A SENIOR DESIGN REPORT

SUBMITTED TO PROFESSOR

DENNIS A. SILAGE

AND THE

SENIOR DESIGN FACULTY OF

THE COLLEGE OF ENGINEERING

TEMPLE UNIVERSITY

DIGITAL WIRELESS DATA COMMUNICATION WITH A SEMI-AUTONOMOUS ROVER

PROJECT NO. EE-2

TEAM MEMBERS: **STEVE HERMAN**

JOHN P. FALCONE

JOHN F. DESSINO

KEMI AHSEBU

Submitted in partial fulfillment of the requirements for Senior Design Project, ENGR W361/362

April 28, 2004

ABSTRACT
Senior Design Team # EE-2

DIGITAL WIRELESS DATA COMMUNICATION WITH A SEMI-AUTONOMOUS ROVER

Team Members: Steve Herman
John P. Falcone, KB3KDM
John F. Dessino
Kemi Ashebu

Faculty Advisor: Dennis A. Silage, PhD., K3DS

Modern space exploration is rapidly advancing, and there is a growing need for technology to collect and transmit data in places that may be unsafe for human travel. Communication, which is used for commands and for transmission of data, is a necessary part of the development of this technology. The aim of the project was to design and build a planetary rover, an operator-controlled base station interface for control of the rover, and protocols that govern the wireless communication between the base station and the rover.

An operator controls the movement and data acquisition of the robot from a base station. The rover is equipped with a number of sensors used to evaluate its surroundings. It collects data and sends it to the base station via the amateur radio 144 MHz frequency band. The base station receives and interprets the data into an appropriate form able to be easily understood by the operator. The operator uses the data feedback for observation of the environment and further control of the rover.

Table of Contents

SECTION 1: BACKGROUND 1

SECTION 2: LITERATURE REVIEW 2

SECTION 3: DESIGN OBJECTIVE 5

SECTION 4: DESIGN PROCESS 6

4.1 RESEARCH 6
4.2 PLATFORM AND HARDWARE 6
4.3 SOFTWARE DEVELOPMENT 7
4.3.1 Rover Software 7
Motor control 8
Sensor Suite Interfacing 9
4.3.2 Base station 10
4.4 HARD-WIRED COMMUNICATION 10
4.5 WIRELESS COMMUNICATION 11
4.6 TESTING 12

SECTION 5: RESULTS 14

5.1 ROVER HARDWARE 14
5.1.1 What is the Rover? 14
5.1.2 The LogicFlex 15
5.1.3 Power Distribution 16
Batteries 16
Voltage Regulators 17
Protection Circuits 18
5.1.4 Driving the Rover 19
H-Bridge 19
Opto-Isolators 19
5.1.5 Sensors 20
Tilt Sensor 21
Compass 21
Accelerometer 22
Camera 23
5.1.6 Rover Communications Hardware 23
Rover Modem 23
Rover Hand Transceiver 27
5.2 BASE STATION HARDWARE 27
5.2.1 Base Station Transceiver 27
5.2.2 Base Station Modem 28
5.2.3 Base Station PC 29
5.3 ROVER SOFTWARE 30
5.3.1 Creating the Borland C Project 30
5.3.2 Mobility 32
5.3.3 Data Acquisition Routines 33
5.3.4 Autonomous Routines 34
5.3.5 Communication Routine 35
5.4 BASE STATION SOFTWARE 36
5.4.1 Joystick Control 37
5.4.2 Sensor Displays 38
5.4.3 Communication Routine 41
5.4.4 Autonomous Routine 42

5.5 PROTOCOLS 43
5.5.1 Introduction *43*
5.5.2 Application Layer *43*
5.5.3 Coding Layer *47*
5.5.4 Physical Layer *48*
5.6 DEVIATIONS FROM PROPOSED DESIGN 51
5.6.1 Opto Isolators *51*
5.6.2 Battery Issues *51*

SECTION 6: SAFETY CONSIDERATIONS 52

6.1 AMATEUR RADIO RULES AND REGULATIONS 52
6.2 ELECTROSTATIC DISCHARGE (ESD) CONSIDERATIONS 52

SECTION 7: CONCLUSION 54

SECTION 8: TIME SCHEDULE 56

SECTION 9: BUDGET 58

SECTION 10: WORKS CITED 59

SECTION 11: BIBLIOGRAPHY 60

SECTION 12: APPENDIX A – SCHEMATIC DIAGRAMS 64

SECTION 13: APPENDIX B – PINOUTS AND WIRING 68

SECTION 14: APPENDIX C – PROGRAM CODE 76

SECTION 15: APPENDIX D – RESUMÉS 129

List of Figures

Figure 4.1: Project Design Flow to guide the team to completion of the project 13
Figure 5.1: Completed version of the Rover 14
Figure 5.2: LogicFlex as provided by JK Microsystems 15
Figure 5.3: Schematic diagram of circuit built to adjust voltage output of the LM317 voltage regulator 18
Figure 5.4: Fredrics Electronic Inclinometer (Tilt Sensor) 21
Figure 5.5: TCM2 Electronic Compass as purchased from pnicorp. 22
Figure 5.6: Crossbow Tri-axial accelerometer 23
Figure 5.7: PC Electronics ATV camera and transmitter 23
Figure 5.8: The MX614 Modem IC 24
Figure 5.9: The MAX232 IC 26
Figure 5.10: Pinout Diagram for the microphone connector on the Kenwood TM-721A 28
Figure 5.11: The LM324 IC 29
Figure 5.12: Borland Target Expert window 31
Figure 5.13: Borland project tree for ROA version 11 32
Figure 5.14: RIA main form 39
Figure 5.15: Compass display 40
Figure 5.16: Inclinometer display 40
Figure 5.17: Accelerometer display 41
Figure 5.18: Protocol stack 43
Figure 5.19: Base station state diagram 44
Figure 5.20: Rover state diagram 44
Figure 5.21: Block Segmentation 48
Figure 5.22: Message Representation 48
Figure 5.23: Physical Layer Block Diagram 50
Figure 8.1: Project time schedule 57
Figure 12.1: Schematic of the circuit for delivering power to the various components of the Rover 64
Figure 12.2: Schematic of the circuit that drives the motors 65
Figure 12.3: Schematic of the modem on the Rover 66
Figure 12.4: Schematic of the modem at the base station 67
Figure 13.1: Diagram of connector locations and devices on board 68

List of Tables

Table 4.1: Status of the motor determined by an instantaneous PWM signal to each H-Bridge input 8
Table 5.1: Logic frequency modes Table 5.2: Baud settings 25
Table 5.3: Transmitting and Receiving modes resulting from the DTR line 25
Table 5.4: Logic levels for different standards 26
Table 5.5: Commands and responses 46
Table 9.1: Project budget 58
Table 13.1: Pinout for the Cable connecting the Protoboard to the LogicFlex Parallel and ADC Ports 69
Table 13.2: Pinout for the Serial Port on LogicFlex to the Protoboard 70
Table 13.3: Pinout for the Compass to the Protoboard 70
Table 13.4: Pinout for the Camera and Transmitter to the Protoboard 70
Table 13.5: Pinout for the Tilt Sensor to the protoboard 71
Table 13.6: Pinout for the Accelerometer to the protoboard 71
Table 13.7: Pinout for the LogicFlex's Driver Ports to the protoboard 71
Table 13.8: Pinout for the LogicFlex power connector to the Protoboard 72
Table 13.9: Pinout for connecting the turning motor to the Protoboard 72
Table 13.10: Pinout for connecting the drive motor to the protoboard 72
Table 13.11: Power Connector Pinouts 72
Table 13.12: Port and Pin assignments for the LogicFlex 73
Table 13.13: Pinout for connecting Base Station PC to Modem 74
Table 13.14: Pinout for connecting Base Station Modem Serial I/O 74
Table 13.15: Pinout for connecting Base Station Modem Transceiver I/O 75

Section 1: Background

In this modern age wireless technology has enhanced the functionality of mobile robots allowing them to be controlled remotely via a personal computer (PC) or an electronic control device. The purpose of this project was to construct a robot capable of performing wireless communication with a PC as part of a base station. This type of mobile robot can be used to navigate distant terrain and to collect and transmit information about its surroundings. The sensors on our robot suit the environment at Temple University. However, the variety of sensors on this robot can be interchanged to appropriately suit other environments. For example, this technology can be applied as a planetary rover deployed in a mission to Mars. It can also be used for mining, tunnel work, rescue teams, swat teams, or any other application where it is dangerous or too expensive to directly send human beings.

Section 2: Literature Review

This project was partially modeled after the Mars Pathfinder project, which was one of the first NASA Discovery class missions. The main intent of NASA's project was to land a rover on the surface of Mars and collect data about its surroundings. The rover, named Sojourner, landed on the surface of Mars on July 4th, 1997. Commands were given to the rover, during its seven-day journey, from a base station communicating on the UHF band. The Sojourner carried a number of instruments, including a stereoscopic imager with spectral filters on an extendible mast, an Alpha Proton X-Ray Spectrometer, and an Atmospheric Structure Instrument/Meteorology package. These instruments provided scientific information about the surface of Mars. Examination of the different surface materials allowed for scientific study of the early evolution of the crust and of the environments and conditions that have existed on Mars. Rover exploration, such as that with the Pathfinder project, is a practical means of obtaining data in places unfit or undesirable for human data collection. They are a relatively low cost means of obtaining important data.

Currently, the NASA Spirit and Opportunity rovers are on the surface of Mars. These rovers are far more complex than the Sojourner rover of the original Pathfinder project. Each rover is "sort of the mechanical equivalent of a geologist walking the surface of Mars" (4). The mission of these rovers also bears some resemblance to the objective of this project. The rovers collect data about their surroundings and communicate with a base station. They carry an array of sensors, but also carry equipment for drilling into rock and analyzing soil. The Spirit and Opportunity also have cameras that provide 360-degree views of their surroundings.

Rovers have been employed in a number of roles. In addition to being useful to send into space at relatively low cost in comparison to manned missions, rovers are also used here on the planet Earth. Police Special Weapons and Tactical (SWAT) teams are responsible for handling high-risk tactical situations involving barricaded suspects, hostage situations, and the serving of warrants. The primary goal of the team is the successful conclusion of high-risk situations through the use of specially equipped and highly trained personnel without injury or loss of life to citizens, suspects, or police officers. Much research concerning the use of rovers in these high-risk situations is currently being conducted. Rovers could take officers out of harms way and provide the surveillance necessary to make the operation more likely to be successful. Many SWAT teams currently employ robots such as the Remotec's Andros robot series. They are capable of climbing stairs, performing two-way audio communication for roles in hostage negotiating, and providing video surveillance. These rovers are similar in concept to the rover conceived in this project. It requires control by an operator communicating with it wirelessly from a base station (5). Rather than being a sensor platform, however, these rovers are usually intended for surveillance, audio conversations, or even disabling vehicles with special equipment that they carry. The rover of this project could also be employed in a surveillance role as it carries a fast scan camera and amateur television (ATV) transmitter for live video transmission.

The set of protocols that define the rules of communication is an important aspect of digital wireless communications. The definition of the Satlink communication protocol written by Adi Rustanbegovic of Sutron Inc. explains the details of how the Satlink meteorological measurement device communicates with a satellite. Rustanbegovic describes a client-server relationship, where only one party will be able to initiate communication,

between devices (6). This is a similar relation necessary to the rover created in this project, where the base station was the client and the rover was the server. Only the base station could initiate communication over the half-duplex channel. In addition, the document specifies such things as timing considerations, handling of retries and errors, packet contents, packet arrangement, and error detection details. The protocols here, however, only define error detection techniques and requests for retransmission. The communication protocols established in this project employed error detection and correction. The Satlink protocol definition is a working set of protocols between a client and a server. It served as a guide to developing protocols. Another document that includes protocols for error control is "A Mobitex wireless modem implementation"(1). This paper provided templates that for error coding, interleaving, and scrambling (1).

The user's manuals and the datasheets for the electronic components comprise some of the necessary literature for implementation of the project. Each electronic component and integrated circuit (IC) has a datasheet or user's manual concerning the use of the device. This information was necessary for interfacing with each device. The user's manual for the JKMicro LogicFlex embedded 386 microcomputer and its peripherals was essential, since the LogicFlex was the central processing unit for the rover. In addition, a number of programming reference books for Visual Basic, C, and C++ were highly useful because programming was a large part of this project.

Section 3: Design Objective

The main objective of this project was to create a rover that was controlled wirelessly and was capable of providing information about its surroundings. The intent was to demonstrate the usefulness of such a rover. The size of the rover was to remain relatively small to maneuver around obstacles and traverse many different types of terrains. Cost effectiveness and versatility were also objectives in the design. Many off the shelf parts were to be used to aid in the design implementation. The array of sensors could be interchanged for use in other environments. The control station was to be user friendly, employing a Windows based graphical user interface (GUI).

Section 4: Design Process

The design process of the semiautonomous rover, the base station, and the protocols for communication between them consisted of a layered approach. As the implementation of each design layer was completed, the process of building the parts of the project outlined in the next layer was begun. Figure 4.1 at the end of this section graphically depicts the design process.

4.1 Research

The first layer was to research all of the devices and parts used in the design. All of the wiring diagrams and pin configurations are well documented and appear in Appendices A and B. Although research was the first step, it was an ongoing process, as it continued throughout the design process.

4.2 Platform and Hardware

The second layer of the project was the building of the rover platform and hardware. The car platform was stripped of all unnecessary components. The motors were tested using a DC power supply input. Batteries and a container to house all electronic devices were secured to the chassis.

After the development of the rover platform, appropriate hardware was attached. Schematics, shown in Appendix A, were developed to map all connections between the microcomputer, sensors, battery, and other necessary circuitry. Voltage regulation was a concern for each device. H-bridge integrated circuits (IC's) were necessary for motor control. A schematic diagram concerning each specific device was developed, and then all necessary circuitry was assembled on a prototyping board.

The platform and hardware development was a primary part of the design process. The hardware went through many evolutions during the project development. Although some software development could not take place until all hardware had been wired, software development began before this stage was complete.

4.3 Software Development

Software development was the third layer in the design process of the rover. It was approached on two fronts simultaneously—that of the rover and that of the base station. Since the rover modules were developed in C language and the base station required a graphical user interface (GUI) in Visual Basic, this development had many challenges. The communication processes developed in software to handle digital communication in C language were also duplicated for the base station, but they were developed in Visual Basic language.

4.3.1 Rover Software

Rover software was approached with a modular style of development. These modules were programs that worked independently of one another. Each module could be tested or changed, but they did not affect any other module. Slight modification allowed them to be incorporated into a main program named Rover Operations Application (ROA). For example, modules were developed to interface with each sensor. After the successful testing of each module, the modules were combined with a main program that executed all sequences. Interrupt service routines were necessary to allow more than one module to appear to be running simultaneously.

Motor control

The steering and propulsion of the vehicle was implemented using Pulse Width Modulation (PWM) and H-Bridge IC's. PWM was used to vary motor speed. PWM is a technique that modulates the amount of 'on' time for a pulse compared to the 'off' time in a period of time. One period of a PWM signal is a duty cycle. For example, a 50/50 duty cycle is half on and half off for the duration of the period. A higher duty cycle, such as 90/10, creates a higher speed, whereas a lower duty cycle, 10/90, produces lower speed. Repeating these duty cycles quickly produces an output to the motors that makes them appear to run at constant speeds. The real time clock and interrupt service routine programming were necessary to create modules that control the motors in this manner. The routines were first written so that they respond to direct keyboard input to the LogicFlex via a hyperterminal connection RS232 connection.

The motor direction was controlled using H-Bridge IC's. The logic of the bits is sent using PWM techniques and is inputted to the H-Bridge. It determines the motor's direction, as shown in Table 4.1.

Table 4.1: Status of the motor determined by an instantaneous PWM signal to each H-Bridge input

Input 1	Input 2	Motor Status
0	0	Coast
1	0	Clockwise
0	1	C-Clockwise
1	1	Hard Stop

Position feedback information was used to turn the steering motor to the proper direction in order to control steering. The steering motor is mechanically linked to a potentiometer. As the motor turns, the potentiometer turns, and as a result the voltage varies

across the potentiometer. The varying voltage across the potentiometer was read using an Analog to Digital Converter (ADC) port on the LogicFlex. Software modules were written to respond to this feedback and turn the motor to the necessary position based on keyboard input.

Sensor Suite Interfacing

Modules to interface with the sensors were designed concurrently with the modules for mobility. This was able to occur simultaneously since the layers do not directly affect one another. Program code was necessary to read the feedback information from each sensor. Each sensor had its own method of relaying its information to the user. The TCM2 Electronic Compass gave analog output. The tri-axial accelerometer and the tilt sensor also had analog outputs, which were monitored with the ADC ports on the Multi Input/Output (I/O) board. The final, and possibly the most important sensor to be integrated, was the camera and video transmitter. Aside from creating a module to turn on and off the camera and the video transmitter, the microcomputer had no other interaction with this system.

Battery consumption was a major concern while designing the rover. Although it may have been ideal to have all systems constantly on, certain systems and sensors are only turned on when necessary in order to lengthen the life of the batteries. Commands for turning on or off each sensor were necessary in addition to commands to request data from them. All modules were written initially to respond to direct keyboard input to the LogicFlex via an RS232 hyperterminal connection. The sensor suite modules were then combined with the motor control modules so that control of movement and sensors were packaged together in a single main program. External communication was the next level of implementation for the rover.

4.3.2 Base station

Since no member had experience in Visual Basic (VB), example programs that had no direct use for the project were first written. Once the language had been learned, programs that respond to joystick movement, joystick buttons, and mouse events were written for the base station VB GUI. Graphics, command buttons, and menus were added to the GUI. The base station GUI was named Rover Interface Application (RIA).

One of the primary benefits of using Visual Basic is its Windows style format and ease of manipulating graphics. A graphical display was made for each sensor. In addition to any raw data that was displayed in numerical format, a graphical display was also designed. For example, raw data for the compass direction, in degrees, was also displayed graphically with a compass needle.

The primary function of the RIA was to control and communicate with the rover. The relationship between the two is one of client-server. The rover responds to and executes all commands of the base station. It does not give any commands to the base station. The modules to handle communication were of utmost importance. The MSComm control, intrinsic to VB, was used in this part of the development. This ActiveX control allowed for selecting the proper RS232 port, setting the proper baud rate, and sending and receiving information through the port.

4.4 Hard-Wired Communication

The fourth layer of the rover design process was the hard-wired communication. Protocols defining the rules for communication between base station and rover were an important part of the design process. The communication layer was introduced after the ROA and RIA were partially developed. All communication used ASCII characters to form

messages. The ASCII characters were encoded into (11,7) Hamming block code to provide forward error detection and correction. This particular block code can successfully detect two errors and correct one error per ASCII character. A more thorough discussion of the protocols is in Section 5.5.

The protocols were first merely theoretically developed. The protocols that were decided upon were then implemented in software. The same exact processes for coding and decoding messages using (11,7) Hamming code in C language for the rover was implemented in Visual Basic language for the base station. Communication began using a null modem RS232 cable. The null modem cable allowed for full-duplex communication (two parties communicating simultaneously). After the modems were built, they were introduced into the link. The base station was connected to a modem that was hardwired to a modem on the rover proto-board. The modems allowed for half-duplex communication (one talker and one listener) only. Protocols were developed to handle the traffic involved in communication and to avoid collisions of messages.

4.5 Wireless Communication

After modem-to-modem communication was established through a hardwired link, the physical link was broken and replaced by an RF link. This involved creating cables to connect each modem to a transceiver. Delays also had to be introduced into each message sent to account for the time it takes for the transceiver to un-key (squelch). The wireless communication flow occurred as follows: 1.) Operator sends a command from base station PC 2.) Modem modulates the digital data using binary frequency shift keying (FSK) 3.) Each bit is sent wirelessly by the transceiver on the 146.580 MHz amateur radio band 4.) The signal travels to the rover's antenna 5.) The rover's transceiver receives the signal and sends it to the

rover's modem 6.) The modem demodulates each bit using FSK. When a message was completely received by the rover, a response was echoed to the base station. Communication in this direction behaved in the same manner, but it started with the rover and ended with the base station.

4.6 Testing

The final layer of the process was the testing of the functionality of all systems. Testing was also conducted as each portion of the project was completed. This helped to remove programming bugs and hardware design flaws.

The final testing process dealt with ensuring the efficient inter-workings of all modules. Battery life during operation of the completed rover, with various sensors on, was extensively tested to determine the operational life of the rover before recharge. The communication protocols were also tested.

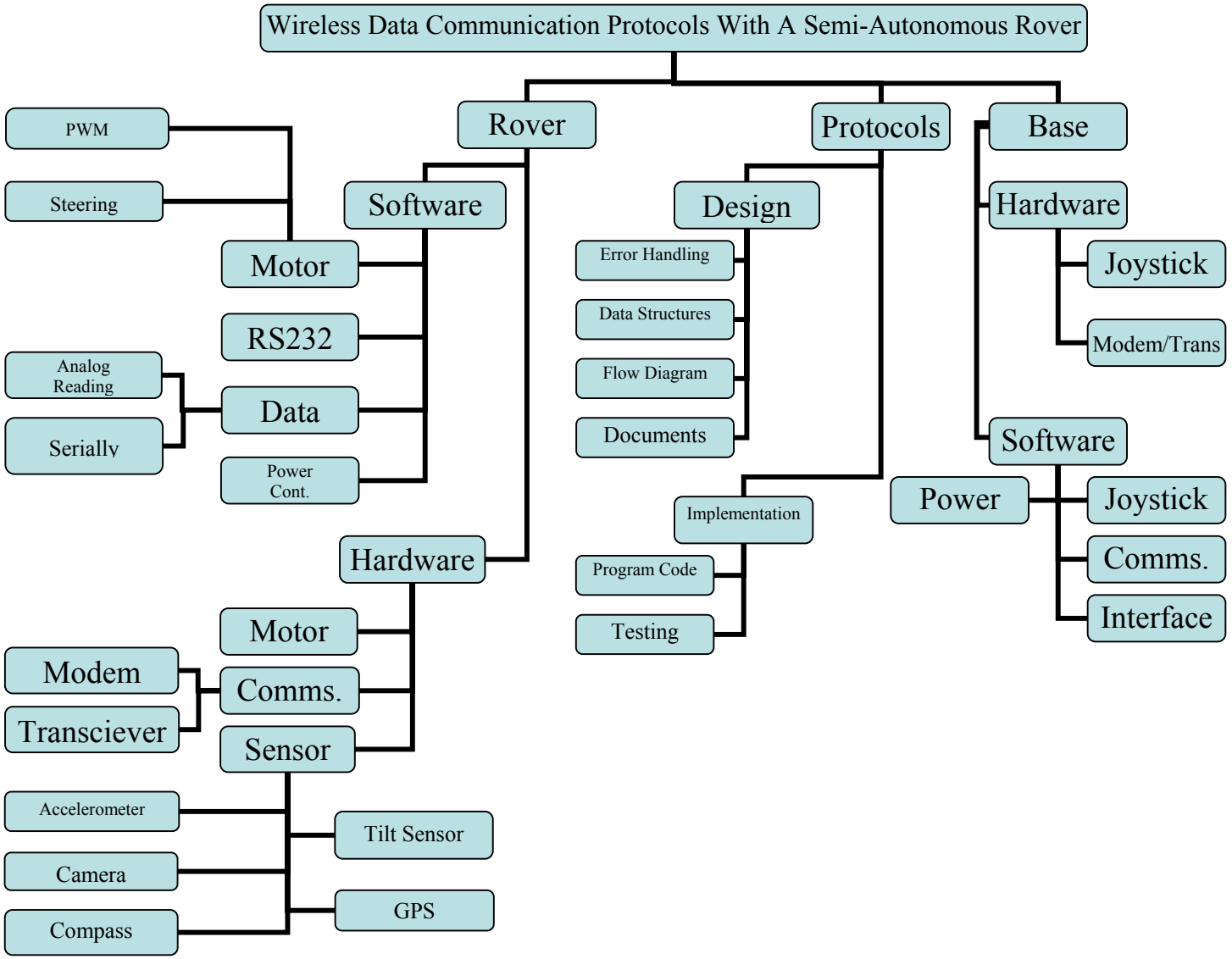


Figure 4.1: Project Design Flow to guide the team to completion of the project

Section 5: Results

5.1 Rover Hardware

In order for proper operation of the rover, circuitry needed to be built to work in concert with the LogicFlex and with each sensor. This section contains discussion of the various pieces of hardware, the circuitry that was built, and how it was integrated.

5.1.1 What is the Rover?

The rover was partially constructed from a remote controlled car. The chassis and suspension offered a ready-made platform that did not require mechanical design. The body of the car and all of its internal components, except for the driving and steering motors, were removed. In order to mount the hardware and peripherals to the vehicle chassis, a plastic storage bin was used. The actual design of the rover and its storage compartments were redesigned a number of times as a result of changes in the type of battery that was to be used. Figure 5.1 shows the final version of the rover layout.

Figure 5.1: Completed version of the Rover

5.1.2 The LogicFlex

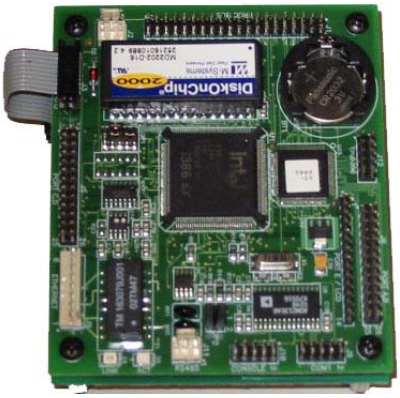


Figure 5.2: LogicFlex as provided by JK Microsystems

The 'brains' of our rover was the LogicFlex embedded microcomputer, a product of JK Microsystems (www.jkmicro.com). Figure 5.2 shows a picture of the LogicFlex. The responsibilities of the LogicFlex included, but were not limited to, receiving and requesting data from the sensors, receiving and requesting data from the remote user, controlling the motor drive speed and turning radius, and delegating power to the different onboard subsystems. The LogicFlex itself has 46 digital Input/Output (I/O) lines that were used for logic level control. It also has an RS232 port that was used for communications. The LogicFlex needed to be powered by a regulated supply of 5V DC with approximately 400mA of current. This specification affected the decisions made in the proto-board circuit design. A regulator capable of meeting these specifications was chosen.

Accompanying the LogicFlex was the Multi I/O peripheral board, also purchased from JK Microsystems, which was also incorporated into the rover design. The Multi I/O board has 8 Analog to Digital Converter (ADC) ports, 4 Digital to Analog Converter (DAC) ports, 8 optically isolated driver ports, and 2 Universal Asynchronous Receiver Transmitter (UART) ports. The LogicFlex runs on a DOS based operating system that can be programmed using C, C++, and assembly language programming. A limitation of the Multi I/O board was that

the ADC ports were only capable of reading analog voltages ranging from 0.000V to 4.095V. Any voltage outside that range could potentially damage the ADC port.

5.1.3 Power Distribution

One major task in any embedded system project is to provide adequate power to every device used in the system. Voltage regulator selection was an important aspect of power distribution. Factors that affected the selection process were the battery usage and the voltage level that each component required. Diode protection circuits were also incorporated in circuits where it was not possible to use voltage regulators.

Batteries

The source of the power supply for the rover was changed numerous times during the building of the rover. A Lead-Acid 12V battery was found to provide ample current supply and contained the necessary voltage to power all of the components of the rover. This power supply source, however, was ruled out because it was too heavy. The weight of the Lead-Acid battery put the motors under such strain that the rover had limited mobility. Lithium-Ion batteries were also considered as a source of power for the rover. They were lighter and had the necessary voltage, but they were unable to provide the current necessary for the motors. The final design contained rechargeable Nickel-Metal Hydride (NiMH) batteries separated into four separate power supplies.

Some of the subsystems on the rover drew relatively large amounts of current. These systems were given a dedicated power supply. Other components were grouped together and given the same supply. In this manner, all components were assigned to the four supplies accordingly, so that no single supply was strained by the components that were being powered. Three of the four supplies contained eight 1.2 V NiMH batteries in series, resulting

in a supply that provided 10.3V. One of these supplies powered the LogicFlex, inclinometer, accelerometer, and compass. Another supply powered only the motors. The third supply powered the transceiver. The fourth power supply consisted of eleven 1.2V NiMH batteries in series providing 14.4V. This supply powered the amateur television (ATV) transmitter and fast scan camera. Each power supply had 2100mAH when fully charged.

Voltage Regulators

Three different voltage regulators were used in the design of the rover. The most important regulator chosen for the rover was the LM323. The LM323 had a metal can package and was supplemented with a heat sink. It provided continual 5V output with a current capacity of 3A. This regulator was chosen to power the LogicFlex, because it was capable of dissipating large amounts of heat, through its package and heat sink, that was generated as a result of large current draw. A number of other voltage regulators were tried in this role but overheated under the strain; consequently, continual 5V supply was not provided to the LogicFlex.

The second kind of regulator used was an LM317 adjustable voltage regulator. These regulators came in a standard 3 pin TO220 package. To make voltage output of the regulator adjustable, the circuit shown in Figure 5.3 was built. To adjust the voltage, the potentiometer R2 was adjusted until V_0 was at the desired output voltage.

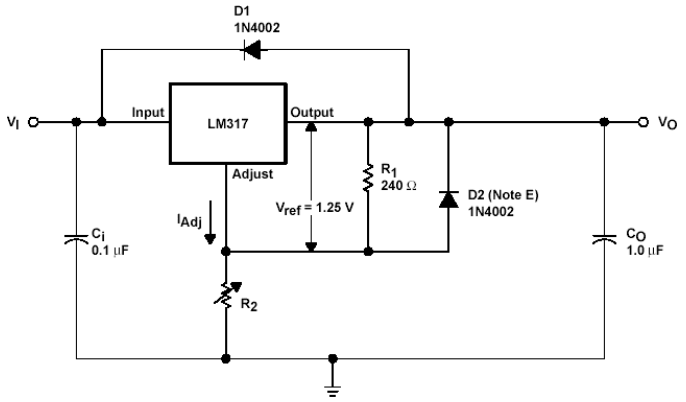


Figure 5.3: Schematic diagram of circuit built to adjust voltage output of the LM317 voltage regulator

The LM317 was used to power a number of different devices. First, one was used to control the voltage provided to the H-Bridge motor controller IC's. Second, an LM317 set to 8 Volts powered the tilt sensor. A third LM317 pre-regulated the voltage input to another type of voltage regulator, a shutdown regulator, that was used.

The third type of voltage regulator used was the LT1129 from Linear Technologies (http://www.linear.com). This regulator had a shutdown pin that was used to turn the regulator off and on. The shutdown pin was controlled from a parallel port output from the LogicFlex. These regulators were used to supply power to the electronic compass, the accelerometer, and the modem circuit. Because the regulators could be turned off, they played an important role in power management and conservation.

Protection Circuits

One of the most important aspects of any project involving electronic devices is protecting those devices from electrical damage. One line of defense against electrical problems is to provide a regulated voltage supply to each device using voltage regulators. Voltage regulators often have a number of safety features built into them that provide enough protection for the device that each regulator is powering. When it is not feasible to use a

voltage regulator, or if the protection provided by the regulator is not sufficient, diode protection circuits and fuses can be used.

The main kind of diode protection circuit used in the rover design was used to protect the radio transceiver. The diode circuits were used to protect the transceiver against reverse polarity, large voltage spikes, and high current caused by a short circuit.

5.1.4 Driving the Rover

Mobility was of utmost importance in this project. Without mobility, the rover could not rove. The rover chassis contained a rear wheel drive DC motor and a front wheel steering motor. The main concerns in driving the rover were development of means to control the speed and direction of each motor. The steering motor presented an additional concern - the need to determine the angle at which the wheels are turned. There was also a need to protect circuitry from electrical disturbances caused by DC motors.

H-Bridge

The TPIC0108B intelligent H-bridge driver by Texas Instruments (www.ti.com) was used to control direction of the motors. The logic input to the H-Bridges determined the polarity of the voltage supplied to the motors. It also, in turn, determined the direction that the motors turned. Pulse width modulation (PWM) techniques were implemented in software to control the speed of the motors. PWM is discussed in more detail in Section 4.3.1.

Opto-Isolators

Opto-isolators were necessary to interface the LogicFlex to the H-bridges. The parallel port outputs on the LogicFlex used 3.3V logic. This meant that when an output pin was turned on, logic 1, the output of that pin was 3.3V. When the output was turned off, logic 0, the output of the pin was 0.0V. The H-bridge, however, required logic greater than 3.3V.

The H-bridges required a voltage ranging from 3.6V to 7V for logic 1. The opto-isolators were used to translate between these two different logic levels. In addition, the opto-isolators provided protection by physically separating the LogicFlex from the H-bridges. Opto-isolators are made up of a light-emitting device and a light sensitive device, which are all wrapped up in a single package. No electrical connection, however, exists between the two. The light emitter is an LED. The light sensitive device is a phototransistor. When the LED is driven with a current, it shines onto the phototransistor, which then allows for electrical conduction (it is turned on). The LED side of the opto-isolator was driven by the parallel port output from the LogicFlex. The phototransistor side was interfaced to the H-bridges with a higher voltage supply. In this manner, opto-isolators provided voltage translation for interfacing to the H-bridges with the added benefit of protection of the embedded microcomputer.

5.1.5 Sensors

Sensors provided useful data feedback about the rover's environment. A tilt sensor, compass, accelerometer, and wireless video camera were added as a sensor suite to the rover. A global positioning device was also incorporated in the original design, but it was removed to minimize the mass of the rover. The LogicFlex monitored each sensor, excluding the camera. Upon request from the base station, data was relayed from the rover to the base station. It was necessary to connect each sensor to its designated port on the rover's circuit board via specially designed cables. The wiring table for each sensor can be found in Appendix B.

Tilt Sensor

A Series 0729 Fredericks bi-axial electrolytic tilt sensor with analog output was used to provide pitch and roll data of the rover (http://www.frederickscom.com). The tilt sensor was interfaced with two ADC ports on the Multi I/O board - one for pitch and one for roll. The electrolytic tilt sensor required a power supply in the range of 7V to 10V and provided an output voltage proportional to tilt angle. The proportional output depended on the voltage supply chosen. An 8V power supply was chosen, and measurements were taken for the output voltage and various degrees of inclination. Figure 5.4 below is a photograph of the inclinometer.

Figure 5.4: Fredrics Electronic Inclinometer (Tilt Sensor)

Compass

A TCM2 Electric Compass (www.pnicorp.com) was also an important part of the sensor suite on the rover (see Figure 5.5). This component contained a tri-axial magnetometer that generated a three dimensional magnetic vector. It computed a polar value from this vector. The compass provided an analog output proportional to the polar value. The compass was interfaced with one ADC port on the Multi I/O board. The ADC port was polled to determine the direction that the vehicle was facing. To satisfy the voltage requirements of this device, a 5V shut down regulator (LT1129) was used. The proportional analog output

ranged from 0V to 5V, where 0V represented 0 degrees from North, and 5V represented 359 degrees from North. However, the ADC ports on the Multi I/O board were only capable of reading voltages ranging from 0V to 4.096V. A voltage divider was introduced into the circuit to correct this problem. The voltage divider scaled the analog voltage output of the compass down to a range acceptable for the ADC port.

Figure 5.5: TCM2 Electronic Compass as purchased from pnicorp.

Accelerometer

A Crossbow Tri-axial Accelerometer (www.xbow.com) was used to measure the three-dimensional acceleration of the rover in units of gravitational force, G's (1 G = 9.81 m/s^2) (see Figure 5.6). To satisfy the voltage requirements of this device, a 5V shut down regulator (LT1129) was used. The accelerometer provided three analog output voltages - one for each axis. The output voltages were proportional to the acceleration in G's, where approximately 2.5V was zero G's with a voltage-per-G ratio of 80 mV/G. Three ADC ports were used to poll the three axes of acceleration.

Figure 5.6: Crossbow Tri-axial accelerometer

Camera

The PC Electronics (www.pcelectronics.com) camera and amateur television (ATV) transmitter were given a 14.4V supply. The power was supplied through a driver port, which was turned on or off depending upon the commands given to the rover, on the Multi I/O board. When on, the ATV transmitter sent a live video feed to the base station transmitting at 440MHz. Figure 5.7 below shows a photograph of the camera and transmitter.

Figure 5.7: PC Electronics ATV camera and transmitter

5.1.6 Rover Communications Hardware

A number of communication devices were built and used in order to communicate information wirelessly between the rover and the base station.

Rover Modem

In order to achieve communication wirelessly, a modem was necessary. The modem acted as an intermediate stage in the communication between the LogicFlex and an amateur

radio transceiver. In order to transmit data, the rover modem took bits of data that were constant output voltages from the LogicFlex and changed them into tones that were then relayed to the base station via the transceiver. The modem converted all received tones from the transceiver into bits of data of constant voltage to be interpreted by the LogicFlex.

The communication system for the rover was based around the MX614 (Figure 5.8) modem integrated circuit. The MX614 is a modem-on-chip that operates under the principle of frequency shift keying. Frequency Shift Keying (FSK) is a type of modulation that uses two distinct tones or frequencies to represent a 1 and a 0. The modem IC is half duplex, meaning that communication is only possible in one direction at any time. The MX614 also has the capability of retiming the data input and output using a clock signal. This feature was unnecessary for the asynchronous communication between the rover and the base station. The MX614 uses Transistor-Transistor Logic (TTL), or 0 and 5 Volt, levels to distinguish between logic 0 and logic 1.

The MX614 had a number of operational modes that utilize different frequencies for data communication. These modes (Table 5.1), as well as the baud rate of the modem (Table 5.2), depended upon the logic levels applied to M1 and M0.

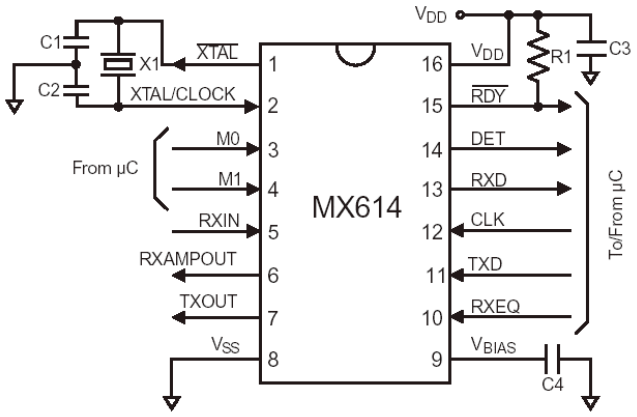


Figure 5.8: The MX614 Modem IC

Table 5.1: Logic frequency modes

M1	M0	Logic 0 frequency	Logic 1 frequency
0	0	488Hz	385Hz
0	1	2.2kHz	1.2kHz
1	0	0Hz	385Hz
1	1	Zero Power	

Table 5.2: Baud settings

M1	M0	Receive bit rate	Transmit bit rate
0	0	1200bps	150bps
0	1	Off	1200bps
1	0	1200bps	Off
1	1	Zero Power	

The modem was operated at 1200 bps for transmitting and receiving, where 2.2kHz represented logic 0 and 1.2kHz represented logic 1. To change the modem between transmit and receive mode, a control line was used from the LogicFlex RS232 port. Since the logic that was applied to M1 and M0 for transmit and receive were logical inverses of each other, a NOT gate was used in conjunction with the control line. Consequently, the control line and logic gate provided logic 10 or 01 to M0 and M1 depending on the desired mode - transmit or receive. The data terminal ready (DTR) line of the RS232 port was used to control between transmit and receive modes. Table 5.3 shows the logic applied to M1 and M0 depending on the logic of the DTR line. Note that when receiving, the modem interprets any value in a range of frequencies as logic 1 and another range is interpreted as logic 0.

Table 5.3: Transmitting and Receiving modes resulting from the DTR line

DTR	M0	M1	Mode	Logic 1	Logic 0
1	1	0	Transmit 1200bps	1.2kHz	2.2kHz
0	0	1	Receive 1200bps	1.04kHz – 1.46kHz	1.91kHz – 2.2 kHz

Although the modem IC contained all the necessary functionality, additional circuitry was necessary to interface with RS232 logic. The modem IC operated using transistor-transistor logic (TTL). The com port of the LogicFlex used RS232 logic. Table 5.4 shows the difference between TTL and RS232 logic standards.

Table 5.4: Logic levels for different standards

TTL Levels		RS232 Logic Levels	
Logic	Voltage	Logic	Voltage
0	0V	0	10V
1	5V	1	-10V

Since the logic levels were incompatible, MAX232 IC's (see Figure 5.9) were needed to convert between the two standards. This IC provided two channels for converting from TTL to RS232 and two channels for converting from RS232 to TTL. One RS232 to TTL channel was used to translate serial data that was to be transmitted from the LogicFlex. One TTL to RS232 channel was used to translate demodulated serial data that came from the transceiver. And one RS232 to TTL channel was used for the DTR control line. The DTR line was first translated by the MAX232 channel and then applied to M0 as well as a NOT gate that was then applied to M1. In this manner, the modem was switched between transmit and receive modes.

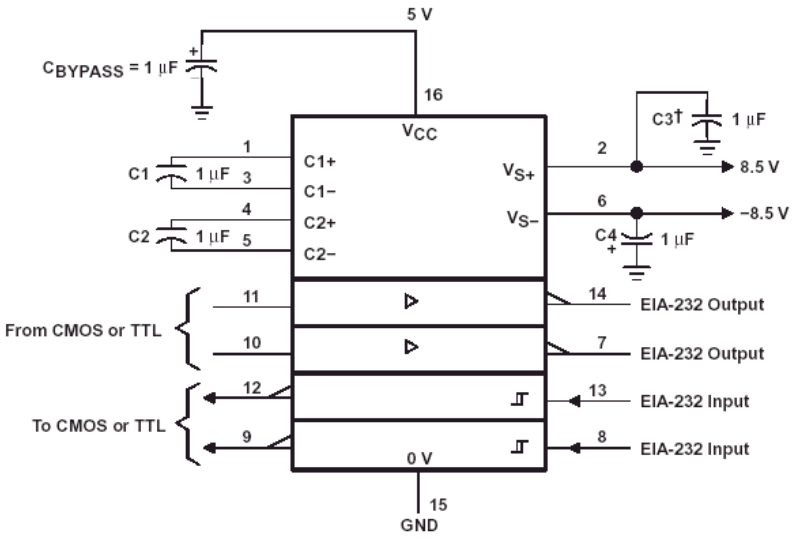


Figure 5.9: The MAX232 IC

According to the specifications of each IC it was also necessary to incorporate capacitors and resistors in the circuit. Also, a crystal was necessary to provide timing to

generate the sinusoidal tones for logic 1 and 0. Cables and connectors were also necessary to connect the modem to the com port and to connect the modem to the transceiver. Each connector was made of two 5-pin strips.

Rover Hand Transceiver

The rover transceiver was a Kenwood, 25AT hand transceiver (HT). This unit required a power supply ranging from 7V to 12V. A protection circuit (as discussed in Section 5.1.3) was needed to safeguard the unit from excessive current spikes. A cable had to be constructed to connect the HT to the rover modem. The microphone input jack of the HT connected to the transmit line of the rover modem. In addition, the push-to-talk line of the modem was connected to this jack to allow for the keying and un-keying of the HT. This line was controlled by the DTR line of the LogicFlex's RS232 port. The cable also created a link between the speaker jack of the HT and the receive line of the modem. Appendix B contains the pinout information for the cable that was constructed between the base station modem and transceiver.

5.2 Base Station Hardware

5.2.1 Base Station Transceiver

The base station transceiver was a Kenwood, FM 144MHz dual bander, TM721A. It was necessary to construct a cable that connected from the base station modem to the microphone connector of the transceiver. The pinout diagram of the Kenwood microphone connector appears below in Figure 5.10.

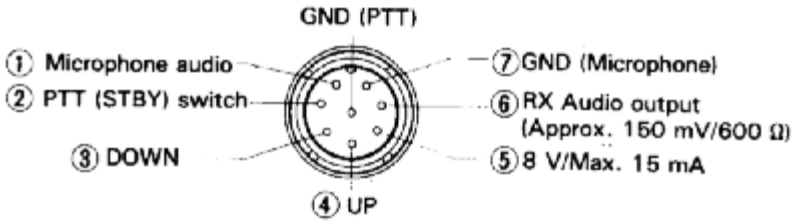


Figure 5.10: Pinout Diagram for the microphone connector on the Kenwood TM-721A

The push-to-talk (PTT) line was used to key and un-key the microphone. This line connected to the modem, which was in turn controlled by the DTR line of the modem. The microphone audio line was used for transmitting all data, and the RX audio output was used for listening to any incoming data. Appendix B contains the pinout information for the cable that was constructed between the base station modem and transceiver. The cable used connected both grounds (PTT and microphone) to a single line. This in turn was connected to the NOT gate in the modem. When the DTR line went high the output of the NOT gate was inverted and both lines were shunted to ground. This made it possible the DTR line to control both the keying of the microphone and the acceptance of a microphone input.

5.2.2 Base Station Modem

The base station modem followed nearly the same design as the rover modem as described above. The design was centered on the MX614 IC that performed the modulation and demodulation using frequency shift keying, and the MAX232 IC that converted between the logic levels. The base station modem was housed in a small plastic casing and connected to the PC com port via an RS232 cable. The case was constructed with a DB9 jack to connect to the RS232 cable. A power jack was also installed to connect to a 5V wall mounted DC power supply. A DIN5 jack was also installed to connect to the base station transceiver.

The main difference between the base station modem and the rover modem is that, in the base station modem, two variable gain amplifiers were installed to boost signals received and

transmitted by the transceiver. The LM324 quad operational amplifier IC (Figure 5.11) was used to perform this function. The LM324 used a single sided 5V power supply that originated from the main power supply to the modem. Since the LM324 contained four op-amps in one package, only one was necessary, saving space on the prototype board.

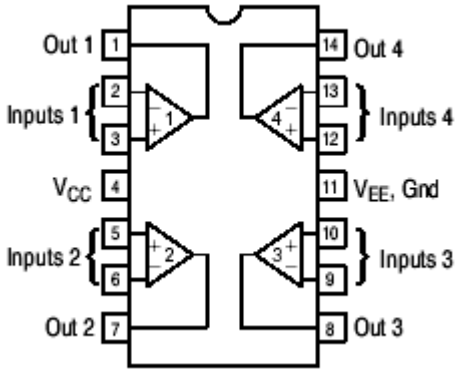


Figure 5.11: The LM324 IC

Each amplifier circuit was constructed with a 500kΩ potentiometers variable feedback resistors and 50kΩ input resistors. The maximum gain achieved by the amplifiers was 100V/V. This meant that an output voltage could be amplified to 100 times that of the input. The output was limited, however, to a maximum of 5V, which was the magnitude of the power supply. This circuit was necessary to send and receive signals of small amplitudes.

5.2.3 Base Station PC

The base station PC was a Compaq, with a Pentium I 166 MHz processor, and 64MB of RAM. The PC operating system was Microsoft Windows 98. The Rover Interface Application (RIA) was a Visual Basic 6.0 application. Consequently, Visual Basic 6.0 was installed on the base station PC in order to run the application. The base station computer also had two RS232 com ports. The com1 serial port was connected to the modem via an

RS232 cable. The PC also contained a 15-pin analog joystick port. A Logitech Wingman Extreme (www.logitech.com) joystick was connected to this port.

5.3 Rover Software

As previously discussed, the central processor for the rover is the LogicFlex i386. The LogicFlex is capable of running C and C++ language executable files compiled using Borland C. The C source code, named Rover Operations Application (ROA) went through many evolutions. When major changes were made, a new version was created. There were eleven versions in total. The final ROA source code appears in Appendix C and was named roverv11.c

5.3.1 Creating the Borland C Project

Setting up a project in the Borland Integrated Development Environment is a critical step in developing operational software for the LogicFlex i386. Figure 5.12 is the Target Expert window of Borland C. The following options were chosen in the appropriate fields: Target Type - Application (.exe), Platform - DOS (Standard), and Target Model - Small. The target type and platform were necessary for operation on the LogicFlex. The small target was chosen to keep the size of executable as small as possible.

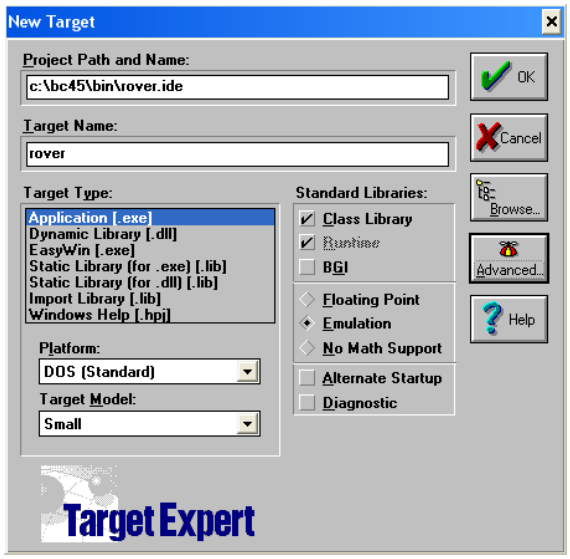


Figure 5.12: Borland Target Expert window

The ROA program was written in the C programming language. Additional files were included in the project to allow for use of the RS232 port and the multi I/O board. RS232.c was written to provide all of the basic functionality needed to employ serial I/O in any application written with Borland C language compilers. JKMicro provided this file with the LogicFlex. The RS232.c file was included in the project tree as a sub-node of our main roverv11.c source code. In addition, it was necessary to have the RS232.h header file in the same folder as the project. These inclusions allowed for use of the functions available in RS232.c. Using the multi I/O board required the inclusion of DRIVERSM.obj. This object file was provided with the LogicFlex and is used for small model applications. If the model target was large, it would have been necessary to include DRIVERLG.obj instead, also provided by JKMicro. This object file contained all the assembly routines necessary for use of the multi I/O board. For example, this file contained routines to turn on or off a driver port, or to read a voltage on an A-D converter port. The DRIVER.h file was also included in the project folder. The project tree for ROA, version 11, appears in Figure 5.13. Although the

RS232.h and DRIVER.h do not appear in the tree, they are saved in the same folder as the project.

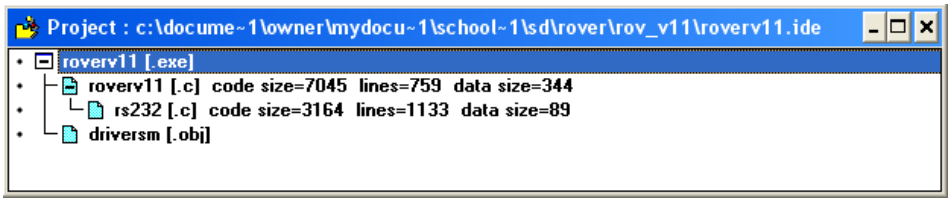


Figure 5.13: Borland project tree for ROA version 11

The project setup procedure described above was the foundation for the source code. Without the proper setup the rover software would not have compiled properly, and the RS232 port and Multi I/O board would not have worked.

5.3.2 Mobility

After the project was created, the main task was to provide mobility to the rover. The key to accomplishing this task was the use of pulse-width modulation (PWM). In order to program PWM it was necessary to use interrupt programming. An interrupt service routine was created and named "irq10". The interrupt routine was triggered at a rate of about 1300 times per second. This rate was based on the argument given to the start_timer function. The start_timer function used the real time clock of the LogicFlex. The argument given to the function determined the rate of interrupts. The function call was as follows: start_timer(0x035F) //1300Hz. At a rate of 1300 times per second, the irq10 interrupt stopped the main function of the rover, executed the lines of code inside of the interrupt and then returned to the main function. The interrupts were used to implement PWM.

If the rover received a drive command, the drive function was called. The drive function set up duty cycle counters for timing the PWM and direction masks for turning on the correct parallel port pins on port B of the LogicFlex. A drive flag was then set to TRUE,

and the interrupts carried out the timing for PWM. In the interrupt, counters were then decremented depending on whether the motor was in its on or off state. When a counter reached zero, the state changed from on to off, or off to on. Fifty interrupts represented one duty cycle. If the ontimerD variable was set to 45 and the offtimerD was set 5, a 90% duty cycle was generated. Also, a counter, driveCount, was used for timing out a drive command. After 26 duty cycles (about 1 second) the drive command ceased.

The turning routine was programmed in a similar manner. The call to the turn function sets up necessary values to complete the turn, and then the interrupts took over to complete the turn. The turning routine used a constant 50% duty cycle. The additional requirement in this routine was that it needed to poll an A-D port connected to a potentiometer that was linked to the steering motor. This potentiometer gave position feedback as to current position of the front wheels. At the beginning of the routine, the A-D port was polled. A determination was made as to direction that the motor had to turn in order to make the wheels steer to the correct position. At the end of each duty cycle, the A-D port was again polled to determine if the potentiometer was voltage was in an acceptable range, in turn indicating that the wheels were steered in the correct direction. If necessary, the steering motor would enter an additional duty cycle, and repeat this process until the wheels were steered in the correct direction.

5.3.3 Data Acquisition Routines

As discussed in Section 5.1.5, the rover carried four sensors: an accelerometer, a compass, an inclinometer, and a video camera with transmitter. To interface with each sensor, it was first necessary to give the rover the capability of turning each sensor on or off. In the case of the accelerometer and compass, an output pin on Port A of the LogicFlex was

assigned to each device. The state of the pin for the given device was changed to 1 to turn on the sensor and 0 to turn off the sensor. This, in turn, turned on or off a 5V shutdown regulator, which provided the required voltage to these sensors. In the case of the inclinometer and video camera driver ports on the multi I/O board were used. This assignment was chosen because these two sensors had different power requirements and could not use the 5V shutdown regulators. The driver ports acted as switches that turn on or off regulators set to the appropriate voltage for the sensor.

After on/off controls were built into each sensor, it was necessary to program methods for querying data. The accelerometer, compass, and inclinometer were connected to A-D converter ports. The accelerometer required three ports, one each for the X, Y and Z-axes. The compass required only one A-D port. The inclinometer required two A-D ports, one for pitch and one for roll. The GetA2D command (driver.h) was used to poll each port. The data was then scaled to appropriate values. The values were then converted to strings for sending back to the base station. The methods can be found in the Acc, Comp, and Tilt functions in the roverv11.c code of Appendix C.

The video camera did not require any further processing to query data from it. Turning on the camera also turns on the National Television System Committee (NTSC) transmitter. The NTSC transmitter communicated directly with a receiver at the base station. It transmitted continuous live video on a separate channel.

5.3.4 Autonomous Routines

Two autonomous routines were built into the functionality of the rover: a quiet mode, and an automatic call sign transmission routine. Quiet mode is a state that the rover entered if it did not receive any message from the base station for three minutes. After a three-minute

period of silence, all sensors were turned off to conserve the life of the battery. This routine was programmed using a counter that decremented every time the interrupt routine interrupts the main module. If at any time a message was received from the base station, the counter was reset to the initial value. If no message was received, the counter eventually decremented to zero after about three minutes, and then entered quiet mode.

To conform to Federal Communications Commission (FCC) regulations, it was necessary to send the operator's call sign every ten minutes. To accomplish this a counter was again used. The counter was decremented at every interrupt to the main module. When the counter reached zero, the counter was reset, and the operator's call sign was transmitted.

5.3.5 Communication Routine

The RS232.c and RS232.h files, provided by JKmicro, were used to handle all rover communications on the LogicFlex via the RS232 port. A readme text file was also provided with these files that documented the functions available in these files. Five functions were used from these files.

First, the rs_initport function was used to open and initialize the RS232 port. This function required a number of arguments to set necessary parameters for the com port. For example, the application required a baud rate of 1200 due to the specifications of the modem IC used. As a result, when the rs_initport function was called the baud rate parameter was given an argument of 1200. Other arguments in the function call included values to set the input and output buffers, the com port used, whether parity is used, and how many stop bits to use in each byte-long transmission.

After the port was initialized for use, data could be sent through the RS232 line. Two functions were used: rs_sndbyt and rs_sndstr. The rs_sndbyt function accepted one argument,

a number from 0 to 255 representing one byte of information. The function then sent the appropriate bits representing this number. The rs_sndstr accepted two arguments, the length of the string, and a pointer to the beginning of the character string to be sent. This function works similarly to rs_sndbyt, but it sent a string of bytes rather than just one. The length argument let the function know how many bytes of the character string it was to send.

The rs_modctrl function was used to turn on or off the data terminal ready (DTR) line of the RS232 port. This in turn turned on or off the push-to-talk line of the modem. This function was necessary for changing the rover from receive mode to transmit mode. The rover would remain in receive mode until it received a command. Upon receiving a command it changed the state of the DTR line to on and then echoed a response to the command. Upon echoing the response, the DTR line was turned off and the rover waited for the next command.

The rs_close function was used to shutdown the port. This function was mainly used in testing. If the port were not closed properly, the port would not be able to be reopened without rebooting the computer. In final testing, this section of code was not reachable. If the rover was instructed to shutdown and exit its main routine, further communication could not be achieved until the rover program was manually restarted.

5.4 Base Station Software

The base station platform was a personal computer (PC). A graphical user interface (GUI) was constructed using Visual Basic 6.0 (VB). VB provided a user-friendly windows style environment for the operator. Creating an easy-to-use control environment was one of the objectives of this project. Visual Basic was a vehicle to this objective. Mouse controls, command buttons, a help menu, tool tips, and many other standard windows features were

incorporated into the GUI. The program created to control the rover was named the Rover Interface Application (RIA).

VB is an event-driven programming language. Event-driven is a term that means different windows on the screen can respond to events. An event is an action that occurs as a result of some user activity. For example, one event is generated when the end-user clicks a mouse button, and a different event is generated when the end-user types on the keyboard. Such events cause the corresponding program function relating to the event to be executed. This style of programming varied greatly from C and C++ iterative style programming. Having no experience in this style of programming, independent study of VB was necessary in order to learn how to create such a program.

5.4.1 Joystick Control

In addition to the keyboard and mouse, RIA used joystick input for control of the rover. In order to accomplish joystick control an ActiveX control, not intrinsic to VB, was used. The ActiveX control used for the project was a free control found on the Internet. In order to use this control it was necessary to paste the free joystick.ocx file into the windows/system32 folder of the PC being used. The control then had to be added by checking the appropriate block in the component manager. The joystick control was then added to the main rover form. Properties, such as the polling interval, were set on the joystick. The polling interval refers to how often the VB program actually checks the state of the joystick. A joystick calibration form was also constructed in order to calibrate the joystick. This form and all of its functions were written specifically for use with RIA. Joystick control added additional ease of use to the control of the rover from the base station. Controlling the rover via joystick was similar to using a joystick to play a video game. It is

likely that any end-user accustomed to video game controls could quickly adjust to controlling the rover. The joystick trigger buttons were used to enable joystick commands. Each of the four trigger buttons corresponded to four different speed commands that could be sent. Only when a button was pressed was a command sent. This protected the system from constant communication even when the joystick was not being used.

5.4.2 Sensor Displays

All data received from the sensors was translated into a graphical form. If a command was sent to the rover to turn on a sensor, and the rover echoed that it completed the task, a green light would turn on. If the sensor was turned off, a red light would illuminate. Figure 5.14 shows a screen shot of the main rover form. As can be seen in this figure, areas of the form are segmented off for each sensor. Buttons are available for turning on or off the sensor. Corresponding lights exist to display whether the sensor is on or off. The accelerometer, inclinometer, and compass also have "Get Data" buttons. These buttons are used to query data from the sensor.

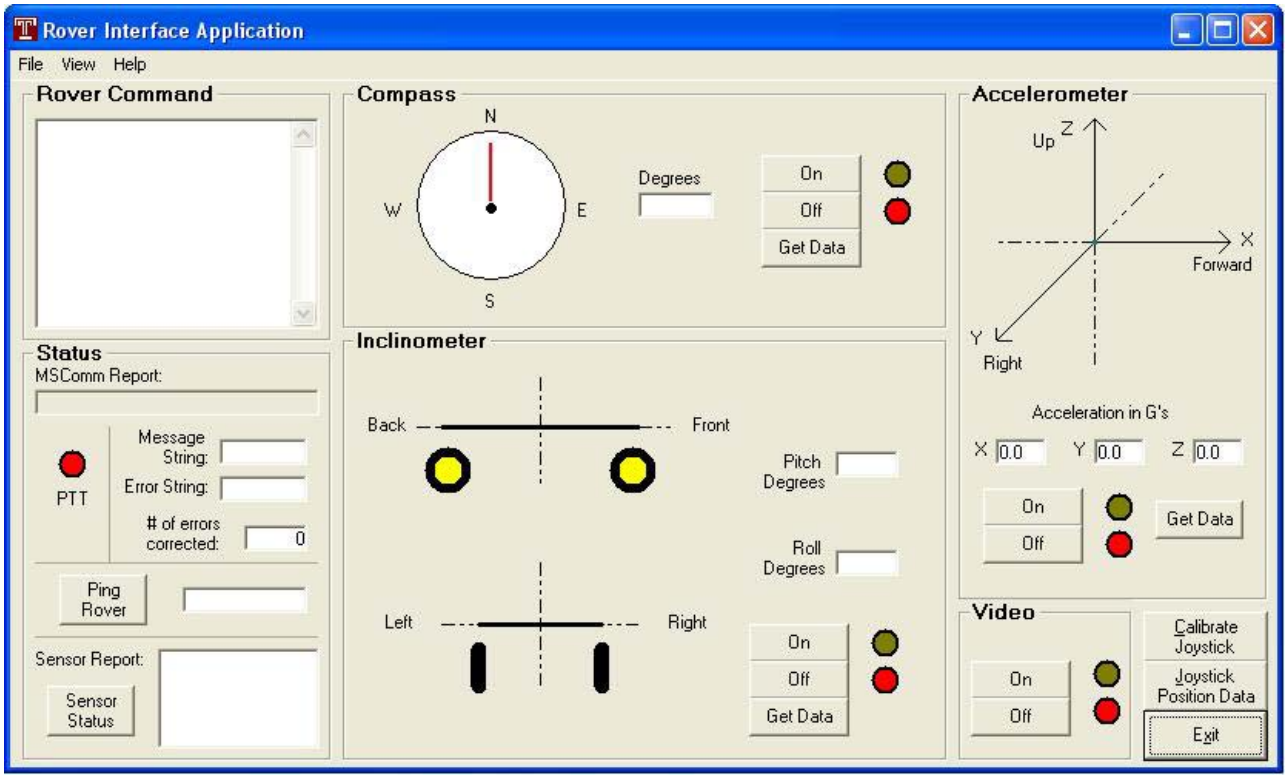


Figure 5.14: RIA main form

The data queried from sensors was displayed in two forms: graphical and numerical. For example, the compass may be queried for data. The rover would receive the corresponding command and then return the appropriate string after polling the necessary A-D port. The data string received by the RIA was first scaled to the appropriate value. The value, in degrees, was then displayed in the degrees text box for the compass sensor. Then the value was used to move the red needle to the appropriate location and represent the direction that the rover is facing. A typical compass readout appears in Figure 5.15.

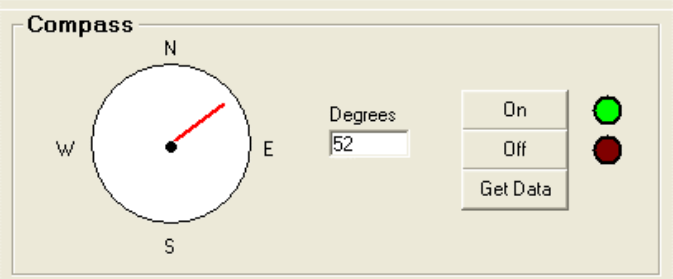


Figure 5.15: Compass display

Similar methods were used to display data for the inclinometer and the accelerometer. A stick drawing of the rover was used to graphically display the inclinometer data. For example if the rover was pitched forward, and rolled to the right, the drawing may appear as in Figure 5.16. Scaled vectors were used to display the acceleration of the rover. A typical readout appears in Figure 5.17.

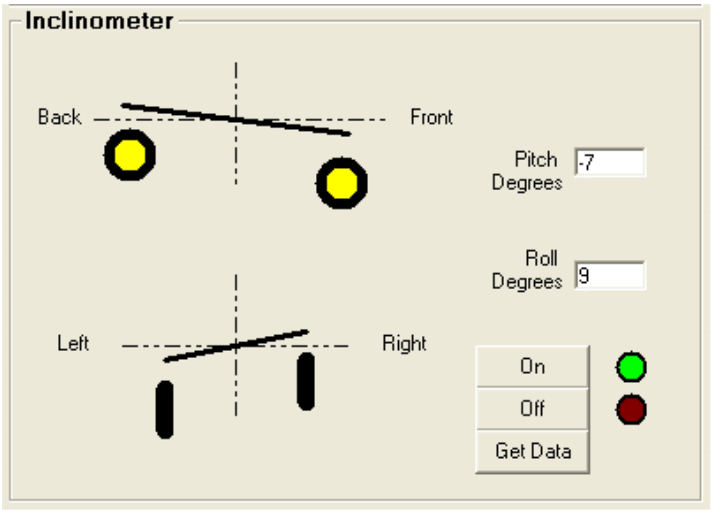


Figure 5.16: Inclinometer display

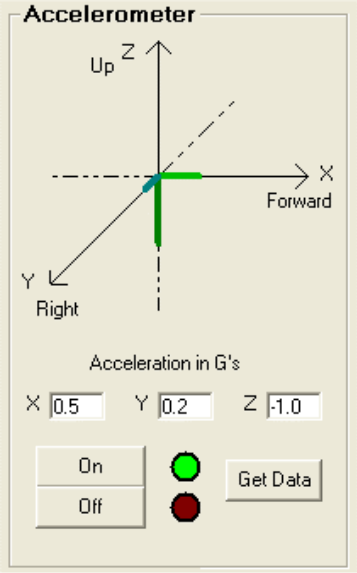


Figure 5.17: Accelerometer display

The use of graphical sensor displays also contributed to the ease of use and ability for an end-user to quickly interpret incoming data. This conforms to today's standards in programming and makes the RIA a powerful platform for collecting data from the rover about its surroundings.

5.4.3 Communication Routine

Communication was handled using the RS232 port. The RS232 port was then connected to a modem built using the MX614 chip, as discussed in Section 5.1.6. In order to program the communication, the MSComm control was used. This control allowed data to be sent and received via the RS-32 port. It also had parameters that could be set for baud rate, start and stop bits, and parity, just as rs_initport had for the Borland C program. As an analogy, the MSComm control gave capability to the RIA VB program that RS232.c and RS232.h gave to the rover C program.

The same baud rate, parity, and stop bit settings were programmed for the MSComm control as were programmed for the rover using rs_initport. Bytes of data were sent using MSComm1.Output statements. Data was received using the MSComm1_OnComm event. As data was received, the input buffer length was checked. If the buffer reached the appropriate length depending on the length of the expected message to be returned by the rover, the input buffer was read using the MSComm1.Input statement. The message was then decoded, and the data was processed.

Another need was to control the state of the data terminal ready (DTR) line. The DTR line was used to control the push-to-talk line (PTT) line of the modem. When the user sent a command, by clicking on a button, by typing a keyboard command, or by using the joystick, the RIA changed to receive mode by changing the state of the DTR line. The DTR line remained off until a response was received. If the RIA did not receive response, a time out would occur. A time out was built in so that the RIA could return to transmit mode whether the rover responded correctly or whether it did not.

5.4.4 Autonomous Routine

To comply with the Federal Communication Commission (FCC) standards, the RIA also required an autonomous routine for sending the user's amateur call sign every ten minutes. Timers in VB have a maximum setting of about sixty-five seconds. As a result, a timer was set to trigger every sixty seconds and then decrement a counter that was set to ten. When the counter reached zero, the call sign was sent, and the counter was reset to ten. In this manner, the user's call sign was sent every ten minutes.

5.5 Protocols

5.5.1 Introduction

The purpose of this section is to define and explain, in detail, the process of wireless digital communication between the base station (client) and the mobile rover (server). To describe the process, a layered approach has been taken. When referred to as a whole, all of the layers will be called the protocol stack. As in other layered methods, each layer in the protocol stack operates with the layers directly adjacent to it. The protocol stack for this project consists of three layers, from top to bottom: Application Layer, Coding Layer, and Physical Layer . Figure 5.18 shows a graphical representation of the protocol stack.

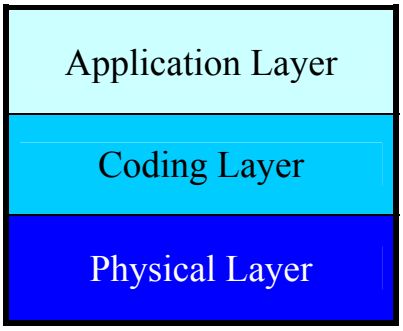


Figure 5.18: Protocol stack

5.5.2 Application Layer

The Application Layer is the highest layer. This layer deals with the applications that are used in communication between client and server. The application layer was responsible for managing the flow of communication, and was also responsible for command and control. For this project, a Rover Interface Application (RIA) was written in Visual Basic for the base station computer, while a Rover Operation Application (ROA) was written in C for the DOS operating system on the rover. In the client-server relationship only the client (base station) could initiate any communication. The server (rover) was expected to respond accordingly.

Only one party was able to communicate at a time over the half-duplex channel. As a result, state diagrams were constructed to describe the flow of communication. Figure 5.20 and 5.21 below are the base station and rover state diagrams respectively.

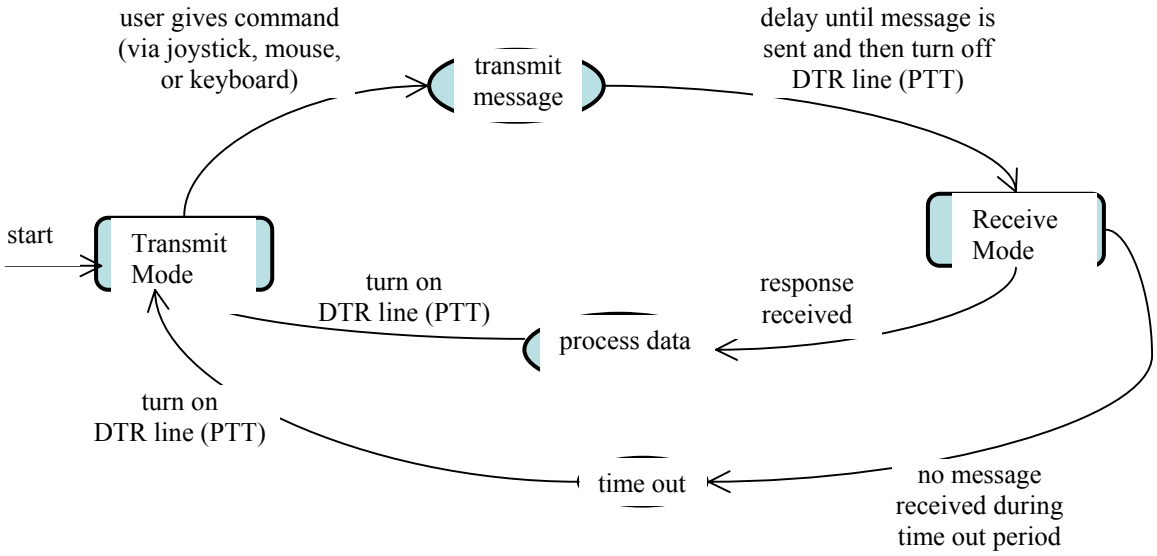


Figure 5.19: Base station state diagram

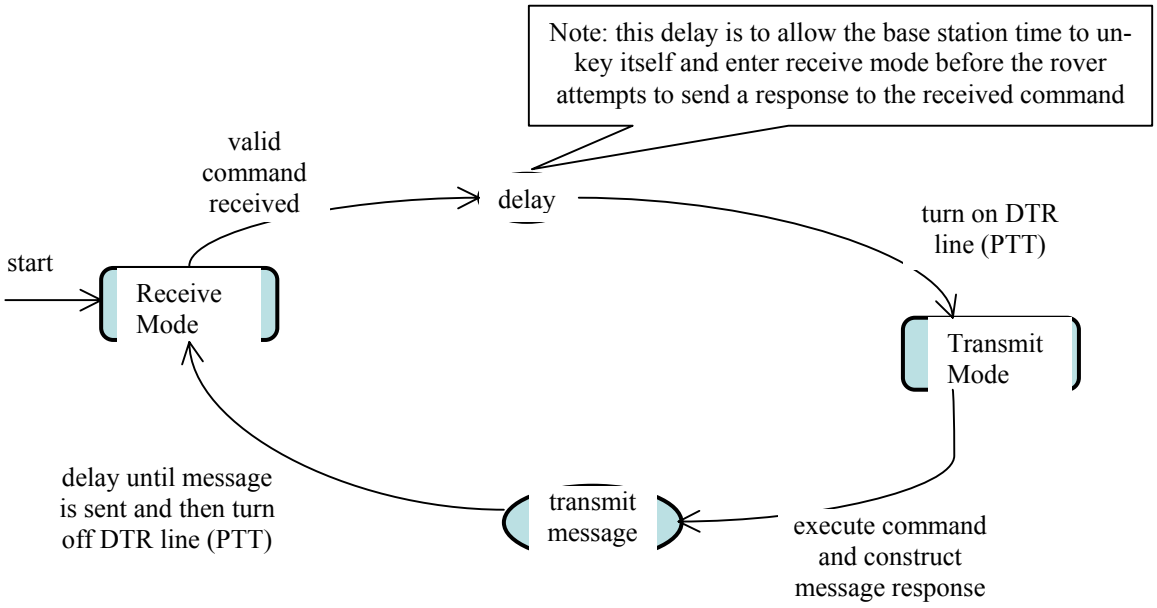


Figure 5.20: Rover state diagram

Once some organized flow of communication was established, commands were assigned to ASCII characters for control of the rover. Table 5.5 details each ASCII character command and its expected response from the rover. In total, twenty-nine ASCII characters were assigned for commands. All of the communication is based on the assumption that each receiving party knows what type of information that it will be receiving prior to receiving the information. The server always expects a one-character command. The client determines what type of information it will be receiving based on the last command that was sent to the server. In the event that the client does not receive a response, a time-out may occur.

Table 5.5: Commands and responses

Command Sent from Base Station	Expected Return String from Rover
A – Query accelerometer	AXXYYZZ – where A is the echo of the command XX is two numerical characters representing the X-axis acceleration, YY is two numerical characters representing the Y-axis acceleration, and ZZ is two numerical characters representing the Z-axis acceleration
B – Query compass	BXXX – where B is the echo f the command and XXX is three numerical characters representing the direction in degrees from North that the rover is facing
C – Query tilt sensor	BXXYY – where C is the echo f the command and XX is two numerical characters representing the roll in degrees and YY is two numerical characters representing the pitch in degrees
D – Update sensor status	D – The status of the sensors is contained in the last 4 bits of the hamming encoded message
a – Reverse speed 4	a – echo of command
b – Reverse speed 3	b – echo of command
c – Reverse speed 2	c – echo of command
d – Reverse speed 1	d – echo of command
e – Stop	e – echo of command
f – Forward speed 1	f – echo of command
g – Reverse speed 2	g – echo of command
h – Forward speed 3	h – echo of command
i – Forward speed 4	i – echo of command
j – Left 3	j – echo of command
k – Left 2	k – echo of command
l – Left 1	l – echo of command
m – Straight	m – echo of command
n – Right 1	n – echo of command
o – Right 2	o – echo of command
p – Right 3	p – echo of command
q – Turn camera ON	q – echo of command
r – Turn camera OFF	r – echo of command
s – Turn compass ON	s – echo of command
t – Turn compass OFF	t – echo of command
u – Turn tilt sensor ON	u – echo of command
v – Turn tilt sensor OFF	v – echo of command
w – Turn accelerometer ON	w – echo of command
x – Turn accelerometer OFF	x – echo of command
# – Ping rover	# – echo ping

5.5.3 Coding Layer

The Coding Layer is responsible for encoding and decoding data for transmission and adding header bytes to the message. Hamming block codes (11,7) were used for this process. The hamming block code allows the correction of one bit error per block of eleven bits. It encodes seven information bits into eleven by adding four additional bits for error correcting. The seven information bits represent an ASCII character. The challenge in designing this layer is that the RIA and ROA used two different languages, VB and C respectively. The same algorithms had to be implemented in two different languages. When the encoding process was complete, two header bytes were appended to the beginning of each message. This marked the beginning of a message and allowed the receiver to discriminate between random information received due to noise and an actual message sent by the transmitting party.

Aside from encoding, a necessity in this layer was to segment the message into bytes. An 11-bit message is segmented into two bytes such that the first byte contained the lower 8 bits, and the second byte contained the upper 3 bits and followed by 5 zeros to fill the byte. The 5 zeros at the end of the bits contain no information and add inefficiency to the system. In fact, when the two header bytes are added to this message, the message is then 4 bytes long. The two header bytes contain no information and are only there to mark the beginning of a message. This, combined with the 5 zero filled bits, adds to 21 bits that contained no information out of 32 that are sent (two bytes = 16, plus 5 zero filled bits, total 21). If out of the 11 information bits sent, only seven of them represent the ASCII character, it can be seen that 32 bits were sent but 25 of them contain no information. Though this inefficiency is a problem, the error-correcting capability and header bytes add efficiency to the system by

correcting errors and discriminating between noise and actual message transmissions. Having a baud rate of 1200 also combats the inefficiency. The relatively high speed of transmission means that it takes well under one second to send 32 bits when the rate of transmission is 1200 bits per second.

Longer messages needed additional segmentation. An example is the accelerometer data string. As discussed in the Application Layer section, the data string sent by the rover contains seven characters. The first character is the echo of the command, and the next six characters, in groups of twos, are numerical values for the X, Y, and Z-axes of acceleration. Each character is encoded into eleven bit blocks, totaling 77 encoded bits. These seven blocks are segmented into 10 bytes (10 bytes contain 80 bits), and the last three bits are set to zero. The encoded blocks are segmented as represented in Figure 5.21.

Encoded Blocks	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block7	0
Byte segmentation	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10

Figure 5.21: Block Segmentation

The header is again appended to the beginning of the message string, giving a message as represented in Figure 5.22.

Header byte 1	Header Byte 2	Ten bytes message

Figure 5.22: Message Representation

5.5.4 Physical Layer

The Physical Layer deals with the actual transmission of bits of data between the base station PC and the rover's LogicFlex. The Physical Layer starts at the serial port of the base station PC, which connects via RS232 to the base station modem. Although RS232 is a full-

duplex standard, the modems are not full-duplex capable, therefore half-duplex operation was implemented with the RS232 standard.

When a message is sent, bits of data enter the modems serially. They are converted to the proper logic and are modulated using FSK at 1200Hz and 2200Hz. The DTR line on the standard serial port was used for changing the state of the modems between transmit and receive. The DTR level propagates through the modem and is used to key and un-key the microphone of the base station transceiver.

After the message was modulated by the modem, and the microphone was keyed, the base station transceiver frequency modulated the FSK signal. The base station transceiver operated on 146.580MHz in the amateur radio band. The signal was then radiated by the shortened monopole antenna at the base station.

The rover also used a shortened monopole antenna for receiving. The rover's FM transceiver demodulated any incoming signal and returned it to baseband. The baseband signal entered the rover's modem and was demodulated into bits. The bits were converted to the proper logic level and then entered the serial port of the LogicFlex. When a message was transmitted from the rover to the base station, the same techniques and equipment were used for communication. Figure 5.23 is a block diagram of a message being transmitted from the base station PC to the rover's LogicFlex over the Physical Layer.

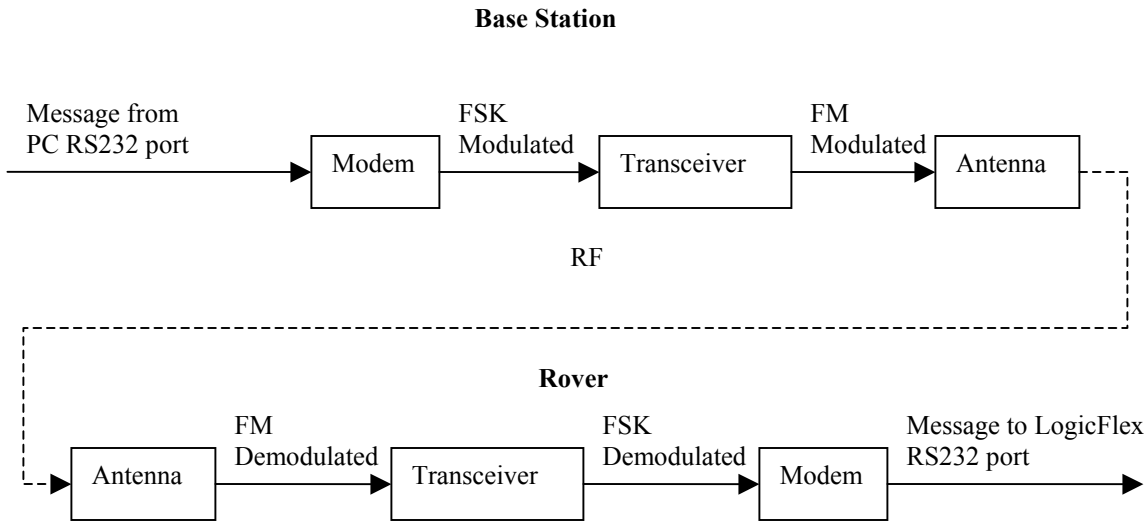


Figure 5.23: Physical Layer Block Diagram

5.6 Deviations From Proposed Design

5.6.1 Opto Isolators

During the design process, it was discovered that the logic level of 3.3V from the LogicFlex was not enough to drive the H-Bridge motor controller. To compensate for this, we initially used a buffer IC. The buffer IC was powered by 5V, and had an input threshold for a logic 1 below 3.3V. So, when 3.3V was applied to an input channel, the corresponding output channel would output 5V. After implementation of the buffer, it was found that it was unsafe for the LogicFlex's parallel port to be electrically connected to the high voltage H-Bridge through the buffer. Hence the decision to implement the opto-isolators that optically separated the logic flex from the H-Bridges.

5.6.2 Battery Issues

As proposed, the rover was going to be powered by a 10.3V NiMH battery pack that was secured to the chassis of the rover. Since the camera and transmitter required 12V to operate, we obtained a second battery. This second battery was a 12V Lead Acid battery, with approximately 3 Amp Hours of charge. During the testing phase of the project, the heavy battery was causing too many limitations in the driving ability of the rover. More specifically, the rover could only drive on level surfaces, not up an incline. Next, we obtained a Lithium Ion battery pack, which was much lighter, and provided 14.4V with slightly less total charge. Again, during testing this battery pack experienced a thermal overload and exploded. Finally, it was decided that four individual lightweight NiMH battery packs would each power different subsystems on the rover.

Section 6: Safety Considerations

6.1 Amateur Radio Rules and Regulations

The project's communication system and television relay signal both take advantage of the Amateur Radio Service for operation at greater distances. This "Service" was created by the Federal Communications Commission (FCC) to fulfill the need for a pool of experts who could provide backup emergency communications in times of need. All frequencies of the amateur radio band are shared. Since no station has the exclusive right of any one channel basic rules must be followed to ensure that users operate and conduct themselves in a safe cooperative manner so that the service can be utilized by all. As a result, the FCC requires that all operators using the service be licensed. Acquisition of this license demonstrates that the operator has a grasp of the basic regulations, operating practices, and electronics theory of amateur radio, with a focus on VHF and UHF applications. Title 47, section 97 of the Code of Federal Regulations (47 CRF 73) outlines the full set of rules governing the Amateur Radio Service. In particular, the rules governing station identification are covered in section 97.119a of the CFR. They state that an amateur station must identify itself at least every ten minutes (2). To comply with this regulation, functions were written into both RIA and ROA to automatically transmit the licensed operator's call sign every ten minutes. All other regulations for communication were followed accordingly.

6.2 Electrostatic Discharge (ESD) Considerations

ESD occurs when electrical charge builds up on a surface (primarily a person) and then jumps to another object in the form of a shock. Although ESD may only seem like a small shock incapable of any real damage, it is a major contributor to the death of many

electronic components. For protection to both individuals and the project from ESD damage, anyone who was working with any electrical component of the rover or base station needed to wear a static strap around their wrist. The static strap was an elastic band with piece of metal on the inside that makes contact with the skin. The bracelet was connected to a high impedance wire that was attached to anything that is part of the earth ground, particularly the earth ground connection on the DC power supply. When someone was wearing the static strap, any charge that was created on their body was transferred directly to ground, rather than building up and discharging violently to a sensitive component.

Section 7: Conclusion

There are many practical applications for a semi-autonomous rover. The versatility of such a rover to complete tasks a human being would not ordinarily be able to under certain conditions is extensive. Tasks that it could perform include, but are not limited to, planetary exploration, mining, and assisting SWAT teams. This project, Digital Wireless Data Communication with a Semi-Autonomous Rover, has demonstrated the use of a rover to act as a sensor relay platform that can be driven by a controller at a base station. The rover is controlled wirelessly from a base station and takes measurements of its surroundings using the various sensors mounted on it.

The project has been an all-inclusive endeavor at building a wireless data communication system and sensor platform from the ground up. Using an existing rover body, microcomputer, and sensors, the interfacing was custom built to fulfill the specific needs of the project. Through circuit design, the various subsystems of the rover were integrated together on a platform. Motor control was first established, allowing the rover to be mobile. The sensors were individually tested and calibrated, and then they were integrated onto the rover with the microcomputer.

Modems were constructed to relay data collected by the rover back to the base station. The modems were originally hardwired together through a cable. This link was broken, and they were connected to amateur radio transceivers to broadcast messages wirelessly. In concert with the modems a complete set of protocols were defined to govern the patterns of communication. The protocols definitions included error correction through Hamming code and random signal blocking. Finally, different modules were programmed onboard the rover

and at the base station to control the robot's driving, data acquisition, and data communication.

This project illustrates one of the many roles in which a semi-autonomous robot can perform. Not only has the project included hardware design and circuit interfacing, but also software design, integration of software and hardware, and the development a complete set of communication protocols. All of these factors contribute to a comprehensive design and unique mobile wireless data acquisition system.

Section 8: Time Schedule

The evolution of the project has been a continuing process. From the summer of 2003 the first hardware pieces were acquired and much research was done into FCC licensing, Visual Basic programming, and the operation of the sensors and microcomputer. Initially, in the Fall 2003 semester, most of the work was done on interfacing the microcomputer, sensors, and rover motors both through hardware and software. Also during this time, the communication systems were designed and built. The first semi-functional prototype of the rover was completed around the time of the semester break. From the conclusion of the break most of the work was performed on the software side of the project and in changes to the existing rover hardware, as older designs became unfeasible. Figure 8.1 further illustrates the time schedule that the project adhered to.

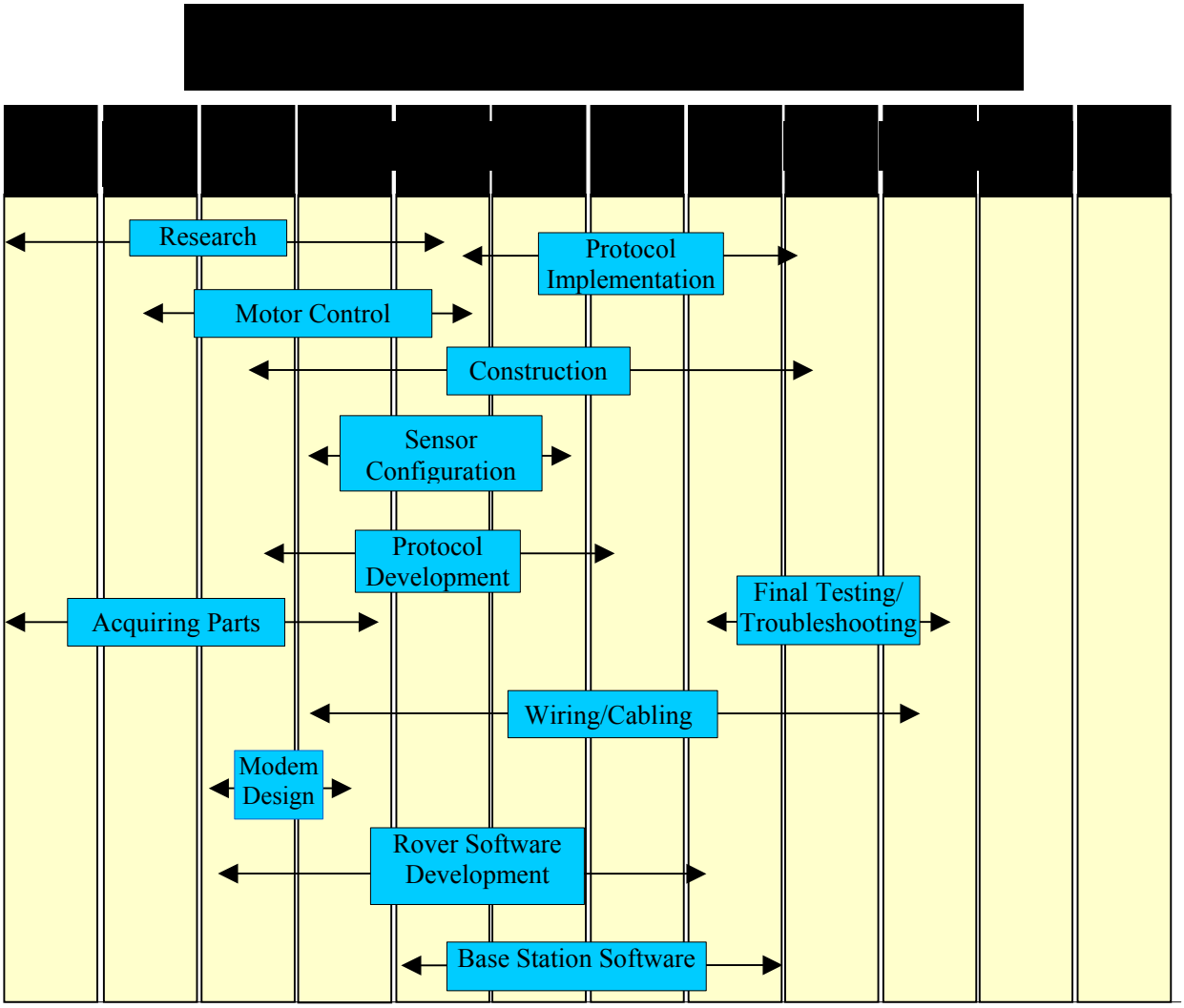


Figure 8.1: Project time schedule

Section 9: Budget

During the course of the project a number of items that were in the original rover design were not included in the final rover design. The budget reflects all parts used at any time for the project. Many of the parts used in the project were from the engineering labs at Temple University, and some parts were donated from outside sources. (An asterisk reflects the parts that were donated from private party individuals). Table 9.1 shows the budget in full.

Table 9.1: Project budget

1	Tilt Sensor*	$149.00	0
1	Precision Navigation TCM 2 Electronic Compass*	$800.00	0
2	TI H-Brigde Motor Controller*	$4.00	0
2	Mxcom MX614 Modem Chip*	$8.00	0
2	Maxim IC MAX232 Chip*	$6.50	0
1	Remote Control Car Chassis*	$10.00	0
5	Plastic Box	$10.00	$10.00
1	Logitech Wingman Extreme Joystick*	$35.00	0
1	Panasonic 14.4 Volt Li Ion Battery Pack*	$45.00	0
1	10.3 Volt Nickel-Metal Hydride Battery Pack*	$25.00	0
1	other Nickel-Metal Hydride Battery Pack*	$25.00	0
46	1.2 Volt Nickel-Metal Hydride Battery cells	$46.00	$46.00
2	Battery Charger	$100.00	0
1	PC Electronics Video Camera and Transmitter*	$175.00	0
1	Kenwood TH-25AT Transceiver*	$280.00	0
1	Kenwood TM-721A Transceiver*	$650.00	0
4	Shortened monopole antennas	$80.00	0
1	Borland C/C++ 4.52*	-	0
1	Visual Basic 6.0*	$450.00	0
1	LM323 Voltage Regulator	$1.79	0
3	LM317 Voltage Regulator	$1.80	$1.80
2	Linear Technologies LT1129 Shutdown Regulator	$2.50	$2.50
1	Opto-isolator	$5.00	$5.00
	Assorted components (resistors, capacitors, solder..etc)*	$100.00	$100.00
	Totals	$3,798.59	$165.30

Section 10: Works Cited

1. Cohen, H.B. "A Mobitex wireless modem implementation." *Proceedings of the Fourth International Conference on Signal Processing Applications and Technology* (1993): pt. 1, p 250-2 vol.1

2. "Code of Federal Regulations, Title 47." Federal Communications Commision. <http://wireless.fcc.gov/rules.html> January 17, 2004.

3. "Mars Pathfinder." NASA Jet Propulsion Laboratory. Pasadena, CA. <http://www.jpl.nasa.gov/news/fact_sheets/mpf.pdf> March 2, 2004.

4. NASA Jet Propulsion Laboratory. Pasadena, CA. <http://marsrovers.jpl.nasa.gov/home/index.html> March 2, 2004.

5. Remotec. <www.remotec-andros.com> March 5, 2004.

6. Rustanbegovic, Adi. "Satlink Transmitter Serial Communications Protocol." Sutron.com. <http://www.sutron.com/downloads/SatlinkProtocolDocumentation/SatlinkProtocolDefinition.html> October 23, 2003.

Section 11: Bibliography

"Code of Federal Regulations, Title 47." Federal Communications Commision. <http://wireless.fcc.gov/rules.html> January 17, 2004.

Technical document specifying the rules and regulations for licensed amateur radio operation.

Cohen, H.B. "A Mobitex wireless modem implementation" *Proceedings of the Fourth International Conference on Signal Processing Applications and Technology* (1993): pt. 1, p 250-2 vol.1

AT&T Microelectronics has implemented two-way, wireless communications using the Mobitex packet radio network in its V.32bis/V.17 modem chipset. Included is a description of synchronization, error coding, interleaving, and scrambling. Also presented is the GMSK modulation and detection required to interface to a radio transceiver.

Ekedahl, Michael, and William Newman. *Programming with Microsoft Visual Basic 6.0: Object-Oriented Approach.* Course Technology, Cambridge, MA. 1999. p 1-612

The text is an introductory course in Visual Basic. It provides a programmer the tools necessary to learn the Visual Basic environment and the procedures necessary for developing Windows based applications. The text assumes that the programmer has little or no prior programming experience. It assumes familiarity with basic Windows 95, 98, or higher concepts.

Farrell, Joyce. *Object-Oriented Programming Using C++.* Course Technology, Cambridge, MA. 1998. p 1-400.

Designed for a second programming course, Farrell assumes that the programmer has a basic programming background. Programmers that have not programmed in C or C++ prior to using this text may need to consult a more introductory text. The aim of the text is to provide an understanding of object-oriented concepts as they apply to programming, and the ability to use these concepts to develop C++ programs.

Hanish, Andrew A., and Tharam S. Dillon. "Communication Protocol design to facilitate re-use

Based on the Object-oriented Paradigm." *Mobile Networks and Applications archive* 2.3 (December 1997)

Object-Oriented (OO) modeling principles are applied to networking protocols to explore the potential for producing re-useable software modules by discovering the underlying generic class structures and behavior of the protocols. Petri Nets (PNs) are used to derive re-useable model elements. The utilization of PNs in the context of object based modeling allows for isolation of the behavioral characterization of objects

into a separate design plane, treated as a meta-level object control. To illustrate the modeling concepts, the paper addresses the problem of inter-layer communication among multiple protocol entities (PEs), assuming the standard ISO/OSI Reference Model.

JK Microsystems Inc.. *LogicFlex User's Manual v1.3*, JK Microsystems Inc., Davis, CA. 2002. p1-28.

This document is the user's manual for the JK Microsystems LogicFlex i386 embedded microprocessor system. It explains, in detail, how to access the operating system on the microcomputer, and how to use its many features.

La Porta , Thomas F., Krishan K. Sabnani, and Richard D. Gitlin. "Challenges for Nomadic Computing: Mobility Management and Wireless Communications." *Mobile Networks and Applications archive.* 1.1 (August 1996): p 3-16.

This paper describes methods in which nomadic robots have been used and designed. The nomadic computing environment is characterized by mobile users that may be connected to the network via wired or wireless means. Three general techniques for addressing these challenges are considered: (1) asymmetric design of applications and protocols, (2) the use of network-based proxies which perform complex functions on behalf of mobile users, and (3) the use of pre-fetching and caching of critical data. We examine how these techniques have been applied to several systems, and present results in an attempt to quantify their relative effectiveness.

Lawlor, Steven C.. *The Art of Programming: Computer Science with C.* PWS Publishing Company, Boston, MA. 1997. p 1-567.

The text is an introductory course in C. The goal is to teach program development—how to efficiently and cleanly develop a computer solution for a problem. In order to fulfill this objective, mastery of language syntax and usage is necessary. The book combines the lessons of language usage and syntax with lessons that aim to demonstrate efficient development of computer solutions for problems.

Low, Kian Hsiang, Wee Kheng Leow, and Marcelo H. Ang, Jr. "A hybrid Mobile Robot Architecture with Integrated Planning and Control." *International Conference on Autonomous Agents archive. Proceedings of the first international joint conference on Autonomous agents and multiagent systems*. Session 3B (2002): p 219- 26.

Two basic approaches have emerged from mobile robot research efforts: deliberative vs. reactive. These two approaches can be distinguished by their different usage of sensed data and global knowledge, speed of response, reasoning capability, and complexity of computation. This paper describes a method for goal-directed, collision-free navigation in unpredictable environments that employs a behavior-based hybrid

architecture with asynchronously operating behavioral modules.

"Mars Pathfinder." NASA Jet Propulsion Laboratory. Pasadena, CA. <http://www.jpl.nasa.gov/news/fact_sheets/mpf.pdf> March 2, 2004.

Website dedicated to NASA's Pathfinder project. It explains many details about the project and provides visual media of the Sojourner.

Matsuki, H., T. Ishiguro, R. Yamamoto, E. Ueno, and T. Oono. "PHS 64 kbit/s data transmission technique." *NTT R & D* 48.2 (1999): p 209-16

This paper proposes a novel data transmission technique over the Personal Handy Phone System (PHS) 64 kbit/s unrestricted digital bearer. It includes two protocols for error control. The error control procedure itself enables higher transmission rates under worse transmitter conditions by applying the acknowledgment number notification method.

McKinney, Alfred L. "A Recent Radical Graphical Approach to Programming" *The Journal of Computing in Small Colleges.* 18.6 (June 2003)

This paper provides a very brief overview of the history of graphical approaches to programming. A more detailed description is given of SoftWIRE, a recent radical graphical approach to programming, which has much potential. SoftWIRE utilizes Visual Basic (VB) in the Integrated Development Environment (IDE) Microsoft Visual Studio. NET.

NASA Jet Propulsion Laboratory. Pasadena, CA. <http://marsrovers.jpl.nasa.gov/home/index.html> March 2, 2004.

Website dedicated to NASA's most recent Mars rovers Spirit and Opportunity. It explains many details about the rovers and provides visual media.

Panagiotis, Louridas. "Some Guidelines for Non-repudiation Protocols." *SIGCOMM Computer Communication Review archive*. 30.5 (October 2000): p 29-38.

Non-repudiation protocols aim at preventing parties in a communication from falsely denying having taken part in that communication; for example, a non-repudiation protocol for digital certified mail should ensure that neither the sender can deny sending the message, nor the receiver can deny receiving it. The guidelines are derived by examining a series of non-repudiation protocols that descend from a single ancestor.

Paul, S., E. Ayanoglu, T.F. La Porta, K. Chen, K.E. Sabnani, and R.D. Gitlin. "An Asymmetric Protocol for Digital Cellular Communications." *Proceedings. IEEE INFOCOM '95. The Conference on Computer Communications. Fourteenth Annual*

Joint Conference of the IEEE Computer and Communications Societies. Bringing Information to People (1995): pt. 3, p 1053-62 vol.3

This paper describes the design, validation, implementation and performance of an asymmetric link-layer protocol for a wireless link. The key ideas in the design consist of placing the bulk of the intelligence in the base station as opposed to placing it symmetrically, in requiring the wireless terminal to combine several acknowledgments into a single acknowledgment to conserve power.

Remotec. <www.remotec-andros.com> March 5, 2004.

Website containing information regarding the Andros Series SWAT Team Rovers.

Rustanbegovic, Adi. "Satlink Transmitter Serial Communications Protocol." Sutron.com. http://www.sutron.com/downloads/SatlinkProtocolDocumentation/SatlinkProtocolDefinition.html. 23 October, 2003

Sutron Inc. has developed a satellite communications protocol to use with their meteorological measuring device called the Satlink. This document is the formal documentation for the description of the protocols used to communicate with the satellites

Schmill, Matthew D., Michael T. Rosenstein, Paul R. Cohen, and Paul Utgoff. "Learning What is Relevant to the Effects of Actions for a Mobile Robot" *Proceedings of the second international conference on Autonomous agents*. (1998): p 247-253.

This conference explains the moral and ethical aspects of mobile robots. It describes the dangers of allowing a semi-autonomous mobile robot being allowed to work amongst humans in dangerous situations.

Shionozaki, Atsushi, and Mario Tokoro. "Control Handling in Real-time Communication Protocols." *Applications, Technologies, Architectures,and Protocols for Computer Communication archive*. (1993): p149-59.

This article concentrated its efforts towards designing protocols that work in "Real Time". It also describes steps taken by a number of engineers to successfully deploy a number of protocols that issued the problem of control handling.

Sklar, Bernard. *Digital Communications: Fundamentals and Applications: 2nd Edition.* Pearson Education Asia, Delhi, India. 2001. p 305-346.

Textbook information regarding rules and procedures for designing digital communication protocols. It explains how to start the process by making flow diagrams, and describes steps to take when implementing the newly designed protocols.

Section 12: Appendix A – Schematic Diagrams

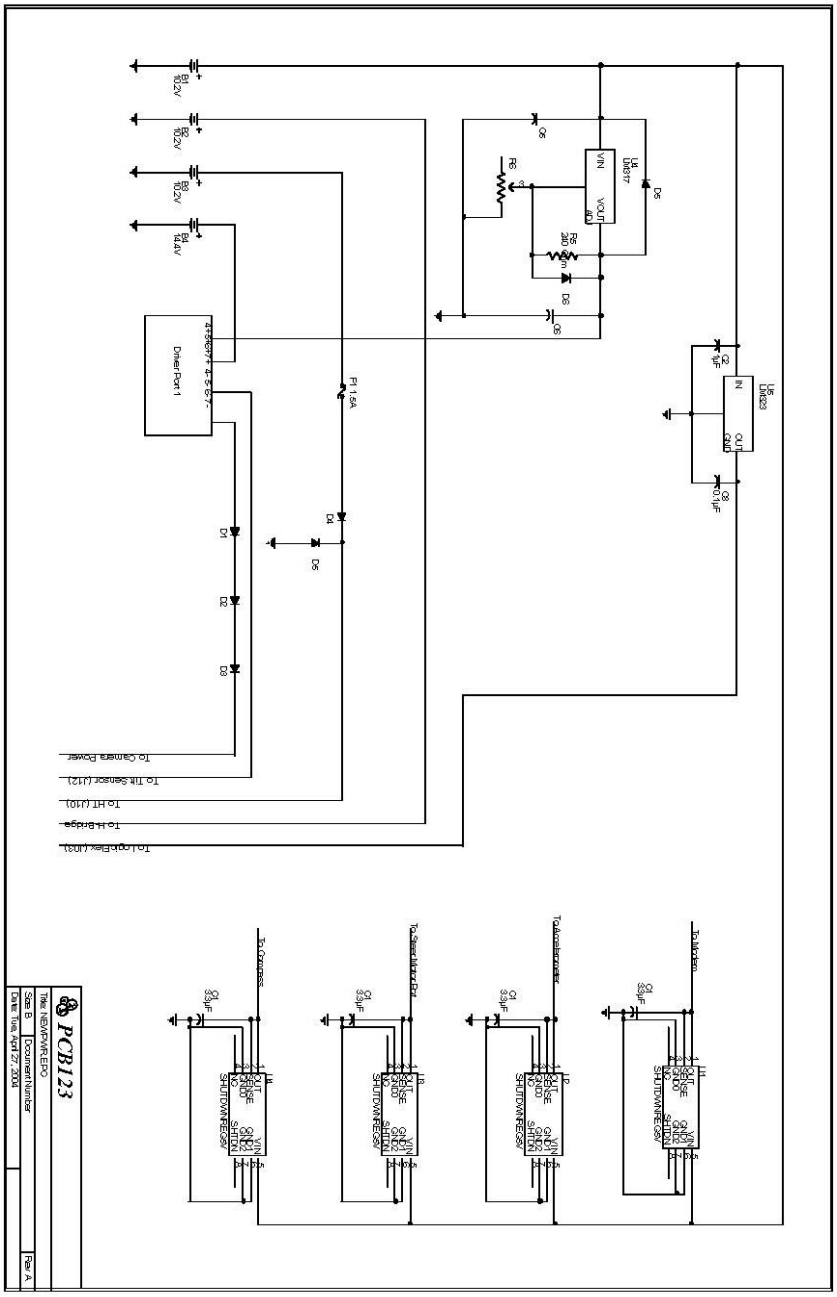


Figure 12.1: Schematic of the circuit for delivering power to the various components of the Rover

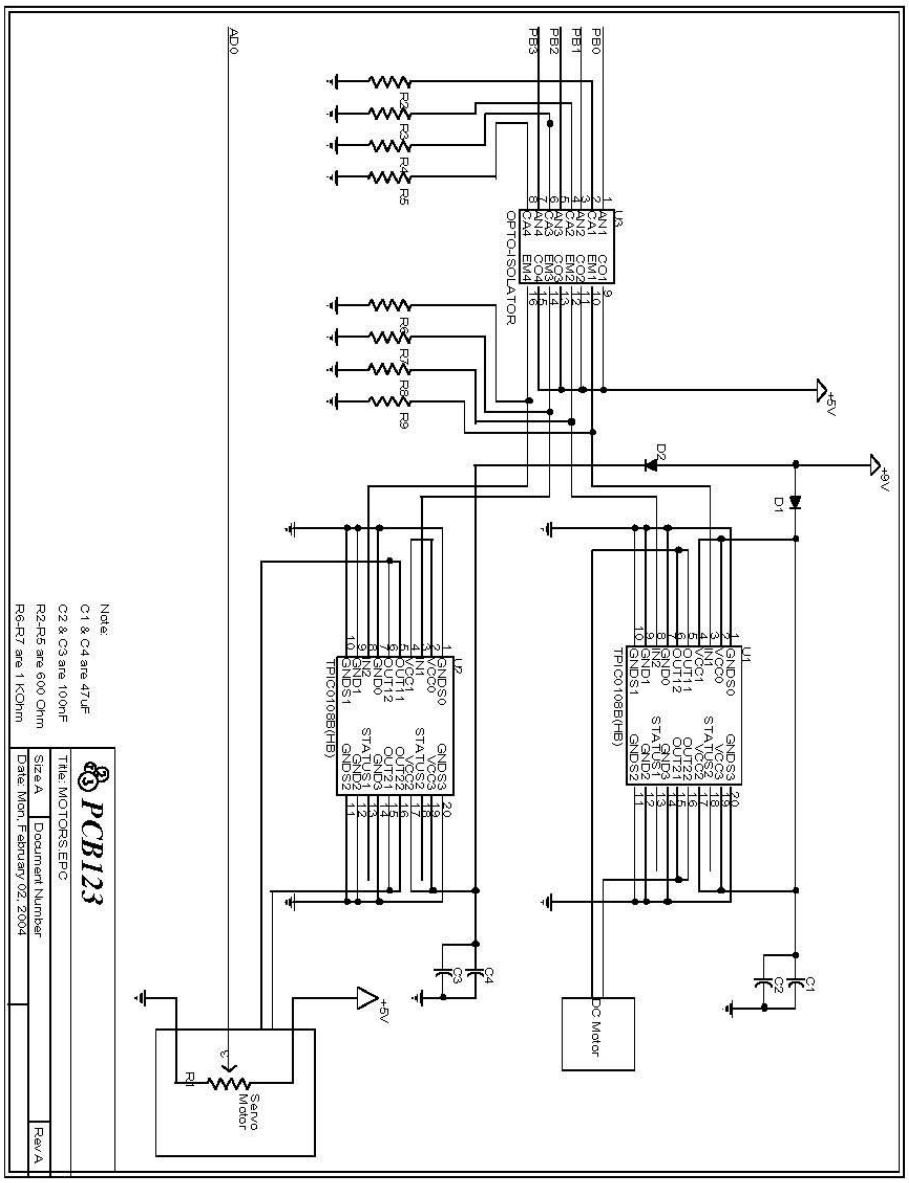


Figure 12.2: Schematic of the circuit that drives the motors

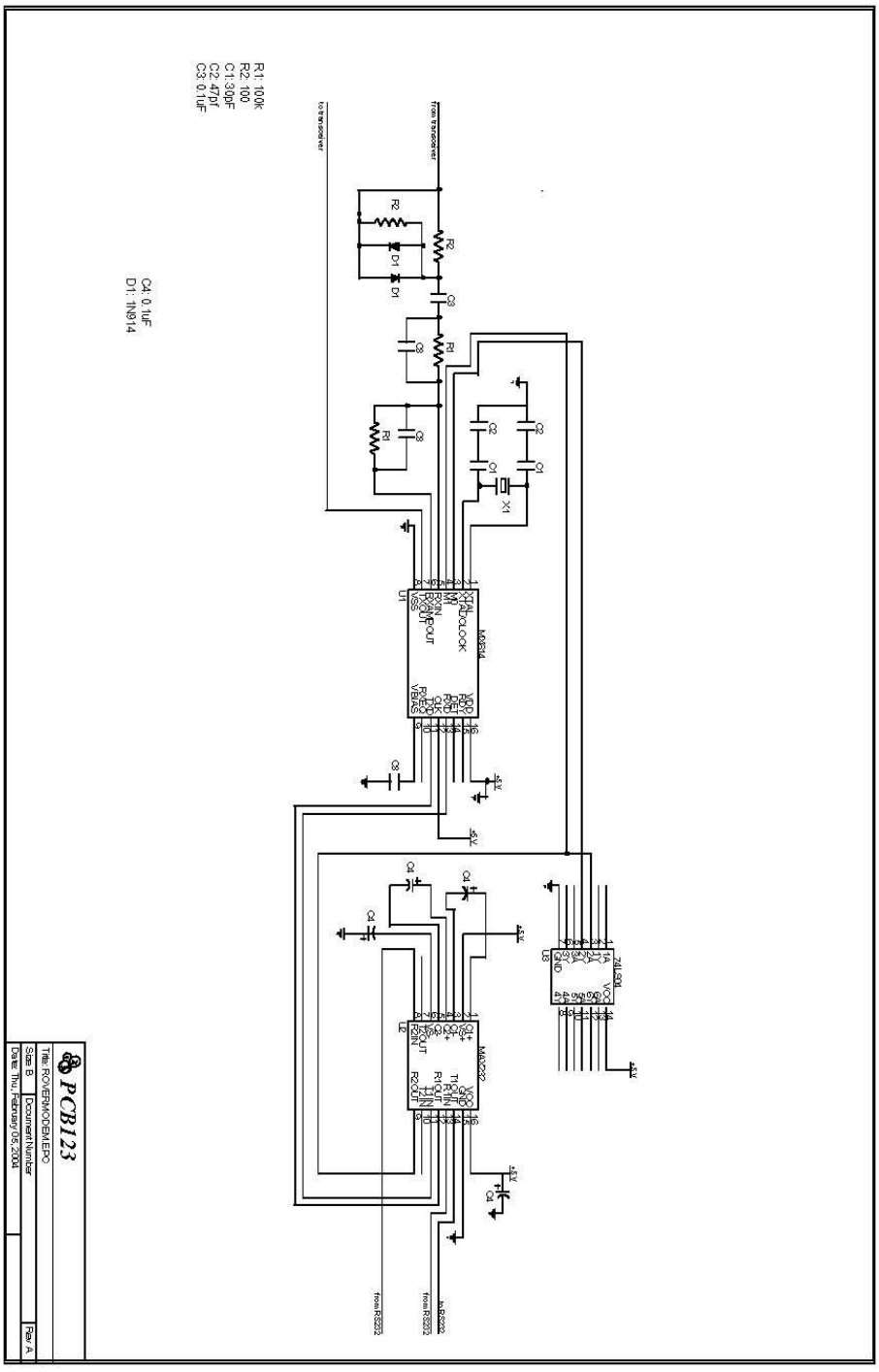


Figure 12.3: Schematic of the modem on the Rover

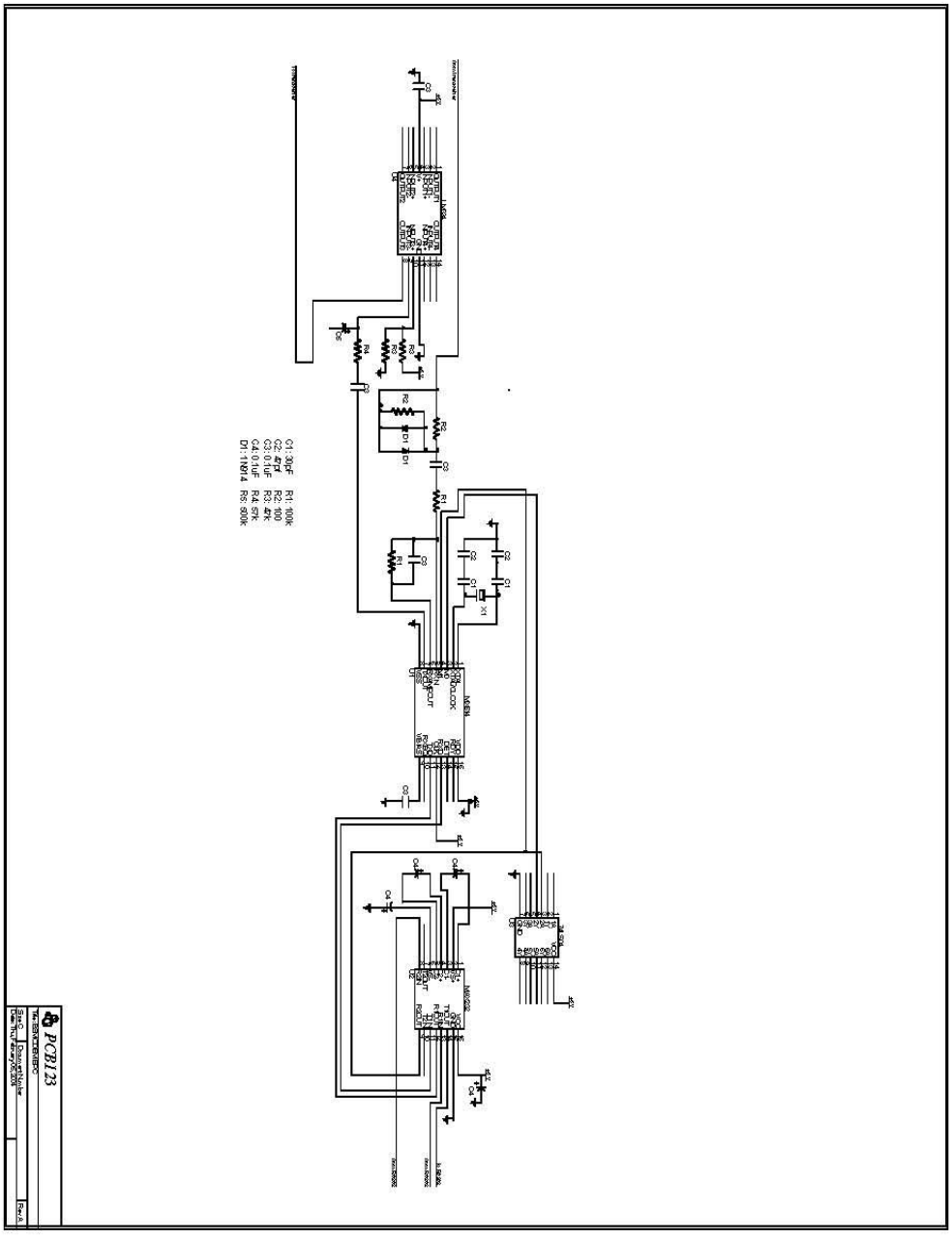


Figure 12.4: Schematic of the modem at the base station

Section 13: Appendix B – Pinouts and Wiring

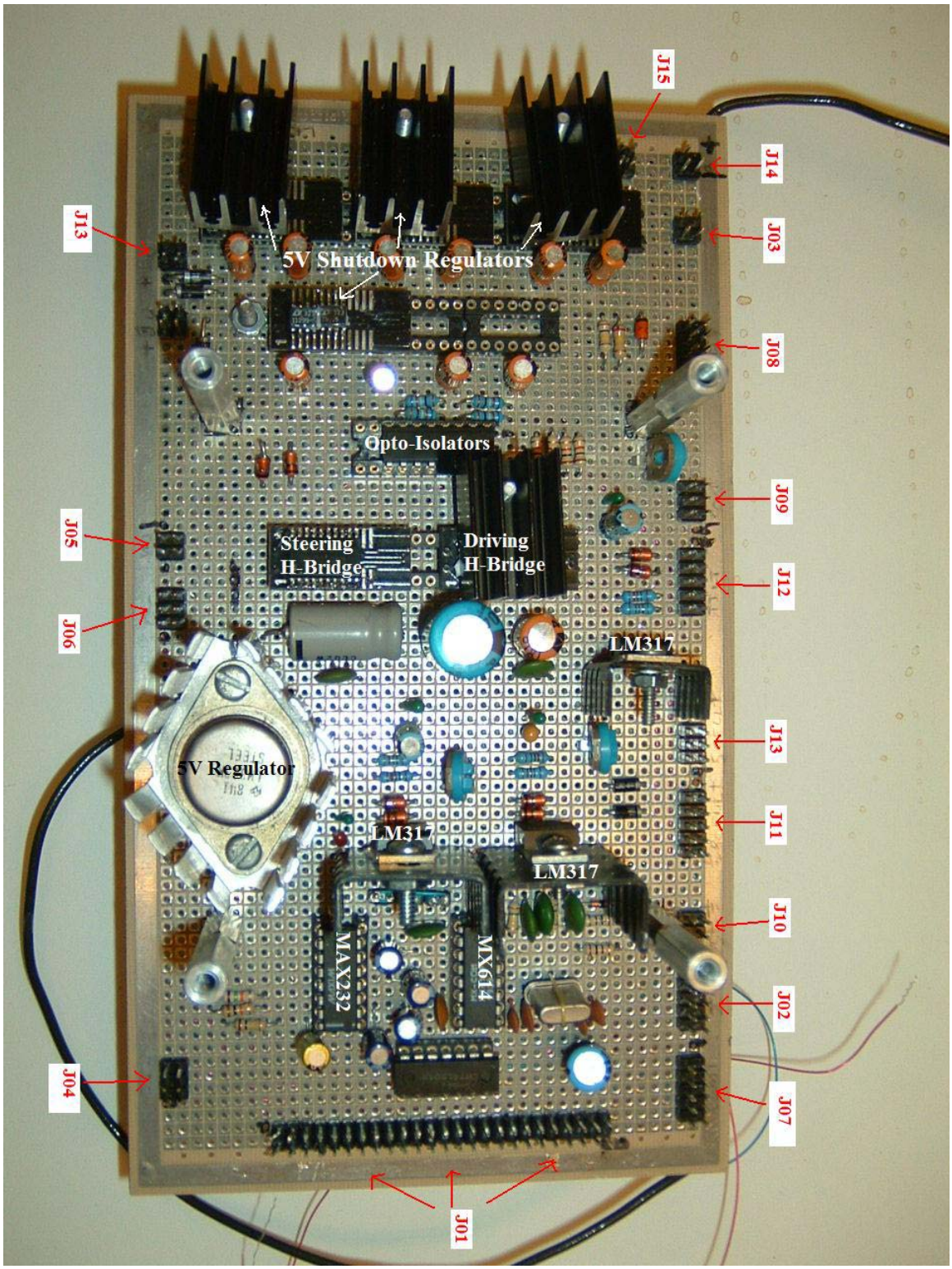


Figure 13.1: Diagram of connector locations and devices on board

Table 13.1: Pinout for the Cable connecting the Protoboard to the LogicFlex Parallel and ADC Ports

I/O Ports and ADC to Protoboard

Connector	Value	Pin	Wire Color	Pin on Proto. (J01)
J7	Vref	1	Red/Grn	26
	AD0	2	Blk/Red	1
	AD1	3	Blue	2
	GND	4	Grn/White/Blk	27
J8	Vref	1	Blue/White	28
	AD2	2	White	3
	AD3	3	Orng/Red	4
	GND	4	Grn/White/Blk	29
J16	Vref	1	Orng/Red	30
	AD4	2	Grn/White/Blk	5
	AD5	3	Blue/White	6
	GND	4	Grn/Blk	31
J17	Vref	1	Grn/Blk	32
	AD6	2	Red/Grn	7
	AD7	3	Grn/White/Blk	8
	GND	4	White	33
J6	GND	1	White/Red	50
	3.3V	2	Orng/Grn	19
	GND	3	Blk/White	25
	3.3V	4	Blk/White	43
	GND	5	Grn/Blk	49
	SSTXCLK	6	White/Red	18
	GND	7	Orng/Grn	24
	SSRXCLK	8	Blk/White	42
	SSIORX	9	Red/Grn	48
	SSIOTX	10	Blue/White	17
	PA.7	11	Grn/Blk	23
	PB.7	12	Blk/Red	41
	PA.6	13	Red/Blk	47
	PB.6	14	Orng/Red	16
	PA.5	15	Orng/Red	22
	PB.5	16	Blk/White/Red	40
	PA.4	17	White	46
	PB.4	18	Blue/White	15
	PA.3	19	Red/White	21
	PB.3	20	Orng/Grn	39
	PA.2	21	White	45
	PB.2	22	White/Red	14
	PA.1	23	Orng/Grn	20
	PB.1	24	Blk/Red/White	38
	PA.0	25	White/Red	44
	PB.0	26	Blk/Red	13

NOTE*: All connectors have a black dot next to pin one on the protoboard. Odd pin numbers are along the same row as pin one. Even pin numbers are along the other row.

Table 13.2: Pinout for the Serial Port on LogicFlex to the Protoboard

Com1 to Protoboard

On Logic Flex				On Protoboard	
Connector	Value	Pin	Wire Color	Pin	Connector
J9	DCD(in)	1	Back to Pin 2		J07
	DSR(in)	2	Back to Pin 1		
	RxD(in)	3		3	
	RTS(out)	4	Back to Pin 6		
	TxD(out)	5		5	
	CTS(in)	6	Back to Pin 4		
	DTR(out)	7	Blue	7	
	N/C	8			
	GND	9		9	
	N/C	10			

Table 13.3: Pinout for the Compass to the Protoboard

Compass to Protoboard

On Compass				On Protoboard	
Connector	Value	Pin	Wire Color	Pin	Connector
	+5V	1	ONG		J09
		2			
	GND	3	BLK		
		4			
		5			
		6			
		7			
		8			
	A out	9	WHT/RED		
	D GND	10	BLU/WHT		

Table 13.4: Pinout for the Camera and Transmitter to the Protoboard

Camera andTransmitter to Protoboard

On Transmitter				On Protoboard	
Connector	Value	Pin	Wire Color	Pin	Connector
	+V	1	RED	I	J13
	+V	2	RED	I	
	GND	3	BLK	O	
	GND	4	BLK	O	

Table 13.5: Pinout for the Tilt Sensor to the protoboard

Tilt Sensor to Protoboard

On Tilt Sensor				On Protoboard	
Connector	Value	Pin	Wire Color	Pin	Connector
	+8V	1	RED/GRN	6	J12
	GND	2	BLK/RED	10	
		3			
	X OUT	4	BLU	4	
	Y OUT	5	WHT	2	
		6			
		7			
		8			
		9			
		10			

Table 13.6: Pinout for the Accelerometer to the protoboard

Accelerometer to Protoboard

On Accelerometer				On Protoboard	
Connector	Value	Pin	Wire Color	Pin	Connector
	+5V		Red/Green	1	J04
	GND		Black/White/Red	2	
	X			3	
	Y			4	
	Z			5	
	NC			6	

Table 13.7: Pinout for the LogicFlex's Driver Ports to the protoboard

Driver Ports To Protoboard

On Multi-I/O				On Protoboard	
Connector	Value	Pin	Wire Color	Pin	Connector
J9	DP0+	1			J08
	DP0-	2			
	DP1+	3	BLUE	5	
	DP1-	4	GRN	6	
J18	DP5+	5	BLK	3	
	DP5-	6	ORA	4	
		7			
		8			

Table 13.8: Pinout for the LogicFlex power connector to the Protoboard

LogicFlex Power to Protoboard

On LF				On Protoboard	
Connector	Value	Pin	Wire Color	Pin	Connector
J2	+5V	1	White/Red	1	J03
	NC	2	-	2	
	NC	3	-	3	
	GND	4	Orng/Red	4	

Table 13.9: Pinout for connecting the turning motor to the Protoboard

Turning (Front) Motor to Protoboard

On Turning Motor		On Protoboard	
Value	Wire Color	Pin	Connector
POT		1	J06
GND		2	
+5V		3	
Out 1		4	
Out 2		5	

Table 13.10: Pinout for connecting the drive motor to the protoboard

Drive (Rear) Motor to Protoboard

On Drive Motor				On Protoboard	
Connector	Value	Pin	Wire Color	Pin	Connector
N/A		1			J05
	In 2	2		2	
	In 1	3		3	
		4			

Table 13.11: Power Connector Pinouts

Power Connectors

Connector	Device	+V Pin	GND PIN
J10	Power to HT	2	3
J11	Motor Battery	1	10
J13	L.F Batttery	2	5
J14	Fan Power	2	4
J15	From HT Battery	1	4
J16	Camera Battery	1	4

Table 13.12: Port and Pin assignments for the LogicFlex

Pin Assignments

Port	Bit	Assignment		Pin On Proto. (J01)
B	0	In 1 (Rear HB)		13
	1	In 2 (Rear HB)		38
	2	In 1 (Front HB)		14
	3	In 2 (Front HB)		39
	4			15
	5			40
	6			16
	7	Transceiver PTT		41
A	0	Servo Pot Reg.		44
	1	GPS Reg.		20
	2	Compass Reg.		45
	3	Accellerometer Reg.		21
	4			46
	5			22
	6			47
	7			23
A2D	0	Servo Pot		1
	1	Compass Reading		2
	2	Accelerometer X		3
	3	Accelerometer Y		4
	4	Accelerometer Z		5
	5	Tilt Sensor X		6
	6	Tilt Sensor Y		7
	7	Battery Voltage		8
Driver 0	0	Camera Power		
	1	Camera Power		
	2			
	5	Tilt Sensor Power		

Table 13.13: Pinout for connecting Base Station PC to Modem

Base Station PC to Base Station Modem

On PC				On Modem Case	
Connector	Value	Pin	Wire Color	Pin	Connector
Serial Cable (DB-9)		1		1	Female Serial Jack (DB-9)
	RXD	2		2	
	TXD	3		3	
	DTR	4		4	
	GND	5		5	
		6		6	
		7		7	
		8		8	
		9		9	

Table 13.14: Pinout for connecting Base Station Modem Serial I/O

Base Station Modem Serial I/O

On Modem Circuit Board				On Modem Case	
Connector	Value	Pin	Wire Color	Pin	Connector
B01		1		1	Female Serial Jack (DB-9)
	RXD	2		2	
	TXD	3		3	
	DTR	4		4	
	GND	5		5	
		6		6	
		7		7	
		8		8	
		9		9	
		10		10	

Table 13.15: Pinout for connecting Base Station Modem Transceiver I/O

Base Station Modem Transceiver I/O

On Modem Circuit Board				On Modem Case	
Connector	Value	Pin	Wire Color	Pin	Connector
B02		1		1	Female DIN-V Jack
	RXD	2		2	
	TXD	3		3	
	DTR	4		4	
	GND	5		5	
		6		6	
		7		7	
		8		8	
		9		9	
		10		10	

Section 14: Appendix C – Program Code

Rover Operations Application – Borland C source code:

```
//*************************************************************************
//      Author: John P. Falcone
//      Title: rover_v11.c  (version 11)
//      Program description: This program is stored in the flash memory of
//                      the LogicFlex i3866.  The rover runs this program during
//                      communication with the base station.  Interrupt routine uses real
//                      time clock (RTC) interrupts at a rate of about 1300Hz.
//*************************************************************************

#include <dos.h>
#include <stdlib.h>
#include <string.h>
#include <bios.h>
#include "driver.h"
#include "rs232.c"

#define FALSE 0
#define TRUE (!FALSE)
#define ON 0x01
#define OFF 0x02

#define IRQ10 0x72                      // irq10 interrupt number

#define TMRCFG      0xF834
#define PINCFG  0xF826
#define TMRCON      0xF043
#define TMR1 0xF041

//function prototypes
void turn(int command);
void drive(int command);
void OnOff(int command);
void Acc(void);
void Comp(void);
void Tilt(void);
void Sensor(void);
void Ping(void);
void QuietMode(void);
void DoCommand(int command);
int DeCode(int M[]);
void EnCode_TX(int size, char DataStr[]);
```

```
//end prototypes

//timer variables
unsigned char tlow, thi;
//PWM flags, timers, and variables
char turnFlag = FALSE, driveFlag = FALSE;  //state of turn and drive
//turning timers and variables
int ontimerLR, offtimerLR;
int volts, lowV, highV;
//Drive timers and variables
int ontimerD, offtimerD;
int driveCount = 0;
int DdutyCyc;
char Dmask1, Dmask2;
//Quiet Mode counter
long QuietModeCnt = 234000l;
//counter for transmitting call sign every 10 minutes
unsigned long CallSignTXCnt = 780000l;
char KB3KDM[] = "KB3KDM"; //Amateur call sign -- John P. Falcone
//PTT off delay counter
unsigned int PTTDelay;
//sensor status byte
char status = 0;

//interrupt vector
void interrupt (*oldirq10)(void) = NULL;

//************* Interrupt Service Routine, IRQ10 (slave IR2) *****************
//---
//At a rate of 1300 times per second, this function interrupts the main
//function, executes the code within the interrupt function, and then returns
//to the point in the main from which it interrupted.
//---
void interrupt irq10(void)
{
        disable();                  // disable ints

//decrement PTTDelay Count
        if (PTTDelay > 0)
                --PTTDelay;

//decrement Quiet Mode Count
        if (QuietModeCnt > 0)
                --QuietModeCnt;

//decrement Call Sign TX counter
```

```
        if (CallSignTXCnt > 0)
                --CallSignTXCnt ;
        else   //10 Minutes elapsed -- Transmit call sign
        {  // turn ON Data Terminal Ready -- Push to talk line
                rs_modctrl(RS_WRTMCR,RS_MCRDTR,RS_LINON);
                rs_sndstr(6, KB3KDM);  //transmit call sign
                PTTDelay = 120;    //set delay to turn off PTT line
                CallSignTXCnt = 780000l;  //reset timer for next 10 minute interval
        }

//service turn PWM
        if (turnFlag == ON) //ON state
        {  if (ontimerLR > 0)
                        --ontimerLR;
                else
                {       PutA2DChannel(0);
                        volts = GetA2D();
                        //check to see if turning is done
                        if(volts <= lowV || volts > highV)
                                turnFlag = OFF; //change flag state for OFF
                        else
                                turnFlag = FALSE; //turning complete
                        //turn OFF motor
                        outportb(0x61, inportb(0x61) & 0xF3);  //PortB xxxx00xx
                }
        }
        if (turnFlag == OFF) //OFF state
        {       if (offtimerLR > 0)
                        --offtimerLR;
                else
                {       //reset duty cycle counters
                        ontimerLR = 25;
                        offtimerLR = 25;

                        turnFlag = ON; //change flag state for ON
                        //ON part of duty cycle
                        if (volts <= lowV)  // turn right
                                outportb(0x61, (inportb(0x61) | 0x08) & 0xFB);
                                                //PortB xxxx10xx
                        else     // turn left
                                outportb(0x61, (inportb(0x61) | 0x04) & 0xF7);
                                                //PortB xxxx01xx
                }
        }

//service drive PWM
```

```
        if (driveFlag == ON) //ON state
        {       if (ontimerD > 0)
                        --ontimerD;
                else //ON complete, goto OFF part of duty cycle
                {       driveFlag = OFF;
                        //turn OFF motor
                        outportb(0x61, inportb(0x61) & 0xFC);  //PortB xxxxxx00
                }
        }
        if (driveFlag == OFF) //OFF state
        {       if (offtimerD > 0)
                        --offtimerD;
                else //OFF complete, back to ON part of duty cycle
                {        if (driveCount > 0)
                        {  --driveCount;
                                driveFlag = ON;  //change flag state for ON
                                //reset duty cycle timers
                                ontimerD = DdutyCyc;
                                offtimerD = 50 - DdutyCyc;
                                //turn ON motor
                                outportb(0x61, (inportb(0x61) | Dmask1) & Dmask2);
                        }
                        else //stop after 39 duty cycles (about 1.5 seconds)
                                driveFlag = FALSE;
                }
        }

        outportb(TMR1, tlow);           // reload timer LSB
        outportb(TMR1, thi);            // reload timer MSB

        outportb(0xA0,0x20);            // send EOI to PIC2
        outportb(0x20,0x20);            // send EOI to PIC1

        enable();                       // re-enable ints
}

//******** configure timer and start the timer for interrupts ***************
//argument i determines the frequency of interrupts
void start_timer(unsigned int i)
{  unsigned char tmp;
  //Setup timer -- timer 1: mode 0
        tmp=inportb(TMRCFG);
        tmp &= ~0x08;                   // connect gate1 to Vcc
        tmp |= 0x04;                    // connect CLKIN1 to TMRCLK1 (1.19 MHz)
        outportb(TMRCFG,tmp);
        outportb(TMRCON,0x70);    // Counter1, rd/wr LSB,MSB, mode 0, binary count
```

```
        thi=i>>8;
        tlow=i&0xFF;
        outportb(TMR1,tlow);          // write LSB to count value
        outportb(TMR1,thi);           // write MSB to count value
        return;
}

//************************** stop timer ************************************
void stop_timer(void)
{
        return;
}

//********************** main rover routine ********************************
main ()
{
        char input_buffer[1024],output_buffer[1024]; //allocate buffers
        int PIC1mask=0,PIC2mask=0;          //define vars and set default values
        int command = 0, M[3] = {0}, i = 0;
        char HF1 = FALSE, HF2 = FALSE;

        //open port 1 with 1200 baud, no parity, 8 data bits, 1 stop bit,
        //a 1024 byte input buffer and a 1024 byte output buffer
        if (rs_initport(RS_PORT1,RS_B1200,RS_NOPAR,RS_DBIT8,RS_SBIT1,
                1024U,input_buffer,1024U,output_buffer) <= 0)
                return 0;

        //PTT False
        rs_modctrl(RS_WRTMCR,RS_MCRDTR,RS_LINOFF);

        //set up port a and b as outputs
        outportb(0x65, inportb(0x65)|0x03);
        //turn off motors
        outportb(0x61, 0x00);  //PortB 00000000
        //turn off sensors
        outportb(0x60, 0x00);  //PortA 00000000
        //tilt off -- driver port 5
        PutDriverChannel(5);
        PutDriver(0);
        //camera OFF -- driver port 0 & 1 in parallel
        PutDriverChannel(0);
        PutDriver(0);
        PutDriverChannel(1);
        PutDriver(0);

        //turn on voltage across potentiometer linked to steering motor
```

```
        outportb(0x60, inportb(0x60) | 0x01);  //PortA xxxxxxx1

                                        // generate IRQ mask for PIC
        PIC1mask = 0x04;                // bit set for IRQ2 (cascade)
        PIC2mask = 0x04;                // bit set for IRQ10

        start_timer(0x035F); //1300 Hz

        disable();                      // disable interrupts

        oldirq10=getvect(IRQ10); // save old vector and set new one
        setvect(IRQ10,irq10);

        outportb(0x21,inportb(0x21)&~PIC1mask); // clear bit for IRQ2 in PIC1
        outportb(0xA1,inportb(0xA1)&~PIC2mask); // clear bit for IRQ10 in PIC2

        enable();                       // re-enable ints
//end interrupt enable

//main loop
        do //run rover foreground -- interrupts in background
        {       if (rs_inrcvd())
                {
                        if (HF2) //Header received, get 2-byte message
                        {       M[i] = rs_getbyt();  //eat inrcvd and make M[i] = received byte

                                if(i < 1) //then only one bite is received so far
                                        ++i;
                                else    //two bytes received
                                {       HF2 = FALSE; //reset flag to false for next message
                                        command = DeCode(M); //decode the received message
                                        DoCommand(command);  //execute the command
                                }
                        }
                        else
                        {       M[0] = rs_getbyt();  //eat inrcvd and assign
                                                     //M[0] = received byte

                                if (HF1)   //then first header byte already received
                                {       if (M[0] == 255) //second header byte received
                                        {       HF2 = TRUE; //make second header flag true
                                                i = 0; //reset i 2-byte message count
                                                //a command is being sent to rover --
                                                //reset Quiet Mode counter
                                                QuietModeCnt = 234000l;
                                        }
```

```
                                        HF1 = FALSE; //make header flag 1 false
                                }
                                if (M[0] == 254) //then first header byte received
                                        HF1 = TRUE; //make first header flag true
                        }
                }

                if (PTTDelay == 0) //PTT False
                        rs_modctrl(RS_WRTMCR,RS_MCRDTR,RS_LINOFF);

                if (QuietModeCnt == 0) //enter quiet mode if timer decremented to 0
                        QuietMode();

                if(command == 27) //if ESC received, exit program
                        break;

        }while(1);

        stop_timer();

// turn off our interrupts and reset vectors
        disable();

        setvect(IRQ10,oldirq10);        // restore old vector

        outportb(0xA1,inportb(0xA1)| PIC2mask); // reset PIC2, do not change PIC1,
                                        // turning off cascade may cause problems

        rs_close();                                     //close the port
        enable();                                       // enable ints

return 0;
}

//*************************** Turn Routine *********************************
void turn(int command)
{       char c[1];

        //stop any other turning commands
        turnFlag = FALSE;
        outportb(0x61, inportb(0x61) & 0xF3);  //PortB xxxx00xx

        switch(command)
        {       case 'j':       lowV = 1550;  //left turn 3
                                highV = 1650;
                                break;
```

```
                case 'k':       lowV = 1750;  /left turn 2
                                highV = 1850;
                                break;
                case 'l':       lowV = 2050;  //left turn 1
                                highV = 2150;
                                break;
                case 'm':       lowV = 2450;  //straight
                                highV = 2550;
                                break;
                case 'n':       lowV = 2750;   //right turn 1
                                highV = 2850;
                                break;
                case 'o':        lowV = 2950;  //right turn 2
                                highV = 3050;
                                break;
                case 'p':       lowV = 3200;   //right turn 3
                                highV = 3300;
                                break;
                default:        lowV = 2450;   //straight
                                highV = 2550;
        }

        //get voltage across middle wire and negative side of potentiometer
        PutA2DChannel(0);
        volts = GetA2D();

        if(volts <= lowV || volts > highV) //then not steered in right range
        {       //set duty cycle counters
                ontimerLR = 25;
                offtimerLR = 25;

                //ON part of duty cycle
                if (volts <= lowV)  // turn right
                        outportb(0x61, (inportb(0x61) | 0x08) & 0xFB);  //PortB xxxx10xx
                else    // turn left
                        outportb(0x61, (inportb(0x61) | 0x04) & 0xF7);  //PortB xxxx01xx
                turnFlag = ON;
        }

  //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
        //echo command
        c[0] = command;
        EnCode_TX(1, c); //encode and TX command
}

//************************** Drive Routine ********************************
```

```
void drive(int command)
{       char c[1];

        //stop any other drive command and reset count
        driveFlag = FALSE;
        outportb(0x61, inportb(0x61) & 0xFC);  //PortB xxxxxx00

        driveCount = 39; //39 cycles until timeout--approximately 1.5 second timeout
        if(command - 101 != 0)
        {       switch (command)
                {       case 'a' : DdutyCyc = 50; //100% reverse
                        case 'b' : DdutyCyc = 30; //60% reverse
                        case 'c' : DdutyCyc = 15; //30% reverse
                        case 'd' : DdutyCyc = 1;  //2% reverse
                        case 'f' : DdutyCyc = 1;  //2% forward
                        case 'g' : DdutyCyc = 15; //30% forward
                        case 'h' : DdutyCyc = 30; //60% forward
                        case 'i' : DdutyCyc = 50; //100% forward
                }
                //set duty cycle timers
                ontimerD = DdutyCyc;
                offtimerD = 50 - DdutyCyc;

                //turn ON motors in proper direction
                if (command - 101 < 0)  //backward command
                {       Dmask1 = 0x02;          //masks to output xxxxxx10 to Port B
                        Dmask2 = 0xFE;
                }
                else  //forward command
                {       Dmask1 = 0x01;          //masks to output xxxxxx01 to Port B
                        Dmask2 = 0xFD;
                }
                //turn on motor
                outportb(0x61, (inportb(0x61) | Dmask1) & Dmask2);
                driveFlag = ON; //interrupts take over
        }

 //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
        //echo command
        c[0] = command;
        EnCode_TX(1, c); //encode and TX command
}

//******************Turn on or Off appropriate sensor**********************
void OnOff(int command)
{       char c[1];
```

```
        switch (command)
        {       //Camera ON    //Driver Port 0 & 1 in parallel
                case 'q':       PutDriverChannel(0);
                                PutDriver(1);
                                PutDriverChannel(1);
                                PutDriver(1);
                                status |= 0x10;
                                break;
                //Camera OFF   //Driver Port 0 & 1 in parallel
                case 'r':       PutDriverChannel(0);
                                PutDriver(0);
                                PutDriverChannel(1);
                                PutDriver(0);
                                status &= 0xE0;
                                break;
                //Compass ON   //PortA xxxxx1xx
                case 's':       outportb(0x60, inportb(0x60) | 0x04);
                                status |= 0x20;
                                break;
                //Compass OFF  //PortA xxxxx0xx
                case 't':       outportb(0x60, inportb(0x60) & 0xFB);
                                status &= 0xD0;
                                break;
                //Inclinometer ON    //Driver Port 5
                case 'u':       PutDriverChannel(5);
                                PutDriver(1);
                                status |= 0x40;
                                break;
                //Inclinometer OFF    //Driver Port 5
                case 'v':       PutDriverChannel(5);
                                PutDriver(0);
                                status &= 0xB0;
                                break;
                //Acc ON    //PortA xxxxxx1x
                case 'w':       outportb(0x60, inportb(0x60) | 0x02);
                                status |= 0x80;
                                break;
                //Acc OFF   //PortA xxxxxx0x
                case 'x':       outportb(0x60, inportb(0x60) & 0xFD);
                                status &= 0x70;
                                break;
        }

//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
        //echo command
```

```
        c[0] = command;
        EnCode_TX(1, c); //encode and TX command
}

//************************Poll Accelerometer*********************************
void Acc(void)
{       float Xfloat, Yfloat, Zfloat;
        unsigned int X, Y, Z;
        char Xstr[3], Ystr[3], Zstr[3], XYZstr[8];

        //A2D channel 2 = X axis, A2D channel 3 = Y axis, A2D channel 4 = Z axis
        //Poll channel, scale data, convert to int

        PutA2DChannel(2);
        Xfloat = GetA2D();
        //scale X
        //raw data +/-80mv/G
        Xfloat = Xfloat/8 - 256.875; //scaling based on measurements
        X = (int) (Xfloat + 0.5);

        if (X < 10) X = 10; //control max and min values
        if (X > 90) X = 90;

        PutA2DChannel(3);
        Yfloat = GetA2D();
        //scale Y
        //raw data  +/-80mv/G
        Yfloat = 362.5 - Yfloat/8;
        Y = (int) (Yfloat + 0.5);

        if (Y < 10) Y = 10;
        if (Y > 90) Y = 90;

        PutA2DChannel(4);
        Zfloat = GetA2D();
        //scale Z
        //raw data +/-80mv/G
        Zfloat = Zfloat/8 - 262.5;
        Z = (int) (Zfloat + 0.5);

        if (Z < 10) Z = 10;
        if (Z > 90) Z = 90;

        //convert X Y and Z ints to strings and send them
        itoa(X, Xstr, 10);
        itoa(Y, Ystr, 10);
```

```
        itoa(Z, Zstr, 10);

        //construct the data string
        strcpy(XYZstr, "A");
        strcat(XYZstr, Xstr);
        strcat(XYZstr, Ystr);
        strcat(XYZstr, Zstr);

 //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
        //echo command and data
        EnCode_TX(7, XYZstr);
}

//***************************Poll Compass**********************************
void Comp(void)
{       float Y;
        unsigned int X;
        char Xstr[4], XXstr[5];

        //A2D channel 1 = compass data
        //Poll channel, scale data, convert to int
        PutA2DChannel(1);
        Y = GetA2D();
        //scale direction X
        Y = Y*359/3882 + 100;

        //round float Y and force to int
        X = (int) (Y + 0.5);

        if (X < 280) //correct compass by 180 degrees
                X = X + 180;
        else
                X = X - 180;

        if (X > 459) X = 459; //control max value

        //convert direction X int to strings and send it
        itoa(X, Xstr, 10);

        //construct string
        strcpy(XXstr, "B");
        strcat(XXstr, Xstr);

 //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
        //echo command and data
        EnCode_TX(4, XXstr);
```

```
}

//************************Poll Tilt sensor**********************************
void Tilt(void)
{       float Pitch, Roll;
        unsigned int Pitchint, Rollint;
        char Xstr[3], Ystr[3], XYstr[6];

        //A2D channel 6 = pitch, A2D channel 5 = roll
        //Poll channel, scale data, convert to int
        PutA2DChannel(6);
        Pitch = GetA2D();

        //scale Pitch
        Pitch = Pitch*0.03316 - 40.467;

        //round float Pitch and force to int
        Pitchint = (int) (Pitch + 0.5);

        if (Pitchint < 10) Pitchint = 10; //control max and min values
        if (Pitchint > 90) Pitchint = 90;

        PutA2DChannel(5);
        Roll = GetA2D();
        //scale Roll
        Roll = Roll*0.03157 - 36.6;

        //round float Roll and force to int
        Rollint = (int) (Roll + 0.5);

        if (Rollint < 10) Rollint = 10;
        if (Rollint > 90) Rollint = 90;

        //convert Pitch and Roll ints to strings and send them
        itoa(Rollint, Xstr, 10);
        itoa(Pitchint, Ystr, 10);

        //construct string
        strcpy(XYstr, "C");
        strcat(XYstr, Ystr);
        strcat(XYstr, Xstr);

 //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
        //echo command and data
        EnCode_TX(5, XYstr);
}
```

```
//********************Update Sensor Status*********************************
void Sensor(void)
{       char CapD[1] = "D";

 //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
        //echo D, and status (status added in encode function)
        EnCode_TX(1, CapD);
}

//*****************************Ping*************************************
*
void Ping(void)
{       char Pound[1] = "#";

 //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
        //echo Ping
        EnCode_TX(1, Pound);
}

//************************Enter Quiet Mode*********************************
//---
//Rover enters quiet mode if QuietModeCnt decremented to 0.  This will happen
//if no command has been sent to Rover for about 3 minutes.
//---
void QuietMode(void)
{       QuietModeCnt = -1;
        //Camera OFF   //Driver Port 0 & 1 in parallel
        PutDriverChannel(0);
        PutDriver(0);
        PutDriverChannel(1);
        PutDriver(0);
        //Inclinometer OFF    //Driver Port 5
        PutDriverChannel(5);
        PutDriver(0);
        //Acc and Compass OFF   //PortA xxxxx00x
        outportb(0x60, inportb(0x60) & 0xF9);
        status = 0; //status 0 -- all sensors off
}

//*****************************Do Command*********************************
//---
//Case statement to decide which command to execute.  If the command is a
//valid command within the set of commands, the appropriate function is
//called.
//---
```

```
void DoCommand(int command)
{       switch(command)
        {  //ascii #  -- Ping
                case 35:
                        Ping();
                        break;
                //ascii A -- Poll Accelerometer
                case 65:
                        Acc();
                        break;
                //ascii B -- Poll Compass
                case 66:
                        Comp();
                        break;
                //ascii C -- Poll Tilt sensor
                case 67:
                        Tilt();
                        break;
                //ascii D -- Sensor update
                case 68:
                        Sensor();
                        break;
                //ascii a through i -- Driving commands
                case 97: case 98:       case 99:        case 100: case 101:
                case 102: case 103: case 104: case 105:
                        drive(command);
                        break;
                //ascii j through p -- Turning commands
                case 106: case 107: case 108: case 109:
                case 110: case 111: case 112:
                        turn(command);
                        break;
                //ascii q through z -- On Off commands
                case 113: case 114: case 115: case 116: case 117:
                case 118: case 119: case 120: case 121: case 122:
                        OnOff(command);
                        break;
                }
}

//******************Hamming Decoding Algorithm*****************************
int DeCode(int M[])
{       int i, C0, C1, C3, C7; //, C;
        int R[11] = {0};
        int Y[7] = {0};
```

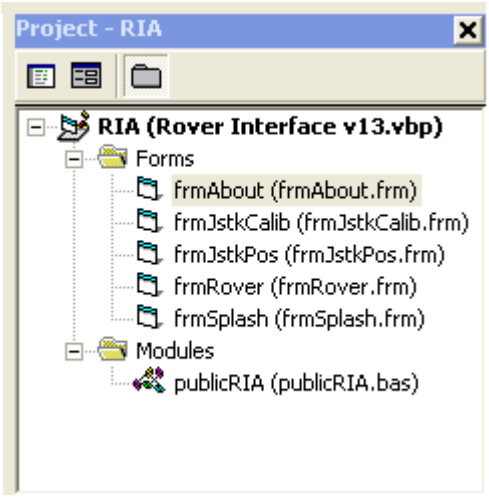
```
	//isolate bits -- place in lowest order position
	for(i = 0; i <= 7; ++i)
		R[i] = (M[0]>>i) & 0x1;

	for(i = 0; i <= 2; ++i)
		R[i + 8] = (M[1]>>i) & 0x1;

	//get error vector
	C0 = R[0] ^ R[2] ^ R[4] ^ R[6] ^ R[8] ^ R[10];
	C1 = R[1] ^ R[2] ^ R[5] ^ R[6] ^ R[9] ^ R[10];
	C3 = R[3] ^ R[4] ^ R[5] ^ R[6];
	C7 = R[7] ^ R[8] ^ R[9] ^ R[10];

	//C = C0 | (C1<<1) | (C3<<2) | (C7<<3);
	//C = error location

	//Decode and correct
	 Y[0] = R[2] ^ (C0 & C1 & ~C3 & ~C7);
	 Y[1] = R[4] ^ (C0 & ~C1 & C3 & ~C7);
	 Y[2] = R[5] ^ (~C0 & C1 & C3 & ~ C7);
	 Y[3] = R[6] ^ (C0 & C1 & C3 & ~C7);
	 Y[4] = R[8] ^ (C0 & ~C1 & ~C3 & C7);
	 Y[5] = R[9] ^ (~C0 & C1 & ~C3 & C7);
	 Y[6] = R[10] ^ (C0 & C1 & ~ C3 & C7);

	//Reconstruct code word from bits
	for(i = 1; i <= 6; i++)
		Y[0] |= (Y[i]<<i);

	return Y[0];
}

//*******************Hamming Encoding Algorithm*****************************
void EnCode_TX(int size, char DataStr[])
{	char S[81] = {0};
	char SendStr[11] = {0};
	int i, n, CodeLength;

	//PTT True
	rs_modctrl(RS_WRTMCR,RS_MCRDTR,RS_LINON);

	//encode string
	//code bitwise in groups of 11
	//size determines how many times needed through loop
	for(i = 0; i < size; ++i)
	{
```

```
        S[2 + (11*i)] = DataStr[i] & 0x1;
        S[4 + (11*i)] = (DataStr[i] & 0x2)>>1;
        S[5 + (11*i)] = (DataStr[i] & 0x4)>>2;
        S[6 + (11*i)] = (DataStr[i] & 0x8)>>3;
        S[8 + (11*i)] = (DataStr[i] & 0x10)>>4;
        S[9 + (11*i)] = (DataStr[i] & 0x20)>>5;
        S[10 + (11*i)] = (DataStr[i] & 0x40)>>6;

        S[0 + (11*i)] = S[2 + (11*i)] ^ S[4 + (11*i)] ^
                                        S[6 + (11*i)] ^ S[8 + (11*i)] ^ S[10 + (11*i)];
        S[1 + (11*i)] = S[2 + (11*i)] ^ S[5 + (11*i)] ^
                                        S[6 + (11*i)] ^ S[9 + (11*i)] ^ S[10 + (11*i)];
        S[3 + (11*i)] = S[4 + (11*i)] ^ S[5 + (11*i)] ^ S[6 + (11*i)];
        S[7 + (11*i)] = S[8 + (11*i)] ^ S[9 + (11*i)] ^ S[10 + (11*i)];
 }

//size determines how many encoded bytes need to be sent
switch(size)
{       case 4: CodeLength = 6;
                break;
        case 5: CodeLength = 7;
                break;
        case 7: CodeLength = 10;
                break;
        default: CodeLength = 2;
}

//pack into bytes
for(n = 0; n < CodeLength; ++n)
{       for (i = 1; i <= 7; ++i)
                S[n*8] |= (S[i + (n*8)]<<i);
        SendStr[n] = S[n*8];
}

//if command 'D' was sent to rover
if(DataStr[0] == 'D')  //put sensor status in upper 4 bits of coded message
        SendStr[1] |= status;

PTTDelay = 215; //215 interrupt delay @ 1300 interrupts per sec
while (PTTDelay > 0)
        ; //wait here until delay decrements to 0

//transmit message
rs_sndbyt(254); //header 1
rs_sndbyt(255); //header 2
for(i = 0; i < CodeLength; ++i)
```

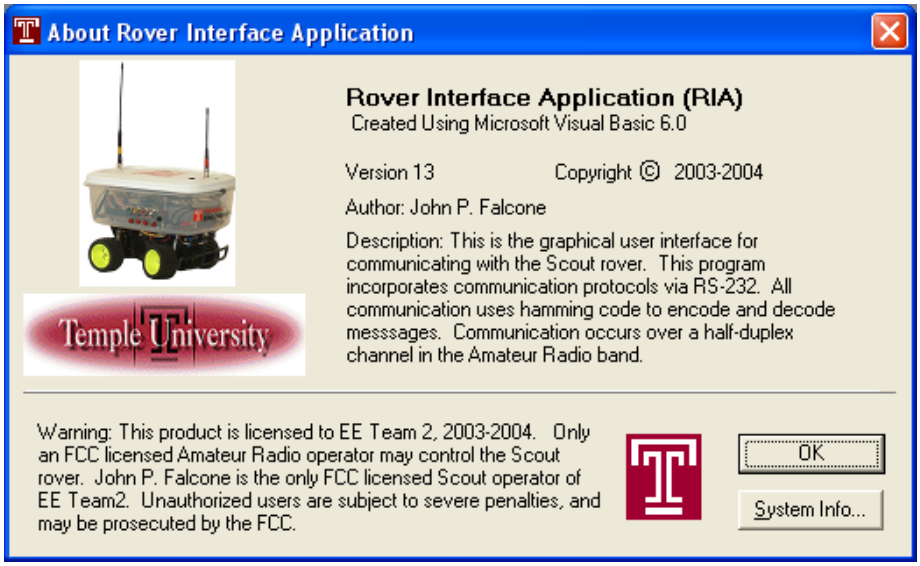
```
            rs_sndbyt(SendStr[i]); //coded message

        //set interrupt counter to delay turning off PTT line (DTR)
        switch(size)
        {       case 4:
                case 5: //215 interrupt delay @ 1300 interrupts per sec
                        PTTDelay = 120;
                        break;
                case 7: //180 interrupt delay @ 1300 interrupts per sec
                        PTTDelay = 180;
                        break;
                default: //50 interrupt delay @ 1300 interrupts per sec
                        PTTDelay = 50;
        }
}
```

Rover Interface Application – Microsoft Visual Basic

Project Tree:



frmAbout:



***frmAbout* source code:**

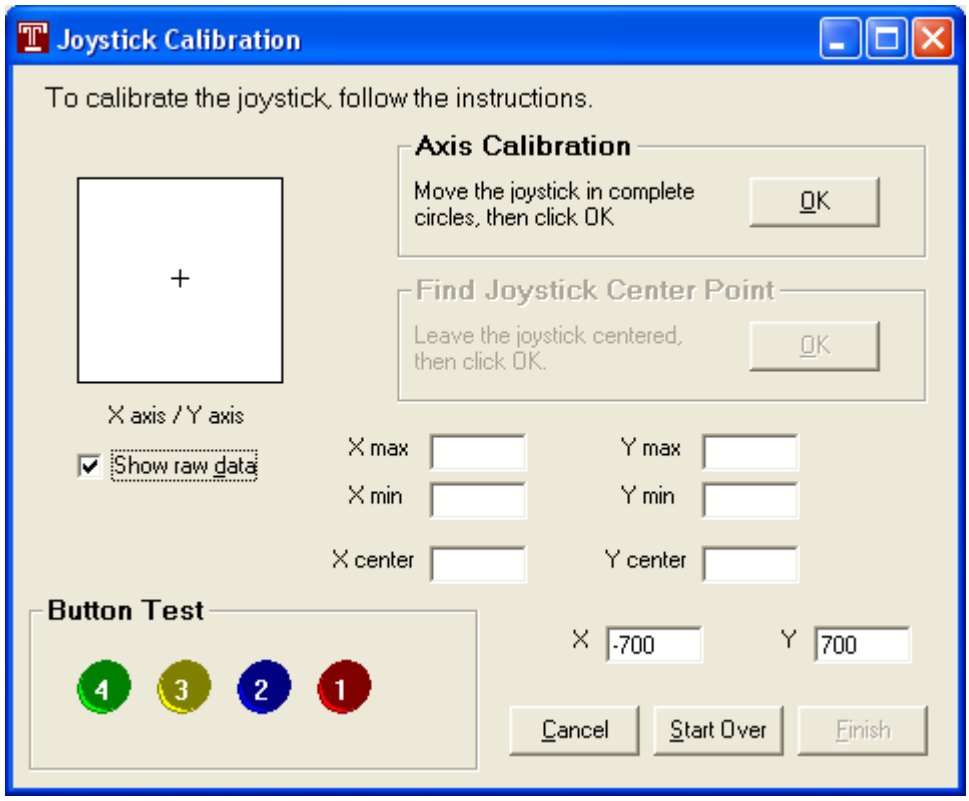
```
Option Explicit

' Reg Key Security Options...
Const READ_CONTROL = &H20000
Const KEY_QUERY_VALUE = &H1
Const KEY_SET_VALUE = &H2
Const KEY_CREATE_SUB_KEY = &H4
Const KEY_ENUMERATE_SUB_KEYS = &H8
Const KEY_NOTIFY = &H10
```

```
Const KEY_CREATE_LINK = &H20
Const KEY_ALL_ACCESS = KEY_QUERY_VALUE + KEY_SET_VALUE + _
                KEY_CREATE_SUB_KEY + KEY_ENUMERATE_SUB_KEYS + _
                KEY_NOTIFY + KEY_CREATE_LINK + READ_CONTROL

' Reg Key ROOT Types...
Const HKEY_LOCAL_MACHINE = &H80000002
Const ERROR_SUCCESS = 0
Const REG_SZ = 1                        ' Unicode nul terminated string
Const REG_DWORD = 4                     ' 32-bit number

Const gREGKEYSYSINFOLOC = "SOFTWARE\Microsoft\Shared Tools Location"
Const gREGVALSYSINFOLOC = "MSINFO"
Const gREGKEYSYSINFO = "SOFTWARE\Microsoft\Shared Tools\MSINFO"
Const gREGVALSYSINFO = "PATH"

Private Declare Function RegOpenKeyEx Lib "advapi32" Alias "RegOpenKeyExA" (By _
Val hKey As Long, ByVal lpSubKey As String, ByVal ulOptions As Long, ByVal _
samDesired As Long, ByRef phkResult As Long) As Long
Private Declare Function RegQueryValueEx Lib "advapi32" Alias "RegQueryValueExA"
(ByVal hKey As Long, ByVal lpValueName As String, ByVal lpReserved As Long, ByRef
lpType As Long, ByVal lpData As String, ByRef lpcbData As Long) As Long
Private Declare Function RegCloseKey Lib "advapi32" (ByVal hKey As Long) As Long


Private Sub cmdSysInfo_Click()
  Call StartSysInfo
End Sub

Private Sub cmdOK_Click()
  Unload Me
End Sub

Public Sub StartSysInfo()
   On Error GoTo SysInfoErr

   Dim rc As Long
   Dim SysInfoPath As String

   ' Try To Get System Info Program Path\Name From Registry...
   If GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFO,  _
                           gREGVALSYSINFO, SysInfoPath) Then
   ' Try To Get System Info Program Path Only From Registry...
   ElseIf GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFOLOC, _
                        gREGVALSYSINFOLOC, SysInfoPath) Then
  ' Validate Existance Of Known 32 Bit File Version
```

```
      If (Dir(SysInfoPath & "\MSINFO32.EXE") <> "") Then
         SysInfoPath = SysInfoPath & "\MSINFO32.EXE"

      ' Error - File Can Not Be Found...
      Else
         GoTo SysInfoErr
      End If
   ' Error - Registry Entry Can Not Be Found...
   Else
      GoTo SysInfoErr
   End If

   Call Shell(SysInfoPath, vbNormalFocus)

   Exit Sub
SysInfoErr:
   MsgBox "System Information Is Unavailable At This Time", vbOKOnly
End Sub

Public Function GetKeyValue(KeyRoot As Long, KeyName As String, SubKeyRef As
String, ByRef KeyVal As String) As Boolean
   Dim i As Long                                ' Loop Counter
   Dim rc As Long                               ' Return Code
   Dim hKey As Long                             ' Handle To An Open Registry Key
   Dim hDepth As Long                           '
   Dim KeyValType As Long                       ' Data Type Of A Registry Key
   Dim tmpVal As String                         ' Tempory Storage For A Registry Key Value
   Dim KeyValSize As Long                       ' Size Of Registry Key Variable
   '------------------------------------------------------------
   ' Open RegKey Under KeyRoot {HKEY_LOCAL_MACHINE...}
   '------------------------------------------------------------
   rc = RegOpenKeyEx(KeyRoot, KeyName, 0, KEY_ALL_ACCESS, hKey) ' Open Registry
                                                                ' Key

   If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError          ' Handle Error...

   tmpVal = String$(1024, 0)                    ' Allocate Variable Space
   KeyValSize = 1024                            ' Mark Variable Size

   '------------------------------------------------------------
   ' Retrieve Registry Key Value...
   '------------------------------------------------------------
   rc = RegQueryValueEx(hKey, SubKeyRef, 0, _
                  KeyValType, tmpVal, KeyValSize)    ' Get/Create Key Value

   If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError          ' Handle Errors
```

```
   If (Asc(Mid(tmpVal, KeyValSize, 1)) = 0) Then          ' Win95 Adds Null Terminated
String...
      tmpVal = Left(tmpVal, KeyValSize - 1)              ' Null Found, Extract From String
   Else                                                  ' WinNT Does NOT Null Terminate String...
      tmpVal = Left(tmpVal, KeyValSize)                  ' Null Not Found, Extract String Only
   End If
   '------------------------------------------------------------
   ' Determine Key Value Type For Conversion...
   '------------------------------------------------------------
   Select Case KeyValType                                ' Search Data Types...
   Case REG_SZ                                           ' String Registry Key Data Type
      KeyVal = tmpVal                                    ' Copy String Value
   Case REG_DWORD                                        ' Double Word Registry Key Data Type
      For i = Len(tmpVal) To 1 Step -1                   ' Convert Each Bit
         KeyVal = KeyVal + Hex(Asc(Mid(tmpVal, i, 1)))   ' Build Value Char. By Char.
      Next
      KeyVal = Format$("&h" + KeyVal)                    ' Convert Double Word To String
   End Select

   GetKeyValue = True                                    ' Return Success
   rc = RegCloseKey(hKey)                                ' Close Registry Key
   Exit Function                                         ' Exit

GetKeyError:      ' Cleanup After An Error Has Occured...
   KeyVal = ""                                           ' Set Return Val To Empty String
   GetKeyValue = False                                   ' Return Failure
   rc = RegCloseKey(hKey)                                ' Close Registry Key
End Function

Private Sub Form_Load() 'enter captions into labels
  lblDescription.Caption = "Description: This is the graphical user interface for " _
   & "communicating with the Scout rover.  This program incorporates communication " _
   & "protocols via RS-232.  All communication uses hamming code to encode and decode" _
   & " messsages.  Communication occurs over a half-duplex channel in the Amateur Radio" _
   & " band."
  lblDisclaimer.Caption = "Warning: This product is licensed to EE Team 2, 2003-2004." _
   & "   Only an FCC licensed Amateur Radio operator may control the Scout rover.  " _
   & "John P. Falcone is the only FCC licensed Scout operator of EE Team2.  Unauthorized" _
   & " users are subject to severe penalties, and may be prosecuted by the FCC."
End Sub
```

***frmJstkCalib*:**



***frmJstkCalib* source code:**

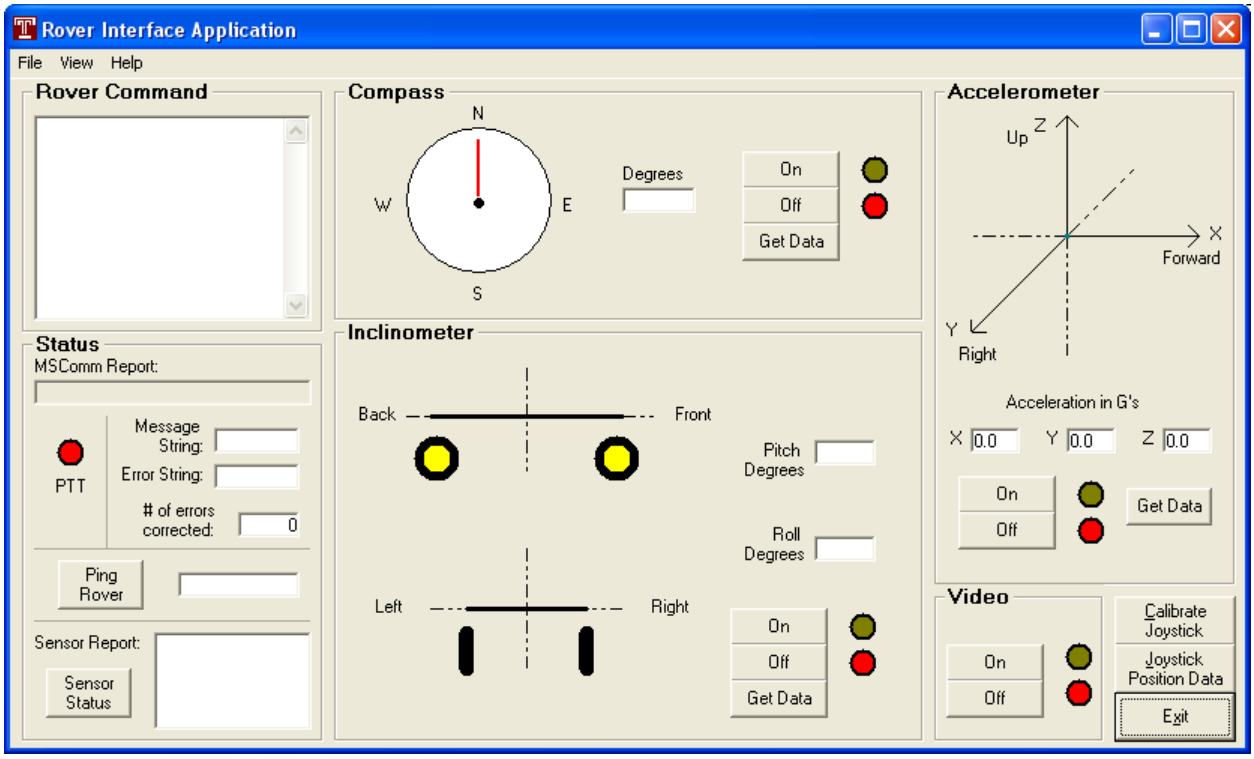
```
'-------------------------------------------------------------------------
'Joystick Calibration form - This form allows user to calibrate the joystick
'-------------------------------------------------------------------------
Option Explicit

'Dim mblnXYSet As Boolean
Dim mlngXcent As Long
Dim mlngYcent As Long
Dim mlngXmax As Long
Dim mlngYmax As Long
Dim mlngXmin As Long
Dim mlngYmin As Long
Public mlngCurrentX As Long
Public mlngCurrentY As Long

'When form activates - set flag to true so that no message are transmitted
'during calibration
Private Sub Form_Activate()
    frmRover.TXFlag = True
```

```
End Sub

'Load form with default values for joystick control and XY positions
Private Sub Form_Load()
   SetDefaultXY
   Joystick1.Interval = 100
   Joystick1.Threshold = 100
   Joystick1.OneShot = False
End Sub

'Unload form.  If calibration has not been performed, values are not saved.
Private Sub Form_Unload(Cancel As Integer)
   frmRover.TXFlag = False 'resume normal communication
   Unload Me
End Sub

'Axis calibration completed
Private Sub cmdAxisOK_Click()
   'disable Axis calibration frame
   fraAxisCal.Enabled = False
   lblAxisCal.Enabled = False
   cmdAxisOK.Enabled = False
   'enable Axis center frame
   fraJCenter.Enabled = True
   lblJCenter.Enabled = True
   cmdCenterOK.Enabled = True
   'enter max min values into text boxes
   txtXmax.Text = mlngXmax
   txtYmax.Text = mlngYmax
   txtXmin.Text = mlngXmin
   txtYmin.Text = mlngYmin
End Sub

'Axis center values found
Private Sub cmdCenterOK_Click()
   'get raw values
   mlngXcent = mlngCurrentX
   mlngYcent = mlngCurrentY
   'prevent abnormal values in case joystick is way off center
   If mlngXcent > 34000 Or mlngXcent < 30000 Then
      mlngXcent = 32000
   End If
   If mlngYcent > 34000 Or mlngYcent < 30000 Then
      mlngYcent = 32000
   End If
   'disable Axis center frame
```

```
    fraJCenter.Enabled = False
    lblJCenter.Enabled = False
    cmdCenterOK.Enabled = False
    'enable finsih button
    cmdFinish.Enabled = True
    'enter center values into text boxes
    txtXcent.Text = mlngXcent
    txtYcent.Text = mlngYcent
    'put focus on finish button
    cmdFinish.SetFocus
End Sub

'Show the raw data frame
Private Sub chkData_Click()
    If (chkData.Value = vbChecked) Then
        fraData.Visible = True
    Else
        fraData.Visible = False
    End If
End Sub

'Return to main form
Private Sub cmdCancel_Click()
    frmRover.TXFlag = False
    Unload Me
End Sub

'Calibration completed - save values and go to main rover form
Private Sub cmdFinish_Click()
    Xmax = mlngXmax
    Ymax = mlngYmax
    Xmin = mlngXmin
    Ymin = mlngYmin
    Xcenter = mlngXcent
    Ycenter = mlngYcent
    frmRover.TXFlag = False
    Unload Me
End Sub

'Start the calibration over
Private Sub cmdStartOver_Click()
    SetDefaultXY 'start with the default values again
    'enable Axis center frame
    fraAxisCal.Enabled = True
    lblAxisCal.Enabled = True
    cmdAxisOK.Enabled = True
```

```
    'disable Finish button
    cmdFinish.Enabled = False
End Sub

'Joystick movement
Private Sub Joystick1_Joystk(X As Long, Y As Long, Direction As Integer)
    'find any new max values
    mlngXmax = max(X, mlngXmax)
    mlngXmin = min(X, mlngXmin)
    mlngYmax = max(Y, mlngYmax)
    mlngYmin = min(Y, mlngYmin)
    'call joystick move subroutine
    mlngCurrentX = X
    mlngCurrentY = Y
    JoystickMove X, Y, Me 'call function from publicRIA module
End Sub

'call public joystick subroutine
Private Sub Joystick1_JoystkButton(ByVal Button As Integer)
    JoystkButton Button, Me 'call function from publicRIA moduel
End Sub

'Set the default joystick calibration values
Public Sub SetDefaultXY()
    mlngXcent = 32000
    mlngYcent = 32000
    mlngXmax = 42000
    mlngYmax = 42000
    mlngXmin = 20000
    mlngYmin = 20000
End Sub
```

***frmRover*:**



***frmRover* source code:**

```
'-------------------------------------------------------------------------
'Title:  Rover Interface Application version 13 (RIA)
'Author: John P. Falcone
'Description: This program is the graphical user interface for communicating
'   with the rover.  This program incorporates communication protocols
'   via RS-232.  All communication uses hamming code to encode and decode
'   messsages.  Communication occurs over a half-duplex channel.
'-------------------------------------------------------------------------
Option Explicit

'Drive and turn flag for alternating between drive and turn commands
Public mblnDrvTurnFlag As Boolean
'Buffer allocation for input from RS-232
Public Buffer As String
'Header flag to see if a valid message is incoming
Public HeadFlag As Boolean
'TXFlag blocks transmission of new commands if Base station is currently
'transmitting.
Public TXFlag As Boolean
'DataReqFlag: 0 for all On/Off, Drive, and Turn commands
```

```
'           1 for Accelerometer data request
'           2 for Compass data request
'           3 for Tilt data request
Public DataReqFlag As Byte

'Turn off Accellerometer
Private Sub cmdAccOFF_Click()
   If Not TXFlag Then 'make sure that no communication is in progress
      DataReqFlag = 0
      EnCode Asc("x") 'send message
   End If
End Sub

'Turn on Accellerometer
Private Sub cmdAccON_Click()
   If Not TXFlag Then 'make sure that no communication is in progress
      DataReqFlag = 0
      EnCode Asc("w") 'send message
   End If
End Sub

'Request Acceleration data
Private Sub cmdAccReq_Click()
   'Make sure sensor is on AND no communication is in progress
   If shpAccON.BackColor = Green And Not TXFlag Then
      DataReqFlag = 1
      EnCode Asc("A") 'send message
   End If
End Sub

'Turn off Compass
Private Sub cmdCompOFF_Click()
   If Not TXFlag Then 'make sure that no communication is in progress
      DataReqFlag = 0
      EnCode Asc("t") 'send message
   End If
End Sub

'Turn on Compass
Private Sub cmdCompON_Click()
   If Not TXFlag Then 'make sure that no communication is in progress
      DataReqFlag = 0
      EnCode Asc("s") 'send message
   End If
End Sub
```

```
'Request Compass data
Private Sub cmdCompReq_Click()
   'Make sure sensor is on AND no communication is in progress
   If shpCompON.BackColor = Green And Not TXFlag Then
      DataReqFlag = 2
      EnCode Asc("B") 'send message
   End If
End Sub

'Exit RIA
Private Sub cmdExit_Click()
   MSComm1.PortOpen = False 'close port
   Unload Me 'Unload form
End Sub

'Turn off Inclinometer
Private Sub cmdIncOFF_Click()
   If Not TXFlag Then 'make sure that no communication is in progress
      DataReqFlag = 0
      EnCode Asc("v") 'send message
   End If
End Sub

'Turn on Inclinometer
Private Sub cmdIncON_Click()
   If Not TXFlag Then 'make sure that no communication is in progress
      DataReqFlag = 0
      EnCode Asc("u") 'send message
   End If
End Sub

'Request Inclinometer data
Private Sub cmdIncReq_Click()
   'Make sure sensor is on AND no communication is in progress
   If shpIncON.BackColor = Green And Not TXFlag Then
      DataReqFlag = 3
      EnCode Asc("C") 'send message
   End If
End Sub

'Click on Joystick Position Data button - goto mnuViewJoyPos_Click function
Private Sub cmdJoystick_Click()
   mnuViewJoyPos_Click
End Sub

'Click on Joystick Calibration - Open joystick calibration form
```

```
Private Sub cmdJstkCal_Click()
   frmJstkCalib.Show vbModal
End Sub

'Ping command
Private Sub cmdPing_Click()
   If Not TXFlag Then 'make sure that no communication is in progress
      lblPing.Caption = "" 'clear ping caption
      DataReqFlag = 0
      EnCode Asc("#") 'send message
      tmrPing.Enabled = True 'start timeout counter
   End If
End Sub

'Sensor status command - Update sensor lights
Private Sub cmdSensorStatus_Click()
   If Not TXFlag Then
      DataReqFlag = 4
      EnCode Asc("D")
   End If
End Sub

'Turn Off Camera
Private Sub cmdTVOFF_Click()
   If Not TXFlag Then 'make sure that no communication is in progress
      DataReqFlag = 0
      EnCode Asc("r") 'send message
   End If
End Sub

'Turn On Camera
Private Sub cmdTVON_Click()
   If Not TXFlag Then 'make sure that no communication is in progress
      DataReqFlag = 0
      EnCode Asc("q") 'send message
   End If
End Sub

'Set properties for controls and initial values of variables when form opens
Private Sub Form_Load()
'Initialize all objects as necessary
   'Startup window state
   mnuViewRestore.Enabled = False
   'joystick variable data
   Xmax = 64000
   Ymax = 65000
```

```
    Xmin = 1500
    Ymin = 0
    Xcenter = 32000
    Ycenter = 32000
    'joystick control properties
    Joystick1.Interval = 100
    Joystick1.Threshold = 500
    Joystick1.OneShot = False
    'RS-232 port setup
    MSComm1.CommPort = 1
    MSComm1.Settings = "1200,N,8,1"
    MSComm1.PortOpen = True
    MSComm1.RThreshold = 1
    'set initial state for Drive or Turn
    mblnDrvTurnFlag = False
    'start lines in correct position
    linXVect.X1 = 1320
    linXVect.X2 = 1320
    linXVect.Y1 = 1560
    linXVect.Y2 = 1560
    linYVect.X1 = 1320
    linYVect.X2 = 1320
    linYVect.Y1 = 1560
    linYVect.Y2 = 1560
    linZVect.X1 = 1320
    linZVect.X2 = 1320
    linZVect.Y1 = 1560
    linZVect.Y2 = 1560
    'set an initial value for Data request flag
    DataReqFlag = 0
    'Count down counter for call sign tranmission
    TXCallSignCnt = 10
    'Start in transmit mode
    PTT True
    'header not received
    HeadFlag = False
    'Timeout timer off
    TXTimeOut.Enabled = False
End Sub

'Sub routine triggered when form is resized
Private Sub Form_Resize()
    'in each state, the appropriate menu option has to be disabled
    If Me.WindowState = 0 Then
        mnuViewRestore.Enabled = False
        mnuViewMinimize.Enabled = True
```

```
        mnuViewMaximize.Enabled = True
    ElseIf Me.WindowState = 1 Then
        mnuViewRestore.Enabled = True
        mnuViewMinimize.Enabled = False
        mnuViewMaximize.Enabled = True
    Else
        mnuViewRestore.Enabled = True
        mnuViewMinimize.Enabled = True
        mnuViewMaximize.Enabled = False
    End If
End Sub

'Unload form
Private Sub Form_Unload(Cancel As Integer)
    Unload frmJstkCalib 'unload other forms
    Unload frmJstkPos
    Unload Me
End Sub

'Joystick movement routine
Private Sub Joystick1_Joystk(X As Long, Y As Long, Direction As Integer)
  'do this only if user is not currently calibrating joystick
  If frmJstkCalib.Visible = False Then
    JoystickMove X, Y, frmJstkPos 'call public function from publicRIA module
  End If

  'do only if joystick button is pushed And no communication is in progress
  If (pButton And Not TXFlag) Then
    If (mblnDrvTurnFlag) Then
        sendTurnCommand frmJstkPos.txtXpos
    Else
        sendDriveCommand frmJstkPos.txtYpos, pButton
    End If
    mblnDrvTurnFlag = Not mblnDrvTurnFlag 'alternate between Drive and Turn
  End If
End Sub

'Joystick sub routine
Private Sub Joystick1_JoystkButton(ByVal Button As Integer)
    pButton = Button 'Get public value from publicRIA module
End Sub

'Unload form on Menu - File - Exit -- click
Private Sub mnuFileExit_Click()
    MSComm1.PortOpen = False
    Unload Me
```

```
End Sub

'Opens About form
Private Sub mnuHelpAbout_Click()
   frmAbout.Show vbModal
End Sub

'menu option - open or close joystick position form
Private Sub mnuViewJoyPos_Click()
   If mnuViewJoyPos.Checked = True Then
      Unload frmJstkPos 'close form
      mnuViewJoyPos.Checked = False 'Fix check mark in menu
   Else
      frmJstkPos.Show 'open form
      mnuViewJoyPos.Checked = True 'Fix check mark in menu
   End If
End Sub

'Keyboard commands
Private Sub Form_KeyPress(KeyAscii As Integer)
   ' If the port is opened AND no communication is in progress
   If MSComm1.PortOpen And Not TXFlag Then
      'determine if keypress is a valid command
      Select Case KeyAscii
         'encode and transmit the keystroke, if a valid command
         Case 65 To 68 'A - D
            DataReqFlag = KeyAscii - 64
         Case 27, 35, 97 To 120 'ESC, #, a - x
            DataReqFlag = 0
      End Select
      TXFlag = True 'communication now in progress
      EnCode KeyAscii 'send the message
   End If
   KeyAscii = 0
End Sub

'Maximize the window
Private Sub mnuViewMaximize_Click()
   Me.WindowState = 2
End Sub

'Minimize the window
Private Sub mnuViewMinimize_Click()
   Me.WindowState = 1
End Sub
```

```
'Restore the window
Private Sub mnuViewRestore_Click()
   Me.WindowState = 0
End Sub

'MSComm routine -- handles all incoming RS232 communication
Private Sub MSComm1_OnComm()
   Dim EVMsg As String
   Dim ERMsg As String
   Dim Ret As Integer
   Dim InByte As Variant

   '--- Branch according to the CommEvent Prop
   Select Case MSComm1.CommEvent
      '--- Event messages
      Case comEvReceive
         If HeadFlag Then
            'Incoming message -- stop time out timer
            TXTimeOut.Enabled = False
            'reset timeout timer
            TXTimeOut.Enabled = True
            Select Case DataReqFlag
               Case 1 'Accelerometer
                  'check buffer count for right length
                  If MSComm1.InBufferCount >= 10 Then
                     HeadFlag = False
                     'Decode the Received message
                     DeCode MSComm1.Input, DataReqFlag
                     TXTimeOut.Enabled = False
                     PTT True 'go back to Transmit
                  End If
               Case 2 'Compass
                  'check buffer count for right length
                  If MSComm1.InBufferCount >= 6 Then
                     HeadFlag = False
                     'Decode the Received message
                     DeCode MSComm1.Input, DataReqFlag
                     TXTimeOut.Enabled = False
                     PTT True 'go back to Transmit
                  End If
               Case 3 'Tilt
                  'check buffer count for right length
                  If MSComm1.InBufferCount >= 7 Then
                     HeadFlag = False
                     'Decode the Received message
                     DeCode MSComm1.Input, DataReqFlag
```

```
                TXTimeOut.Enabled = False
                PTT True 'go back to Transmit
            End If
        Case Else 'all other commands
            'check buffer count for right length
            If MSComm1.InBufferCount >= 2 Then
                HeadFlag = False
                'Decode the Received message
                DeCode MSComm1.Input, DataReqFlag
                TXTimeOut.Enabled = False
                PTT True 'go back to Transmit
            End If
    End Select
  Else
    'get the incoming byte
    InByte = MSComm1.Input
    'concatenate incoming byte with buffer
    Buffer$ = Buffer$ & Chr$(InByte(0))
    'check to see if header is in the buffer
    If InStr(Buffer$, Chr$(254) & Chr$(255)) Then
      'Incoming message -- stop time out timer
      TXTimeOut.Enabled = False
      TXTimeOut.Enabled = True 'reset timeout timer
      Buffer$ = "" 'clear buffer
      HeadFlag = True 'header has been received
    End If
    If Len(Buffer$) > 512 Then 'keep buffer limited in size
      Buffer$ = Mid(Buffer$, 257)
    End If
  End If

'event messages
Case comEvCTS
  EVMsg = "Change in CTS Detected"
Case comEvDSR
  EVMsg = "Change in DSR Detected"
Case comEvCD
  EVMsg = "Change in CD Detected. "
Case comEvRing
  EVMsg = "The Phone is Ringing"
Case comEvEOF
  EVMsg = "End of File Detected"
Case comEventBreak
  EVMsg = "Break Received"
'--- Error messages
Case comEventCTSTO
```

```
            ERMsg = "CTS Timeout"
        Case comEventDSRTO
            ERMsg = "DSR Timeout"
        Case comEventFrame
            EVMsg = "Framing Error"
        Case comEventOverrun
            PTT True
            ERMsg = "Overrun Error"
        Case comEventCDTO
            ERMsg = "Carrier Detect Timeout"
        Case comEventRxOver
            ERMsg = "Receive Buffer Overflow"
        Case comEventRxParity
            EVMsg = "Parity Error"
        Case comEventTxFull
            ERMsg = "Transmit Buffer Full"
       End Select

    If Len(EVMsg) Then
        '--- Display event messages in label
        lblStatus.Caption = EVMsg
        EVMsg = ""
    ElseIf Len(ERMsg) Then
        '--- Display error messages in an alert message box
        Beep
        Ret = MsgBox(ERMsg, vbOKCancel, "Press Cancel to Quit, Ok to ignore.")
        ERMsg = ""
        '--- If Cancel (2) was pressed
        If Ret = vbCancel Then
            MSComm1.PortOpen = False    'Close the port and quit
        End If
    End If
End Sub

'Command Prompt -- Prompt user of command that has been executed
Private Sub CommandPrompt(Data As String)
    'Depending on incoming command, update appropriate text in command
    'prompt or other labels.  Update any on off lights necessary.
    Select Case Data
        Case "#"
            tmrPing.Enabled = False
            lblPing.Caption = "Pinged!"
            Beep
        Case "a"
            txtCmdPrompt.Text = "Reverse: 4" & vbCrLf & _
            txtCmdPrompt.Text
```

```
Case "b"
   txtCmdPrompt.Text = "Reverse: 3" & vbCrLf & _
   txtCmdPrompt.Text
Case "c"
   txtCmdPrompt.Text = "Reverse: 2" & vbCrLf & _
   txtCmdPrompt.Text
Case "d"
   txtCmdPrompt.Text = "Reverse: 1" & vbCrLf & _
   txtCmdPrompt.Text
Case "e"
   txtCmdPrompt.Text = "Stop" & vbCrLf & _
   txtCmdPrompt.Text
Case "f"
   txtCmdPrompt.Text = "Drive: 1" & vbCrLf & _
   txtCmdPrompt.Text
Case "g"
   txtCmdPrompt.Text = "Drive: 2" & vbCrLf & _
   txtCmdPrompt.Text
Case "h"
   txtCmdPrompt.Text = "Drive: 3" & vbCrLf & _
   txtCmdPrompt.Text
Case "i"
   txtCmdPrompt.Text = "Drive: 4" & vbCrLf & _
   txtCmdPrompt.Text
Case "j"
   txtCmdPrompt.Text = "Left: 3" & vbCrLf & _
   txtCmdPrompt.Text
Case "k"
   txtCmdPrompt.Text = "Left: 2" & vbCrLf & _
   txtCmdPrompt.Text
Case "l"
   txtCmdPrompt.Text = "Left: 1" & vbCrLf & _
   txtCmdPrompt.Text
Case "m"
   txtCmdPrompt.Text = "Straight" & vbCrLf & _
   txtCmdPrompt.Text
Case "n"
   txtCmdPrompt.Text = "Right: 1" & vbCrLf & _
   txtCmdPrompt.Text
Case "o"
   txtCmdPrompt.Text = "Right: 2" & vbCrLf & _
   txtCmdPrompt.Text
Case "p"
   txtCmdPrompt.Text = "Right: 3" & vbCrLf & _
   txtCmdPrompt.Text
Case "q"
```

```
        txtCmdPrompt.Text = "Camera Activated" & vbCrLf & _
        txtCmdPrompt.Text
        shpTVON.BackColor = Green
        shpTVOFF.BackColor = DkRed
    Case "r"
        txtCmdPrompt.Text = "Camera Deactivated" & vbCrLf & _
        txtCmdPrompt.Text
        shpTVON.BackColor = DkYellow
        shpTVOFF.BackColor = Red
    Case "s"
        txtCmdPrompt.Text = "Compass Activated" & vbCrLf & _
        txtCmdPrompt.Text
        shpCompON.BackColor = Green
        shpCompOFF.BackColor = DkRed
    Case "t"
        txtCmdPrompt.Text = "Compass Deactivated" & vbCrLf & _
        txtCmdPrompt.Text
        shpCompON.BackColor = DkYellow
        shpCompOFF.BackColor = Red
    Case "u"
        txtCmdPrompt.Text = "Inclinometer Activated" & vbCrLf & _
        txtCmdPrompt.Text
        shpIncON.BackColor = Green
        shpIncOFF.BackColor = DkRed
    Case "v"
        txtCmdPrompt.Text = "Inclinometer Deactivated" & vbCrLf & _
        txtCmdPrompt.Text
        shpIncON.BackColor = DkYellow
        shpIncOFF.BackColor = Red
    Case "w"
        txtCmdPrompt.Text = "Accelerometer Activated" & vbCrLf & _
        txtCmdPrompt.Text
        shpAccON.BackColor = Green
        shpAccOFF.BackColor = DkRed
    Case "x"
        txtCmdPrompt.Text = "Accelerometer Deactivated" & vbCrLf & _
        txtCmdPrompt.Text
        shpAccON.BackColor = DkYellow
        shpAccOFF.BackColor = Red
End Select

'goto subroutine if q through x was received
If Data >= "q" And Data <= "x" Then
    UpdateStatusBox
End If
```

```
End Sub

'Send Turn Command - 3 left turn radii, straight, and 3 right turn radii
Private Sub sendTurnCommand(turn As Integer)
    Dim strCommand As String

    'Pick appropriate turn command based on joystick position
    If (turn > 500) Then
        strCommand = "p" 'right 3
    ElseIf (turn <= 500 And turn > 300) Then
        strCommand = "o" 'right 2
    ElseIf (turn <= 300 And turn > 100) Then
        strCommand = "n" 'right 1
    ElseIf (turn <= 100 And turn >= -100) Then
        strCommand = "m" 'straight
    ElseIf (turn < -100 And turn >= -300) Then
        strCommand = "l" 'left 1
    ElseIf (turn < -300 And turn >= -500) Then
        strCommand = "k" 'left 2
    Else '(turn < -500)
        strCommand = "j" 'left 3
    End If

   'Send the command
    DataReqFlag = 0
    EnCode Asc(strCommand)
End Sub

'Send Drive command - Forward speeds 1-4, stop, and reverse 1-4
Private Sub sendDriveCommand(drive As Integer, pButton As Integer)
   Dim strCommand As String

   'Pick right command based on button pressed and joystick position
   If (drive > 300) Then
       If pButton = 1 Then
           strCommand = "f" 'drive 1
       ElseIf pButton = 4 Then
           strCommand = "g" 'drive 2
       ElseIf pButton = 8 Then
           strCommand = "h" 'drive 3
       ElseIf pButton = 2 Then
           strCommand = "i" 'drive 4
       End If
   ElseIf (drive <= 300 And drive >= -300) Then
       strCommand = "e" 'stop
   Else '(drive < -300)
```

```
        If pButton = 1 Then
            strCommand = "d" 'reverse 1
        ElseIf pButton = 4 Then
            strCommand = "c" 'reverse 2
        ElseIf pButton = 8 Then
            strCommand = "b" 'reverse 3
        ElseIf pButton = 2 Then
            strCommand = "a" 'reverse 4
        End If
    End If

    'Send the command
    DataReqFlag = 0
    EnCode Asc(strCommand)
End Sub

'Process the Accelerometer Data
Private Sub ProcessAcc(AccData As String)
    'check to see if received data is valid
    If Mid(AccData, 1, 1) = "A" And IsNumeric(Mid(AccData, 2, 6)) Then
        'update command prompt
        txtCmdPrompt.Text = "Accelerometer Polled" & vbCrLf & _
            txtCmdPrompt.Text

        'insert data into label box for further manipulation -- label box
        'is not visible on form
        lblAccData.Caption = AccData
        'manipulate data - normalize data and update vectors
        'X acceleration is 2nd and 3rd character
        lblXdata.Caption = (Val(Mid(AccData, 2, 2)) / 10) - 5
            linXVect.X2 = linXVect.X1 + (lblXdata.Caption * 400)
        'Y acceleration is 3rd and 4th character
        lblYdata.Caption = ((Val(Mid(AccData, 4, 2)) / 10) - 5) * (-1#)
            linYVect.Y2 = linYVect.Y1 + (lblYdata.Caption * 283)
            linYVect.X2 = linYVect.X1 - (lblYdata.Caption * 283)
        'Z acceleration is 5th and 6th character
        lblZdata.Caption = (Val(Mid(AccData, 6, 2)) / 10) - 5
            linZVect.Y2 = linZVect.Y1 + (lblZdata.Caption * -400)
    Else
        'invalid data - update command prompt
        txtCmdPrompt.Text = "Accelerometer Polled - Invalid data received" & _
            vbCrLf & txtCmdPrompt.Text
    End If
End Sub

'Process the Compass Data
```

```
Private Sub ProcessComp(CompData As String)
   Dim NormalizedDeg As Integer

   'check to see if received data is valid
   If Mid(CompData, 1, 1) = "B" And IsNumeric(Mid(CompData, 2, 3)) Then
      'update command prompt
      txtCmdPrompt.Text = "Compass Polled" & vbCrLf & _
         txtCmdPrompt.Text

      'insert data into label box for further manipulation -- label box
      'is not visible on form
      lblCompData.Caption = CompData
      'normalize data
      lblDegData.Caption = Val(Mid(CompData, 2, 3)) - 100
      NormalizedDeg = lblDegData.Caption + 90
      'move needle
      linNeedle.X1 = 1440 - Cos(NormalizedDeg * PI / 180) * 600
      linNeedle.Y1 = 1200 - Sin(NormalizedDeg * PI / 180) * 600
   Else
      'invalid data - update command prompt
      txtCmdPrompt.Text = "Compass Polled - Invalid data received" & _
         vbCrLf & txtCmdPrompt.Text
   End If
End Sub

'Process the Tilt sensor Data
Private Sub ProcessTilt(TiltData As String)
   'check to see if received data is valid
   If Mid(TiltData, 1, 1) = "C" And IsNumeric(Mid(TiltData, 2, 4)) Then
      'update command prompt
      txtCmdPrompt.Text = "Inclinometer Polled" & vbCrLf & _
         txtCmdPrompt.Text

      'insert data into label box for further manipulation -- label box
      'is not visible on form
      lblTiltData.Caption = TiltData
      'move rover's "pitch mockup" - lines and wheels
      lblPitchData.Caption = Val(Mid(TiltData, 2, 2)) - 50 - 6
         linPitch.X1 = 1920 - Cos(lblPitchData.Caption * PI / 180) * 960
         linPitch.X2 = 1920 + Cos(lblPitchData.Caption * PI / 180) * 960
         linPitch.Y1 = 960 + Sin(lblPitchData.Caption * PI / 180) * 960
         linPitch.Y2 = 960 - Sin(lblPitchData.Caption * PI / 180) * 960
         BackWheel.Left = linPitch.X1 - 120
         BackWheel.Top = linPitch.Y1 + 240
         FrontWheel.Left = linPitch.X2 - 120
         FrontWheel.Top = linPitch.Y2 + 240
```

```
        'move rover's "roll mockup" - lines and wheels
        lblRollData.Caption = Val(Mid(TiltData, 4, 2)) - 50 - 10
            linRoll.X1 = 1920 - Cos(lblRollData.Caption * PI / 180) * 600
            linRoll.X2 = 1920 + Cos(lblRollData.Caption * PI / 180) * 600
            linRoll.Y1 = 2880 + Sin(lblRollData.Caption * PI / 180) * 600
            linRoll.Y2 = 2880 - Sin(lblRollData.Caption * PI / 180) * 600
            LeftWheel.X1 = linRoll.X1
            LeftWheel.X2 = linRoll.X1
            LeftWheel.Y1 = linRoll.Y1 + 240
            LeftWheel.Y2 = linRoll.Y1 + 600
            RightWheel.X1 = linRoll.X2
            RightWheel.X2 = linRoll.X2
            RightWheel.Y1 = linRoll.Y2 + 240
            RightWheel.Y2 = linRoll.Y2 + 600
    Else
        'invalid data - update command prompt
        txtCmdPrompt.Text = "Inclinometer Polled - Invalid data received" & _
            vbCrLf & txtCmdPrompt.Text
    End If
End Sub

'Update Sensor status
Private Sub UpdateStatusBox()
    'Update status label based on the lights that are currently on
    If shpTVON.BackColor = Green Then
        lblSensorStat.Caption = "Camera ON" & vbCrLf
    Else
        lblSensorStat.Caption = "Camera OFF" & vbCrLf
    End If
    If shpCompON.BackColor = Green Then
        lblSensorStat.Caption = "Compass ON" & vbCrLf & _
        lblSensorStat.Caption
    Else
        lblSensorStat.Caption = "Compass OFF" & vbCrLf & _
        lblSensorStat.Caption
    End If
    If shpIncON.BackColor = Green Then
        lblSensorStat.Caption = "Inclinometer ON" & vbCrLf & _
        lblSensorStat.Caption
    Else
        lblSensorStat.Caption = "Inclinometer OFF" & vbCrLf & _
        lblSensorStat.Caption
    End If
    If shpAccON.BackColor = Green Then
        lblSensorStat.Caption = "Accelerometer ON" & vbCrLf & _
        lblSensorStat.Caption
```

```
    Else
        lblSensorStat.Caption = "Accelerometer OFF" & vbCrLf & _
        lblSensorStat.Caption
    End If
End Sub

'Sensor update - update sensors based on message received from rover
'----
'If the base station has not sent any message to the rover for 3 minutes
'the rover will have entered quiet mode and shut down all sensors.  This
'routine can be used to update the status of the sensors.  The light displays
'may reflect that some sensors are still on, even though the rover has entered
'quiet mode.
'----
Private Sub SensorUpdate(S1 As Byte, S2 As Byte, S3 As Byte, S4 As Byte)
    'update command prompt
    txtCmdPrompt.Text = "Sensor Status Updated" & vbCrLf & _
    txtCmdPrompt.Text

    'S1, S2, S3, S4 come from the upper four bits of the message that was sent
    If S1 Then
        lblSensorStat.Caption = "Camera ON" & vbCrLf
        shpTVON.BackColor = Green
        shpTVOFF.BackColor = DkRed
    Else
        lblSensorStat.Caption = "Camera OFF" & vbCrLf
        shpTVON.BackColor = DkYellow
        shpTVOFF.BackColor = Red
    End If
    If S2 Then
        lblSensorStat.Caption = "Compass ON" & vbCrLf & _
        lblSensorStat.Caption
        shpCompON.BackColor = Green
        shpCompOFF.BackColor = DkRed
    Else
        lblSensorStat.Caption = "Compass OFF" & vbCrLf & _
        lblSensorStat.Caption
        shpCompON.BackColor = DkYellow
        shpCompOFF.BackColor = Red
    End If
    If S3 Then
        lblSensorStat.Caption = "Inclinometer ON" & vbCrLf & _
        lblSensorStat.Caption
        shpIncON.BackColor = Green
        shpIncOFF.BackColor = DkRed
    Else
```

```
        lblSensorStat.Caption = "Inclinometer OFF" & vbCrLf & _
        lblSensorStat.Caption
        shpIncON.BackColor = DkYellow
        shpIncOFF.BackColor = Red
    End If
    If S4 Then
        lblSensorStat.Caption = "Accelerometer ON" & vbCrLf & _
        lblSensorStat.Caption
        shpAccON.BackColor = Green
        shpAccOFF.BackColor = DkRed
    Else
        lblSensorStat.Caption = "Accelerometer OFF" & vbCrLf & _
        lblSensorStat.Caption
        shpAccON.BackColor = DkYellow
        shpAccOFF.BackColor = Red
    End If
End Sub

'Ping time out - event occurs if ping is not received during 600ms receive mode
Private Sub tmrPing_Timer()
    lblPing.Caption = "Timed Out"
    tmrPing.Enabled = False
End Sub

'Routine for automatically transmitting call sign every 10 minutes.
'The timer is triggered after every minute, then counter counts down
'from 10, giving 10 minutes.  The VB timer control has a maximum countdown
'time of about 65 seconds.  As a result, a counter and timer were used
'in combination.
Private Sub tmrTXCallSign_Timer()
    If (TXCallSignCnt > 0) Then '10 minutes have not elapsed
        TXCallSignCnt = TXCallSignCnt - 1 '1 min elapsed, decrement counter
    Else
        TXCallSignCnt = 10 're-initialize to 10, for 10 minute count
        PTT True 'turn to Transmit mode
        MSComm1.Output = "KB3KDM" 'Transmit John P. Falcone's call sign
    End If
End Sub

'RX mode timer - if a message is not received, RIA returns to TX mode
Private Sub TXTimeOut_Timer()
    TXTimeOut.Enabled = False 'turn timer off
    HeadFlag = False 'reset header flag
    'update command prompt
    txtCmdPrompt.Text = "Transmission Timed Out" & vbCrLf & _
    txtCmdPrompt.Text
```

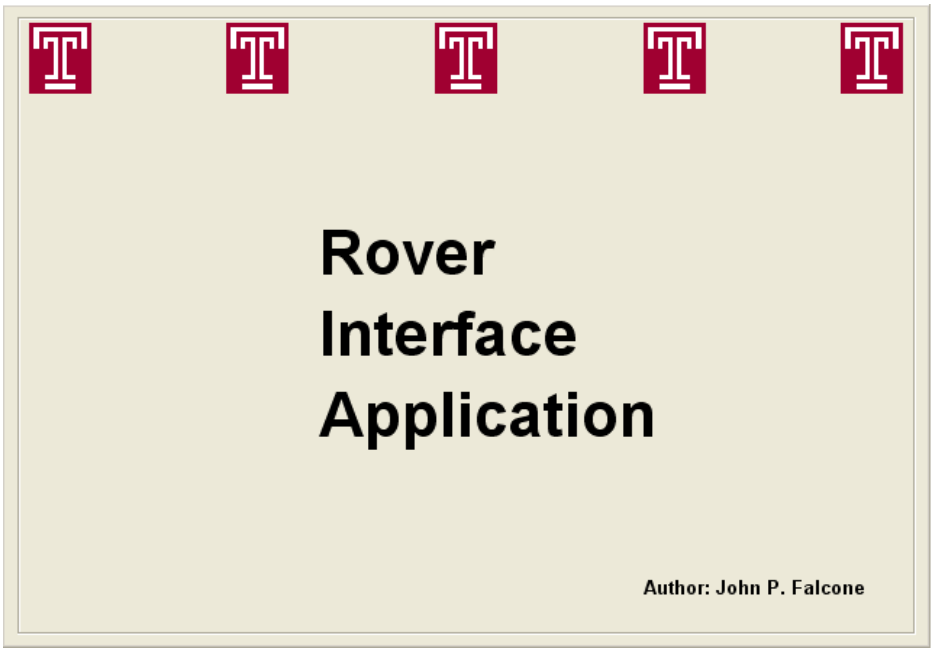
```
   PTT True 'revert to transmit
End Sub

'Push to talk routine - Key or unkey the mic, fix the PTT light
Private Sub PTT(TX As Boolean)
   If TX Then
      MSComm1.DTREnable = True 'key mic
      shpTX.BackColor = Red 'Red light for transmitting
      Sleep 150 'delay to allow rover to unsquelch and go to receive
      TXFlag = False 'transmission complete
   Else
      TXFlag = True 'transmission in progress
      MSComm1.DTREnable = False 'unkey mic
      shpTX.BackColor = DkYellow 'subdued light for receiving
   End If
End Sub

'Hamming encoding routine
Private Sub EnCode(Command As Integer)
   Dim S(10) As Integer
   Dim M(1) As Byte
   Dim Index As Byte

   S(2) = (Command And &H1) * 4     'S(2) = 00000000 00000x00
   S(4) = (Command And &H2) * 8     'S(4) = 00000000 000x0000
   S(5) = (Command And &H4) * 8     'S(5) = 00000000 00x00000
   S(6) = (Command And &H8) * 8     'S(6) = 00000000 0x000000
   S(8) = (Command And &H10) * 16   'S(8) = 0000000x 00000000
   S(9) = (Command And &H20) * 16   'S(9) = 000000x0 00000000
   S(10) = (Command And &H40) * 16 'S(10) = 00000x00 00000000

   S(0) = (S(2) \ 4) Xor _
          (S(4) \ 16) Xor _
          (S(6) \ 64) Xor _
          (S(8) \ 256) Xor _
          (S(10) \ 1024)          'S(0) = 00000000 0000000c
   S(1) = (S(2) \ 2) Xor _
          (S(5) \ 16) Xor _
          (S(6) \ 32) Xor _
          (S(9) \ 256) Xor _
          (S(10) \ 512)           'S(1) = 00000000 000000c0
   S(3) = (S(4) \ 2) Xor (S(5) \ 4) Xor (S(6) \ 8)  'S(3) = 00000000 0000c000
   S(7) = (S(8) \ 2) Xor (S(9) \ 4) Xor (S(10) \ 8) 'S(7) = 00000000 c0000000

   For Index = 1 To 10
      S(0) = S(0) Or S(Index)
```

```
    Next 'S(0) = 00000xxx CxxxCxCC

    'segment S into two bytes
    M(0) = S(0) And &HFF
    M(1) = (S(0) And &HFF00) \ 256

    Transmit M 'transmit M
End Sub

'Transmit the command
Private Sub Transmit(M() As Byte)
    'send essage with 2 header bytes
    MSComm1.Output = Chr$(254) & Chr$(255) & Chr$(M(0)) & Chr$(M(1))  'send message
    Sleep 60   'delay 60 ms
    PTT False  'delay finished--turn off PTT
    TXTimeOut.Enabled = True  'start up timer for turning back to TX mode
End Sub

'Hamming Decoding and error correcting routine
Private Sub DeCode(M As Variant, Size As Byte)
    Dim R(80) As Byte
    Dim Index, Count, UpperBound, N As Byte
    Dim ShiftX As Integer
    Dim Y(6) As Byte
    Dim Error, Message As String
    Dim C As Byte
    Dim C0, C1, C3, C7 As Boolean

    'determine how long the message is
    'UpperBound + 1 is number of byes in message
    'N + 1 is number 11 bit blocks
    Select Case Size
        Case 1 'Accelerometer
            UpperBound = 9
            N = 6
        Case 2 'Compass
            UpperBound = 5
            N = 3
        Case 3 'Tilt
            UpperBound = 6
            N = 4
        Case Else 'all other commands
            UpperBound = 1
            N = 0
    End Select
```

```
'segment message into bits -- loop through UpperBound + 1 times
For Count = 0 To UpperBound
   ShiftX = 1
   For Index = 0 To 7
      R(Index + (Count * 8)) = (M(Count) \ ShiftX) And &H1
      ShiftX = ShiftX * 2
   Next
Next

'Decode message -- loop N + 1 times
For Count = 0 To N
   Index = Count * 11
   C0 = R(0 + Index) Xor R(2 + Index) Xor _
      R(4 + Index) Xor R(6 + Index) Xor _
      R(8 + Index) Xor R(10 + Index)
   C1 = R(1 + Index) Xor R(2 + Index) Xor _
      R(5 + Index) Xor R(6 + Index) Xor _
      R(9 + Index) Xor R(10 + Index)
   C3 = R(3 + Index) Xor R(4 + Index) Xor _
      R(5 + Index) Xor R(6 + Index)
   C7 = R(7 + Index) Xor R(8 + Index) Xor _
      R(9 + Index) Xor R(10 + Index)

   Y(0) = R(2 + Index) Xor (C0 And C1 And Not C3 And Not C7)
   Y(1) = R(4 + Index) Xor (C0 And Not C1 And C3 And Not C7)
   Y(2) = R(5 + Index) Xor (Not C0 And C1 And C3 And Not C7)
   Y(3) = R(6 + Index) Xor (C0 And C1 And C3 And Not C7)
   Y(4) = R(8 + Index) Xor (C0 And Not C1 And Not C3 And C7)
   Y(5) = R(9 + Index) Xor (Not C0 And C1 And Not C3 And C7)
   Y(6) = R(10 + Index) Xor (C0 And C1 And Not C3 And C7)

   'determine if an error was received
   C = 0
   If C0 Then
      C = 1
   End If
   If C1 Then
      C = C Or 2
   End If
   If C3 Then
      C = C Or 4
   End If
   If C7 Then
      C = C Or 8
   End If
   'C still equals zero here if no error was received
```

```
        If C <> 0 Then 'increment number of errors
            lblNumOfErrs.Caption = Val(lblNumOfErrs.Caption) + 1
        End If

        ShiftX = 2 'pack bits back into a byte
        For Index = 1 To 6
            Y(0) = Y(0) Or (Y(Index) * ShiftX)
            ShiftX = ShiftX * 2
        Next

        'concatenate byte to the message string
        Message = Message & Chr$(Y(0))
        Error = Error & Hex(C) 'concatenate error to the error string

    Next

    'put message and error into labels
    lblMessageString.Caption = Message
    lblErrorString.Caption = Error

    'depending on size call appropriate routine
    Select Case Size
        Case 1
            ProcessAcc Message
        Case 2
            ProcessComp Message
        Case 3
            ProcessTilt Message
        Case 4
            If Message = "D" Then 'sensor update - pass upper 4 bits to routine
                SensorUpdate R(12), R(13), R(14), R(15)
            End If
        Case Else
            CommandPrompt Message
    End Select

End Sub
```

***frmSplash*:**



***frmSplash* source code:**

```
'--------------------------------------------------------------------------
'Boot up Splash Screen -- Displays name of application and author
'Temple Logos scroll across top of screen
'--------------------------------------------------------------------------
Option Explicit

'When form is activated scroll logos across top
Private Sub Form_Activate()
    Dim pintCounter As Integer
    Dim pintLeft As Integer
    Dim pintLNum As Integer

    '5 logos scroll across
    For pintLNum = 0 To 4
        picLogo(pintLNum).Visible = True
        pintLeft = picLogo(pintLNum).Left
        'Move logo from left to right
        For pintCounter = 1 To (pintLeft / 16)
            picLogo(pintLNum).Move (pintCounter * 16), 150
            frmSplash.Refresh
        Next
    Next
```

```
End Sub

'timer triggers after three seconds - unload splash form after 3 seconds
'goto main rover form
Private Sub tmr1_Timer()
    frmRover.Show
    Unload frmSplash
End Sub
```

***publicRIA* source code:**

```
'--------------------------------------------------------------------------------
'Public module--This module contains: public variables used by multiple forms,
'  declarations of constants, functions that are used by multiple
'  forms, and a declaration to call a function from the kernel32 library.
'--------------------------------------------------------------------------------
Option Explicit

'Allow Sleep function to be used from kernel32 library
Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'Public variables
Public Xref As Long
Public Yref As Long
Public Xcenter As Long
Public Ycenter As Long
Public Xmax As Long
Public Xmin As Long
Public Ymax As Long
Public Ymin As Long
Public pButton As Integer
Public Data() As Byte
Public TXCallSignCnt As Integer

'Declare constants
Public Const PI = 3.14159265
Public Const DkRed = &H80&
Public Const DkBlue = &H800000
Public Const DkGreen = &H8000&
Public Const DkYellow = &H8080&
Public Const Red = &HFF&
Public Const Blue = &HFF0000
Public Const Green = &HFF00&
Public Const Yellow = &HFFFF&
Public Const Orange = &H80FF&
```

```
'Find the max of two numbers
Public Function max(a As Long, b As Long) As Variant
   If (a > b) Then
      max = a
   Else
      max = b
   End If
End Function

'Find the min of two numbers
Public Function min(a As Long, b As Long) As Variant
   If (a > b) Then
      min = b
   Else
      min = a
   End If
End Function

'Normalize the joystick values
Public Function Normalize(N, max, min, center, HtWd) As Variant
   Normalize = (N - center) * (HtWd - 150) / (max - min)
End Function

'Joystick movement routine
Public Sub JoystickMove(X, Y, Form)
   Xref = Normalize(X, Xmax, Xmin, Xcenter, Form.shpJstField.Width)
   Yref = Normalize(Y, Ymax, Ymin, Ycenter, Form.shpJstField.Height)

   'move crosshairs to current joystick position

   'calculate X coordinate value and move
   Dim pintCursorX As Integer
   pintCursorX = Xref + Form.shpJstField.Left + (Form.shpJstField.Width - 150) / 2
   If pintCursorX <= Form.shpJstField.Left + 15 Then
      Form.fraJstPtr.Left = Form.shpJstField.Left + 15
   ElseIf pintCursorX >= Form.shpJstField.Left + Form.shpJstField.Width - _
         Form.fraJstPtr.Width - 15 Then
      Form.fraJstPtr.Left = Form.shpJstField.Left + Form.shpJstField.Width - _
         Form.fraJstPtr.Width - 15
   Else
      Form.fraJstPtr.Left = pintCursorX
   End If

   'calculate Y coordinate value and move
   Dim pintCursorY As Integer
   pintCursorY = Yref + Form.shpJstField.Top + (Form.shpJstField.Height - 150) / 2
```

```
    If pintCursorY <= Form.shpJstField.Top + 15 Then
       Form.fraJstPtr.Top = Form.shpJstField.Top + 15
    ElseIf pintCursorY >= Form.shpJstField.Top + Form.shpJstField.Height - _
          Form.fraJstPtr.Height - 15 Then
       Form.fraJstPtr.Top = Form.shpJstField.Top + Form.shpJstField.Height - _
          Form.fraJstPtr.Height - 15
    Else
       Form.fraJstPtr.Top = pintCursorY
    End If

    'Put numerical X and Y coordinate values into text boxes
    Form.txtXpos.Text = Form.fraJstPtr.Left + (Form.fraJstPtr.Width / 2) _
       - Form.shpJstField.Left - (Form.shpJstField.Width / 2)
    Form.txtYpos.Text = -Form.fraJstPtr.Top - (Form.fraJstPtr.Height / 2) _
       + Form.shpJstField.Top + (Form.shpJstField.Height / 2)

End Sub

'Joystick button press routine
Public Sub JoystkButton(Button, Form)
    'get value of button pressed
    pButton = Button
    'update button lights
    If (Button And 1) Then
       Form.Shape1(0).BackColor = Red
       Form.Shape2(0).BackColor = DkRed
    Else
       Form.Shape1(0).BackColor = DkRed
       Form.Shape2(0).BackColor = Red
    End If
    If (Button And 2) Then
       Form.Shape1(1).BackColor = Blue
       Form.Shape2(1).BackColor = DkBlue
    Else
       Form.Shape1(1).BackColor = DkBlue
       Form.Shape2(1).BackColor = Blue
    End If
    If (Button And 4) Then
       Form.Shape1(3).BackColor = Yellow
       Form.Shape2(3).BackColor = DkYellow
    Else
       Form.Shape1(3).BackColor = DkYellow
       Form.Shape2(3).BackColor = Yellow
    End If
    If (Button And 8) Then
       Form.Shape1(2).BackColor = Green
```

```
        Form.Shape2(2).BackColor = DkGreen
    Else
        Form.Shape1(2).BackColor = DkGreen
        Form.Shape2(2).BackColor = Green
    End If
End Sub
```

Section 15: Appendix D – Resumés

John P. Falcone

3800 Manayunk Ave, 2nd Floor (267) 240-6397
Philadelphia, PA 19128-5108 jfalcone@temple.edu

Objective

To obtain a Master of Science and a Doctor of Philosophy in Engineering at Temple University's College of Engineering, department of Electrical and Computer Engineering

Education

Temple University, Philadelphia, PA 2001-present
Bachelor of Science in Engineering, Electrical and Computer Engineering
Current GPA: 4.0
Expected graduation: Spring 2004

Delaware County Community College, Media, PA 2000-2001
Associate in Arts, Liberal Arts
GPA: 4.0

Central Texas College, Killeen, TX 1999
Associate in General Studies
GPA: 3.875

U.S. Army, Non-Commissioned Officer Academy, Fort Hood, TX 1998
Diploma, Primary Leadership Development Course

U.S. Army, Tactical Assembly Area Lion, Kuwait 1995
Certificate, Combat Lifesaver Course

Honors, Awards, and Organizations

Eta Kappa Nu, Electrical and Computer Engineering Honors Society – member

National Society of Collegiate Scholars – member

Boeing Engineering Scholarship – awardee

Institute of Electrical and Electronics Engineers – member

Experience

Temple University, Electrical Engineering Dept., Philadelphia, PA 2002-present
Tutor

- Provide individual instruction and guidance to electrical engineering under-graduate students
- Prepare students for examination on subjects such as nodal and mesh analysis, discrete and continuous signals, Laplace transforms, Fourier transforms, and digital design

Temple University, Speech Processing Laboratory, Philadelphia, PA 2003
Research Assistant/Electrical Engineer

- Assisted in research and development of usable speech measures
- Attended weekly research seminars to review professional papers
- Created a digital phoneme library for use in speech identification

Temple University, Geology Department, Philadelphia, PA 2002
Electrical Engineer

- Assembled an electronic system to automatically log data such as temperature and pressure
- Troubleshot and tested devices using various lab equipment and computer interface
- Coordinated with an outside institution to have hex code loaded to microprocessors

U.S. Army, C company 1-8 Cavalry, Fort Hood, TX 1995-2000
Armor Crewman

- Trained and performed in various positions on an M1A1 and M1A2 tank.
- Worked together as part of a 4-man crew and on a larger scale as part of a 14-tank company
- Operated and maintained a 68-ton, $6.5 million, tracked vehicle during live fire drills and qualification courses

2080 Peevy Rd
East Greenville, PA 18041

215-767-0132
sherman@temple.edu

STEVEN D. HERMAN

Summary: Temple University Electrical Engineering graduate with 3 years of corporate computer networking and troubleshooting experience. Work well in teams, and independently.

Education: B.S. in Electrical Engineering at Temple University. Expected May 2004.

Capstone Design: Was team leader in a team of four that designed a semi-autonomous data collecting rover that communicated wirelessly with a base station.

Relevant Skills:

Control Systems	Matlab	DOS
Netware 4 and 5	Microsoft Networking	C, C++
Microsoft Office 98,NT,2K,XP	Assembly Language	Windows

Certifications:

Cisco Certified Network Associate (CCNA) June 2000
Took a year long class in high school to learn modern computer networking and prepare for the CCNA exam.

Novell Certified Network Administrator (CNA) August 2001
Taught myself the necessary information using textbooks and practice exams.

Employment

2/03 – Present
Tutor
Temple University ACT 101 Program Philadelphia, PA
- Tutored for engineering and upper level math students.

6/03 – 2/04
Technology Specialist
Germantown Academy Fort Washington, PA
- Supported over 150 PC's with one other person
- Successfully deployed a number of IT projects

12/99 – 8/02
IT Analyst and Helpdesk Operator
Accu-Sort Systems Telford, PA
- Responsible for receiving all IT support calls, troubleshooting them and entering them in the support database.
- Played major role in decreasing the number of simultaneous unresolved support calls by over 50%
- Participated in teams to improve overall departmental efficiency, including hardware and software tracking.

9/00 – 5/02
Computer Lab Consultant/Technician
Temple University Ambler Computer Services Ambler, PA
- Went out on jobs to troubleshoot and repair PC issues throughout the campus.
- Assisted students in the computer lab

307 Pearl Avenue
Horsham, PA 19044
(215) 672-4672
JFDessino@aol.com

John F. Dessino

EDUCATION

Nanofabrication Manufacturing Technologies (NMT) Program
Pennsylvania State University
State College, PA
August 2001

A.S. in Engineering Sciences
Montgomery County Community College
Blue Bell, PA
May 2001

CURRENT EDUCATION

B.S. in Electrical Engineering
Temple University
Philadelphia, PA

EXPERIENCE

- Trained in a class 10/100 cleanroom
- NMT machines trained on:
  - -Applied Materials P5000 Magnetically Enhanced Reactive Ion Etcher (MERIE)
  - -Plasma Therm SLR 720 Reactive Ion Etcher (RIE)
  - -Plasma Therm SLR 770 Electron Cyclotron Resonance (ECR) Etcher
  - -APX magnetron sputtering system
  - -Karl Suss MJ3 contact lithography system.
- NMT machines with experience on:
  - -Kiethly CV/IV meter
  - -Leica Scanning Electron Microscope (SEM)
  - -Tencor Alpha-step 500 Profilometer
  - -AET Rapid Thermal Annealer (RTA)

COMPUTER SKILLS

Matlab
SystemView

EMPLOYMENT

04/02 – Present Target Corporation
Electronics Department Sales Associate
-Answers customer questions on software and electronic devices

Kemi Ashebu

7701 Revere Street, Apt B11 Philadelphia, PA 19149.
(215) 624 6417 (267) 216 7778. Cell
kemash@temple.edu

OBJECTIVE:

To obtain an internship position in an Electrical Engineering field that would display my engineering and leadership skills

EDUCATION:

Temple University, Philadelphia, PA

Major: Electrical Engineering Expected graduate date: Dec 2004
Cumulative GPA: 3.07 out of 4

Canberra Institute of Technology, Canberra Australia

Major: Electronic Engineering June 00 – Sept 01

RELATED COURSE WORK:

Signal and Systems	Engineering Static	Electromagnetic fields
Programming in C	Microelectronics	Microprocessors
Probability	Differential Equations	Classical and Modern Controls

Telecommunication
Senior design project: Designing Digital Wireless Data Communications with semi autonomous robot

SKILLS:

Operating Systems: Windows 98, Windows 2000, and Mac
Software: Matlab, Maple, System view, Proficient in Microsoft office suite.

WORK EXPERIENCE:

Upwards bound Temple University **Summer 2003**
Science Teacher/Mentor

- Taught high school kid sciences over the summer: my leadership skills enabled the kids to achieve academic progress - 90% of the students passed at the end of the six weeks of tutoring them

Math & Science Research Center Temple University Sep 2002 –Present
Student Worker

- Assist students in acquiring knowledge in their math and science subjects
- Accurate input of information in the computer
- Performed general administrative duties
- Worked as a team with fellow tutors

Honors, Award & Organization:

* IEEE – "*Institute of Electronic and Electrical Engineers"* Member
* SWE – *"Society of Women Engineers"* Coordinator Chair
* NSBE – *"National Society of Black Engineers"* Member