImport("kernel32.dll")> Private Shared Function
WaitForSingleObject(ByVal hHandle As Int32, ByVal dwMilliseconds As
Int32) As Int32
   End Function
   <DllImport("kernel32.dll")> Private Shared Function
GetOverlappedResult(ByVal hFile As Int32, ByRef lpOverlapped As
OVERLAPPED, ByRef lpNumberOfBytesTransferred As Int32, ByVal bWait As
Int32) As Int32

   End Function
   <DllImport("kernel32.dll")> Private Shared Function
SetCommMask(ByVal hFile As Int32, ByVal lpEvtMask As Int32) As Int32
   End Function


   Private Declare Function FormatMessage Lib "kernel32" Alias
    "FormatMessageA" (ByVal dwFlags As Int32, ByVal lpSource As Int32,
    ByVal dwMessageId As Int32, ByVal dwLanguageId As Int32,
    ByVal lpBuffer As StringBuilder, ByVal nSize As Int32, ByVal
    Arguments As Int32) As Int32


#End Region
#Region "Events"
   Public Event DataReceived(ByVal Source As Rs232, ByVal DataBuffer()
As Byte)
   Public Event TxCompleted(ByVal Source As Rs232)
   Public Event CommEvent(ByVal Source As Rs232, ByVal Mask As
EventMasks)
#End Region
```

```
Public Property Port() As Integer
   '=================================================
   '
   'Description:           Comunication Port
   'Created:               21/09/2001 - 11:25:49
   '
   '*Parameters Info*
   '
   'Notes:
   '=================================================
   Get
      Return miPort
   End Get
   Set(ByVal Value As Integer)
      miPort = Value
   End Set
End Property
Public Overridable Property Timeout() As Integer
   '=================================================
   '
   'Description:     Comunication timeout in seconds
   'Created:         21/09/2001 - 11:26:50
   '
   '*Parameters Info*
   '
   'Notes:
   '=================================================
   Get
      Return miTimeout
   End Get
   Set(ByVal Value As Integer)
      miTimeout = CInt(IIf(Value = 0, 500, Value))
      '// If Port is open updates it on the fly
      pSetTimeout()
   End Set
End Property
Public Property Parity() As DataParity
   '=================================================
   '
   'Description:     Comunication parity
   'Created:         21/09/2001 - 11:27:15
   '
   '*Parameters Info*
   '
   'Notes:
   '=================================================
   Get
      Return meParity
   End Get
   Set(ByVal Value As DataParity)
      meParity = Value
   End Set
End Property
```

```
Public Property StopBit() As DataStopBit
   '=================================================
   '
   'Description:     Comunication StopBit
   'Created:         21/09/2001 - 11:27:37
   '
   '*Parameters Info*
   '
   'Notes:
   '=================================================
   Get
      Return meStopBit
   End Get
   Set(ByVal Value As DataStopBit)
      meStopBit = Value
   End Set
End Property
Public Property BaudRate() As Integer
   '=================================================
   '
   'Description:     Comunication BaudRate
   'Created:         21/09/2001 - 11:28:00
   '
   '*Parameters Info*
   '
   'Notes:
   '=================================================
   Get
      Return miBaudRate
   End Get
   Set(ByVal Value As Integer)
      miBaudRate = Value
   End Set
End Property
Public Property DataBit() As Integer
   '=================================================
   '
   'Description:     Comunication DataBit
   'Created:         21/09/2001 - 11:28:20
   '
   '*Parameters Info*
   '
   'Notes:
   '=================================================
   Get
      Return miDataBit
   End Get
   Set(ByVal Value As Integer)
      miDataBit = Value
   End Set
End Property
```

```
   Public Property BufferSize() As Integer
      '====================================================
      '
      'Description:     Receive Buffer size
      'Created:         21/09/2001 - 11:33:05
      '
      '
      *Parameters Info*
      '
      '           Notes                    :
      '====================================================
      Get
         Return miBufferSize
      End Get
      Set(ByVal Value As Integer)
         miBufferSize = Value
      End Set
   End Property
   Public Overloads Sub Open()
      '====================================================
      '
      'Description:     Initializes and Opens comunication port
      'Created:         21/09/2001 - 11:33:40
      '
      '*Parameters Info*
      '
      'Notes:
      '====================================================
      '// Get Dcb block,Update with current data
      Dim uDcb As DCB, iRc As Int32
      '// Set working mode
      Dim iMode As Int32 = Convert.ToInt32(IIf(meMode =
Mode.Overlapped, FILE_FLAG_OVERLAPPED, 0))
      '// Initializes Com Port
      If miPort > 0 Then
         Try
            '// Creates a COM Port stream handle
            mhRS = CreateFile("COM" & miPort.ToString, _
            GENERIC_READ Or GENERIC_WRITE, 0, 0, _
            OPEN_EXISTING, iMode, 0)
            If mhRS <> -1 Then
               '// Clear all comunication errors
               Dim lpErrCode As Int32
               iRc = ClearCommError(mhRS, lpErrCode, 0&)
               '// Clears I/O buffers
               iRc = PurgeComm(mhRS, PurgeBuffers.RXClear Or
PurgeBuffers.TxClear)
               '// Gets COM Settings
               iRc = GetCommState(mhRS, uDcb)
               '// Updates COM Settings
               Dim sParity As String = "NOEM"
               sParity = sParity.Substring(meParity, 1)
               '// Set DCB State
               Dim sDCBState As String = String.Format("baud={0}
parity={1} data={2} stop={3}", miBaudRate, sParity, miDataBit,
CInt(meStopBit))
               iRc = BuildCommDCB(sDCBState, uDcb)
```

```
                uDcb.Parity = CByte(meParity)
                iRc = SetCommState(mhRS, uDcb)
                If iRc = 0 Then
                   Dim sErrTxt As String = pErr2Text(GetLastError())
                   Throw New CIOChannelException("Unable to set COM
state0" & sErrTxt)
                End If
                '// Setup Buffers (Rx,Tx)
                iRc = SetupComm(mhRS, miBufferSize, miBufferSize)
                '// Set Timeouts
                pSetTimeout()
             Else
                '// Raise Initialization problems
                Throw New CIOChannelException("Unable to open COM" &
miPort.ToString)
             End If
          Catch Ex As Exception
             '// Generica error
             Throw New CIOChannelException(Ex.Message, Ex)
          End Try
       Else
          '// Port not defined, cannot open
          Throw New ApplicationException("COM Port not defined,use Port
property to set it before invoking InitPort")
       End If
    End Sub
    Public Overloads Sub Open(ByVal Port As Integer, ByVal BaudRate As
Integer, ByVal DataBit As Integer, ByVal Parity As DataParity, ByVal
StopBit As DataStopBit, ByVal BufferSize As Integer)
       '=================================================
       '
       'Description:     Opens comunication port (Overloaded method)
       'Created:         21/09/2001 - 11:33:40
       '
       '*Parameters Info*
       '
       'Notes:
       '=================================================
       Me.Port = Port
       Me.BaudRate = BaudRate
       Me.DataBit = DataBit
       Me.Parity = Parity
       Me.StopBit = StopBit
       Me.BufferSize = BufferSize
       Open()
    End Sub
    Public Sub Close()
       '=================================================
       '
       'Description:     Close comunication channel
       'Created:         21/09/2001 - 11:38:00
       '
       '*Parameters Info*
       '
       'Notes:
       '=================================================
```

```
      If mhRS <> -1 Then
         CloseHandle(mhRS)
         mhRS = -1
      End If
   End Sub
   ReadOnly Property IsOpen() As Boolean
      '====================================================
      '
      'Description:     Returns Port Status
      'Created:         21/09/2001 - 11:38:51
      '
      '*Parameters Info*
      '
      'Notes:
      '====================================================
      Get
         Return CBool(mhRS <> -1)
      End Get
   End Property
   Public Overloads Sub Write(ByVal Buffer As Byte())
      '====================================================
      '
      'Description:     Transmit a stream
      'Created:         21/09/2001 - 11:39:51
      '
      '*Parameters Info*
      '     Buffer:     Array of Byte() to write
      'Notes:
      '====================================================
      Dim iBytesWritten, iRc As Integer
      '---------------------------------
      If mhRS = -1 Then
         Throw New ApplicationException("Please initialize and open
port before using this method")
      Else
         '// Transmit data to COM Port
         Try
            If meMode = Mode.Overlapped Then
               '// Overlapped write
               If pHandleOverlappedWrite(Buffer) Then
                  Throw New ApplicationException("Error in overllapped
write")
               End If
            Else
               '// Clears IO buffers
               PurgeComm(mhRS, PURGE_RXCLEAR Or PURGE_TXCLEAR)
               iRc = WriteFile(mhRS, Buffer, Buffer.Length,
iBytesWritten, Nothing)
               If iRc = 0 Then
                  Throw New ApplicationException("Write Error - Bytes
Written " & iBytesWritten.ToString & " of " & Buffer.Length.ToString)
               End If
            End If
         Catch Ex As Exception
            Throw
         End Try
      End If
```

```
    End Sub
    Public Overloads Sub Write(ByVal Buffer As String)
        '===================================================
        '
        'Description:     Writes a string to RS232
        'Created:         04/02/2002 - 8:46:42
        '
        '*Parameters Info*
        '
        'Notes:           24/05/2002 Fixed problem with ASCII Encoding
        '===================================================
        Dim oEncoder As New System.Text.ASCIIEncoding()
        Dim oEnc As Encoding = oEncoder.GetEncoding(1252)
        '-----------------------------------------------------------
        Dim aByte() As Byte = oEnc.GetBytes(Buffer)
        Me.Write(aByte)
    End Sub
    Public Function Read(ByVal Bytes2Read As Integer) As Integer
        '===================================================
        '
        'Description:           Read Bytes from Port
        'Created:               21/09/2001 - 11:41:17
        '
        '*Parameters Info*
        '     Bytes2Read:       Bytes to read from port
        '     Returns:          Number of readed chars
        '
        'Notes:
        '===================================================
        Dim iReadChars, iRc As Integer
        '-------------------------------------
        '// If Bytes2Read not specified uses Buffersize
        If Bytes2Read = 0 Then Bytes2Read = miBufferSize
        If mhRS = -1 Then
            Throw New ApplicationException("Please initialize and open
port before using this method")
        Else
            '// Get bytes from port
            Try
                '// Purge buffers
                'PurgeComm(mhRS, PURGE_RXCLEAR Or PURGE_TXCLEAR)
                '// Creates an event for overlapped operations
                If meMode = Mode.Overlapped Then
                    pHandleOverlappedRead(Bytes2Read)
                Else
                    '// Non overlapped mode
                    ReDim mabtRxBuf(Bytes2Read - 1)
                    iRc = ReadFile(mhRS, mabtRxBuf, Bytes2Read, iReadChars,
Nothing)
                    If iRc = 0 Then
                        '// Read Error
                        Throw New ApplicationException("ReadFile error " &
iRc.ToString)
                    Else
                        '// Handles timeout or returns input chars
                        If iReadChars < Bytes2Read Then
                            Throw New IOTimeoutException("Timeout error")
```

```
                  Else
                     mbWaitOnRead = True
                     Return (iReadChars)
                  End If
               End If
            End If
         Catch Ex As Exception
            '// Others generic erroes
            Throw New ApplicationException("Read Error: " & Ex.Message,
Ex)
         End Try
      End If
   End Function
   Overridable ReadOnly Property InputStream() As Byte()
      '=================================================
      '
      'Description:     Returns received data as Byte()
      'Created:         21/09/2001 - 11:45:06
      '
      '
      *Parameters Info*
      '
      'Notes:
      '=================================================
      Get
         Return mabtRxBuf
      End Get
   End Property
   Overridable ReadOnly Property InputStreamString() As String
      '=================================================
      '
      'Description:     Return a string containing received data
      'Created:         04/02/2002 - 8:49:55
      '
      '*Parameters Info*
      '
      'Notes:
      '=================================================
      Get
         Dim oEncoder As New System.Text.ASCIIEncoding()
         '-------------------------------------------------------------
         Return oEncoder.GetString(Me.InputStream)
      End Get
   End Property
   Public Sub ClearInputBuffer()
      '=================================================
      '
      'Description:     Clears Input buffer
      'Created:         21/09/2001 - 11:45:34
      '
      '*Parameters Info*
      '
      'Notes:           Gets all character until end of buffer
      '=================================================
      If Not mhRS = -1 Then
         PurgeComm(mhRS, PURGE_RXCLEAR)
      End If
```

```
    End Sub
    Public WriteOnly Property Rts() As Boolean
        '=====================================================
        '
        'Description:     Set/Resets RTS Line
        'Created:         21/09/2001 - 11:45:34
        '
        '*Parameters Info*
        '
        'Notes:
        '=====================================================
        Set(ByVal Value As Boolean)
            If Not mhRS = -1 Then
                If Value Then
                    EscapeCommFunction(mhRS, Lines.SetRts)
                Else
                    EscapeCommFunction(mhRS, Lines.ClearRts)
                End If
            End If
        End Set
    End Property
    Public WriteOnly Property Dtr() As Boolean
        '=====================================================
        '
        'Description:     Set/Resets DTR Line
        'Created:         21/09/2001 - 11:45:34
        '
        '*Parameters Info*
        '
        'Notes:
        '=====================================================
        Set(ByVal Value As Boolean)
            If Not mhRS = -1 Then
                If Value Then
                    EscapeCommFunction(mhRS, Lines.SetDtr)
                Else
                    EscapeCommFunction(mhRS, Lines.ClearDtr)
                End If
            End If
        End Set
    End Property
    Public ReadOnly Property ModemStatus() As ModemStatusBits
        '=====================================================
        '
        'Description:     Gets Modem status
        'Created:         28/02/2002 - 8:58:04
        '
        '*Parameters Info*
        '
        'Notes:
        '=====================================================
        Get
            If mhRS = -1 Then
                Throw New ApplicationException("Please initialize and open
port before using this method")
            Else
                '// Retrieve modem status
```

```
            Dim lpModemStatus As Int32
            If Not GetCommModemStatus(mhRS, lpModemStatus) Then
               Throw New ApplicationException("Unable to get modem
status")
            Else
               Return CType(lpModemStatus, ModemStatusBits)
            End If
         End If
      End Get
   End Property
   Public Function CheckLineStatus(ByVal Line As ModemStatusBits) As
Boolean
      '====================================================
      '
      'Description:     Check status of a Modem Line
      'Created:         28/02/2002 - 10:25:17
      '
      '*Parameters Info*
      '
      'Notes:
      '====================================================
      Return Convert.ToBoolean(ModemStatus And Line)
   End Function
   Public Property WorkingMode() As Mode
      '====================================================
      '
      'Description:     Set working mode (Overlapped/NonOverlapped)
      'Created:         28/02/2002 - 15:01:18
      '
      '*Parameters Info*
      '
      'Notes:
      '====================================================
      Get
         Return meMode
      End Get
      Set(ByVal Value As Mode)
         meMode = Value
      End Set
   End Property
   Public Overloads Sub AsyncWrite(ByVal Buffer() As Byte)
      '====================================================
      '
      'Description:     Write bytes using another thread,
      '                 TxCompleted raised when done
      'Created:         01/03/2002 - 12:00:56
      '
      '*Parameters Info*
      '
      'Notes:
      '====================================================
      If meMode <> Mode.Overlapped Then Throw New
ApplicationException("Async Methods allowed only when
WorkingMode=Overlapped")
      If mbWaitOnWrite = True Then Throw New
ApplicationException("Unable to send message because of pending
transmission.")
```

```
      mabtTmpTxBuf = Buffer
      moThreadTx = New Thread(AddressOf _W)
      moThreadTx.Start()
   End Sub
   Public Overloads Sub AsyncWrite(ByVal Buffer As String)
      '====================================================
      '
      'Description:     Overloaded Async Write
      'Created:         01/03/2002 - 12:00:56
      '
      '*Parameters Info*
      '
      'Notes:
      '====================================================
      Dim oEncoder As New System.Text.ASCIIEncoding()
      '------------------------------------------------------------
      Dim aByte() As Byte = oEncoder.GetBytes(Buffer)
      Me.AsyncWrite(aByte)
   End Sub
   Public Overloads Sub AsyncRead(ByVal Bytes2Read As Int32)
      '====================================================
      '
      'Description:     Read bytes using a different thread,
      '                 RxCompleted raised when done
      'Created:         01/03/2002 - 12:00:56
      '
      '*Parameters Info*
      '
      'Notes:
      '====================================================
      If meMode <> Mode.Overlapped Then Throw New
ApplicationException("Async Methods allowed only when
WorkingMode=Overlapped")
      miTmpBytes2Read = Bytes2Read
      moThreadTx = New Thread(AddressOf _R)
      moThreadTx.Start()
   End Sub

#Region "Finalize"
   Protected Overrides Sub Finalize()
      '====================================================
      '
      'Description:     Closes COM port if object is garbage collected
      '                 and still owns COM port reosurces
      '
      'Created:         27/05/2002 - 19:05:56
      '
      '*Parameters Info*
      '
      'Notes:
      '====================================================
      Try
         If Not mbDisposed Then Close()
      Finally
      End Try
   End Sub
#End Region
```

```
#Region "Thread related functions"
   Public Sub _W()
      '=================================================
      '
      'Description:     Method invoked by thread to perform an async
      '                 write
      'Created:         01/03/2002 - 12:23:08
      '
      '*Parameters Info*
      '
      'Notes:           Do not invoke this method from code
      '=================================================
      Write(mabtTmpTxBuf)
   End Sub
   Public Sub _R()
      '=================================================
      '
      'Description:     Method invoked by thread to perform an async
      '                 read
      'Created:         01/03/2002 - 12:23:08
      '
      '*Parameters Info*
      '
      'Notes:           Do not invoke this method from code
      '=================================================
      Dim iRet As Int32 = Read(miTmpBytes2Read)
   End Sub
#End Region

#Region "Private Routines"
   Private Sub pSetTimeout()
      '=================================================
      '
      'Description:     Set comunication timeouts
      'Created:         21/09/2001 - 11:46:40
      '
      '
      *Parameters Info*
      '
      'Notes:
      '=================================================
      Dim uCtm As COMMTIMEOUTS
      '// Set ComTimeout
      If mhRS = -1 Then
         Exit Sub
      Else
         '// Changes setup on the fly
         With uCtm
            .ReadIntervalTimeout = 0
            .ReadTotalTimeoutMultiplier = 0
            .ReadTotalTimeoutConstant = miTimeout
            .WriteTotalTimeoutMultiplier = 10
            .WriteTotalTimeoutConstant = 100
         End With
         SetCommTimeouts(mhRS, uCtm)
      End If
   End Sub
```

```
   Private Sub pHandleOverlappedRead(ByVal Bytes2Read As Int32)
      '====================================================
      '
      'Description:     Handles overlapped read
      'Created:         28/02/2002 - 16:03:06
      '
      '*Parameters Info*
      '
      'Notes:
      '====================================================
      Dim iReadChars, iRc, iRes, iLastErr As Int32
      '------------------------------------------------------------------
      muOverlapped.hEvent = CreateEvent(Nothing, 1, 0, Nothing)
      If muOverlapped.hEvent = 0 Then
         '// Can't create event
         Throw New ApplicationException("Error creating event for
overlapped read.")
      Else
         '// Ovellaped reading
         If mbWaitOnRead = False Then
            ReDim mabtRxBuf(Bytes2Read - 1)
            iRc = ReadFile(mhRS, mabtRxBuf, Bytes2Read, iReadChars,
muOverlapped)
            If iRc = 0 Then
               iLastErr = GetLastError()
               If iLastErr <> ERROR_IO_PENDING Then
                  Throw New ArgumentException("Overlapped Read Error: "
& pErr2Text(iLastErr))
               Else
                  '// Set Flag
                  mbWaitOnRead = True
               End If
            Else
               '// Read completed successfully
               RaiseEvent DataReceived(Me, mabtRxBuf)
            End If
         End If
      End If
      '// Wait for operation to be completed
      If mbWaitOnRead Then
         iRes = WaitForSingleObject(muOverlapped.hEvent, miTimeout)
         Select Case iRes
            Case WAIT_OBJECT_0
               '// Object signaled,operation completed
               If GetOverlappedResult(mhRS, muOverlapped, iReadChars,
0) = 0 Then
                  '// Operation error
                  iLastErr = GetLastError()
                  If iLastErr = ERROR_IO_INCOMPLETE Then
                     Throw New ApplicationException("Read operation
incomplete")
                  Else
                     Throw New ApplicationException("Read operation
error " & iLastErr.ToString)
                  End If
               Else
                  '// Operation completed
```

```
                RaiseEvent DataReceived(Me, mabtRxBuf)
                mbWaitOnRead = False
            End If
         Case WAIT_TIMEOUT
            Throw New IOTimeoutException("Timeout error")
         Case Else
            Throw New ApplicationException("Overlapped read error")
      End Select
   End If
End Sub
Private Function pHandleOverlappedWrite(ByVal Buffer() As Byte) As
Boolean
   '==================================================
   '
   'Description:     Handles overlapped Write
   'Created:         28/02/2002 - 16:03:06
   '
   '*Parameters Info*
   '
   'Notes:
   '==================================================
   Dim iBytesWritten, iRc, iLastErr, iRes As Integer, bErr As
Boolean
   '-------------------------------------------------------------------
   muOverlappedW.hEvent = CreateEvent(Nothing, 1, 0, Nothing)
   If muOverlappedW.hEvent = 0 Then
      '// Can't create event
      Throw New ApplicationException("Error creating event for
overlapped write.")
   Else
      '// Overllaped write
      PurgeComm(mhRS, PURGE_RXCLEAR Or PURGE_TXCLEAR)
      mbWaitOnRead = True
      iRc = WriteFile(mhRS, Buffer, Buffer.Length, iBytesWritten,
muOverlappedW)
      If iRc = 0 Then
         iLastErr = GetLastError()
         If iLastErr <> ERROR_IO_PENDING Then
            Throw New ArgumentException("Overlapped Read Error: " &
pErr2Text(iLastErr))
         Else
            '// Write is pending
            iRes = WaitForSingleObject(muOverlappedW.hEvent,
INFINITE)
            Select Case iRes
               Case WAIT_OBJECT_0
                  '// Object signaled,operation completed
                  If GetOverlappedResult(mhRS, muOverlappedW,
iBytesWritten, 0) = 0 Then
                     bErr = True
                  Else
                     '// Notifies Async tx completion,stops thread
                     mbWaitOnRead = False
                     RaiseEvent TxCompleted(Me)
                  End If
            End Select
         End If
```

```
            Else
                '// Wait operation completed immediatly
                bErr = False
            End If
        End If
        CloseHandle(muOverlappedW.hEvent)
        Return bErr
    End Function
    Private Function pErr2Text(ByVal lCode As Int32) As String
        '=================================================
        '
        'Description:     Translates API Code to text
        'Created:         01/03/2002 - 11:47:46
        '
        '*Parameters Info*
        '
        'Notes:
        '=================================================
        Dim sRtrnCode As New StringBuilder(256)
        Dim lRet As Int32
        '----------------------
        lRet = FormatMessage(&H1000, 0, lCode, 0, sRtrnCode, 256, 0)
        If lRet > 0 Then
            Return sRtrnCode.ToString
        Else
            Return "Error not found."
        End If

    End Function
    Private Sub pDispose() Implements IDisposable.Dispose
        '=================================================
        '
        'Description:     Handles correct class disposing Write
        'Created:         27/05/2002 - 19:03:06
        '
        '*Parameters Info*
        '
        'Notes:
        '=================================================
        If Not mbDisposed AndAlso mhRS <> -1 Then
            '// Closes Com Port releasing resources
            Try
                Close()
            Finally
                mbDisposed = True
                '// Suppress unnecessary Finalize overhead
                GC.SuppressFinalize(Me)
            End Try
        End If


    End Sub


#End Region
End Class
#End Region
```

```
#Region "Exceptions"
Public Class CIOChannelException : Inherits ApplicationException
      '=================================================
      '
      'Module:          CChannellException
      'Description:     Customized Channell Exception
      'Created:         17/10/2001 - 10:32:37
      '
      'Notes:           This exception is raised when NACK error found
      '=================================================
      Sub New(ByVal Message As String)
            MyBase.New(Message)
      End Sub
      Sub New(ByVal Message As String, ByVal InnerException As
Exception)
            MyBase.New(Message, InnerException)
      End Sub
End Class
Public Class IOTimeoutException : Inherits CIOChannelException
      '=================================================
      '
      'Description:     Timeout customized exception
      'Created:         28/02/2002 - 10:43:43
      '
      '*Parameters Info*
      '
      'Notes:
      '=================================================
      Sub New(ByVal Message As String)
            MyBase.New(Message)
      End Sub
      Sub New(ByVal Message As String, ByVal InnerException As
Exception)
            MyBase.New(Message, InnerException)
      End Sub
End Class

#End Region
```

The following code was developed at the MIT BioInstrumentation Lab. It relies on the previous section of code to provide serial communication support. It was written by Jeffrey Hoff and Brett Lockyer.

```
'vbcrlf is new line
Public Class Form1
    Inherits System.Windows.Forms.Form

    '// Private members
    Private miComPort As Integer
    Friend WithEvents btnOpenCom As System.Windows.Forms.Button
    Friend WithEvents btnCloseCom As System.Windows.Forms.Button
    Friend WithEvents btnTx As System.Windows.Forms.Button
    Friend WithEvents Label2 As System.Windows.Forms.Label
    Friend WithEvents Label3 As System.Windows.Forms.Label
    Friend WithEvents txtTx As System.Windows.Forms.TextBox
    Friend WithEvents txtRx As System.Windows.Forms.TextBox
    Friend WithEvents btnRx As System.Windows.Forms.Button
    Friend WithEvents Label5 As System.Windows.Forms.Label
    Friend WithEvents txtBytes2Read As System.Windows.Forms.TextBox
    Friend WithEvents GroupBox1 As System.Windows.Forms.GroupBox
    Friend WithEvents optCom2 As System.Windows.Forms.RadioButton
    Friend WithEvents optCom1 As System.Windows.Forms.RadioButton
    Friend WithEvents txtTimeout As System.Windows.Forms.TextBox
    Friend WithEvents Label4 As System.Windows.Forms.Label
    Friend WithEvents txtBaudrate As System.Windows.Forms.TextBox
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents chkAutorx As System.Windows.Forms.CheckBox
    Private WithEvents moRS232 As Rs232
    Private mlTicks As Long

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As
Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub
    Private components As System.ComponentModel.IContainer
```

```
    'Required by the Windows Form Designer

    'NOTE: The following procedure is required by the Windows Form
    '      Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents lbHex As System.Windows.Forms.ListBox
    Friend WithEvents Label6 As System.Windows.Forms.Label
    Friend WithEvents Timer1 As System.Windows.Forms.Timer
    Friend WithEvents cmdClear As System.Windows.Forms.Button
    Friend WithEvents cmdRxAlways As System.Windows.Forms.Button
    Friend WithEvents LblIdle As System.Windows.Forms.Label
    Friend WithEvents cmdLeft As System.Windows.Forms.Button
    Friend WithEvents cmdRight As System.Windows.Forms.Button
    Friend WithEvents cmdDown As System.Windows.Forms.Button
    Friend WithEvents cmdUp As System.Windows.Forms.Button
    Friend WithEvents cmdForward As System.Windows.Forms.Button
    Friend WithEvents cmdBack As System.Windows.Forms.Button
    Friend WithEvents cmdWarning As System.Windows.Forms.Button
    Friend WithEvents tmrBlink As System.Windows.Forms.Timer
    Friend WithEvents OptCom6 As System.Windows.Forms.RadioButton
    Friend WithEvents OptCom5 As System.Windows.Forms.RadioButton
    Friend WithEvents OptCom4 As System.Windows.Forms.RadioButton
    Friend WithEvents OptCom3 As System.Windows.Forms.RadioButton
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        Me.components = New System.ComponentModel.Container
        Me.btnTx = New System.Windows.Forms.Button
        Me.txtTimeout = New System.Windows.Forms.TextBox
        Me.txtTx = New System.Windows.Forms.TextBox
        Me.chkAutorx = New System.Windows.Forms.CheckBox
        Me.btnOpenCom = New System.Windows.Forms.Button
        Me.txtBaudrate = New System.Windows.Forms.TextBox
        Me.btnRx = New System.Windows.Forms.Button
        Me.txtBytes2Read = New System.Windows.Forms.TextBox
        Me.Label5 = New System.Windows.Forms.Label
        Me.Label1 = New System.Windows.Forms.Label
        Me.Label4 = New System.Windows.Forms.Label
        Me.Label3 = New System.Windows.Forms.Label
        Me.Label2 = New System.Windows.Forms.Label
        Me.txtRx = New System.Windows.Forms.TextBox
        Me.optCom1 = New System.Windows.Forms.RadioButton
        Me.btnCloseCom = New System.Windows.Forms.Button
        Me.optCom2 = New System.Windows.Forms.RadioButton
        Me.GroupBox1 = New System.Windows.Forms.GroupBox
        Me.OptCom6 = New System.Windows.Forms.RadioButton
        Me.OptCom5 = New System.Windows.Forms.RadioButton
        Me.OptCom4 = New System.Windows.Forms.RadioButton
        Me.OptCom3 = New System.Windows.Forms.RadioButton
        Me.lbHex = New System.Windows.Forms.ListBox
        Me.Label6 = New System.Windows.Forms.Label
        Me.Timer1 = New System.Windows.Forms.Timer(Me.components)
        Me.cmdClear = New System.Windows.Forms.Button
        Me.cmdRxAlways = New System.Windows.Forms.Button
        Me.LblIdle = New System.Windows.Forms.Label
        Me.cmdLeft = New System.Windows.Forms.Button
        Me.cmdRight = New System.Windows.Forms.Button
```

```
        Me.cmdDown = New System.Windows.Forms.Button
        Me.cmdUp = New System.Windows.Forms.Button
        Me.cmdForward = New System.Windows.Forms.Button
        Me.cmdBack = New System.Windows.Forms.Button
        Me.cmdWarning = New System.Windows.Forms.Button
        Me.tmrBlink = New System.Windows.Forms.Timer(Me.components)
        Me.GroupBox1.SuspendLayout()
        Me.SuspendLayout()
        '
        'btnTx
        '
        Me.btnTx.Enabled = False
        Me.btnTx.Location = New System.Drawing.Point(8, 192)
        Me.btnTx.Name = "btnTx"
        Me.btnTx.Size = New System.Drawing.Size(56, 24)
        Me.btnTx.TabIndex = 7
        Me.btnTx.Text = "Tx"
        '
        'txtTimeout
        '
        Me.txtTimeout.BorderStyle =
System.Windows.Forms.BorderStyle.FixedSingle
        Me.txtTimeout.Location = New System.Drawing.Point(96, 32)
        Me.txtTimeout.Name = "txtTimeout"
        Me.txtTimeout.Size = New System.Drawing.Size(65, 21)
        Me.txtTimeout.TabIndex = 3
        Me.txtTimeout.Text = "500"
        Me.txtTimeout.TextAlign =
System.Windows.Forms.HorizontalAlignment.Center
        '
        'txtTx
        '
        Me.txtTx.BorderStyle =
System.Windows.Forms.BorderStyle.FixedSingle
        Me.txtTx.CharacterCasing =
System.Windows.Forms.CharacterCasing.Upper
        Me.txtTx.Location = New System.Drawing.Point(7, 160)
        Me.txtTx.Name = "txtTx"
        Me.txtTx.Size = New System.Drawing.Size(193, 21)
        Me.txtTx.TabIndex = 6
        Me.txtTx.Text = "W"
        '
        'chkAutorx
        '
        Me.chkAutorx.Checked = True
        Me.chkAutorx.CheckState =
System.Windows.Forms.CheckState.Checked
        Me.chkAutorx.Location = New System.Drawing.Point(216, 160)
        Me.chkAutorx.Name = "chkAutorx"
        Me.chkAutorx.Size = New System.Drawing.Size(96, 32)
        Me.chkAutorx.TabIndex = 13
        Me.chkAutorx.Text = "Automatically receive bytes"
        Me.chkAutorx.Visible = False
        '
        'btnOpenCom
        '
        Me.btnOpenCom.Location = New System.Drawing.Point(211, 19)
```

```
        Me.btnOpenCom.Name = "btnOpenCom"
        Me.btnOpenCom.Size = New System.Drawing.Size(95, 27)
        Me.btnOpenCom.TabIndex = 1
        Me.btnOpenCom.Text = "Open COM Port"
        '
        'txtBaudrate
        '
        Me.txtBaudrate.BorderStyle =
System.Windows.Forms.BorderStyle.FixedSingle
        Me.txtBaudrate.Location = New System.Drawing.Point(96, 80)
        Me.txtBaudrate.Name = "txtBaudrate"
        Me.txtBaudrate.Size = New System.Drawing.Size(65, 21)
        Me.txtBaudrate.TabIndex = 5
        Me.txtBaudrate.Text = "57600"
        Me.txtBaudrate.TextAlign =
System.Windows.Forms.HorizontalAlignment.Center
        '
        'btnRx
        '
        Me.btnRx.Enabled = False
        Me.btnRx.Location = New System.Drawing.Point(8, 224)
        Me.btnRx.Name = "btnRx"
        Me.btnRx.Size = New System.Drawing.Size(56, 24)
        Me.btnRx.TabIndex = 10
        Me.btnRx.Text = "Rx"
        '
        'txtBytes2Read
        '
        Me.txtBytes2Read.BorderStyle =
System.Windows.Forms.BorderStyle.FixedSingle
        Me.txtBytes2Read.Location = New System.Drawing.Point(232, 120)
        Me.txtBytes2Read.Name = "txtBytes2Read"
        Me.txtBytes2Read.Size = New System.Drawing.Size(65, 21)
        Me.txtBytes2Read.TabIndex = 12
        Me.txtBytes2Read.Text = "1"
        Me.txtBytes2Read.TextAlign =
System.Windows.Forms.HorizontalAlignment.Center
        '
        'Label5
        '
        Me.Label5.Location = New System.Drawing.Point(232, 104)
        Me.Label5.Name = "Label5"
        Me.Label5.Size = New System.Drawing.Size(72, 14)
        Me.Label5.TabIndex = 11
        Me.Label5.Text = "Bytes to read"
        Me.Label5.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'Label1
        '
        Me.Label1.Location = New System.Drawing.Point(96, 16)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(75, 14)
        Me.Label1.TabIndex = 2
        Me.Label1.Text = "Timeout (ms)"
        Me.Label1.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
```

```
        '
        'Label4
        '
        Me.Label4.Location = New System.Drawing.Point(96, 64)
        Me.Label4.Name = "Label4"
        Me.Label4.Size = New System.Drawing.Size(59, 14)
        Me.Label4.TabIndex = 4
        Me.Label4.Text = "BaudRate"
        Me.Label4.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'Label3
        '
        Me.Label3.Location = New System.Drawing.Point(448, 16)
        Me.Label3.Name = "Label3"
        Me.Label3.Size = New System.Drawing.Size(82, 14)
        Me.Label3.TabIndex = 8
        Me.Label3.Text = "Received Data"
        '
        'Label2
        '
        Me.Label2.Location = New System.Drawing.Point(7, 144)
        Me.Label2.Name = "Label2"
        Me.Label2.Size = New System.Drawing.Size(82, 14)
        Me.Label2.TabIndex = 5
        Me.Label2.Text = "Data to Tx"
        '
        'txtRx
        '
        Me.txtRx.Anchor = CType((System.Windows.Forms.AnchorStyles.Top
Or System.Windows.Forms.AnchorStyles.Bottom),
System.Windows.Forms.AnchorStyles)
        Me.txtRx.BorderStyle =
System.Windows.Forms.BorderStyle.FixedSingle
        Me.txtRx.Location = New System.Drawing.Point(448, 32)
        Me.txtRx.Multiline = True
        Me.txtRx.Name = "txtRx"
        Me.txtRx.ScrollBars = System.Windows.Forms.ScrollBars.Vertical
        Me.txtRx.Size = New System.Drawing.Size(120, 158)
        Me.txtRx.TabIndex = 9
        Me.txtRx.Text = ""
        '
        'optCom1
        '
        Me.optCom1.Location = New System.Drawing.Point(8, 16)
        Me.optCom1.Name = "optCom1"
        Me.optCom1.Size = New System.Drawing.Size(64, 16)
        Me.optCom1.TabIndex = 0
        Me.optCom1.Text = "COM &1"
        '
        'btnCloseCom
        '
        Me.btnCloseCom.Enabled = False
        Me.btnCloseCom.Location = New System.Drawing.Point(211, 51)
        Me.btnCloseCom.Name = "btnCloseCom"
        Me.btnCloseCom.Size = New System.Drawing.Size(95, 27)
        Me.btnCloseCom.TabIndex = 2
```

```
Me.btnCloseCom.Text = "Close COM Port"
'
'optCom2
'
Me.optCom2.Location = New System.Drawing.Point(8, 32)
Me.optCom2.Name = "optCom2"
Me.optCom2.Size = New System.Drawing.Size(66, 16)
Me.optCom2.TabIndex = 1
Me.optCom2.Text = "COM &2"
'
'GroupBox1
'
Me.GroupBox1.Controls.Add(Me.txtTimeout)
Me.GroupBox1.Controls.Add(Me.Label4)
Me.GroupBox1.Controls.Add(Me.txtBaudrate)
Me.GroupBox1.Controls.Add(Me.Label1)
Me.GroupBox1.Controls.Add(Me.OptCom6)
Me.GroupBox1.Controls.Add(Me.OptCom5)
Me.GroupBox1.Controls.Add(Me.OptCom4)
Me.GroupBox1.Controls.Add(Me.OptCom3)
Me.GroupBox1.Controls.Add(Me.optCom2)
Me.GroupBox1.Controls.Add(Me.optCom1)
Me.GroupBox1.Location = New System.Drawing.Point(7, 11)
Me.GroupBox1.Name = "GroupBox1"
Me.GroupBox1.Size = New System.Drawing.Size(198, 133)
Me.GroupBox1.TabIndex = 0
Me.GroupBox1.TabStop = False
Me.GroupBox1.Text = "COM Setup"
'
'OptCom6
'
Me.OptCom6.Location = New System.Drawing.Point(8, 96)
Me.OptCom6.Name = "OptCom6"
Me.OptCom6.Size = New System.Drawing.Size(66, 16)
Me.OptCom6.TabIndex = 9
Me.OptCom6.Text = "COM &6"
'
'OptCom5
'
Me.OptCom5.Location = New System.Drawing.Point(8, 80)
Me.OptCom5.Name = "OptCom5"
Me.OptCom5.Size = New System.Drawing.Size(64, 16)
Me.OptCom5.TabIndex = 8
Me.OptCom5.Text = "COM &5"
'
'OptCom4
'
Me.OptCom4.Checked = True
Me.OptCom4.Location = New System.Drawing.Point(8, 64)
Me.OptCom4.Name = "OptCom4"
Me.OptCom4.Size = New System.Drawing.Size(66, 16)
Me.OptCom4.TabIndex = 7
Me.OptCom4.TabStop = True
Me.OptCom4.Text = "COM &4"
'
'OptCom3
'
```

```
        Me.OptCom3.Location = New System.Drawing.Point(8, 48)
        Me.OptCom3.Name = "OptCom3"
        Me.OptCom3.Size = New System.Drawing.Size(64, 16)
        Me.OptCom3.TabIndex = 6
        Me.OptCom3.Text = "COM &3"
        '
        'lbHex
        '
        Me.lbHex.BorderStyle =
System.Windows.Forms.BorderStyle.FixedSingle
        Me.lbHex.Location = New System.Drawing.Point(320, 32)
        Me.lbHex.Name = "lbHex"
        Me.lbHex.Size = New System.Drawing.Size(120, 158)
        Me.lbHex.TabIndex = 16
        '
        'Label6
        '
        Me.Label6.Location = New System.Drawing.Point(320, 16)
        Me.Label6.Name = "Label6"
        Me.Label6.Size = New System.Drawing.Size(114, 14)
        Me.Label6.TabIndex = 17
        Me.Label6.Text = "Received Data (Hex)"
        '
        'Timer1
        '
        Me.Timer1.Interval = 500
        '
        'cmdClear
        '
        Me.cmdClear.Location = New System.Drawing.Point(464, 208)
        Me.cmdClear.Name = "cmdClear"
        Me.cmdClear.Size = New System.Drawing.Size(72, 24)
        Me.cmdClear.TabIndex = 28
        Me.cmdClear.Text = "Clear"
        '
        'cmdRxAlways
        '
        Me.cmdRxAlways.Location = New System.Drawing.Point(96, 208)
        Me.cmdRxAlways.Name = "cmdRxAlways"
        Me.cmdRxAlways.Size = New System.Drawing.Size(104, 24)
        Me.cmdRxAlways.TabIndex = 29
        Me.cmdRxAlways.Text = "Auto Rx is Off"
        '
        'LblIdle
        '
        Me.LblIdle.BorderStyle =
System.Windows.Forms.BorderStyle.FixedSingle
        Me.LblIdle.Location = New System.Drawing.Point(336, 208)
        Me.LblIdle.Name = "LblIdle"
        Me.LblIdle.Size = New System.Drawing.Size(88, 16)
        Me.LblIdle.TabIndex = 30
        Me.LblIdle.Text = "IDLETIME"
        Me.LblIdle.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'cmdLeft
        '
```

```
Me.cmdLeft.BackColor = System.Drawing.SystemColors.Control
Me.cmdLeft.Location = New System.Drawing.Point(80, 320)
Me.cmdLeft.Name = "cmdLeft"
Me.cmdLeft.Size = New System.Drawing.Size(56, 32)
Me.cmdLeft.TabIndex = 47
Me.cmdLeft.Text = "Left"
'
'cmdRight
'
Me.cmdRight.BackColor = System.Drawing.SystemColors.Control
Me.cmdRight.Location = New System.Drawing.Point(224, 320)
Me.cmdRight.Name = "cmdRight"
Me.cmdRight.Size = New System.Drawing.Size(56, 32)
Me.cmdRight.TabIndex = 48
Me.cmdRight.Text = "Right"
'
'cmdDown
'
Me.cmdDown.BackColor = System.Drawing.SystemColors.Control
Me.cmdDown.Location = New System.Drawing.Point(152, 360)
Me.cmdDown.Name = "cmdDown"
Me.cmdDown.Size = New System.Drawing.Size(56, 32)
Me.cmdDown.TabIndex = 49
Me.cmdDown.Text = "Down"
'
'cmdUp
'
Me.cmdUp.BackColor = System.Drawing.SystemColors.Control
Me.cmdUp.Location = New System.Drawing.Point(152, 288)
Me.cmdUp.Name = "cmdUp"
Me.cmdUp.Size = New System.Drawing.Size(56, 32)
Me.cmdUp.TabIndex = 50
Me.cmdUp.Text = "Up"
'
'cmdForward
'
Me.cmdForward.BackColor = System.Drawing.SystemColors.Control
Me.cmdForward.Location = New System.Drawing.Point(312, 296)
Me.cmdForward.Name = "cmdForward"
Me.cmdForward.Size = New System.Drawing.Size(64, 32)
Me.cmdForward.TabIndex = 53
Me.cmdForward.Text = "Forward"
'
'cmdBack
'
Me.cmdBack.BackColor = System.Drawing.SystemColors.Control
Me.cmdBack.Location = New System.Drawing.Point(312, 344)
Me.cmdBack.Name = "cmdBack"
Me.cmdBack.Size = New System.Drawing.Size(64, 32)
Me.cmdBack.TabIndex = 52
Me.cmdBack.Text = "Back"
'
'cmdWarning
'
Me.cmdWarning.BackColor = System.Drawing.Color.Tomato
Me.cmdWarning.Location = New System.Drawing.Point(400, 312)
Me.cmdWarning.Name = "cmdWarning"
```

```
        Me.cmdWarning.Size = New System.Drawing.Size(96, 48)
        Me.cmdWarning.TabIndex = 51
        Me.cmdWarning.Text = "Warning!"
        '
        'tmrBlink
        '
        Me.tmrBlink.Interval = 300
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 14)
        Me.ClientSize = New System.Drawing.Size(578, 416)
        Me.Controls.Add(Me.cmdLeft)
        Me.Controls.Add(Me.cmdRight)
        Me.Controls.Add(Me.cmdDown)
        Me.Controls.Add(Me.cmdUp)
        Me.Controls.Add(Me.cmdForward)
        Me.Controls.Add(Me.cmdBack)
        Me.Controls.Add(Me.cmdWarning)
        Me.Controls.Add(Me.LblIdle)
        Me.Controls.Add(Me.cmdRxAlways)
        Me.Controls.Add(Me.cmdClear)
        Me.Controls.Add(Me.txtBytes2Read)
        Me.Controls.Add(Me.txtRx)
        Me.Controls.Add(Me.txtTx)
        Me.Controls.Add(Me.Label6)
        Me.Controls.Add(Me.lbHex)
        Me.Controls.Add(Me.chkAutorx)
        Me.Controls.Add(Me.GroupBox1)
        Me.Controls.Add(Me.Label5)
        Me.Controls.Add(Me.Label3)
        Me.Controls.Add(Me.btnRx)
        Me.Controls.Add(Me.Label2)
        Me.Controls.Add(Me.btnTx)
        Me.Controls.Add(Me.btnCloseCom)
        Me.Controls.Add(Me.btnOpenCom)
        Me.Font = New System.Drawing.Font("Tahoma", 8.25!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
CType(0, Byte))
        Me.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog
        Me.MaximizeBox = False
        Me.Name = "Form1"
        Me.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen
        Me.Text = "VB.NET Tactile Vest GUI"
        Me.GroupBox1.ResumeLayout(False)
        Me.ResumeLayout(False)

    End Sub

#End Region


    Private Sub btnOpenCom_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnOpenCom.Click
```

```
        moRS232 = New Rs232
        Try
            '// Setup parameters
            With moRS232
                .Port = miComPort
                .BaudRate = CInt(txtBaudrate.Text)
                .DataBit = 8
                .StopBit = Rs232.DataStopBit.StopBit_1
                .Parity = Rs232.DataParity.Parity_None
                .Timeout = CInt(txtTimeout.Text)
                .WorkingMode = CType(Rs232.Mode.NonOverlapped,
Rs232.Mode)
            End With
            '// Initializes port
            moRS232.Open()
            '// Set state of RTS / DTS
            moRS232.Dtr = True
            moRS232.Rts = True
        Catch Ex As Exception
            MessageBox.Show(Ex.Message, "Connection Error",
MessageBoxButtons.OK)
        Finally
            btnCloseCom.Enabled = moRS232.IsOpen
            btnOpenCom.Enabled = Not moRS232.IsOpen
            btnTx.Enabled = moRS232.IsOpen
            btnRx.Enabled = moRS232.IsOpen
        End Try
    End Sub

    Private Sub btnCloseCom_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnCloseCom.Click
        moRS232.Close()
        btnCloseCom.Enabled = moRS232.IsOpen
        btnOpenCom.Enabled = Not moRS232.IsOpen
        btnTx.Enabled = moRS232.IsOpen
        btnRx.Enabled = moRS232.IsOpen
    End Sub

    ' This function attempts to open the passed Comm Port. If it is
    '   available, it returns True, else it returns False. To determine
    '   availability a Try-Catch block is used.
    Private Function IsPortAvailable(ByVal ComPort As Integer) As
Boolean
        Try
            moRS232.Open(ComPort, 57600, 8,
Rs232.DataParity.Parity_None, _
                Rs232.DataStopBit.StopBit_1, 4096)
            ' If it makes it to here, then the Comm Port is available.
            moRS232.Close()
            Return True
        Catch
            ' If it gets here, then the attempt to open the Comm Port
            '   was unsuccessful.
            Return False
        End Try
    End Function
```

```
    Private Sub Button1_Click_1(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnTx.Click
        moRS232.Write(txtTx.Text)
        '// Clears Rx textbox
        txtRx.Text = String.Empty
        txtRx.Refresh()
        lbHex.Items.Clear()
    End Sub

    Private Sub Form1_Closing(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
        If Not moRS232 Is Nothing Then
            If moRS232.IsOpen Then moRS232.Close()
        End If
    End Sub

    Private Sub BtnRx_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnRx.Click
        Try
            moRS232.Read(CInt(txtBytes2Read.Text))
            txtRx.Text = moRS232.InputStreamString
            txtRx.ForeColor = Color.Black
            txtRx.BackColor = Color.White
            '// Fills listbox with hex values
            Dim aBytes As Byte() = moRS232.InputStream
            Dim iPnt As Int32
            For iPnt = 0 To aBytes.Length - 1
                lbHex.Items.Add(iPnt.ToString & ControlChars.Tab &
String.Format("0x{0}", aBytes(iPnt).ToString("X")))
            Next
        Catch Ex As Exception
            txtRx.BackColor = Color.Red
            txtRx.ForeColor = Color.White
            txtRx.Text = "Error occurred " & Ex.Message & "  data
fetched: " & moRS232.InputStreamString
        End Try
    End Sub

    Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
        Me.Close()
    End Sub

    Private Sub optCom1_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles optCom1.CheckedChanged,
optCom2.CheckedChanged, OptCom3.CheckedChanged, OptCom4.CheckedChanged,
OptCom5.CheckedChanged, OptCom6.CheckedChanged
        If sender Is optCom1 Then
            miComPort = 1
        ElseIf sender Is optCom2 Then
            miComPort = 2
        ElseIf sender Is OptCom3 Then
            miComPort = 3
        ElseIf sender Is OptCom4 Then
            miComPort = 4
        ElseIf sender Is OptCom5 Then
            miComPort = 5
```

```
        ElseIf sender Is OptCom6 Then
            miComPort = 6
        End If
    End Sub

    Private Sub btnCheck_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs)
        '================================================
        '
        'Description:   Check passed status line
        'Created:       28/02/2002 - 10:35:54
        '
        '*Parameters Info*
        '
        'Notes:
        '================================================
        If Not moRS232 Is Nothing Then
            Dim bState As Boolean
            Dim output As String
            bState =
moRS232.CheckLineStatus(Rs232.ModemStatusBits.ClearToSendOn)
            output = "Clear to Send: " & IIf(bState, "On",
"Off").ToString & vbCrLf

            bState =
moRS232.CheckLineStatus(Rs232.ModemStatusBits.DataSetReadyOn)
            output &= "Dataset Ready: " & IIf(bState, "On",
"Off").ToString & vbCrLf

            bState =
moRS232.CheckLineStatus(Rs232.ModemStatusBits.RingIndicatorOn)
            output &= "Ring Indicator: " & IIf(bState, "On",
"Off").ToString & vbCrLf

            bState =
moRS232.CheckLineStatus(Rs232.ModemStatusBits.CarrierDetect)
            output &= "Carrier Detect: " & IIf(bState, "On",
"Off").ToString & vbCrLf
        End If
    End Sub

    Private Sub moRS232_DataReceived(ByVal Source As Rs232, ByVal
DataBuffer() As Byte) Handles moRS232.DataReceived
        Dim lTicks As Long = DateTime.Now.Ticks
        txtRx.Text = Source.InputStreamString
        txtRx.ForeColor = Color.Black
        txtRx.BackColor = Color.White
        '// Fills listbox with hex values
        Dim aBytes As Byte() = Source.InputStream
        Dim iPnt As Int32
        For iPnt = 0 To aBytes.Length - 1
            lbHex.Items.Add(iPnt.ToString & ControlChars.Tab &
String.Format("0x{0}", aBytes(iPnt).ToString("X")))
        Next
    End Sub
```

```
    Private Sub btnAsyncTx_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs)
        '// Clears Rx textbox
        txtRx.Text = String.Empty
        txtRx.Refresh()
        lbHex.Items.Clear()
        mlTicks = DateTime.Now.Ticks
        moRS232.AsyncWrite(txtTx.Text)
        If chkAutorx.Checked Then btnAsync_Click(Nothing, Nothing)
    End Sub

    Private Sub btnAsync_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs)
        Try
            moRS232.AsyncRead(CInt(txtBytes2Read.Text))
            Dim lTicks As Long = DateTime.Now.Ticks
        Catch Ex As Exception
            txtRx.BackColor = Color.Red
            txtRx.ForeColor = Color.White
            txtRx.Text = "Error occurred " & Ex.Message & "  data
fetched: " & moRS232.InputStreamString
        End Try
    End Sub

    Private Sub cmdClear_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles cmdClear.Click
        txtRx.Text = ""
        IdleTime = 0
        lbHex.Items.Clear()
    End Sub

    Private Sub cmdRxAlways_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles cmdRxAlways.Click
        If Timer1.Enabled = True Then
            Timer1.Enabled = False
            cmdRxAlways.Text = "Auto Rx is Off"
        Else
            Timer1.Enabled = True
            cmdRxAlways.Text = "Auto Rx is On"
        End If
    End Sub

    Dim NewLine As Boolean
    Dim IdleTime As Integer

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        Try
            moRS232.Read(CInt(txtBytes2Read.Text))
            txtRx.Text += moRS232.InputStreamString
            NewLine = True
            IdleTime = 0
            LblIdle.Text = "Active"
            Timer1.Interval = 100
            txtRx.ForeColor = Color.Black
            txtRx.BackColor = Color.White
            '// Fills listbox with hex values
```

```
            Dim aBytes As Byte() = moRS232.InputStream
            Dim iPnt As Int32
            For iPnt = 0 To aBytes.Length - 1
                lbHex.Items.Add(iPnt.ToString & ControlChars.Tab &
String.Format("0x{0}", aBytes(iPnt).ToString("X")))
            Next
        Catch Ex As Exception
            IdleTime += Timer1.Interval
            LblIdle.Text = (IdleTime / 1000).ToString
            If NewLine = True Then
                txtRx.Text &= vbCrLf
                NewLine = False
                Timer1.Interval = 500
            End If
        End Try
    End Sub

    Private Sub TextBox1_KeyPress(ByVal sender As System.Object, ByVal
e As System.Windows.Forms.KeyPressEventArgs)
    End Sub

    Private Sub TextBox1_KeyUp(ByVal sender As System.Object, ByVal e
As System.Windows.Forms.KeyEventArgs)
        Select Case e.KeyCode
            Case Keys.Left
                cmdLeft_Click(sender, e)
            Case Keys.Right
                cmdRight_Click(sender, e)
            Case Keys.Up
                cmdUp_Click(sender, e)
            Case Keys.Down
                cmdDown_Click(sender, e)
            Case Keys.F
                cmdForward_Click(sender, e)
            Case Keys.B
                cmdBack_Click(sender, e)
            Case Keys.Space
                cmdWarning_Click(sender, e)

        End Select
    End Sub
    Private Sub cmdUp_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdUp.Click
        txtTx.Text = "U"
        Button1_Click_1(sender, e)
    End Sub
    Private Sub cmdDown_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdDown.Click
        txtTx.Text = "D"
        Button1_Click_1(sender, e)
    End Sub
    Private Sub cmdLeft_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdLeft.Click
        txtTx.Text = "L"
        Button1_Click_1(sender, e)
    End Sub
```

```
    Private Sub cmdRight_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles cmdRight.Click
        txtTx.Text = "R"
        Button1_Click_1(sender, e)
    End Sub
    Private Sub cmdForward_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles cmdForward.Click
        txtTx.Text = "F"
        Button1_Click_1(sender, e)
    End Sub
    Private Sub cmdBack_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdBack.Click
        txtTx.Text = "B"
        Button1_Click_1(sender, e)
    End Sub
    Private Sub cmdWarning_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles cmdWarning.Click
        txtTx.Text = "W"
        Button1_Click_1(sender, e)
    End Sub
    Private Sub cboStatusLine_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs)
    End Sub
    Private Sub TextBox1_TextChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs)
    End Sub
    Private Sub LblStatus_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs)
    End Sub
    Private Sub txtBaudrate_TextChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles txtBaudrate.TextChanged
    End Sub
    Private Sub lbAsync_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs)
    End Sub
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    End Sub
    Private Sub lbHex_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
lbHex.SelectedIndexChanged
    End Sub
End Class
```